UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA

EN INFORMÁTICA

PROYECTO FIN DE CARRERA

reTaskXML: Especificación de modelos de tareas a partir de especificaciones de interfaces de usuario

Abraham Martínez Martínez

Diciembre, 2007

UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA POLITÉCNICA SUPERIOR

Departamento de Sistemas Informáticos

PROYECTO FIN DE CARRERA

reTaskXML: Especificación de modelos de tareas a partir de especificaciones de interfaces de usuario

Autor: Abraham Martínez Martínez

Director: Francisco Montero Simarro

Diciembre, 2007

Dedicatoria y Agradecimientos

No quiero dejar pasar la oportunidad de agradecer a todas aquellas personas que de una manera u otra me han ayudado para la elaboración de este proyecto fin de carrera, así como las personas que he conocido y me han ayudado durante mis estudios de Ingeniería Técnica en Informática de Sistemas así como en los años de segundo ciclo de Ingeniería Informática. Sin su ayuda y apoyo no hubiese llegado hasta aquí.

Primeramente, me gustaría agradecer la dedicación, confianza y esfuerzo puesto en mi por mi director de Proyecto Francisco Montero, que me ha ayudado en todos los problemas que me han ido surgiendo y solucionado mis dudas.

Me gustaría agradecer a mi familia su paciencia y apoyo desinteresado y constante durante mi vida, ellos me han dado alas para seguir formándome.

A toda la gente que he conocido durante estos años de universidad en Albacete y de convivencia en las Residencias universitarias Campus y José Prat, han sido un ejemplo de amistad y espero haberles correspondido y seguir haciéndolo en el futuro de igual manera.

No quisiera olvidarme de todos mis amigos del pueblo, gente a la que conozco de toda la vida, que con su compañía y ánimos has hecho todo esto más llevadero.

Si al leer esto me he olvidado de ti al escribir estos agradecimientos, acepta mi agradecimiento y mis disculpas.

Resumen

Hoy en día, casi todo el mundo interactúa con ordenadores personales de una forma u otra. Los usamos ya sea en casa o en el trabajo, por entretenimiento, por la necesidad de búsqueda de información, o por querer dar a conocer nuestras opiniones o nuestros conocimientos.. En la actualidad, fruto de esta necesidad de interacción entre hombre y máquina siempre que alguien se siente delante de un ordenador lo manejará a través de una interfaz de usuario. Actualmente, se interactúa con los ordenadores, principalmente, usando un ratón, pulsado sobre iconos, y manipulando varias ventanas sobre la pantalla e interactuando sobre controles gráficos en ella ofrecidos.

Por otro lado, en Ingeniería del Software, existe la tendencia actual de trabajar utilizando modelos, arquitectura dirigida por modelos o MDA. Con esta tendencia de desarrollo es posible obtener unos modelos para, a partir de la información explicita de la especificación de la propia aplicación y llevar a cabo un proceso automático o semiautomático de generación de software, con el que finalmente el usuario podrá interactuar. En el desarrollo de interfaces de usuario se sigue una tendencia similar desde hace algunos años (principios de los noventa) y son distintos modelos y notaciones los que se utilizan para la especificación y generación de interfaces de usuario de una forma automática o casi automática.

Tomando como máxima esta tendencia y apuesta descarada por el modelo como elemento de especificación y desarrollo de software y de interfaces de usuario, en este proyecto final de carrera hemos querido centrarnos en el proceso inverso al natural que conlleva dichos desarrollos, es decir, aquí partiremos de modelos y obtendremos otros modelos, partiremos de especificaciones de interfaz de usuario a nivel abstracto y proporcionaremos especificaciones de tareas utilizando notaciones propuestas en Interacción Persona-Ordenador. Es decir, partiremos de especificaciones de la interfaz de usuario a nivel abstracto, que son independientes de la plataforma y de la forma de interactuar, y lograremos modelos, también independientes de la plataforma, donde la especificación estará centrada en la tarea. Con la realización de este proyecto se constatan las posibilidades adicionales que pueden ofrecer las herramientas de prototipado para el desarrollo y especificación de otros modelos de una forma transparente que tradicionalmente requieren el conocimiento de notaciones adicionales y con las que el diseñador debería familiarizarse. El uso de notaciones próximas a la especificación de interfaces de usuario sería mucho más intuitivo y directo para todos los diseñadores.

Este proyecto se encuadra dentro de las actividades llevadas a cabo dentro de la propuesta y definición de un marco de trabajo que permita la especificación y el desarrollo

de interfaces de usuario utilizando el lenguaje de especificación de interfaces de usuario propuesto con UsiXML (Limbourg et al., 2004).

Albacete, 2 de diciembre de 2007

Índice de Contenido

Índice de Figuras

Índice de tablas

Capítulo 1. INTRODUCCIÓN

1.1 Contexto y ámbito en el que se realiza este proyecto

La Interfaz de Usuario (IU en adelante), de un programa es el conjunto de elementos hardware y software de una computadora que presentan información al usuario y le permiten interactuar con la información y con el ordenador. También se puede considerar parte de la IU la documentación (manuales, ayuda, referencia, tutoriales) que acompaña al hardware y al software.

Si la IU está bien diseñada, el usuario encontrará la respuesta que espera a sus acciones. Si no es así puede resultar frustrante su interacción con la máquina, ya que el usuario habitualmente tiende a culparse a sí mismo por no saber usar cualquier dispositivo. El interés por las interfaces de usuario y por su desarrollo se resalta debido a que los programas se usan por usuarios con distintos niveles de conocimientos, desde principiantes hasta expertos, es por ello que no existe una interfaz válida para todos los usuarios y todas las tareas. Debe permitirse libertad al usuario para que elija el modo de interacción que más se adecue a sus objetivos en cada momento. La mayoría de los programas y sistemas operativos ofrecen varias formas de interacción al usuario.

La investigación en la interacción hombre-máquina ha tenido gran éxito y ha cambiado drásticamente el mundo de la computación. La interacción entre el ser humano y la máquina se realiza a través de una IU que facilita intercomunicación traduciendo del lenguaje binario de la máquina al lenguaje humano y viceversa, pudiendo ésta adoptar múltiples modalidades (gráfica, textual, vocal,...) e incluso combinarlas en las llamadas interfaces de usuario multimodales.

Las interfaces de usuario han evolucionado desde interfaces textuales, donde cada orden debía ser escrita usando el teclado, hasta interfaces de usuario gráficas de gran complejidad.

Las aplicaciones basadas en sistemas de información cuidan, cada vez más, sus interfaces de usuario. Atrás quedaron los tiempos donde los ordenadores eran máquinas para una élite: bastaba con que el sistema hiciese bien el trabajo no importando lo compleja que fuese la IU. Más allá de su finalidad esencial de interacción entre hombre y máquina, las interfaces de usuario conforman también la cara visible o escaparate de las aplicaciones tal y como las percibe el cliente final que las usa.

Como consecuencia directa, las interfaces de usuario entregadas en productos finales están cada vez más y más cuidadas para ofrecer al usuario sensaciones de

seguridad, fiabilidad, ergonomía, sencillez de uso y precisión en la aplicación suministrada. Todas estas cualidades se transmiten subliminalmente a través de la IU.

Desde el punto de vista del marketing podríamos considerar a las aplicaciones informáticas, como cualquier otro producto, entran por la vista. En este sentido, una cuidada presentación es esencial para promover su venta. No es de extrañar que, ante tal panorama, en la producción de software comercial cada vez sea más frecuente la inversión de mayores esfuerzos en el desarrollo de interfaces de usuario de alta calidad.

La creciente popularidad de los ordenadores y su uso cotidiano han cambiado la relación de los usuarios frente a la máquina y, por ende, lo que los usuarios esperan de ella. Nuevas formas de ordenadores como asistentes para datos personales(PDA), teléfonos móviles, sistemas empotrados o nuevas generaciones de electrodomésticos comienzan a popularizarse, obligándonos a interactuar con sus interfaces que no siempre son todo lo naturales ni intuitivas que uno hubiera deseado.

Los usuarios exigen cualidades al software destinadas a facilitar su trabajo, ahorra tiempo – de uso y de aprendizaje-, evitar y corregir los errores. Los procesos de desarrollo de aplicaciones dedican cada vez mayores esfuerzos a diseñar y mantener las interfaces de usuario de sus aplicaciones. Gran parte de este trabajo se realiza a mano por diseñadores y programadores que crean y dotan de funcionalidad a las interfaces construidas. Este proceso consume muchos recursos: personal, herramientas, tiempo y dinero.

Por otro lado, la emergente Sociedad de la información de este nuevo milenio incrementa día a día el volumen de datos producidos. Las aplicaciones necesarias para extraer información valiosa del océano de datos se convierten en una necesidad imperiosa. Los consumidores de aplicaciones demandan cada vez mejor y más aplicaciones en menos tiempo.

El panorama descrito se beneficia de cualquier nuevo avance que permita incrementar la productividad en el desarrollo de interfaces de usuario.

1.2 Motivación

Este trabajo tiene como motivación principal contribuir a la mejora de la calidad de las aplicaciones software, incidiendo sobre el diseño de la IU y la mejora de sus características de usabilidad y **reutilización**.

Herramientas conocidas y recogidas en este documento realizan la conversión de modelo de tareas (CTT) a especificación abstracta de IU (notación de UsiXML). El principal objetivo de este proyecto es **crear una aplicación que realice la transformación inversa, es decir, aquella que partiendo de modelos de especificación abstracta de la interfaz** (Limbourg et al., 2004) **proporcione modelos de especificación de tareas**. La justificación de este proyecto está en que no se conocen herramientas que realicen este cambio y puede ser de gran utilidad para migrar sistemas legados y facilitar la especificación y elaboración de distintos modelos sin necesidad de conocer todas y cada una de las notaciones que precisen los mencionados modelos.

En este sentido, resulta de interés poder pasar de un modelo de presentación realizado a nivel abstracto, que debe ser independiente tanto de la plataforma destino como de la modalidad de interacción elegida, dejando al usuario la libertad de realizar una IU concreta con criterio y creatividad; a una especificación de tareas realizada utilizando la notación CTT (Paternò, 1999) que a su vez pueden estar ligadas a los casos de uso y aportan respecto a ellos dos facilidades adicionales. Por un lado facilitan la descripción más detallada de estos últimos y, por otro, mantienen la especificación a un nivel de abstracción deseable en la fase de diseño de un producto software en la que se considera su utilización.

La disponibilidad del logro principal de este proyecto final de carrera está en que la transformación perseguida **permitiría comprobar que el diseño de interfaz que se ha realizado está acorde con los requisitos del sistema a implementar**, gracias a la correspondencia que puede establecerse entre tareas y casos de uso.

1.3 Objetivos

Derivados de la principal motivación de este proyecto final de carrera, los principales objetivos que deben abordarse son:

- Estudio y familiarización con notaciones relacionadas con la especificación de modelos de tareas (CTT) y de interfaces de usuario, de especial interés será la familiarización con las especificaciones de interfaz de usuario abstractas.

- Estudio de la propuesta UsiXML, prestando especial atención al entorno IdealXML.

- Elaboración de una herramienta que permita la generación del modelo de tareas asociado a una interfaz abstracta especificada previa y gráficamente.

- Determinación del mecanismo más ventajoso para abordar la transformación entre modelos: procedural o declarativa.

- Conocimiento y familiarización con lenguajes de especificación de transformaciones, especialmente XSLT que facilita la transformación de documentos en XML.

- Selección y utilización de un API que permita realizar transformaciones desde una aplicación Java.

- Comprobación empírica del desarrollo realizado.

1.4 Estructura de la memoria

Este documento está estructurado de la siguiente manera:

En este capítulo primero el lector encontrará una introducción a la materia, así como la motivación que ha justificado la realización de este proyecto final de carrera y los objetivos del trabajo dadas las motivaciones expuestas.

A continuación, el capítulo segundo describe el estado actual de los trabajos relacionados con el tema tratado en este proyecto final de carrera. Trabajos previos relevantes a la especificación y construcción de interfaces de usuario son descritos en este capítulo. Especial atención y espacio se dedicará a presentar las notaciones CTT y la propuesta UsiXML, especialmente será de interés la notación propuesta para llevar a cabo la especificación abstracta de interfaces de usuario.

El capítulo tercero está dedicado a la explicación de todos los elementos de análisis y diseño del proyecto. Así como de los detalles de implementación del editor realizado. Por último, en este capítulo se muestra un breve manual de funcionamiento del producto software desarrollado.

En el capítulo cuarto se utilizan unos casos de estudio para demostrar la utilidad práctica de la herramienta implementada en este proyecto fin de carrera. Se muestran la conversión de interfaces de usuario abstractas a modelo de tareas, utilizando para dicha transformación ficheros que cumplen los estándares impuestos por UsiXML.

Finalmente, el capítulo quinto recoge las conclusiones alcanzadas tras la realización de este proyecto, se comentan trabajos asociados y se proponen algunos trabajos futuros.

Capítulo 2. ANTECEDENTES, ESTADO DE LA CUESTION

En este capítulo se describe el desarrollo de interfaces de usuario basado en modelos. En este sentido se revisarán notaciones y propuestas de especificación de interfaces de usuario. Especial atención por su repercusión en el trabajo realizado y en su presencia en capítulos posteriores tendrán las notaciones relacionadas con el modelado de tareas ConcurTaskTrees (CTT) y las que permiten la especificación de modelos de presentación, como por ejemplo el lenguaje usiXML. En este capítulo también se hará referencia a otros lenguajes y herramientas disponibles en la actualidad que guardan relación con la herramienta ligada a este proyecto final de carrera.

2.1. La importancia de la IU

La interfaz de usuario es la forma en que los usuarios pueden comunicarse con una computadora, y comprende todos los puntos de contacto entre el usuario y el equipo. Sus principales funciones son:

- Manipulación de archivos y directorios
- Herramientas de desarrollo de aplicaciones
- Comunicación con otros sistemas
- Información de estado
- Configuración de la propia interfaz y entorno
- Intercambio de datos entre aplicaciones
- Control de acceso
- Sistema de ayuda interactivo.

Nos encontramos con dos tipos de interfaz de usuario:

- Interfaces alfanuméricas (intérpretes de mandatos).
- Interfaces gráficas de usuario (GUI, *Graphics User Interfaces*), las que permiten comunicarse con el ordenador de una forma muy rápida e intuitiva.

Además, las interfaces también pueden clasificarse en hardware o software:

- En el primer caso se trata de un conjunto de dispositivos que permiten la interacción hombre-máquina, de modo que permiten ingresar y tomar datos del ordenador.
- También están las interfaces de software que son programas o parte de ellos que permiten expresar nuestros deseos y ordenes al ordenador.

La interfaz de usuario es el elemento que permite la interacción del usuario con cualquier sistema o programa. La importancia de la interfaz de usuario radica en ser el primer elemento con el que el usuario tiene que interactuar por lo que su diseño será relevante para conseguir una experiencia del usuario agradable y que cumpla sus expectativas.

Una mala interfaz de usuario puede provocar el fracaso de un sistema:
Un interfaz de usuario mal diseñado puede ser la causa de la falta de uso de un sistema o de un sitio web, y puede contribuir a que el usuario cometa errores.
En la web no hay manuales de uso (de un vistazo debe quedar claro como se utiliza).
Los usuarios muchas veces no reciben una formación extensa sobre las herramientas que utilizan. Suelen formarse con el uso.

Resulta necesario tener muy en cuenta el proceso de desarrollo de interfaces de usuario y dedicar el tiempo y los recursos necesarios para que dicho desarrollo se realice con las máximas garantías de éxito posibles.

2.2 Factores que afectan al desarrollo de IU

El desarrollo de interfaces de usuario plantea una serie de desafíos sin precedentes debido a la influencia de varias variables:

Diversidad de usuarios: los usuarios finales de la misma aplicación interactiva no puede considerarse similares cuando ellos exhiben varias habilidades, capacidades, niveles de experiencia y preferencias que deberían ser reflejadas en el IU. En vez de tener un sólo IU para todos los usuarios, una familia de diferentes IUS debería ser desarrollada para enfrentarse con las diferencias de múltiples categorías de usuarios, incluyendo los que son perjudicados.

Riqueza de culturas: cuando una aplicación interactiva va a ser global, su IU no puede ser el mismo para todas las lenguas, países, y culturas. Más bien puede ser sometido a un proceso de localización para adaptar el IU a restricciones particulares o a un proceso de globalización para adaptar el IU a la población más grande posible.

Diversidad de dispositivos y estilos de interacción: La Interacción Hombre-máquina se sabe que es capaz de tratar con una amplia variedad de dispositivos de interacción (por ejemplo, ratón *bimanual*, punteros de 3D, punteros láser,…) y estilos (el rastreo 3D, el rastreo de ojo, el reconocimiento de gesto, el reconocimiento vocal y la síntesis). El

manejo de eventos generados por estos dispositivos y su sólida incorporación en un estilo de interacción requiere muchas habilidades de programación que a menudo van más allá de las capacidades clásicas de un desarrollador medio de un sistema de información. Incluso más, cuando varias modalidades se combinan, como en una aplicación multimodal, esta complejidad aumenta. Lo mismo para realidad virtual, realidad aumentada y aplicaciones de realidad mixta.

Heterogeneidad de plataformas: el mercado de plataformas de computo es sometido a una introducción constante de nuevas plataformas, cada una viniendo con un nuevo juego de restricciones para ser impuestas al IU que debería funcionar sobre ellas. Por ejemplo, una restricción significativa es la resolución de pantalla y las capacidades de interacción que en gran parte varían dependiendo de la plataforma de cómputo: el teléfono móvil, smartphone, Pocket PC, Blacberry, Handbag PC, Tablet PC, quiosco interactivo, ordenador portátil, ordenador de escritorio, etc. Todas estas plataformas no necesariamente funcionan con el mismo sistema operativo y el IU no necesariamente es desarrollado con el mismo lenguaje de marcado (por ejemplo, WML, cHTML, HTML, DHTML, VoiceXML, X+V, VRML97, X3D) o lenguaje de programación (por ejemplo, Visual Basic, C ++, Java, C#). Incluso cuando un mismo lenguaje podría ser usado, varias particularidades están presentes sobre cada plataforma, así impidiendo al desarrollador reutilizar el código de una plataforma a otro. El usuario final típico usa hoy al menos tres plataformas, a veces con alguna sincronización entre ellas.

Múltiples ambientes de trabajo: los usuarios finales hoy en día se enfrentan a una serie de ambientes físicos diferentes donde se supone que ellos trabajan con la misma fiabilidad y eficacia. Pero cuando el ambiente se hace más restrictivo, por ejemplo, con estrés, con el ruido, con la luminosidad, con la disponibilidad de recursos de red, el IU no necesariamente se adapta a esas variaciones.

Múltiples contextos de uso: si un contexto de uso dado es definido como una categoría de usuario particular que trabaja con una plataforma dada en un ambiente específico, entonces la serie de los contextos potenciales de empleo explota. Desde luego, no todas las variaciones contextuales deberían ser consideradas e interpretadas en un cambio significativo del IU, pero al menos una cantidad razonable de diferencias existen para aplicaciones de contexto conocido. Desde la perspectiva del usuario, pueden ocurrir varios escenarios:

- Los usuarios pueden moverse entre plataformas diferentes mientras están implicados en una tarea: para comprar una película en DVD un usuario al principio podría buscarlo en su ordenador de sobremesa, leer la crítica del DVD sobre una

PDA en el tren de camino a casa desde el trabajo, y luego pedirlo usando un teléfono móvil WAP.

- El contexto de uso puede cambiar mientras el usuario está interactuando: el tren puede entrar en un túnel oscuro entonces la pantalla de la PDA se oscurece, el nivel de ruido se eleva así el volumen de regeneración de audio aumenta para que pueda ser oído.
- Los usuarios pueden querer colaborar en una misma tarea usando plataformas heterogéneas: el usuario decide telefonear a un amigo que ha visto la película y ha visto las críticas con él, una persona usando WebTV y la otra utilización su ordenador portátil, entonces la misma información es presentada radicalmente de manera diferente.

Múltiples arquitecturas de software: debido a las susodichas variaciones, el IU debería ser desarrollado con la arquitectura de software dedicada en mente, que explícitamente dirige las variaciones consideradas para la informática móvil, la informática ubicua, usos contexto conocido.

Complejidad en el mantenimiento de sistemas legados: son sistemas informáticos o aplicaciones, que continúan en uso porque el usuario (generalmente una organización) no quieres reemplazarlos o rediseñarlos. Típicamente sus costos de mantenimiento se incrementan y normalmente son difíciles de mejorar y expandir porque existe una escasez general de entendimiento del sistema. Desarrollados antes de que el uso de las técnicas de ingeniería de software estuvieran difundidas. No están estructurados ni documentados.

Una de las aportaciones directas que permite abordar la realización de este proyecto final de carrera está especialmente ligada a este último punto. La reingeniería de sistemas, se encarga de re-estructurar o re-escribir parte o todo el sistema legado sin cambiar su funcionalidad. Es aplicable donde algo, pero no todo el subsistema de un gran sistema, requiere mantenimiento. Re-ingeniería incluye adición de esfuerzo para hacer más fácil el mantenimiento. El sistema puede ser re-estructurado y re-documentado.

2.3 Desarrollo de IU basado en modelos

El desarrollo de interfaces de usuario basado en modelos (MB-UIDE – *Model-Based User Interface Development Environment)* consiste en la especificación de la IU utilizando modelos declarativos que describen las distintas facetas y artefactos involucrados en el desarrollo de una IU. La creación de los modelos necesarios suele realizarse mediante herramientas visuales donde el usuario hace uso de una notación gráfica que permite la especificación de los distintos modelos de una manera sencilla. Las aproximaciones

basadas en modelos persiguen aumentar el nivel de abstracción usado en el diseño de la IU, dejando los detalles de implementación a generadores de código, y permitiendo una generación total o parcial de la IU de forma sencilla cuando los requisitos cambian. De igual manera, dentro de dichas aproximaciones también se persigue la portabilidad de las interfaces de usuario, de forma que un mismo diseño declarativo pueda ser convertido en código ejecutable para distintas plataformas o lenguajes sin necesidad de un rediseño de la interfaz.

Actualmente, no existe un estándar que defina cuáles son los modelos que debería tener un entorno de desarrollo basado en modelos, aunque sí que existen una serie de modelos comunes que aparecen en prácticamente todas las aproximaciones, como son los modelos de tareas, dominio, usuario, diálogo y presentación. La falta de consenso en la adopción de un conjunto estándar de modelos es debida en gran medida a las distintas disciplinas de los grupos de investigación involucrados en la definición de los MB-UIDE, a continuación se muestra su ciclo de vida típico (Fig. 2-1).

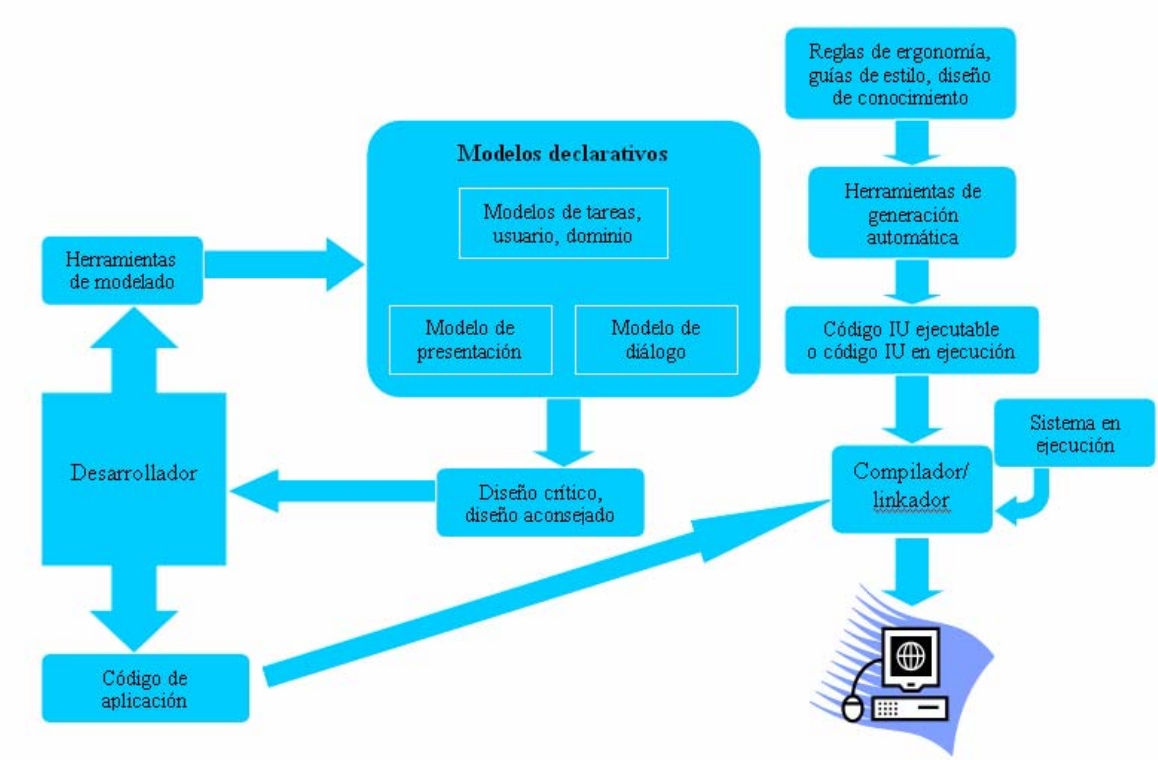

Figura 2-1. Ciclo de vida típico del desarrollo basado en modelos.

2.3.1 El modelo de tareas

El modelo de tareas expresa cuáles son las tareas que va a realizar el usuario de la aplicación a través de la IU. Las tareas se descomponen en acciones atómicas que representan los pasos necesarios para alcanzar los objetivos de la tarea. Dentro de los datos

capturados en este modelo también se recogen los requisitos no funcionales de las tareas, como son por ejemplo los requisitos de tiempo de respuesta.

Para la especificación del modelo de tareas se han utilizado distintas aproximaciones, entre las que destacan los métodos textuales basados en el análisis cognitivo como GOMS *(Goals, Operators, Methods, Selection rules)* o los métodos basados en formalismos como ConcurTaskTrees(Paternò, 2000), GTA, HTA, CUA, etc.

Habitualmente, la captura de los requisitos de las distintas tareas que se realizarán con la IU suele realizarse utilizando diagramas de casos de uso.

El análisis jerárquico de tareas (*HTA Hierarchical Task Analysis*)(Annett y Duncan, 1967), es la técnica de análisis de tareas más conocido y antiguo.

La descripción de la información se realiza en forma de tabla o en forma de diagrama de árbol que describa las relaciones entre tareas y subtareas. Un ejemplo sería la descripción HTA de la preparación del té(Fig. 2-2).

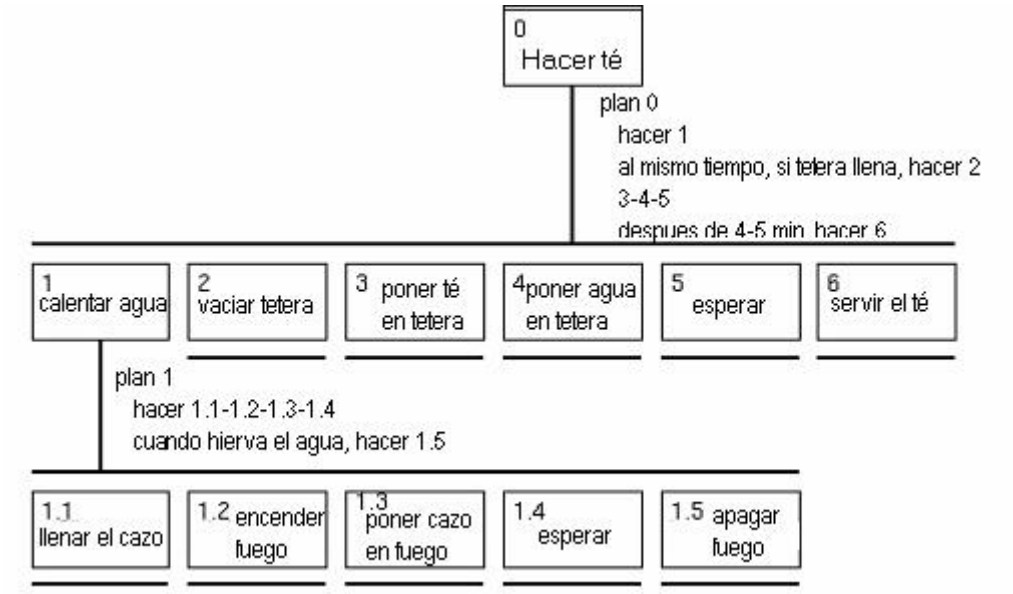

Figura 2-2. Descripción HTA de la preparación del té.

0. Hacer té
    1. Calentar el agua
        1.1 Llenar cazo
        1.2 Encender fuego
        1.3 Poner cazo en fuego
        1.4 Esperar
        1.5 Apagar fuego

2. Vaciar tetera
3. Poner hojas de té en tetera
4. Verter el agua
5. Esperar
6. Servir el té

Plan 0: **hacer** 1.
si tetera está llena,
entonces hacer 2 al mismo tiempo
hacer 3-4-5
Cuando el té ha reposado, **hacer** 6

Plan 1: **hacer** 1.1-1.2-1.3-1.4
Cuando el agua está hirviendo, **hacer** 1.5

2.3.2 El modelo de dominio

El modelo de dominio incluye una visión de los objetos sobre los que actúan las tareas capturadas en el modelo de tareas. La especificación del modelo de dominio va muy ligada a las especificaciones realizadas dentro del modelador del dominio de la parte funcional. Esto puede producir una duplicidad de información con el consecuente peligro de aparición de incoherencias.

Algunas aproximaciones utilizan para la representación del modelo de dominio diagramas entidad/relación, como por ejemplo *GENIUS* (Janssen et al., 1993), mientras que otras utilizan técnicas de orientación a objetos (diagramas de clases principalmente), por ejemplo *MECANO* (Puerta, 2006) o *IDEAS* (Lozano, 2001).

IDEAS (***I****nterface* ***D****evelopment* ***E****nvironment within O****AS****IS*) pretende ser un sistema de desarrollo automático de Interfaces de Usuario integrado en el marco de trabajo de OASIS para soportar de forma automática la producción de interfaces de usuario de alta calidad.

El Modelo de Dominio para IDEAS (Fig. 2-3) viene dado por el Diagrama de Secuencia asociado a cada uno de los casos de uso (tareas) identificados previamente, que define el comportamiento del sistema, y el Modelo de Roles, que define la estructura de las clases que intervienen en el diagrama de secuencia asociado junto con la relación existente entre ellas especificando el rol que desempeña cada una de las clases en ese caso concreto.

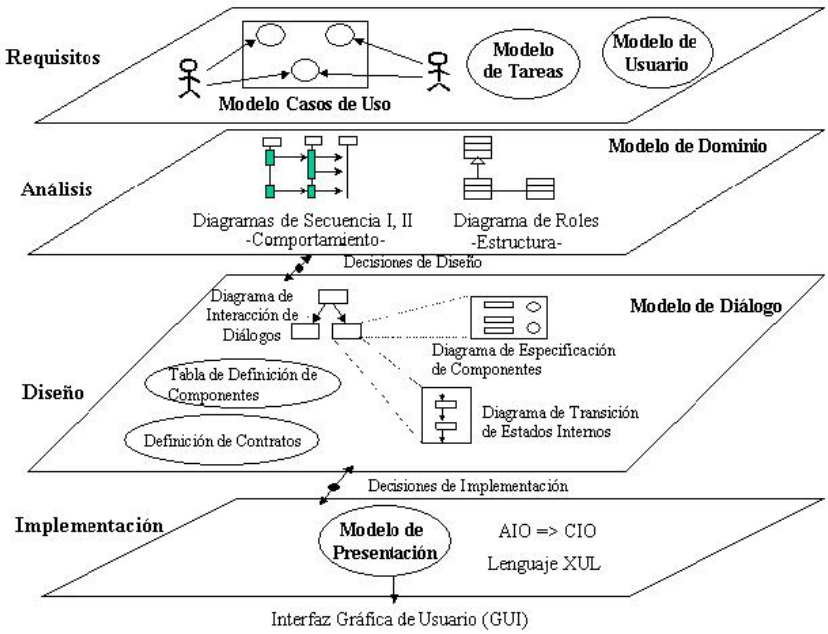

Figura 2-3. Proceso de Desarrollo de IU propuesto en IDEAS.

2.3.3 El modelo de usuario

El modelo de usuario captura las características y requisitos individuales de cada usuario o grupo de usuarios. Habitualmente cada uno de los tipos de usuarios que interactuarán con la aplicación es denominado *rol* (papel). El objetivo de este modelo es ofrecer una IU que se ajuste a las características y requisitos de cada usuario. La adaptación de la IU al usuario puede realizarse en tiempo de diseño o en tiempo de ejecución. En tiempo de diseño se puede definir, entre otras muchas cosas, cuales serán las tareas disponibles para cada tipo de usuario. Por otra parte, la adaptación en tiempo de ejecución requiere un modelo de usuario más completo.

Hasta la fecha los modelos de usuario introducidos dentro del desarrollo de interfaces de usuario basado en modelos han sido bastante reducidos, e insuficientes para la adaptación de la IU en tiempo de ejecución.

Las características del usuario contenidas en el modelo de usuario se suelen clasificar en dependientes de la aplicación e independientes de la aplicación. Las características independientes de la aplicación podrán ser reutilizadas de una aplicación a otra para el mismo usuario, mientras que las dependientes de la aplicación deberán ser elaboradas para cada nueva aplicación. Los modelos de usuario más sencillos serían diagramas de clases o clases que representan a cada usuario o rol almacenando sus preferencias y características a la hora de elegir o utilizar un determinado interfaz de usuario.

2.3.4 El modelo de diálogo

El modelo de diálogo describe las posibles conversaciones entre la IU y el usuario. Representa cuando el usuario puede introducir datos, seleccionar o cuando se muestran los datos. En general representa una secuencia de entradas y salidas.

Para la representación del modelo de diálogo se utilizan tanto técnicas textuales como visuales. Entre estas técnicas las más utilizadas son las distintas variaciones de los diagramas de transición, las redes de Petri o los diagramas de secuencia o estados de UML. Aquí lo que se lleva son diagramas de estados al modo y manera, o muy similares, a los diagramas de estados propuestos en UML, donde cada estado representa una instancia o "ventana" de nuestra aplicación.

2.3.5 El modelo de presentación

El modelo de presentación contiene una descripción de la IU final con la que el usuario interactuará. Este modelo contiene los componentes que contienen la IU, su disposición y su apariencia.

En algunos entornos existen dos modelos de presentación ampliamente aceptados, por una parte se construye un modelo abstracto, el cual describe la IU en función de objetos abstractos de interacción (AIO – *Abstract Interaction Object)*, y por otro lado se construye un modelo de presentación concreto, que se conformará con objetos concretos de interacción (CIO – *Concrete Interaction Object)*. El conjunto de objetos concretos de interacción para una plataforma no tiene por qué coincidir con los objetos de interacción concretos de otra plataforma. Esta separación en nivel abstracto y concreto del modelo de presentación permite una generación de la IU para distintas plataformas a partir de una misma descripción abstracta de la interfaz, donde será necesario seleccionar los objetos concretos de interacción correspondientes a cada objeto abstracto de interacción a partir de heurísticas, guías de estilo o patrones de interacción. Por otra parte, a partir de la información recogida en el modelo de usuario se pueden también crear distintas presentaciones concretas de la IU dependiendo de las características y habilidades del usuario que está utilizando la aplicación en cada momento. TERESA (Mori et al., 2004), usiXML (Limbourg et al., 2004), XUL (Mozilla), XAML (Microsoft), XIML (Puerta et al., 2004), son sólo algunos ejemplos de lenguajes o notaciones asociados o que ofrecen facilidades para especificar modelos de presentación.

2.4 La misma idea: Arquitectura dirigida por modelos (MDA)

Los desarrollos basados en modelos tratados en el punto anterior no son una propuesta aislada e independiente de la actual tendencia en el desarrollo de software. Más bien al contrario se encuentran en la misma forma de trabajar propuesta y buscada en Ingeniería del Software. En este sentido han surgido en esta última disciplina los desarrollos dirigidos por modelos (MDD), un ejemplo notable de esta tendencia lo constituye la propuesta de arquitectura dirigida por modelos. La arquitectura dirigida por modelos (*Model Driven Architecture*) es una especificación detallada por el OMG (*Object Management Group*) que integra diferentes especificaciones y estándares definidos por la misma organización con la finalidad de ofrecer una solución a los problemas relacionados con los cambios en los modelos de negocio, la tecnología y la adaptación de los sistemas de información a los mismos.

MDA nos permite el despliegue de aplicaciones empresariales, diseñadas sin dependencias de plataforma de despliegue y expresado su diseño mediante el uso de UML y otros estándares, potencialmente en cualquier plataforma existente, abierta o propietaria, como servicios web, .NET, CORBA, J2EE, u otras.

La especificación de las aplicaciones y la funcionalidad de las mismas se expresan en un modelo independiente de la plataforma que permite una abstracción de las características técnicas específicas de las plataformas de despliegue. Mediante transformaciones y trazas aplicadas sobre el modelo independiente de la plataforma se consigue la generación automática de código específico para la plataforma de despliegue elegida, lo que proporciona finalmente una independencia entre la capa de negocio, y la tecnología empleada. De esta manera es mucho más simple la incorporación de nuevas funcionalidades, o cambios en los procedimientos de negocio sin tener que llevar a cabo los cambios en todos los niveles del proyecto. Simplemente se desarrollan los cambios en el modelo independiente de la plataforma, y éstos se propagarán a la aplicación, consiguiendo por tanto una considerable reducción del esfuerzo en el equipo de desarrollo, en los errores que tienden a producirse en los cambios introducidos en las aplicaciones mediante otros métodos de desarrollo, y por consiguiente, la reducción de costes y aumento de productividad que conlleva, tan demandados tanto la industria de desarrollo de software como el resto de las empresas.

MDA se apoya sobre los siguientes estándares para llevar a cabo su función:

- UML: empleado para la definición de los modelos independientes de la plataforma y los modelos específicos de las plataformas de destino. Es un estándar para el modelado introducido por el OMG.

- MOF: establece un marco común de trabajo para las especificaciones del OMG, a la vez que provee de un repositorio de modelos y metamodelos.
- XMI: define una traza que permite transformar modelos UML en XML para poder ser tratados automáticamente por otras aplicaciones.
- CWM: define la transformación de los modelos de datos en el modelo de negocio a los esquemas de base de datos.

MDA distingue entre los siguientes modelos:

- Modelos independientes de la plataforma (PIM): es una vista de un sistema desde un punto de vista independiente de la plataforma. Un PIM muestra un grado específico de independencia de plataforma para poder ser usado con un número de plataformas diferentes de tipo similar. Describe el sistema, pero no muestra los detalles del empleo de plataformas.
- Modelos dependientes de la plataforma (PSM) : es una vista de un sistema desde el punto de vista específico de la plataforma. Un PSM combina las especificaciones del PIM con los detalles que especifican como el sistema usa un tipo particular de plataforma.

En su modelo de transformación, PIM e información adicional se combinan para producir el PSM (Fig. 2-4).

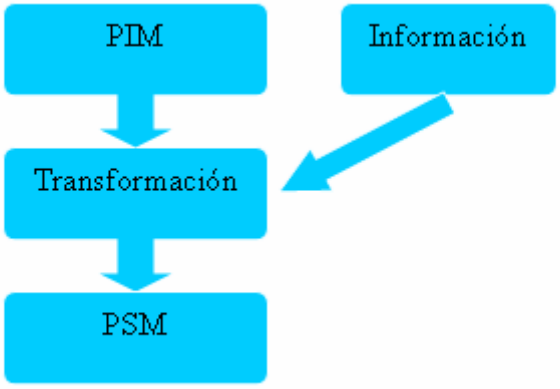

Figura 2-4. Transformación de PIM a PSM.

La principal diferencia entre el desarrollo de IU y del software es que el segundo tiene un lenguaje unificado y aceptado que le da soporte (UML). En el terreno del desarrollo de IU se está lejos de contar con algo aceptado universalmente y se cuenta con diferentes propuestas, por ejemplo CTT, UsiXML, etc. El trabajo realizado en este proyecto y que se describirá en detalle en el capítulo tercero está localizado a nivel independiente de la plataforma, PIM, ya que de la especificación que se partirá y la que se alcanzará están situados en ese nivel de abstracción.

2.5 Análisis de tareas: La notación ConcurTaskTrees

ConcurTaskTrees (CTT) es una notación desarrollada por Fabio Paternò (Paternò, 1999) cuyo principal finalidad es la de poder representar las relaciones temporales existentes entre las actividades y usuarios que son necesarios para llevar a cabo en las tareas.

Una de las principales ventajas de esta notación es su facilidad de uso, lo que hace que sea aplicable a proyectos reales con aplicaciones de un tamaño medio–largo y que conllevan especificaciones de cierta complejidad. La notación genera una representación gráfica en forma de árbol de la descomposición jerárquica de las tareas existentes en el sistema. Se permite la utilización de un conjunto de operadores, sacados de la notación de Lotos, para describir las relaciones temporales entre tareas (secuencialidad, concurrencia, recursión, iteración...).

Se pueden reutilizar partes de especificación para la creación de "árboles de tareas concurrentes" e identificarlo como un patrón de tarea. Podemos identificar 4 categorías de tareas en función del actor que la llevará a cabo (Tabla 2-1).

Tipo de tarea(icono)	Descripción
	Tareas del usuario. Tareas realizadas completamente por el usuario, son tareas cognitivas o físicas que no interactúan con el sistema. Describen procesos realizados por el usuario usando la información que recibe del entorno (por ejemplo seleccionar dentro de un conjunto de información la que se necesita en un instante determinado para la realización de otra tarea).
	Tareas de la aplicación. Tareas realizadas por la aplicación y activadas realizadas por la propia aplicación. Pueden obtener información interna del sistema o producir información hacia el usuario. Como ejemplo podemos ver una tarea que presente los resultados obtenidos de una consulta a una base de datos.
	Tareas de interacción. Son tareas que realiza el usuario interactuando con la aplicación por medio de alguna técnica de interacción. Un ejemplo puede ser seleccionar un elemento de una lista desplegable.
	Tareas Abstractas . Tareas que requieren acciones complejas y que por ello no es fácil decidir donde se van a realizar exactamente. Son tareas que van a ser descompuestas en un conjunto de nuevas subtareas.

Tabla 2-1. Tipos de tareas definidas en CTT.

Para la descripción se utilizan además una serie de operadores temporales que facilitan la descripción de las relaciones temporales existentes entre tareas. Estos operadores se han obtenido como una extensión de los operadores existentes en Lotos. El uso de estos operadores facilita la descripción de comportamientos complejos. Los operadores temporales que podemos usar son (Tabla 2-2):

Operador	Descripción
T1 ||| T2.	Concurrencia. Las acciones de las dos tareas pueden realizarse en cualquier orden.
T1 |[]| T2	Sincronización (Concurrencia con intercambio de Información). Las dos tareas tienen que sincronizarse en alguna de sus acciones para intercambiar información.
T1 >> T2	Activación. Cuando termina la T1 se activa la T2. Las dos tareas se realizan de forma secuencial.
T1 []>> T2	Activación con paso de información. Cuando termina T1 genera algún valor que se pasa a T2 antes de ser activada.
T1 [] T2	Elección. Selección alternativa entre dos tareas. Una vez que se esta realizando una de ellas la otra no esta disponible al menos hasta que termine la que esta activa.
T1 [> T2	Desactivación. Cuando se da la primera acción de T2, la tarea T1 se desactiva.
T1 | > T2	Suspender/Resumir. T2 tiene la posibilidad de interrumpir a T1 que podrá ser retomada cuando aquella finalice.
T1*	Iteración. La tarea T1 se realiza de forma repetitiva. Se estará realizando hasta que otra tarea la desactive.
T1(n)	Iteración finita. La tarea T1 puede darse n veces. Se utiliza cuando el diseñador conoce cuantas veces tiene que realizarse la tarea.
[T1]	Tarea opcional. No es obligatorio que se realice la tarea. Cuando describimos las subtareas existente en la tarea de rellenar un formulario algunas de las subtareas pueden ser opcionales (las de los campos que sean opcionales).

Tabla 2-2. Operadores temporales de CTT.

Actualmente, ConcurTaskTrees se ha convertido en la notación para especificación de tareas más utilizada, y va camino de convertirse en el estándar de facto en la especificación de modelos de tareas. Igual que comentaba en una sección previa, sería interesante incluir o poder referenciar un ejemplo incluido en este documento en el que se pudiese comprobar la facilidad de uso o las características que para el modelado de tareas ofrece CTT.

2.6 Lenguajes de especificación de interfaces: UsiXML

UsiXML (Limbourg et al., 2004) es un lenguaje para la descripción de interfaces de usuario que permite la especificación de las características más habituales usadas en el desarrollo de interfaces de usuario basadas en modelos, y almacenarlas en un fichero en formato XML. UsiXML describe a un alto nivel de abstracción los elementos de una interfaz, pero no sólo los *widgets* o componentes gráficos, sino también los modos de interacción. Diferentes *renderers* permiten su ejecución en diferentes dispositivos y plataformas.

UsiXML (el cuál se presenta como candidato para el lenguaje de marcado extensible de interfaz de usuario) es un lenguaje XML de marcado adaptado que describe la IU para múltiples contextos de uso tales como Interfaces de Usuario de Caracteres (CUIs), Interfaces de Usuario gráficas (GUIs), Interfaces de Usuario de Auditoría, Interfaces de Usuario Multimodales. En otras palabras, las aplicaciones interactivas con diferentes tipos de técnicas de interacción, modalidades de uso, y plataformas de computación pueden ser descritas de la forma que preserva el diseño independientemente de las características peculiares de la plataforma de computación física.

El lenguaje se basa en el marco de desarrollo de interfaces de usuario desarrollado dentro del proyecto europeo Cameleon, cuyos niveles de abstracción son ilustrados en la figura (Fig. 2-5).

Figura 2-5. Niveles de abstracción en UsiXML.

Las **tareas y los conceptos** representan las tareas que el usuario podrá realizar a través de la IU y los objetos del dominio que dichas tareas deben manipular para ser llevadas a cabo por el usuario. La forma de especificar las tareas en Usixml está inspirada en CTT. En este sentido, Usixml contempla los mismos tipos de tareas y de operadores temporales definibles entre ellas.

La **IU abstracta (AUI – Abstract User Interface)** describe la interfaz de usuario con la cual el usuario va a interaccionar, pero usando un alto nivel de abstracción. Este alto nivel de abstracción hace que la especificación de la interfaz de usuario abstracta sea independiente tanto de la plataforma final donde será ejecutada como de la modalidad usada para interaccionar con la IU.

La **IU concreta (CUI – Concrete User Interface)** representa la IU de una forma abstracta, pero en un nivel de abstracción menor al usado en la IU abstracta. En este caso la especificación de la IU concreta es independiente, de la plataforma donde se ejecutará la aplicación, pero es dependiente de la modalidad que será usada (gráfica, vocal o textual).

La **IU final (FUI – Final User Interface)** representa el código que será ejecutado en la plataforma destino para mostrar la IU. Por lo tanto, esta versión de la IU es dependiente tanto de la plataforma donde se ejecutará la interfaz como de la modalidad que será usada para interactuar con ella.

Una IU en usiXML (Fig. 2-6) es descrita utilizando una serie de modelos que representan los modelos más habitualmente usados en los desarrollos de interfaces de usuario basados en modelos, y adicionalmente algunos modelos que permiten representar posibles transformaciones que se puedan aplicar sobre un modelo, siguiendo el paradigma de MDA.

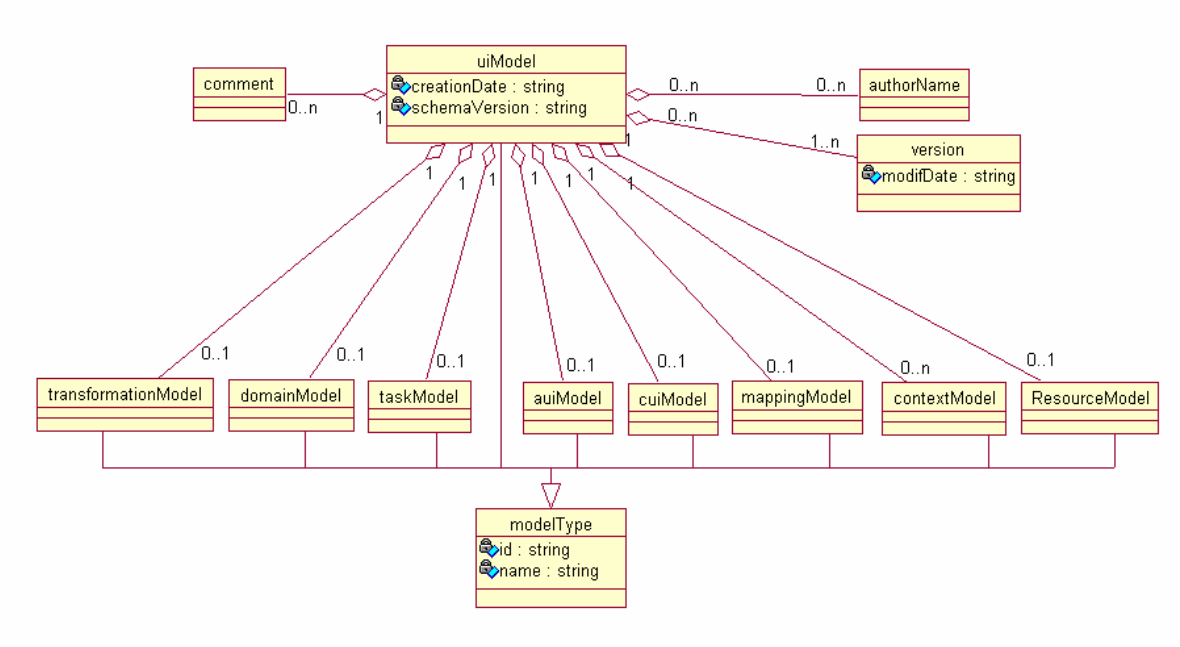

Figura 2-6. Modelos asociados con la especificación de una interfaz de usuario en usiXML.

El modelo de interfaz abstracta (Fig. 2-7) permite representar la IU utilizando un alto nivel de abstracción. Básicamente, la interfaz se describe usando contenedores

abstractos y elementos individuales abstractos. A los elementos individuales abstractos se les puede asociar facetas que describen las capacidades de interacción que presentan (entrada, salida, navegación o control). De igual forma, también es posible la definición de relaciones abstractas entre los distintos elementos abstractos:

- *spatioTemporal*: es una relación que describe restricciones abstractas en el tiempo y espacio entre AIOs, independientemente de cualquier modalidad de interacción.
- *mutualEmphasis*: relación que expresa exclusión mutua entre AIOs.
- *abstractAdjacency*: permite expresar una relación de adyacencia entre dos AIOs.
- *auiDialogControl*: habilita la especificación de un diálogo de control en términos de operadores de LOTOS entre AIOs.
- *abstractContainment*: permite especificar que un abstractContrainment tiene incrustados uno o varios abstractContrainment o uno o varios abstractIndividualComponents.

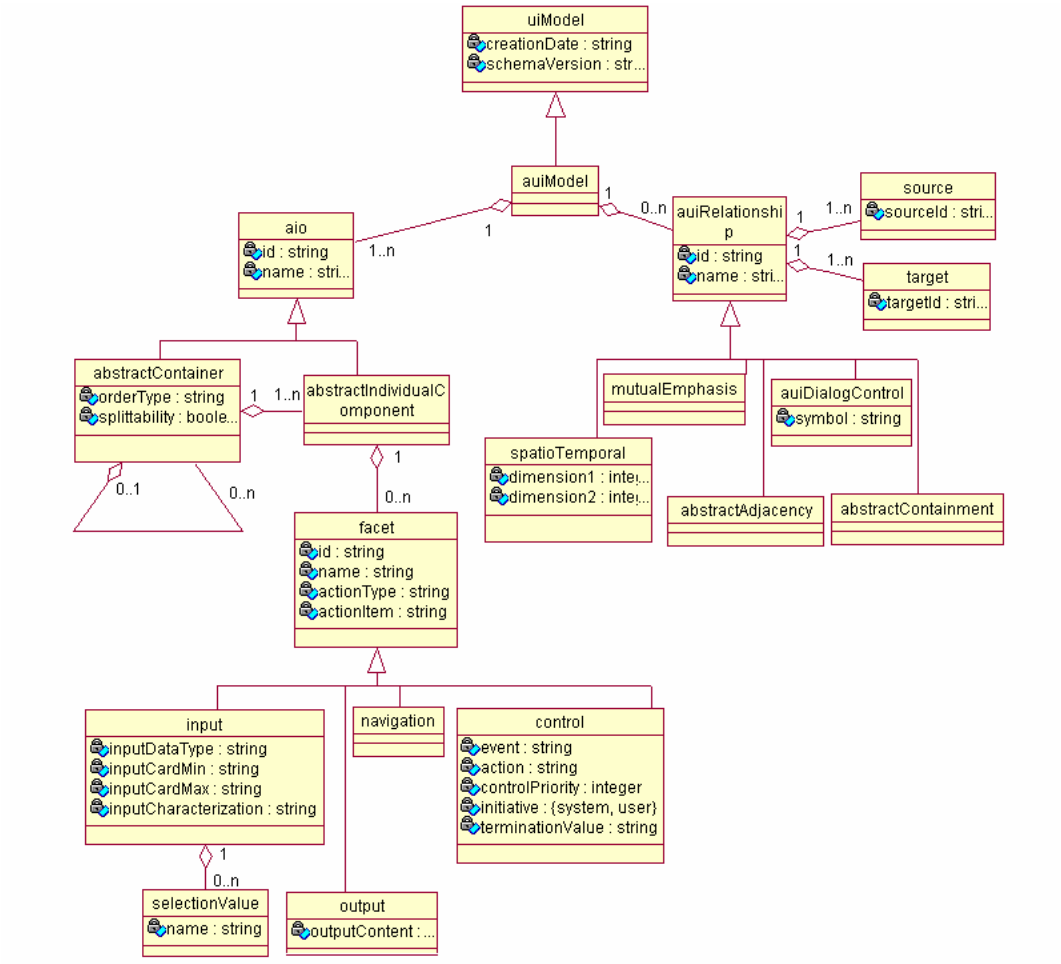

Figura 2-7. Modelo de interfaz de usuario abstracta para usiXML.

2.6.1 Modelo de interfaz de usuario abstracta (AUI)

Un modelo Interfaz de Usuario Abstracto (AUI) es un modelo de IU que representa una expresión canónica de las interpretaciones y la manipulación de los conceptos de dominio y funciones en un camino que es tan independiente como posible de modalidades y especificidades de plataforma informáticas. Un AUI está poblado por Objetos de Interacción Abstractos y relaciones de IU Abstractas. Los Objetos de Interacción Abstractos (AIO) pueden ser de dos tipos: Componentes Abstractos Individuales (AIC) y Contenedores Abstractos (AC). Un Componente Abstracto Individual es una abstracción que permite la descripción de objetos de interacción en un camino que es independiente de la modalidad en la cual será dado en el mundo físico. Un AIC puede ser compuesto de múltiples facetas. Cada faceta describe una función particular que se endosa a un AIC en el mundo físico (Tabla 2-3).

Se identifican cuatro facetas principales: una faceta de entrada describe la acción de entrada soportada por un AIC, una faceta de salida describe que datos pueden ser presentados al usuario por un AIC, una faceta de navegación describe la transición posible que un contenedor AIC particular puede permitir, y una faceta de control describe los eslabones entre un AIC y funciones de sistema (por ejemplo, métodos del modelo de dominio cuando existen). Un AC es una entidad que permite a una agrupación lógica de otros contenedores abstractos o componentes abstractos individuales. AC se dice que apoya la ejecución de un juego de tareas lógicamente/semánticamente conectadas. AIC y AC pueden ser referenciados en el nivel concreto, en uno o varios contenedores gráficos como cuadros de diálogo de ventanas, cajas de disposición o intervalos de tiempo en el caso de interfaces de usuario de auditoria. En este modelo es posible establecer relaciones. Una relación importante es la relación de control de Diálogo. Esta relación permite una especificación de un flujo de control entre los objetos de interacción abstractos y puede ser sacada de relaciones de modelo de tarea.

De un modo similar, como otros modelos, los patrones pueden ser usado aquí construir el IU abstracto. Muchos modelos que puede ser encontrado en la literatura y en sitios Web pueden ser descritos usando UsiXML y editados en idealXML, y estos esbozos que usan componentes abstractos son mapeados con tareas siguiendo los mapeos *isExecutedIn*.

Icono	Descripción
	Icono asociado a un objeto Container. Objeto de interacción abstracto que permite aglutinar componentes en su interior.
	Icono asociado a un Componente. Objeto de interacción abstracto que mediante la incorporación de distintas facetas definidas en usiXML permite dar soporte a la interacción mantenida entre usuario y sistema.
	Iconos asociados a las diferentes facetas definidas en usiXML. A través de ellas es posible dotar a los componentes de interacción de facilidades para dar soporte a diferentes acciones de interacción, respectivamente, entrada de datos, salida de datos, operaciones de control y de navegación.

Tabla 2-3. Iconografía utilizada en IdealXML para elaborar modelos de presentación.

El desarrollo basado en modelos es compatible con la tendencia que sigue el propio desarrollo del software. En este sentido, el desarrollo dirigido por modelos(MDA) es una apuesta descarada por el modelo y su utilización como elemento activo en el desarrollo de productos software.

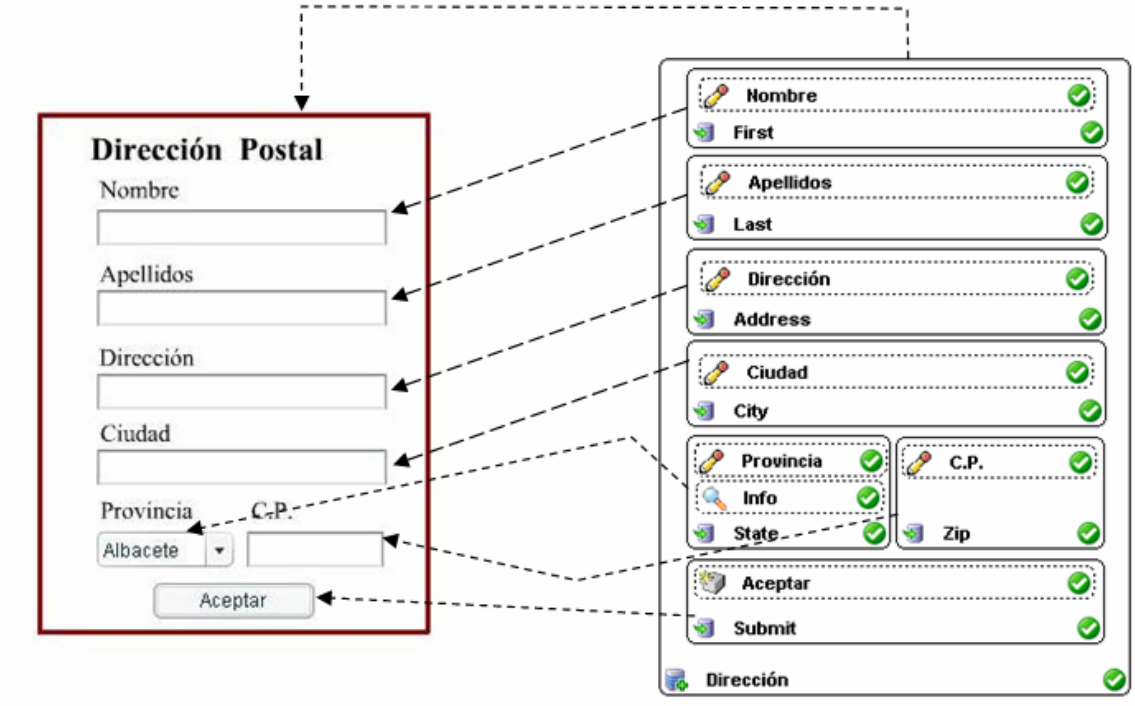

Figura 2-8. Ejemplo de uso de AUI.

Con un sencillo ejemplo de formulario para indicar la dirección postal(Fig. 2-8), en el que se hace uso de componentes con facetas y contenedores, se demuestra la capacidad

de modelado que ofrece la notación de especificación de modelos de presentación abstracta definida en usiXML.

En este ejemplo están marcados con flechas las correspondencias entre el diagrama AUI y el formulario. Sólo existe un contenedor que será la ventana principal y el resto serán componentes que dependiendo del campo tendrán unas facetas u otras.

2.7 Estudio de herramientas de soporte al desarrollo basado en modelos

Siguiendo, cuando menos la filosofía de desarrollo y especificación presentada en los apartados previos, seguidamente se presentan diferentes herramientas propuestas en el ámbito de MBUID.

2.7.1 El entorno TERESA: de las tareas a la presentación

Teresa multimodal (Mori et al., 2004) es un entorno basado en transformaciones, que permite el diseño y desarrollo de interfaces de usuario multiservicio a partir de múltiples descripciones lógicas. Se intenta proporcionar un entorno completo y semiautomático que soporta un número de transformaciones útiles para los diseñadores para construir y analizar su diseño en distintos niveles de abstracción y como consecuencia generar la IU para un tipo específico de plataforma(Fig. 2-10).

Esta herramienta soporta las siguientes transformaciones:

- Generar grupo de tareas habilitadas (ETS). Son el grupo de tareas que son habilitadas en el mismo periodo de tiempo, de acuerdo a las restricciones indicadas en el modelo de tareas. Esta característica permite a los diseñadores obtener tales grupos.

- De modelo de tareas a IU abstracta (Fig. 2-9). Existe una correspondencia directa de ETSs y las presentaciones abstractas resultantes, ya que las tareas pertenecientes al mismo ETS pueden pertenecer a la misma presentación. La especificación de la IU abstracta, en términos de su estructura (la parte de presentación) y el comportamiento dinámico (la parte de diálogo), es obtenida en esta fase y salvada para transformaciones y análisis adicionales.

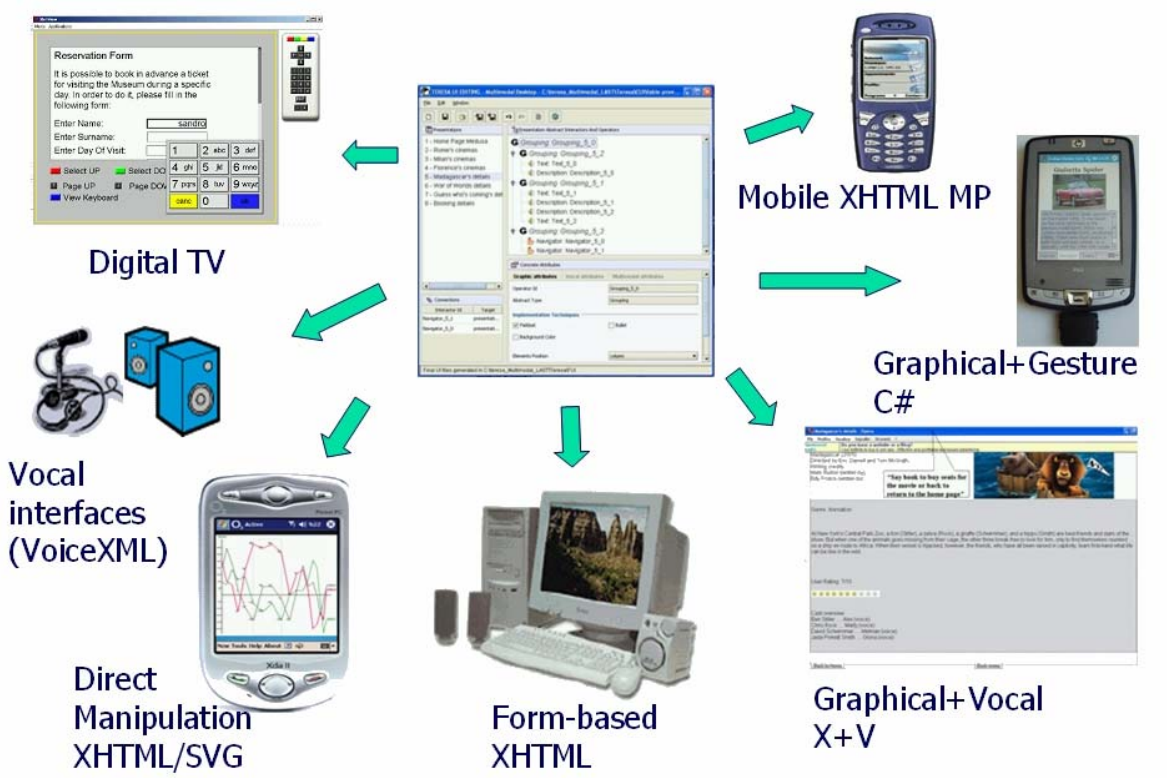

Figura 2-9. TERESA. del modelo de tareas a la interfaz de usuario para distintas plataformas.

De IU abstracta a IU concreta para sistemas de escritorio. Esta transformación comienza con la carga de una interfaz abstracta previamente salvada para una aplicación de escritorio y produce la interfaz concreta asociada. Una serie de parámetros asociados al uso de la IU concreta se ponen a disposición del diseñador.

De IU abstracta a IU concreta para dispositivos móviles. Con esta característica es posible derivar a interfaces de usuario concretas para un teléfono móvil desde una interfaz abstracta. Los diseñadores tendrán la posibilidad de seleccionar un número de parámetros que les permiten el uso de las interfaces resultantes para el móvil.

Generación de IU automáticas. Con esta característica el diseñador puede generar automáticamente la IU final, comenzando con la el modelos de tareas visualizado actualmente (plataforma única), y haciendo uso de un número de elementos de la configuración por defecto asociada a la generación de la interfaz.

Como puede deducirse de la lista de transformaciones, si está cubierto el paso de IU abstracta a concreta, pero no dispone de la transformación inversa.

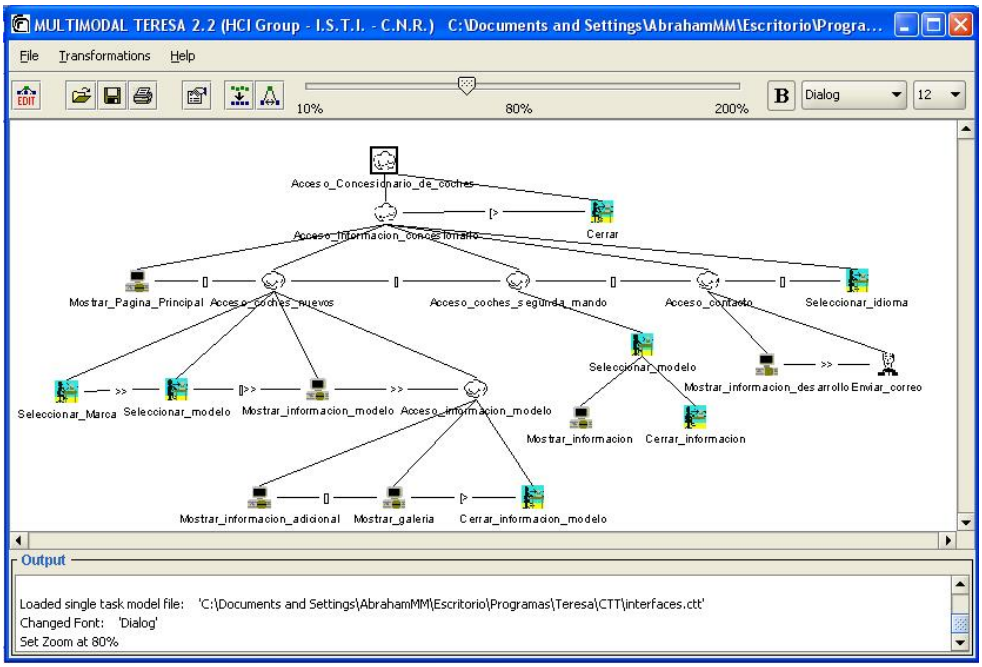

Figura 2-10. Entorno TERESA.

Versión de especificación de presentación propuesta por (Mori et al., 2004)(Fig 2-11).

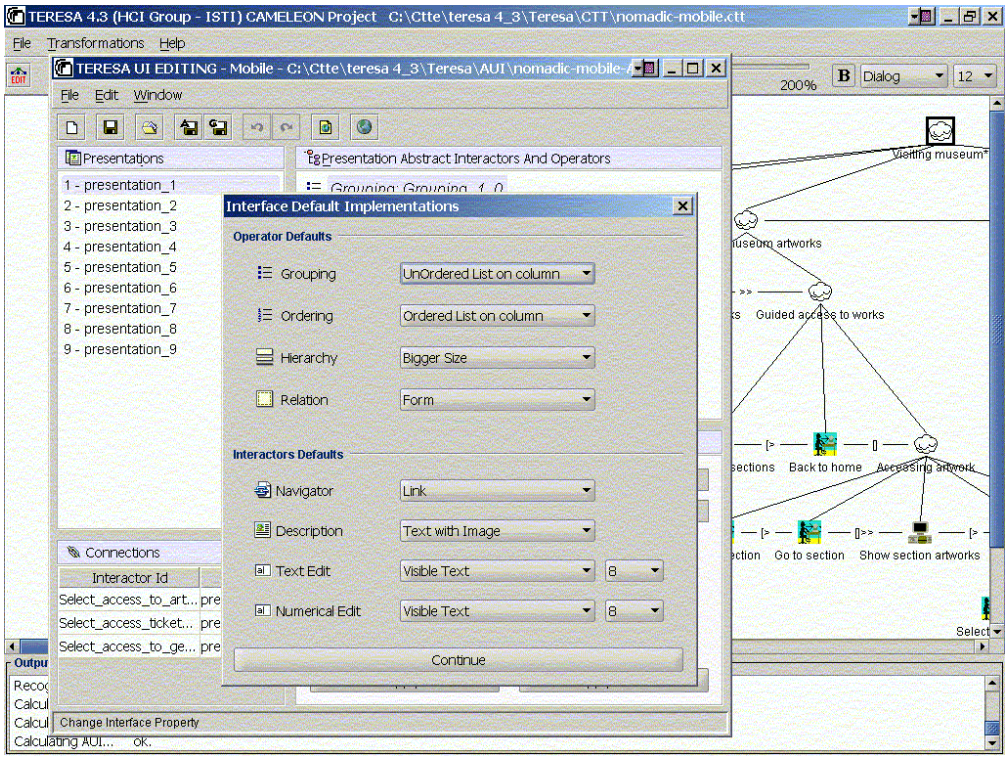

Figura 2-11. TERESA diferentes técnicas para implementar operadores abstractos.

2.7.2 Herramientas asociadas a UsiXML

Las herramientas recogidas en la página de UsiXML (Limbourg et al., 2004), más relevantes y asociadas con el propósito de este proyecto son (Fig. 2-12):

TransformiXML (Limbourg et al., 2004) es una aplicación basada en Java con la que es posible definir, almacenar, manipular y ejecutar especificaciones realiza-das utilizando gramáticas de grafos y con la que es posible realizar transformaciones de modelo a modelo.

KnowiXML (Furtado et al., 2004) consta de un sistema experto basado en Protégé (Noy et al., 2001) que automáticamente produce diferentes especificaciones abstractas a partir de modelos de dominio y de tareas elaborados considerando diferentes contextos.

GrafiXML (Limbourg et al., 2004) es una aplicación que permite editar especificaciones concretas de IU y del modelo de contexto, y también es capaz de generar automáticamente el código equivalente a la especificación realizada de la interfaz en HTML, XHTML, XUL y Java mediante una serie de plug-ins.

VisiXML es un plug-in elaborado para Microsoft Visio con el que es posible especificar interfaces de usuario a nivel concreto y almacenarlos utilizando el lenguaje usiXML.

SketchiXML (Coyette et al., 2004) es una herramienta que permite hacer prototipos de IU considerando múltiples usuarios, plataformas y contextos de uso. Esta herramienta está implementada sobre un sistema de agentes como es Jack (Howden, et al., 2001).

FormiXML es un editor dedicado a la elaboración de formularios interactivos donde se dan facilidades para la reutilización de componentes. Los formularios generados pueden almacenarse en código Java.

VisualiXML (Schlee et al., 2004) permite la personalización y manipulación de interfaces de usuario producidos utilizando técnicas de programación generativas.

ReversiXML (Bouillon et al., 2004) obtiene las especificaciones concretas y abstractas de interfaces de usuario en usiXML relacionadas aquellas interfaces que aparecen en páginas Web elaboradas utilizando HTML.

La generación automática y directa de aplicaciones utilizando transformaciones y disponibles en java (**JaviXML**), Flash (**FlashiXML**) y en el entorno Tcl/Tk (**QtkXML**).

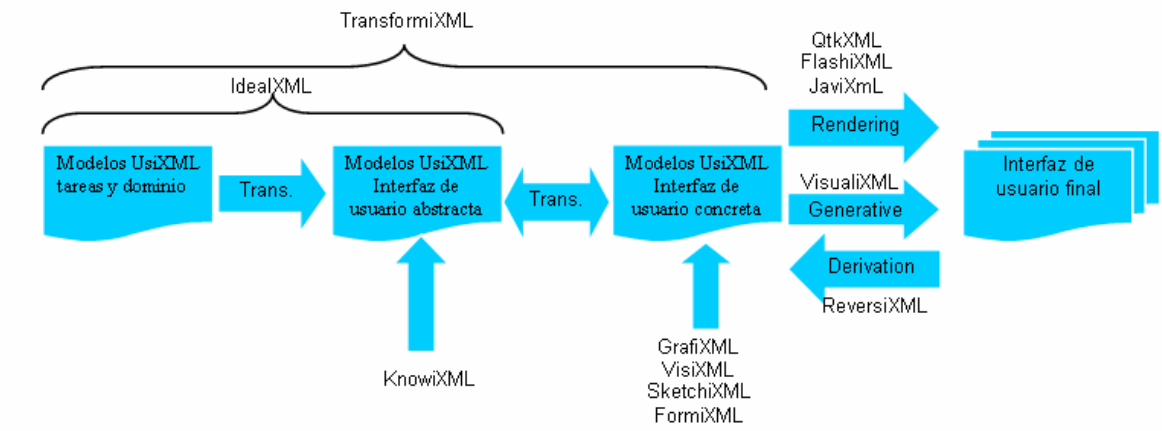

Figura 2-12. Herramientas de UsiXML estructuradas de acuerdo con la clasificación de MDA.

Hay muchas herramientas asociadas a UsiXML pero pocas de ellas van en la dirección que se considera en este proyecto final de carrera. La gran mayoría van de la especificación a la generación de código final asociado a la interfaz de usuario. Este proyecto se centra en el proceso contrario de modelos más próximos a la interfaz a modelos más próximos a la especificación original de una interfaz.

2.7.3 IdealXML

IdealXML (*Interface Development Environment for Applications specified in UsiXML*), es un entorno que facilita la gestión y manipulación de experiencia recopilada en forma de patrones y la posterior utilización de la misma en labores de especificación de productos software.

Se trata de un entorno desarrollado para facilitar la documentación de distintos tipos de patrones, en especial de los patrones de interacción. Tradicionalmente, dichos patrones se documentan utilizando descripciones en lenguaje natural donde quedan recogidos diferentes secciones significativas.

Con el entorno IdealXML se pueden documentar, gestionar, recuperar, compartir y utilizar un catálogo de patrones, y también es posible asignar a cada patrón criterios de calidad que afectan a la usabilidad y se constituyen en modelo de calidad.

IdealXML permite crear un nuevo repositorio, y luego distribuir ese repositorio a otras personas, esto es esencial. Se pueden editar elementos textuales asociados con un patrón, como: nombre, alias, problema, contexto, solución, resumen, racionamiento, etc. Y

se pueden editar diagramas que usan notaciones significativas (diagramas de clase y CTT) de la ingeniería de software y la interacción hombre-máquina. Así, criterios ergonómicos, fuerzas, diagramas, ejemplos y la información del autor son asociados con un patrón también. Los diagramas son asociados usando UsiXML y los patrones son almacenados usando PLML. PLML es una plantilla, mediante la especificación de un DTD de XML, dicha plantilla sólo persigue unificar criterios a la hora de documentar patrones de clara relación con Interacción Persona-Ordenador.

Esta herramienta puede ser también usada para derivar el AUI en UsiXml a partir del modelo de tareas. La siguiente tabla ilustra las equivalencias utilizadas en IdealXML(Tabla 2-4).

<table>
<tr><th>CTT</th><th colspan="2">AUI</th></tr>
<tr><td>Tarea abstracta</td><td colspan="2">contenedor</td></tr>
<tr><td rowspan="3">Tarea de interacción</td><td rowspan="2">es hoja: componente</td><td>entrada</td></tr>
<tr><td>salida</td></tr>
<tr><td>control</td></tr>
<tr><td></td><td colspan="2">navegación</td></tr>
<tr><td rowspan="3">Tarea de aplicación</td><td colspan="2">no es hoja: contenedor</td></tr>
<tr><td rowspan="2">es hoja: componente</td><td>salida</td></tr>
<tr><td>navegación</td></tr>
</table>

Tabla 2-4. Paso de CTT a AUI en IdealXML.

Una aplicación de este proceso de transformación puede ser en una sencilla página web de un concesionario de vehículos(Fig. 2-13). A partir de esa página es posible extraer su modelo de tareas correspondiente utilizando el editor de IdealXML(Fig. 2-14). Con este diagrama es posible obtener su correspondencia de interfaz de usuario abstracta(Fig. 2-15)utilizando la opción correspondiente en el menú de IdealXML. Para que en este diagrama aparezcan la facetas deseadas se ha indicado para cada tarea en la propiedad “user action” el tipo de faceta correspondiente.

Figura 2-13. Ejemplo de página de un concesionario de coches.

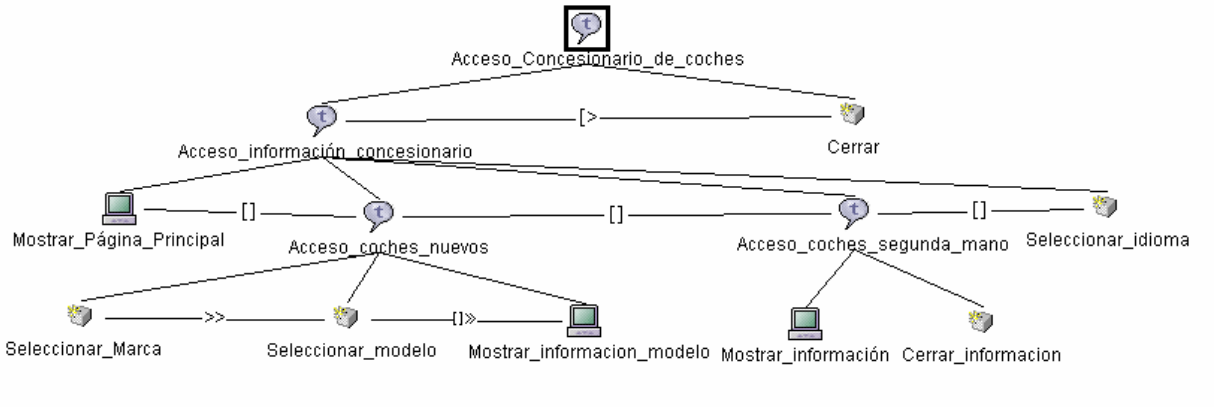

Figura 2-14. Modelo de tareas correspondiente a la página del concesionario.

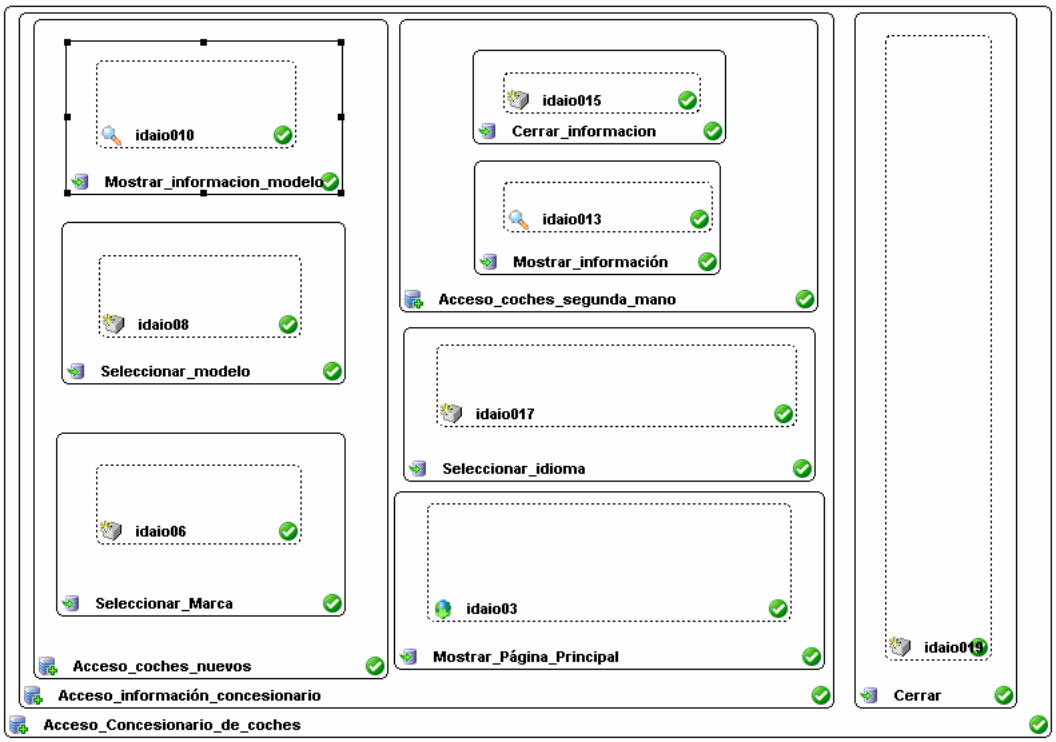

Figura 2-15. Diagrama AUI obtenido a partir del modelo de tareas.

2.8 Resumen y Conclusiones

Hace unos cuantos años, en la Ingeniería del Software, la falta de estandarización en la manera de representar gráficamente un modelo impedía que los diseños gráficos realizados pudieran compartirse fácilmente entre distintos diseñadores lo que suponía un verdadero obstáculo. Se necesitaba un lenguaje no sólo para comunicar ideas a otros desarrolladores sino también para servir de apoyo en los procesos de análisis de un problema. Con este objetivo nació el Lenguaje Unificado de Modelado (UML). La estandarización que se ha aportado con UML, adoptado como lenguaje de modelado no ha llegado todavía al terreno de la interacción persona-ordenador. En este capítulo hemos visto algunas propuestas reseñables nacidas en el ámbito académico como son TERESA o USIXML, en ellas se proponen lenguajes de especificación, pero dichos lenguajes están lejos de ser un estándar en la actualidad.

El aumento del nivel de abstracción a la hora de considerar una aplicación ha sido más creciente, primándose el diseño frente a la implementación y más tarde la elicitación de requisitos frente al diseño. En ese mismo sentido, la propia programación ha cambiado pasando del uso de lenguajes de programación de alto nivel a la elaboración de especificaciones utilizando UML, que tras ser compiladas proporcionan el correspondiente código. En IS se ha apostado, definitivamente, por un desarrollo dirigido por modelos, bajo una visión arquitectónica y donde priman los factores internos (MDA).

Desde el punto de vista de la interacción persona-ordenador también se han desarrollado metodologías, lenguajes y herramientas que, aunque no han alcanzado el grado de estandarización del que se dispone en IS, si está extendido su uso y gozan de gran popularidad.

Las aproximaciones basadas en modelos han proporcionado al desarrollo de interfaces de usuario la sistematización necesaria para la reutilización tanto del código como en parte de la experiencia, y han permitido la generación automática o semiautomática de las interfaces de usuario para distintas plataformas.

En general, la mayoría de las propuestas que utilizan el paradigma basado en modelos han aportado herramientas y aplicaciones capaces de controlar el proceso de generación automática de las interfaces. Sin embargo, apenas se ha hecho hincapié en el aspecto ingenieril del asunto, donde cabe plantearse tratamientos más automatizados y posibilidades de depuración y verificación automáticas si es posible llegar, a partir de la interfaz generada, a los modelos que se utilizaron para especificar la interfaz en el momento de su generación.

Para justificar el uso de MDA se va a presentar que tiene de positivo y negativo este tipo de desarrollo. La utilización de este tipo de aproximaciones presenta una serie de ventajas:

- Permiten una descripción más abstracta de la IU que los métodos de descripción de interfaces de usuario más tradicionales.
- Permiten diseñar e implementar las interfaces de usuario de forma sistemática, ya que facilitan: el modelado de la IU en distintos niveles de abstracción, refinar incrementalmente los modelos y reutilizar las especificaciones de los modelos de usuario.
- Proporcionan la infraestructura necesaria para la automatización de parte de las tareas de diseño y generación de la IU.

Por otra parte, estas aproximaciones también presentan actualmente ciertos problemas por resolver:

- La complejidad de los modelos provoca que no sea habitualmente fácil aprender su funcionamiento, aunque se espera que el desarrollo de herramientas de diseño visuales reduzca la complejidad en gran medida.
- La integración de la parte funcional de la aplicación y de su IU todavía no está totalmente resuelta.

- No existe consenso sobre cuáles son los modelos más adecuados para modelar una IU. Ni siquiera lo existe sobre cuáles son los aspectos de la IU que deben ser modelados.

La ingeniería inversa es un proceso difícil que sirve para analizar un sistema existente con el objetivo de identificar sus componentes y sus relaciones y crear otras representaciones del mismo o crear alguna abstracción sobre el propio sistema. La ingeniería inversa se utiliza usualmente para el rediseño de un nuevo sistema, mejorando así su mantenimiento, o simplemente para producir una copia del sistema si no se dispone de la fuente a partir de la cual fue originalmente producido.

El desarrollo de interfaces de usuario, se presta a la ingeniería inversa, debido a que la mayoría de las migraciones se realizan por cuestiones de adaptación del propio software a nuevas librerías gráficas, lenguajes o incluso plataformas de acceso.

Uno de los aspectos más interesantes de la modelización de interfaces de usuario es cómo poder aprovechar esta especificación utilizada en la definición de la interfaz no solamente para la generación automática de la misma, sino para llevar a cabo, de forma automática, procesos más laboriosos que involucres técnicas de re-ingeniería para el mantenimiento de las propias interfaces. Es decir, sería deseable la situación de que, una vez generada la interfaz, se pueden hacer cambios persistentes sobre dicha interfaz que afecten a la futura generación de interfaces del mismo tipo, o que se pueda dar el caso de poder cambiar el aspecto de la interfaz para transportarlas a otros lenguajes o incluso crear distintas vistas de la presentación de la propia interfaz.

En los próximos capítulos se presentarán los pasos seguidos hasta realizar la herramienta realizada para abordar la transformación entre modelos independientes de la plataforma, así como una serie de casos de estudio donde se mostrarán las aportaciones y aplicaciones de uso de dicha herramienta.

Capítulo 3. ANALISIS Y DISEÑO

En este capítulo se describen los pasos seguidos hasta conseguir la herramienta objetivo de este proyecto final de carrera. Dicha herramienta debe facilitar la edición y transformación de diagramas de presentación a los de tareas. El proceso sugerido en este proyecto final de carrera consistirá a grandes rasgos en partir de un diagrama de especificación abstracta de la interfaz (AUI), obtener un fichero XML con su contenido, y transformar éste en otro fichero XML de temática tareas, gracias a la identificación de las transformaciones entre un mundo y otro. Este nuevo archivo contendrá el modelo de tareas correspondiente al modelo de IU abstracta dado inicialmente. El entorno de especificación utilizado será gráfico tanto para la especificación de entrada como para la visualización de los resultados finalmente obtenidos.

3.1 Identificación y especificación de requisitos funcionales

Una de las primeras labores que hay que abordar en todo proceso de desarrollo es la identificación y especificación de aquellos requisitos indispensables y ligados al producto software que se pretende realizar. En este sentido, los requisitos funcionales, son aquellos que describen lo que debe hacer el sistema en cuanto a: funciones de actualización de datos, funciones de consulta, informes proporcionados, datos manejados e interacción con otros sistemas.

Para abordar la especificación de los requisitos funcionales se utiliza la técnica de diagramas de casos de uso. Un diagrama de casos de uso describe lo que hace un sistema desde el punto de vista de un observador externo, debido a esto, un diagrama de este tipo generalmente es de los más sencillos de interpretar en UML, ya que su razón de ser se concentra en: "*Qué hace el sistema*", a diferencia de otros diagramas UML que intentan dar respuesta a: "*Cómo logra su comportamiento el sistema*". Un caso de uso especifica una manera de usar un sistema sin revelar la estructura interna del mismo. De esta forma el conjunto de casos de uso especifica todas las posibles formas de usar un sistema, sin revelar cómo esto es implementado por el sistema.

En definitiva, las tareas de alto nivel con las que el usuario de una aplicación logrará sus objetivos se describen mediante la utilización de casos de uso. Los casos de uso muestran de forma clara e intuitiva las metas del usuario, en un lenguaje que puede ser entendido tanto por el usuario como el desarrollador.

En este proyecto final de carrera se han identificado tres casos de uso: "Editar especificaciones abstractas de interfaz de usuario", "Visualizar especificaciones de tareas"

y “Convertir de AUI a CTT”. Además, hay un actor denominado “Usuario” que representa a cualquier individuo que interactúa con la aplicación.

A continuación se muestra el diagrama de casos de uso planteado para el sistema.(Fig. 3-1).

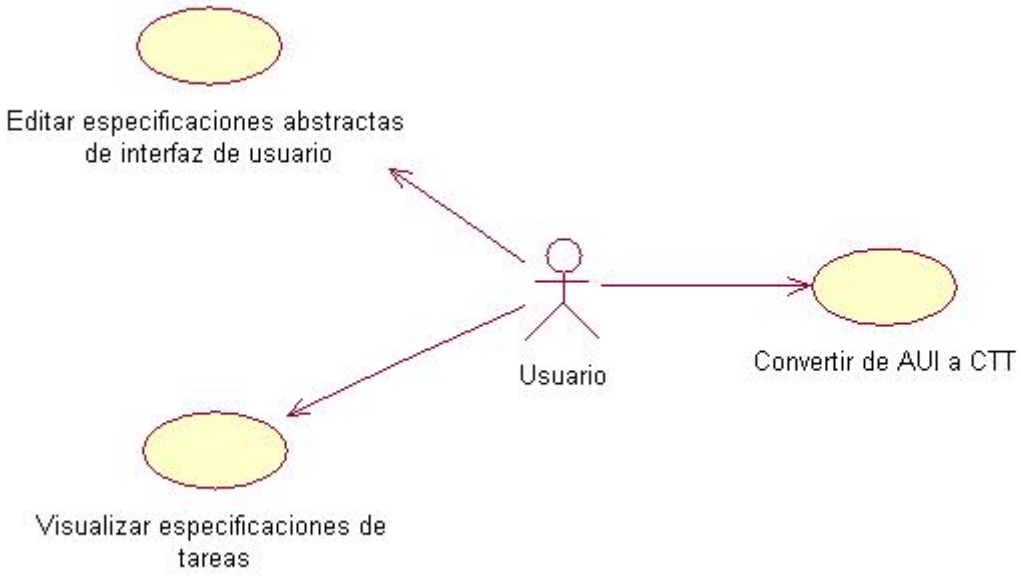

Figura 3-1. Diagrama de casos de uso del sistema.

En las siguientes tablas se describe el actor y los casos de uso de la figura anterior. Se comienza con la definición del único actor existente en el diagrama de casos de uso “Usuario”:

ACT-01	Usuario
Versión	1.0
Autor	Abraham Martínez
Descripción	Este actor representa a cualquier usuario que interactúe con la aplicación creada en este proyecto.
Comentarios	Ninguno

Tabla 3-1. ACT-01: Usuario.

La definición de los Casos de Uso del diagrama anterior, se muestra en las siguientes tablas:

UC-01	Editar especificaciones abstractas de interfaz de usuario	
Versión	1.0	
Autor	Abraham Martínez	
Descripción	El usuario puede realizar una serie de operaciones sobre la especificación abstracta, como: crear un diagrama nuevo, guardar un diagrama, cargar un diagrama, editar un diagrama,… Editar un diagrama consiste en añadir contenedores o componentes abstractos al diagrama, eliminarlos, cambiarlos de posición, modificar sus atributos,... Además puede añadir una serie de facetas a los componentes abstractos.	
Precondición	Ninguna	
Secuencia normal	Paso	Acción
	1	El usuario solicita al sistema la realización de una operación sobre el diagrama concreto.
	2	El sistema realiza la operación solicitada.
Postcondición	Ninguna	
Excepciones	Paso	Acción
	2	Si la operación solicitada por el usuario no es correcta, el sistema no la realiza, y avisa al usuario de la imposibilidad de realizar dicha operación.
Rendimiento	Paso	Cota de tiempo
	2	0.4 segundos
Frecuencia	Alta	
Estabilidad	Alta	
Comentarios	Ninguno	

Tabla 3-2. UC-01: Editar especificaciones abstractas de interfaz de usuario.

UC-02	Convertir de AUI a CTT	
Versión	1.0	
Autor	Abraham Martínez	
Descripción	Se realiza la conversión de la Interfaz de Usuario Abstracta que el usuario ha creado, y se obtiene la especificación de tareas equivalente.	
Precondición	Ninguna	
Secuencia normal	Paso	Acción
	1	El usuario solicita al sistema obtención de modelo de tareas a partir del diagrama abstracto editado.
	2	El sistema crea un fichero temporal del diagrama AUI existente.
	3	El sistema aplica la transformación XSLT al fichero AUI, generándose el fichero temporal correspondiente con el diagrama CTT.
	4	El sistema añade el tipo a las tareas generadas en el fichero de CTT, recorriendo el fichero temporal AUI.
	5	El sistema muestra en el editor de CTT el modelo generado en la transformación.
Postcondición	Ninguna	
Excepciones	Paso	Acción
	3	Si el sistema detecta algún parámetro incorrecto, avisa al usuario de que no puede llevarse a cabo la transformación.
Rendimiento	Paso	Cota de tiempo
	3	0.4 segundos
	4	0.4 segundos
Frecuencia	Alta	
Estabilidad	Alta	
Comentarios	Ninguno	

Tabla 3-3. UC-02: Convertir de AUI a CTT.

UC-03	Visualizar especificaciones de tareas	
Versión	1.0	
Autor	Abraham Martínez	
Descripción	El usuario puede realizar una serie de operaciones con el modelo de tareas, como: guardar un diagrama, cargar un diagrama, editar las tareas existente en el diagrama, eliminarlas, cambiarlas de posición, modificar sus atributos, modificar el tipo de las tareas, añadir relaciones,...	
Precondición	Ninguna	
Secuencia normal	Paso	Acción
	1	El usuario solicita al sistema la realización de una operación sobre el modelo de tareas.
	2	El sistema realiza la operación solicitada.
Postcondición	Ninguna	
Excepciones	Paso	Acción
	2	Si la operación solicitada por el usuario no es correcta, el sistema no la realiza, y avisa al usuario de la imposibilidad de realizar dicha operación.
Rendimiento	Paso	Cota de tiempo
	2	0.4 segundos
Frecuencia	Alta	
Estabilidad	Alta	
Comentarios	Ninguno	

Tabla 3-4. UC-03: Visualizar especificaciones de tareas.

El caso de uso “UC-02 Convertir de AUI a CTT”, es el que mayor interés despierta para este proyecto final de carrera, ya que es el que posibilita alcanzar el objetivo principal de este proyecto final de carrera. Por ello se ha creído conveniente estudiar los escenarios asociados a su realización, para ello se han utilizado diagramas de secuencia como los mostrados en la Fig. 3-2.

Un diagrama de secuencia muestra las interacciones entre objetos ordenadas en secuencia temporal. Muestra los objetos que se encuentran en el escenario y la secuencia de mensajes intercambiados entre los objetos para llevar a cabo la funcionalidad descrita por el escenario.

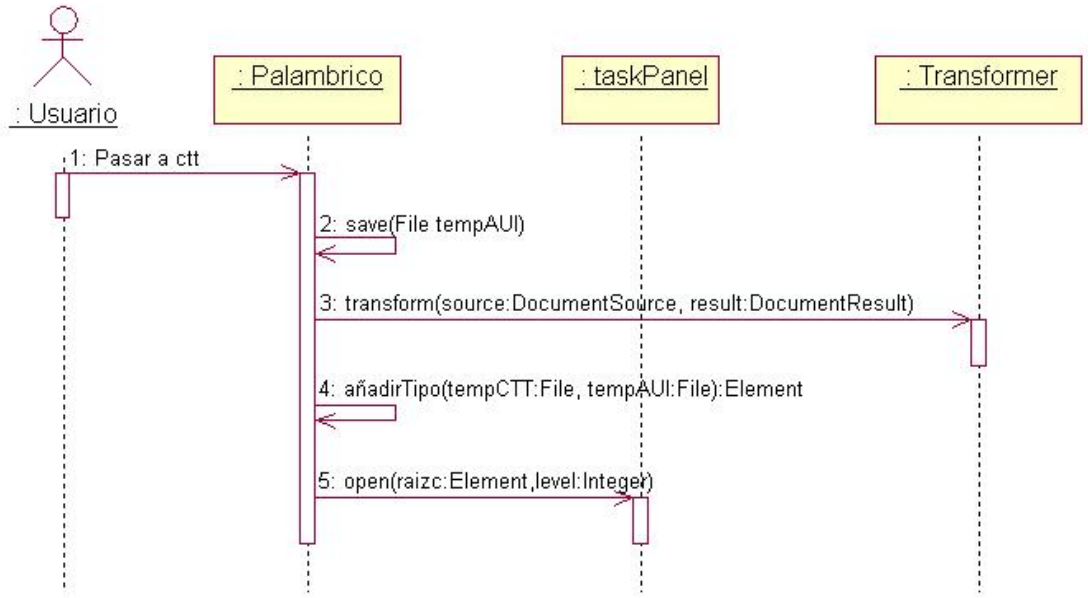

Figura 3-2. Diagrama de secuencia UC-02.

Este diagrama de secuencia refleja el proceso llevado a cabo por el programa desde que el usuario selecciona el botón “Pasar a ctt” hasta que el sistema muestra el modelo de tareas correspondiente. En este proceso, se generaran dos ficheros temporales, para almacenar los diagramas en formato xml y la transformación se realizara aplicando al fichero AUI una transformación XSLT. El último paso antes de mostrar el diagrama CTT, será añadir el tipo de las tareas de este diagrama, que se realizará inspeccionando la especificación abstracta.

3.2 Descripción estática de la aplicación

El diagrama de clases correspondiente al trabajo realizado con la herramienta que permite la transformación de AUI a CTT queda reflejado a continuación (Fig. 3-3):

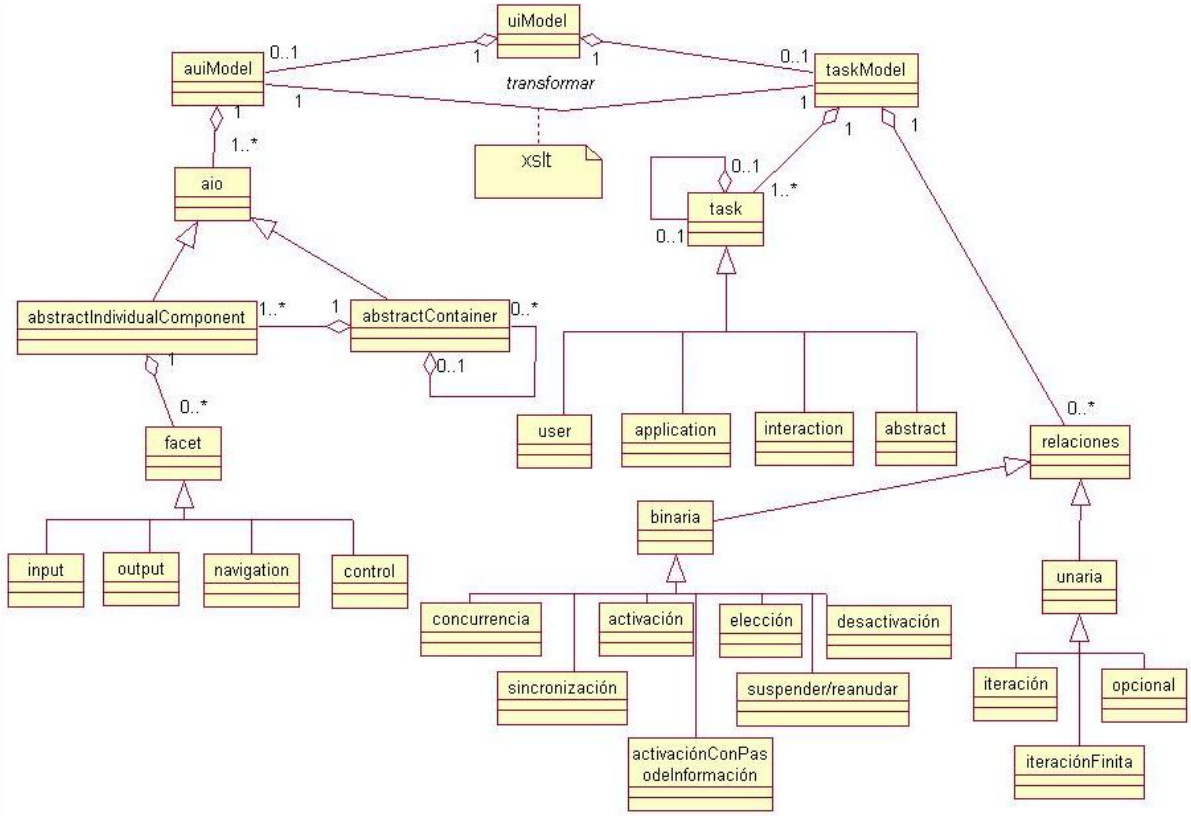

Figura 3-3. Diagrama de clases de la herramienta reTaskXML.

Una IU puede ser descrita, por varios tipos de modelos entre los que se encuentran el modelo de tareas y el modelo de IU abstracta.

El modelo AUI está compuesto por componentes y contenedores. Estos últimos a su vez pueden contener a otros componentes o contenedores. Los componentes pueden a su vez tener facetas que pueden ser de cuatro tipos. La principal diferencia de este diagrama con el presentado en el apartado de UsiXML para AUI, es que en este se ha decidido no añadir las relaciones abstractas entre objetos abstractos propuesta, no se han añadido ya que se consideran poco descriptivas y se resalta una mayor importancia de las relaciones temporales aportadas por CTT, por lo que se ha decidido añadir un atributo en el container con el tipo de relación temporal.

Desde un modelo AUI podemos pasar a su modelo de tareas correspondientes que es la tarea fundamental de este proyecto, como se indica este diagrama este proceso de transformación se realizará gracias a un documento XSLT que se encargará de pasar de un modelo a otro a partir del documento XML con el diagrama de AUI.

El diagrama CTT resultante contará con una jerarquía de tareas, que podrán estar relacionadas mediante relaciones temporales tanto unarias como binarias propias de la notación.

Para llevar a cabo este trabajo, se ha implementado en Java las clases que aparecen en el siguiente diagrama (Fig. 3-4), estas clases están agrupadas por paquetes, donde cada paquete corresponde con un subsistema del sistema general.

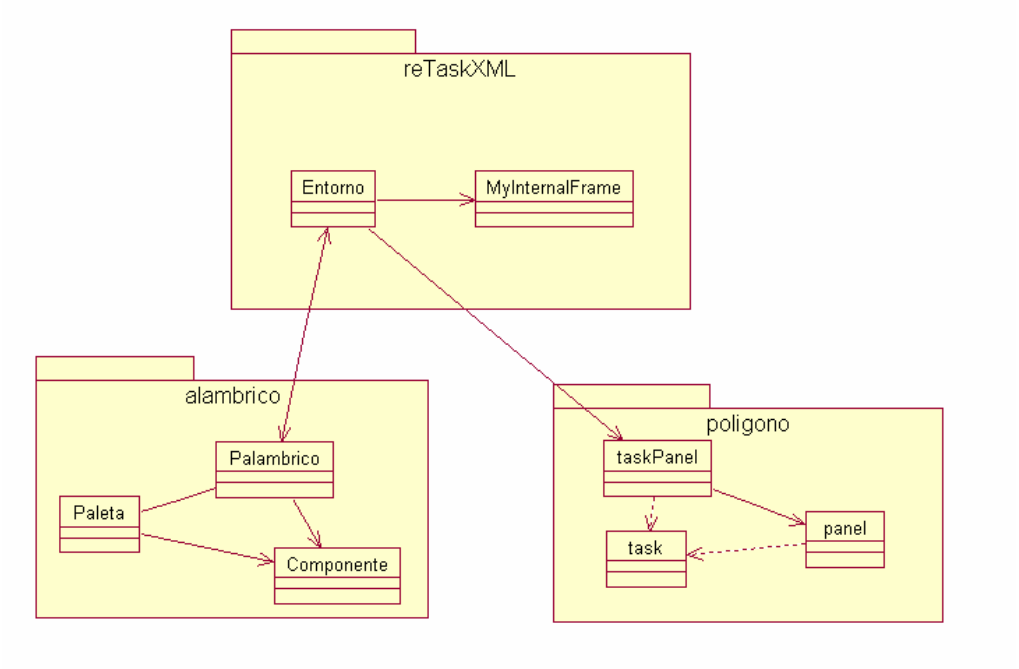

Figura 3-4. Diagrama de clases de implementación de la herramienta.

En el paquete alambrico es donde se encuentran las clases correspondientes al editor de AUI, en la clase Palambrico se encuentra el método convert() encargado de la transformación, mientras que en el paquete polígono es donde están las clases implementadas para el editor de CTT. Para la interfaz general, donde se mostrarán los editores y para el proceso de transformación se ha implementado el paquete ReTaskXML.

En los siguientes apartados se explican más en detalle los editores de ambos tipos de modelos y como se ha realizado la transformación, el producto final de este apartado es la herramienta ReTaskXML.

3.3 La especificación de interfaces de usuario a nivel abstracto

En este proyecto se ofrece la posibilidad de realizar especificaciones gráficas de interfaz a nivel abstracto. El editor de AUI facilitado se basa en la notación UsiXML para el modelo de interfaz abstracta. Donde la interfaz se describe usando contenedores abstractos y elementos individuales abstractos. También se incluyen los cuatro tipos de facetas que se pueden asociar a los componentes.

Como novedad en la gestión de relaciones entre componentes de interfaz, en este proyecto final de carrera se ha decidido no utilizar las relaciones entre elementos abstractos tal y como se sugieren en la propuesta UsiXML, hemos considerado una notación, a nuestro juicio más flexible basada en los operadores temporales que aporta la notación CTT, al ser más descriptivos y completos. Además, de esta manera se facilitan las transformaciones entre el mundo abstracto y el mundo de las tareas.

Al realizar la transformación de una especificación abstracta de IU a notación CTT, por tanto, apareció la necesidad de reflejar de algún modo las relaciones temporales que aparecen el los diagramas de CTT, ya que de lo contrario el mapeo de un diagrama a otro no sería completo. Se decidió añadir a los container un nuevo atributo que fuese el tipo de relación, que se corresponde con las existentes en CTT, de modo que al seleccionar un valor para ese atributo, todos los elementos que contenga ese contenedor estarán relacionados en su correspondiente diagrama de CTT con la relación seleccionada.

La herramienta desarrollada en este proyecto, reTaskXML, ha sido realizada con el editor JBuilder en leguaje Java. El editor para la generación de diagramas AUI (Fig. 3-5), permite además de la edición, el guardado y apertura de diagramas en formato XML.

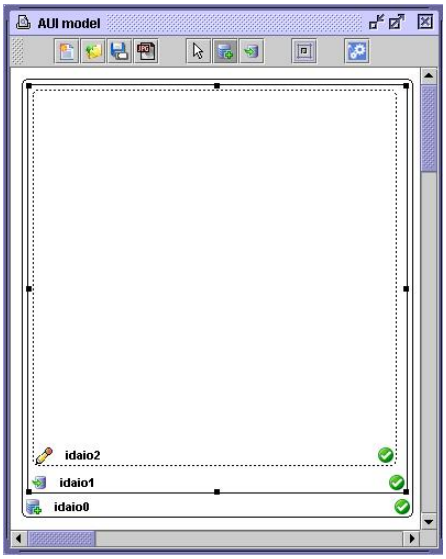

Figura 3-5. Editor de AUI.

3.4 Transformación de AUI a CTT

La tabla de equivalencias de una notación, apoyada por la transformación inversa mostrada en el apartado de IdealXML, se muestra en la tabla siguiente (Tabla 3-5):

AUI		CTT
contenedor		tarea abstracta
		tarea de interacción si no es hoja
		tarea de aplicación si no es hoja
componente	entrada	tarea de interacción si es hoja
	salida	tarea de interacción si es hoja
		tarea de interacción si es hoja
	control	tarea de interacción si es hoja
		tarea de aplicación si es hoja
	navegación	tarea de interacción si es hoja
		tarea de aplicación si es hoja

Tabla 3-5. Transformación de AUI a CTT.

El principal problema de este mapeo se presenta en que para la mayoría de las facetas de un componente, tienen como equivalencia una tarea de aplicación o una tarea de interacción, dependiendo de si la tarea la lleva a cabo la máquina directamente o si se necesita la intervención del usuario para completarla.

Esto hace necesario añadir un nuevo atributo a las facetas que permita determinar quién realiza esa tarea y así tener claro que tipo de tarea le corresponde. Este atributo denominado actionType, estará por defecto al valor “interaction”. Para las facetas de entrada sólo se permitirá el tipo interacción, mientras para el resto de facetas se puede seleccionar entre interacción y aplicación.

La localización de la herramienta presentada en este proyecto dentro del marco de trabajo de UsiXML(Fig. 3-6) es similar a IdealXML, pero se encarga de la transformación en sentido inverso, permitiendo la edición de modelos UsiXML de IU abstracta, para su transformación a modelos de tareas.

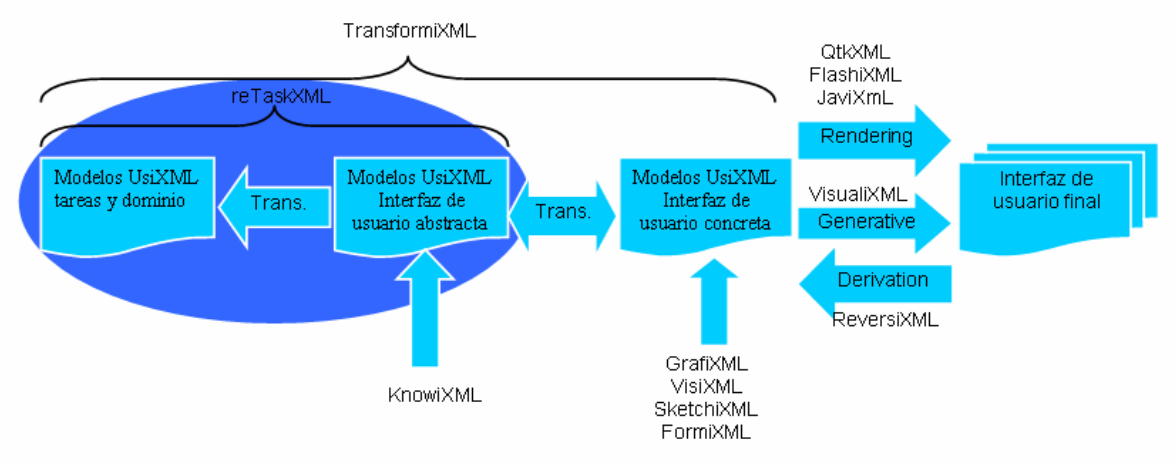

Figura 3-6. Localización de la herramienta ReTaskXML dentro de UsiXML

Una vez realizado con el editor el diagrama de AUI, este será guardado en lenguaje xml. Para realizar, generación del modelo de tareas asociado a esa una interfaz abstracta dada por el documento, será necesario realizar la transformación de el documento xml con la interfaz abstracta, en otro documento xml que pueda ser leído por el editor de CTT. Para ellos se hace necesaria la utilización del lenguaje XSLT.

3.5 Transformación de documentos XML mediante XSLT

La familia de recomendaciones del W3C para definir la transformación y presentación de documentos XML. Tiene tres partes: XSLT, XPath (lenguaje de expresión usado por XSLT y por XLink, para acceder o referirse a partes de un documento XML) y XSL-FO o propiamente XSL (vocabulario XML para especificar una semántica de formato).

XSLT o XSL Transformaciones es un estándar que presenta una forma de transformar documentos XML en otros e incluso a formatos que no son XML, como HTML o algún otro lenguaje basado en texto. Las hojas de estilo (aunque el termino de hojas de estilo no se aplica sobre la función directa del XSLT) XSLT realizan la transformación del documento utilizando una o varias reglas de plantilla: unidas al documento fuente a transformar, esas reglas de plantilla alimentan a un procesador de XSLT, el cual realiza las transformaciones deseadas colocando el resultado en un archivo de salida o, como en el caso de una página Web, directamente en un dispositivo de presentación, como el monitor de un usuario.

Cuando se escribe un documento XSL, es necesario tener en cuenta que se va a trabajar con una serie de conceptos fundamentates. Estos conceptos son los siguientes:

- **Plantilla**: Las plantillas XSL se utilizan para definir los patrones que se aplicarán a cada tipo de elemento que se encuentre en el documento de datos o XML.

- **Patrón**: Los patrones definen qué elementos se transformarán mediante las transformación XSL.

- **Plantilla raíz**: Realmente no es un elemento especial, aunque puede tratarse así ya que se trata de la plantilla que define el comportamiento de la transformación para el elemento raíz del documento XML.

- **Plantilla predeterminada**: Esta plantilla es la que se aplicará a todos aquellos elementos del documento XML que no tengan ninguna plantilla definida para ellos.

Actualmente, XSLT es muy usado en la edición Web, generando páginas HTML o XHTML. La unión de XML y XSLT permite separar contenido y presentación, aumentando así la productividad.

Ventajas del uso de XSLT:

- Facilidad de mostrar datos formateados en el navegador.

- Facilidad de modificar cuando cambia el formato de los datos XML más que modificar el código de parsers SAX y DOM.

- Se pueden utilizar con consultas de base de datos que devuelven XML.

Desventajas:

- Necesidad de aprender un nuevo lenguaje.

- Dificultad de implementar complejas reglas de negocio.

- Intensidad de memoria y sufrimiento de una penalización en desarrollo.

- No se puede cambiar el valor de las variables.

3.5.1 Especificación de las transformaciones definidas

Para facilitar el trabajo con XSLT se ha utilizado la herramienta Altova XMLSpy, que permite editar y depurar documentos XSLT aplicándoles un documento XML.

Uno de los primeros problemas que se planteó para la transformación fue la necesidad de especificar relaciones temporales a nivel de la especificación abstracta para reflejar en el modelo de tareas generado esas relaciones. Para ello se decidió asociar esos operadores a los *Containers* añadiéndoles el atributo *temporalOperator* (Fig. 3-7). Si a un componente se le ha editado dicho atributo todos los elementos abstractos que son hijos suyos estarán relacionados con una relación de ese tipo.

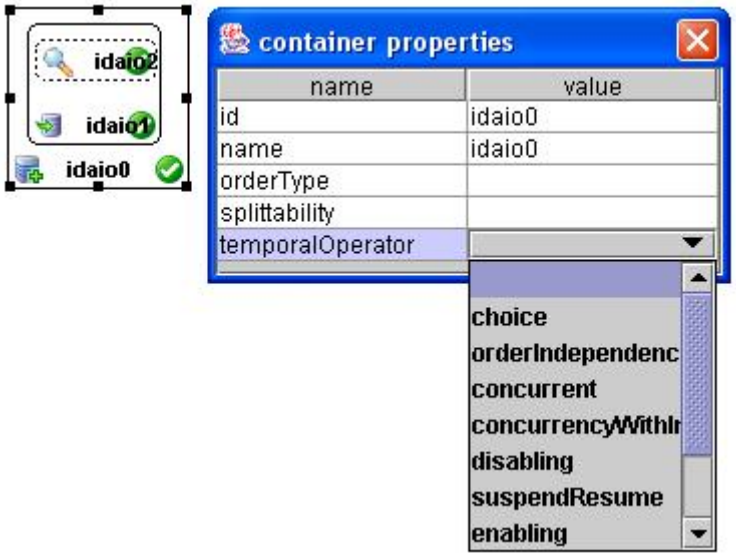

Figura 3-7. Edición de propiedades temporales para un container.

Inicialmente se desarrolló una codificación XSLT con XMLSpy, compuesta por un documento con las *templates* y otro con las funciones necesarias para establecer el tipo de las tareas correspondientes a los AIO. Una vez conseguido el correcto funcionamiento en XMLSpy, al pasar esta versión de XSLT a java e intentar realizar la transformaciones comenzaron a aparecer errores y no se consiguió hacer funcionar la parte de las funciones para extraer el tipo de la tarea. Tras varios intentos fallidos y correcciones para hacer compatible el documento para hacerlo funcionar en java, se decidió mantener el fichero con las *templates* que permite obtener las tareas correspondientes al fichero AUI y las relaciones temporales que se indicasen en los contenedores. La parte problemática para extraer el tipo de las tareas a partir de los AIO hijos, se decidió realizar mediante código java haciendo uso de la librería de código abierto Dom4j.

“Dom4j” (**M**odelo de **O**bjetos del **D**ocumento para **J**ava) para facilitar el trabajo con los ficheros XML que se tienen que generar y manejar en esta aplicación. “Dom4j” permite trabajar con ficheros XML en la plataforma Java.

Esta librería permite ver el mismo documento en otro formato, describiendo el contenido del documento como un conjunto de objetos, y un programa Java puede actuar sobre ellos. Su página oficial es: www.dom4j.org, y en ella se puede encontrar mucha información sobre esta librería.

En el fichero XSLT xsltransforma.xsl, al encontrar la etiqueta auimodel (Fig. 3-8) se realizarán dos barridos del contenido de esta etiqueta, para distinguirlos se ha añadido el atributo mode=”mode1” en xsl:template para el primero y se utiliza el dado por defecto para el segundo recorrido. El primero servirá para obtener todas las tareas con sus atributos correspondientes, para ello comprobará las etiquetas correspondientes a abstractContainer y a abstractIndividualComponent. La segunda vez que se recorre el documento es para extraer las relaciones existentes que existirán si alguno de los contenedores tiene el atributo temporalOperator, por ello sólo interesará la etiqueta abstractContainer para enlazar a sus hijos.

```
- <xsl:template match="auimodel">
  - <taskmodel>
      <xsl:apply-templates mode="mode1"/>
      <xsl:apply-templates/>
    </taskmodel>
  </xsl:template>
```

Figura 3-8. Recorrido del xsl a partir de la etiqueta auimodel.

El mayor problema encontrado para realizar la transformación ha sido el sacar el tipo de las tareas a partir de los elementos del diagrama AUI. Las decisiones tomadas para sacar el tipo de las tareas son:

Para un contenedor: si todos los AIO que contiene son del mismo tipo entonces el tipo del contenedor será el mismo que el de sus AIO. De lo contrario, al contenedor le corresponderá una tarea de tipo abstracto.

En el ejemplo (Fig. 3-9), el tipo del contenedor vendrá determinado por el tipo de los dos componentes que contiene. Al componente idaio1 le corresponderá siempre una tarea de interacción por su faceta de entrada. El tipo de la tarea para el contenedor, vendrá por tanto determinada por el tipo de la tarea correspondiente al componente idaio3. Al tener una faceta de control el tipo de la tarea correspondiente dependerá de la propiedad *actionType* de la faceta, si la propiedad dice que la acción es de interacción (valor por defecto) al componente idaio3 le corresponderá una tarea de interacción y por tanto la tarea

correspondiente a idaio0 será de interacción. En el caso de que la propiedad indique que es de aplicación, la tarea idaio3 será de aplicación y por tanto idaio0 será una tarea abstracta al no coincidir los tipos de sus tareas hijas.

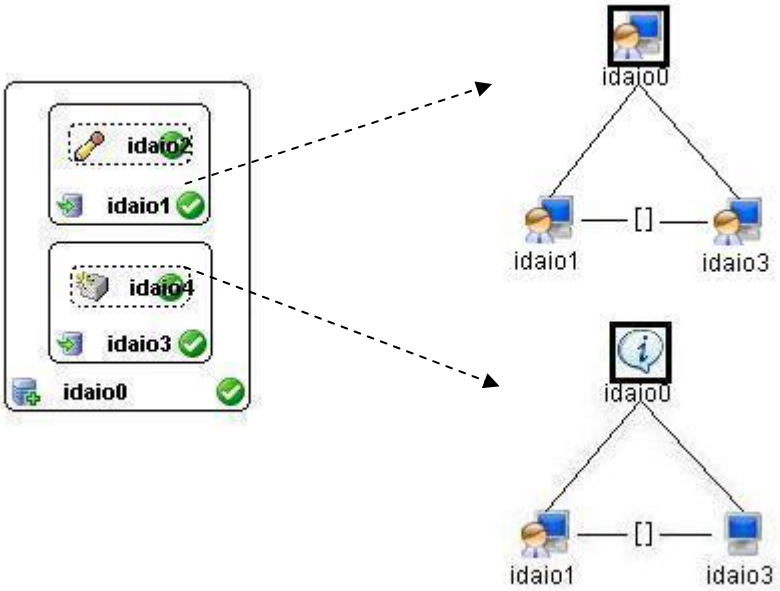

Figura 3-9. Equivalencia de tareas para un contenedor.

Para un componente: su tipo vendrá determinado por las facetas que contiene. Si todas sus facetas son del mismo tipo y además todas tienen el mismo valor para su atributo tipo, entonces la tarea del componente será de ese tipo. En otro caso, al componente le corresponderá una tarea de tipo abstracto.

Para una faceta: su tipo vendrá determinado por el valor de su atributo actionType, si este no se ha especificado se le dará el valor por defecto interaction que es válido para todas las facetas. Se puede consultar la equivalencia de las facetas en Tabla 3-1.

3.5.2 Utilización e invocación de las transformaciones definidas

Una vez creado el diagrama deseado con el editor de AUI, se debe seleccionar “pasar a CTT” del menú herramientas. Al hacer esto, en primer lugar se crea un fichero temporal XML del diagrama de AUI. A ese fichero se le aplica la transformación XSLT obteniendo otro fichero temporal XML que puede ser abierto por el editor de CTT (Fig. 3-10).

Para aplicar el XSLT al documento XML con el diagrama de AUI, se han utilizado las funciones que proporciona la librería dom4j, que cuenta con una función para transformar un documento XML en otro aplicándole la hoja de estilo XSLT deseada.

Este proceso será totalmente transparente al usuario ya que éste sólo verá como desde su diagrama de IU abstracta, se genera el correspondiente diagrama de tareas CTT en una nueva ventana.

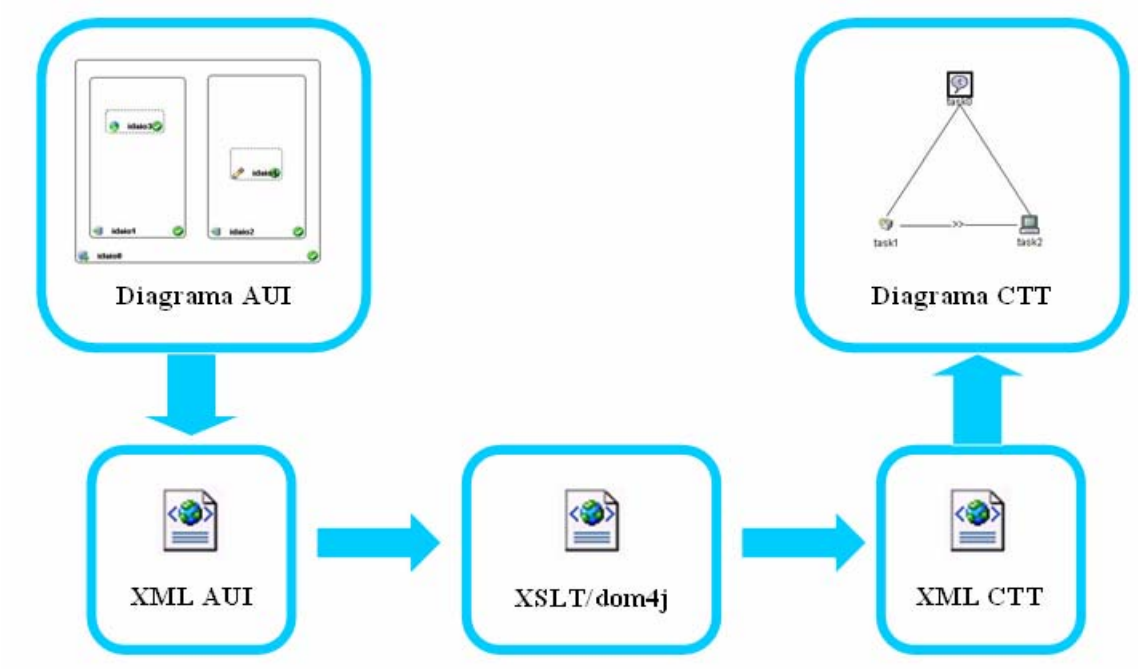

Figura 3-10. Proceso de transformación realizado por la herramienta reTaskXML.

Como se ha comentado anteriormente, se ha optado por una solución en la que con la transformación realizada con el documento XSLT se obtiene la estructura básica del modelo de tareas CTT, obteniendo las tareas correspondientes a ese diagrama, así como las relaciones temporales. Esta diagrama se completa añadiendo mediante código java el tipo a esas tareas, para ello se han implementado los métodos necesarios es la clase Palambrico del paquete alambrico. Estos métodos harán un recorrido del documento XML correspondiente a AUI e irán estableciendo los tipos para las tareas dependiendo de los hijos de los componentes y contenedores.

3.6 La especificación tareas siguiendo la notación CTT

El editor de CTT (Fig. 3-11) es el utilizado en la herramienta IdealXML. Tras realizar la transformación desde AUI, se muestra el modelo de tareas equivalente, que es posible editar para adaptarlo a las necesidades del usuario.

El modelo de tareas equivalente, puede ser guardado en formato XML para usos posteriores. Este editor permite también abrir ficheros XML que contengan diagramas CTT. También se puede visualizar el código XML generado correspondiente al diagrama mostrado.

Al completar este paso, se ha conseguido el objetivo principal de este proyecto fin de carrera, que era elaborar una herramienta que permitiese la generación del modelo de tareas asociado a una interfaz abstracta especificada previa y gráficamente.

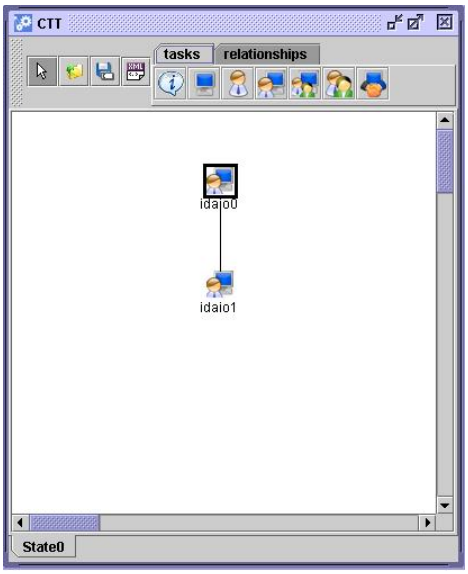

Figura 3-11. Editor de CTT.

Los tipos de tareas utilizados por la herramienta y sus iconos correspondientes son muy similares a los utilizados en la notación CTT(Tabla 3-6). Los tipos de relaciones temporales son exactamente las mismas que las de la notación.

Tipo de tarea(icono)	Descripción
	Tarea de usuario.
	Tarea de la aplicación.
	Tarea de interacción.
	Tarea abstracta .
	Cooperación en tareas de interacción.
	Tareas de cooperación.
	Colaboración en tareas de interacción.

Tabla 3-6. Tipos de tareas para el editor de CTT.

3.7 Manual de la herramienta reTaskXML

A continuación se presenta la información de mayor importancia para cualquier usuario que desee uso de la herramienta presentada en este proyecto, ya que es fundamental contar con un manual para asesorarle en el funcionamiento y que le sirva de referencia mientras se está trabajando con la herramienta.

3.6.1 Manual de Ayuda

Hoy en día, cualquier programa dispone de un manual de ayuda que los usuarios pueden consultar para aprender su manejo y resolver dudas que les puedan ir surgiendo. Por este motivo se consideró oportuno realizar un manual de ayuda, que los usuarios pudieran consultar durante el trabajo con la herramienta.

En este manual se realiza una pequeña introducción a la herramienta, para que el usuario conozca la funcionalidad de esta. También se explica como el usuario puede crear y editar sus propios diagramas y como realizar la transformación de especificación abstracta a modelo de tareas. Se han realizado dos ejemplos prácticos para orientar al usuario y que este pueda observar paso a paso el funcionamiento de esta. La estructura de este manual de ayuda es la siguiente:

1.- ¿Qué es reTaskXML?
2.- ¿Cómo utilizar la herramienta reTaskXML?
3.- Ejemplos
    3.1.- Ejemplo1: Sistema de control de acceso
    3.2.- Ejemplo2: Sistema de reserva de habitación de hotel
4.- Acerca de

El usuario puede encontrar el manual de ayuda y la información general acerca de la herramienta, en la barra de menú pestaña “Ayuda” (Fig. 3-12):

Figura 3-12. Menú para abrir la ayuda.

Al seleccionar “Manual Ayuda” aparecerá una nueva ventana, donde el usuario podrá navegar por todos los apartados del manual (Fig. 3-13).

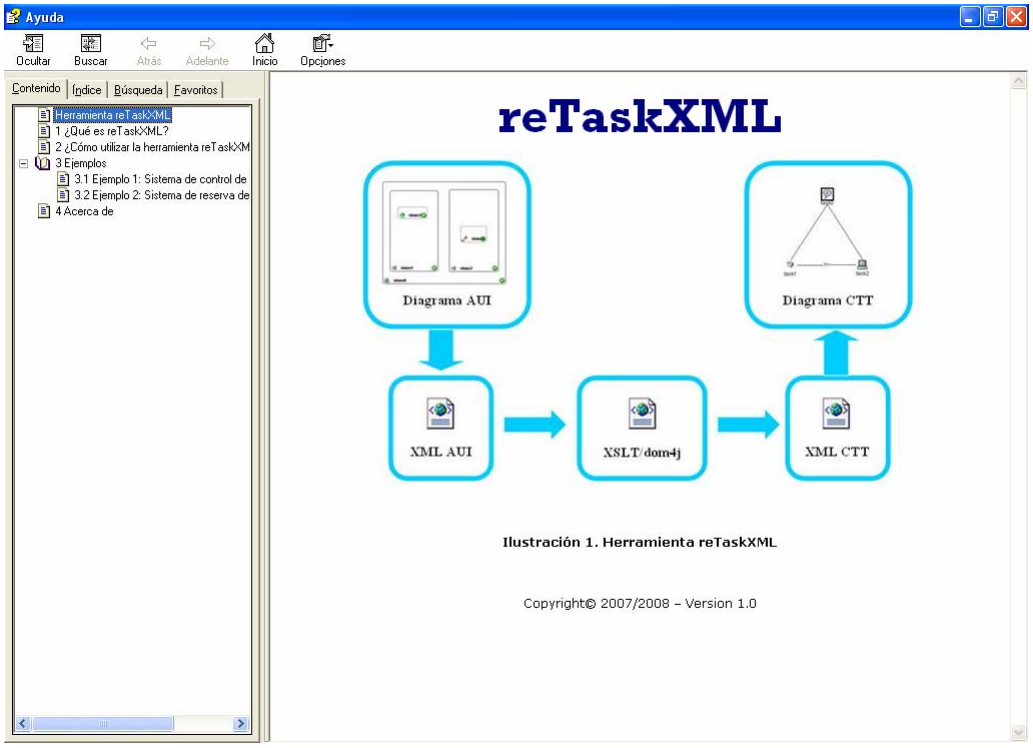

Figura 3-13. Manual de ayuda.

3.6.2 ¿Cómo utilizar la herramienta reTaskXML?

Al ejecutar la herramienta aparece la ventana general donde se muestra en la barra superior un menú general (Fig. 3-14). En la pestaña Archivo se permite crear nuevo diagrama, abrir y guardar así como salir de la aplicación. En el menú Edición están las pestañas para deshacer, rehacer y limpiar el contenido del diagrama. Herramientas cuenta con la opción de pasar a CTT y Ayuda cuenta con Acerca de con información general de la herramienta. En la parte central se muestra la ventana con el editor de AUI.

El editor de AUI cuenta con un acceso directo a las opciones más comunes para este tipo de diagramas, como crear nuevo, guardar, abrir,…, que aparecen con sus correspondientes iconos en la parte superior de la ventana.

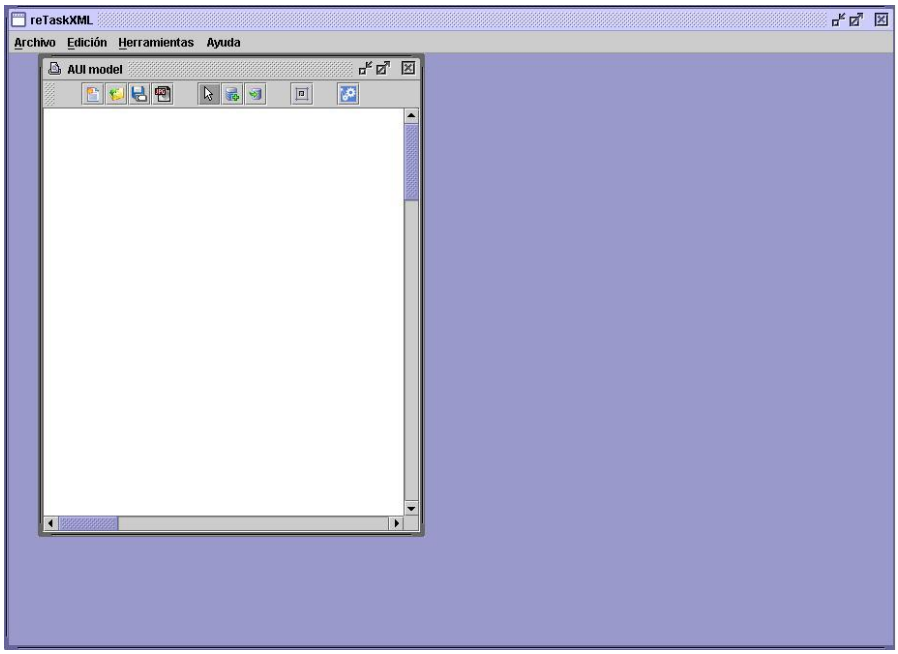

Figura 3-14. Aspecto inicial de la herramienta.

Para comenzar un diagrama debemos seleccionar el botón para crear un componente o contenedor. Al agregar un contenedor, podemos visionar las opciones permitidas haciendo clic derecho sobre él (Fig.3-15). En este menú podremos añadirle nuevos componentes o contenedores, editarlo para modificar sus propiedades, cambiar su color, ordenarlo e incluso borrarlo.

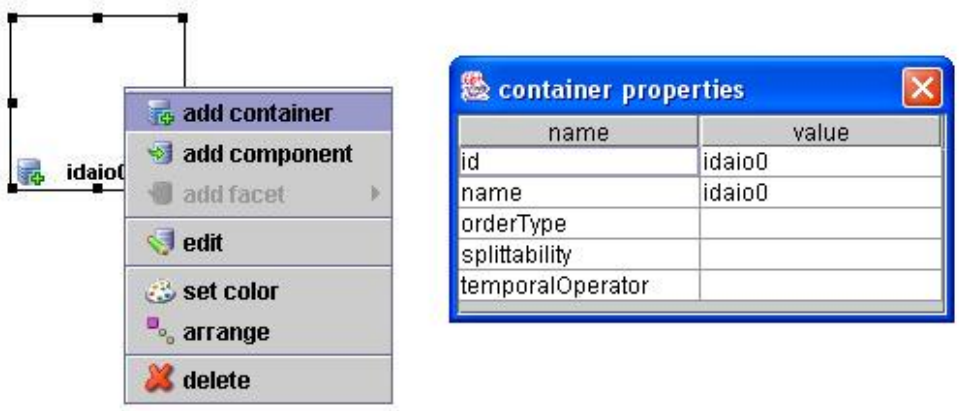

Figura 3-15. Acciones permitidas y propiedades editables de un contenedor.

Al añadir un componente las opciones permitidas haciendo clic derecho(Fig. 3-16), son añadirle facetas y las opciones comunes de editarlo, cambiar su color, ordenarlo y borrar.

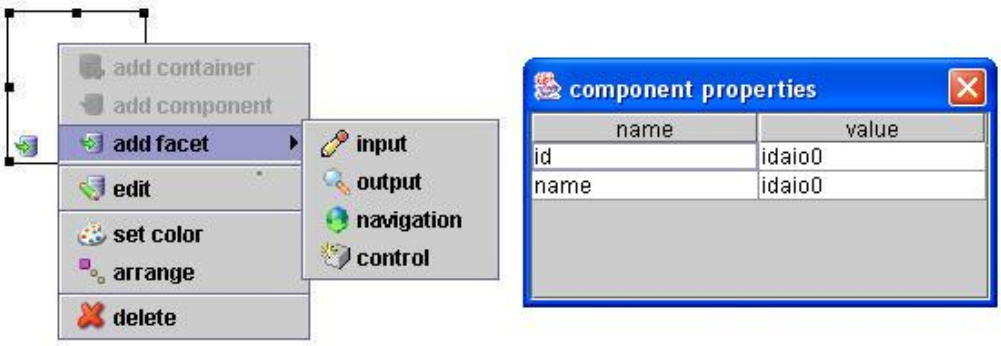

Figura 3-16. Acciones permitidas y propiedades editables de un componente.

Podemos añadir facetas a los componentes(Fig. 3-17), con la opción del componente “add facet”, para ello seleccionaremos su tipo, en una faceta lo más interesante será editar sus propiedades.

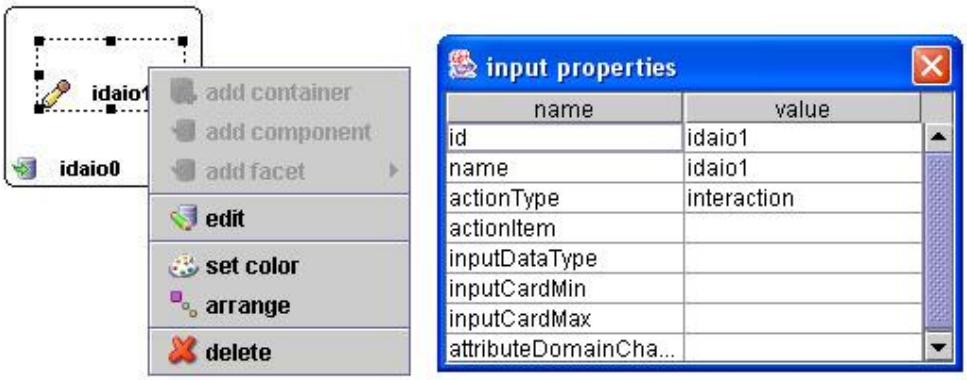

Figura 3-17. Acciones permitidas y propiedades editables de una faceta.

También podemos ordenar los elementos con el botón , para que estén mejor organizados y ocupen toda la ventana(Fig. 3-18).

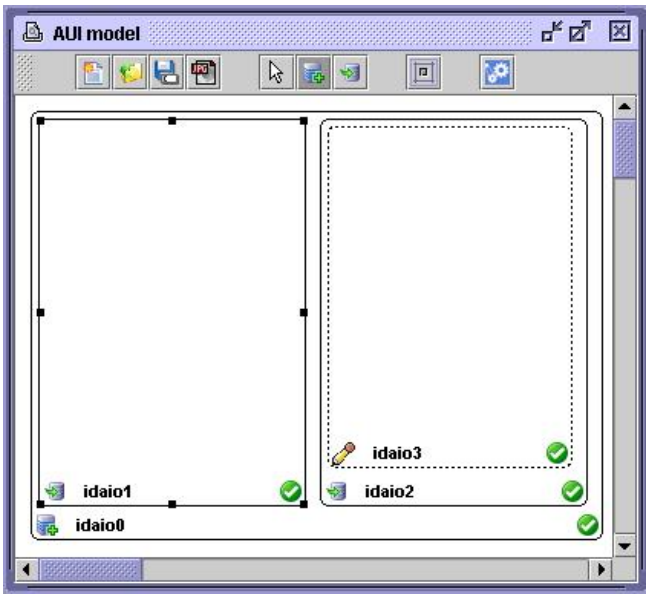

Figura 3-18. Diagrama de AUI ordenado.

La opción que más nos interesa una vez creado nuestro diagrama en AUI, es la que nos permite pasar a CTT, para ello podemos seleccionar Herramientas/pasar a CTT ó pulsar el icono que se muestra en la ventana AUI.

Una vez seleccionada la opción de transformación se nos mostrará en pantalla la ventana con el diagrama CTT equivalente (Fig. 3-19). Este diagrama puede ser editado por el usuario, haciendo uso de las opciones disponibles en la ventana.

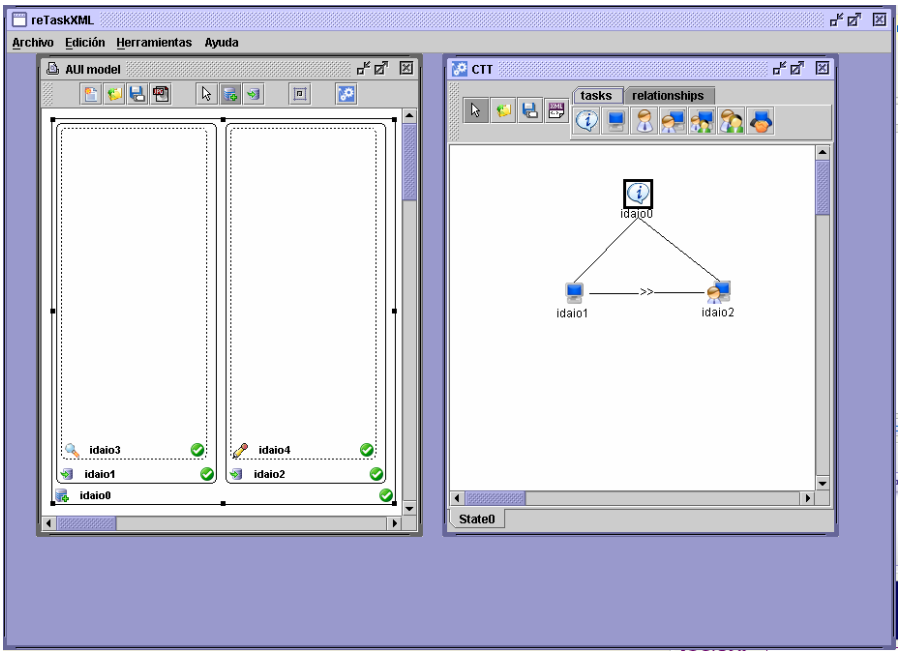

Figura 3-19. Diagrama de AUI ordenado.

Cualquier tarea del diagrama puede ser editada para modificar sus propiedades (Fig. 3-20), para que el usuario pueda adaptar el modelo de tareas resultante.

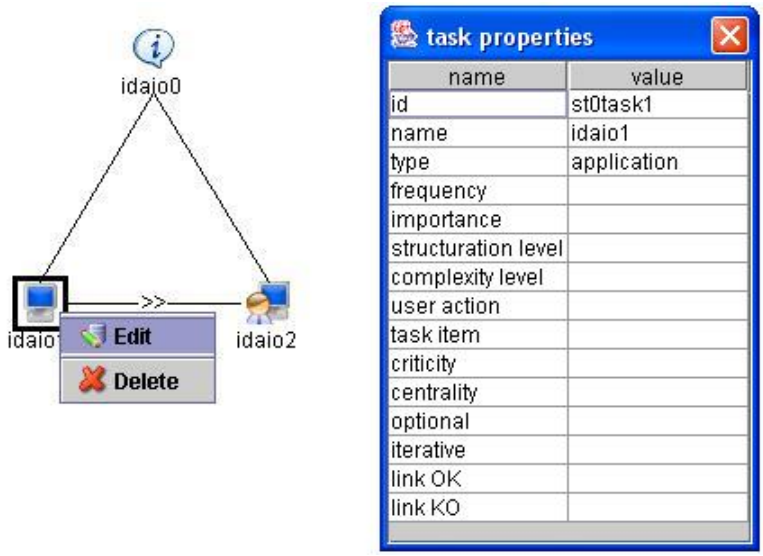

Figura 3-20. Edición de propiedades de tareas.

El editor de tareas también cuenta con un acceso directo a las opciones más comunes para este tipo de diagramas, como guardar, abrir o ver el código xml correspondiente al diagrama, que aparecen con sus correspondientes iconos en la parte superior de la ventana. También cuenta con unas pestañas para poder ir agregando tareas o cambiar de pestaña para añadir relaciones.

3.8 Análisis y conclusiones

En la figura de la página siguiente se muestra un esquema de las fases seguidas en el desarrollo de reTaskXML(Fig. 3-21).

El proceso comienza con la etapa de adquisición de requisitos funcionales para la futura aplicación. En la etapa de análisis de requisitos se estudian los requisitos descritos anteriormente y realizan diagramas de secuencia y de clases. En la etapa de diseño ha sido fundamental el definir cómo realizar la transformación de AUI a CTT. Finalmente, la etapa de implementación genera el código de la aplicación que será ejecutado dentro de la arquitectura.

Esta última fase de implementación, se decidió utilizar código Java, pero se le ha dado una gran importancia al lenguaje XML como formato para abrir y guardar los diagramas generados, para lo cuál se ha seguido el estandar UsiXML. En la transformación se optó por utilizar XSLT aunque se ha necesitado la ayuda de código por problemas de compatibilidad con Java.

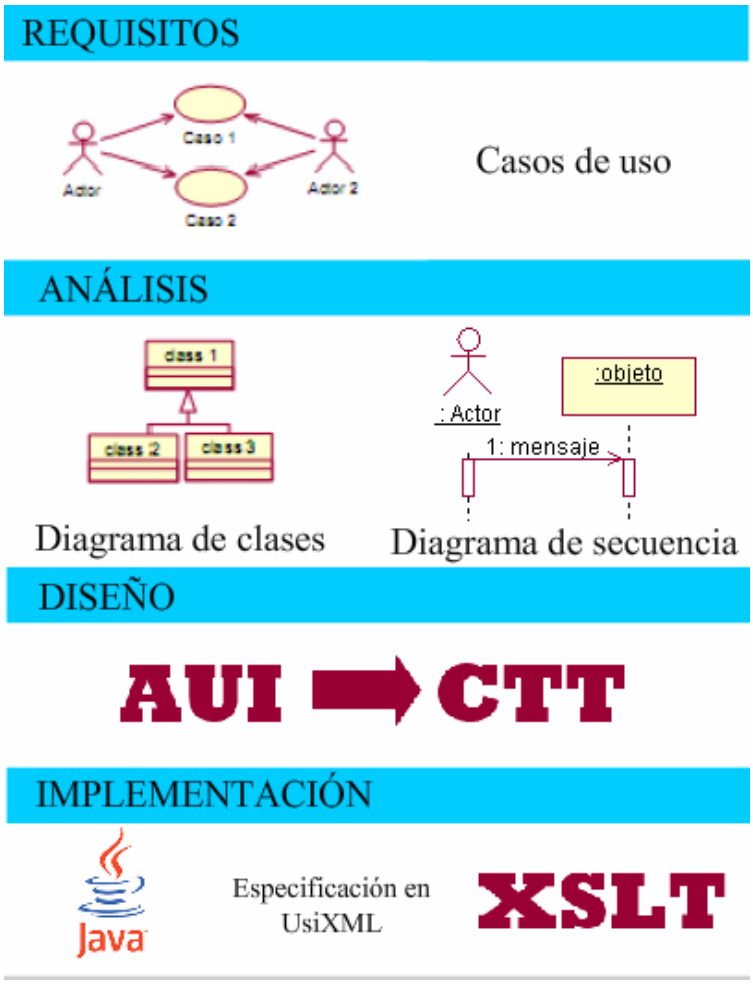

Figura 3-21. Fases de desarrollo para reTaskXML.

En el siguiete capítulo se pondrá en práctica todo este desarrollo, mediante una serie de casos de prueba, que sirven como guía para el futuro usuario y demuestran el funcionamiento de la herramienta.

Capítulo 4. CASOS DE ESTUDIO

A continuación se muestran unos ejemplos prácticos realizados con la nueva herramienta creada: reTaskXML.

4.1 Caso de estudio 1: Sistema de control de acceso

Este primer caso consiste en un sencillo sistema para control de acceso mediante un login y un password, para permitir sólo el acceso a usuarios autorizados (Fig. 4-1). Primero debemos introducir el login, al introducirlo se permite introducir la clave correspondiente y al introducir ambos se permite pulsar aceptar.

Figura 4-1. Interfaz de usuario concreta de control de acceso.

Esta interfaz concreta se compone a nivel abstracto de un contenedor, que hemos llamado Acceso, este contenedor contiene tres componentes. Login para introducir el usuario y que tiene asociada una faceta de entrada. El segundo componente es Password y contiene una faceta de entrada para introducir la contraseña. El último componente se corresponde el botón Aceptar y por tanto tiene asociada una faceta de control de tipo interaccion(Fig. 4-2).

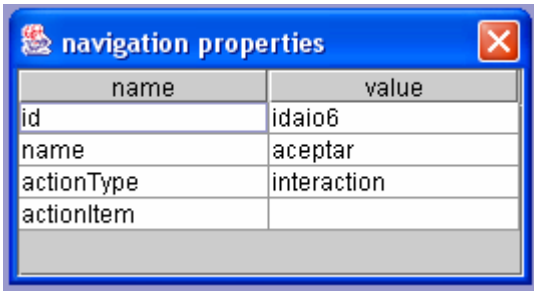

Figura 4-2. Propiedades de la faceta correspondiente al componente aceptar.

Como se ha dicho que al poner el usuario se habilita el introducir la contraseña y esta habilita la opción de aceptar el contenedor tendrá como operador temporal enabling(Fig. 4-

3), que relaciona a sus componentes y esto quedará posteriormente reflejado en el modelo de tareas correspondiente.

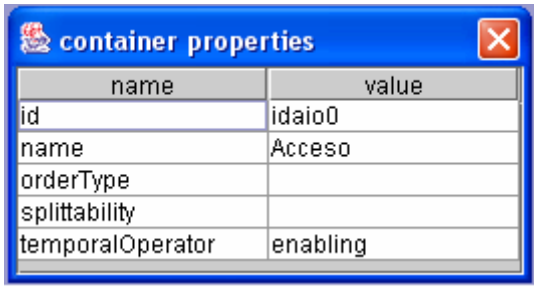

Figura 4-3. Propiedades del contenedor de control de acceso.

La IU abstracta correspondiente a este ejemplo de control de acceso, editado con la herramienta ReTaskXML(Fig. 4-4), le corresponde la especificación a nivel abstracto(Fig. 4-5) siguiendo el estandar UsiXML.

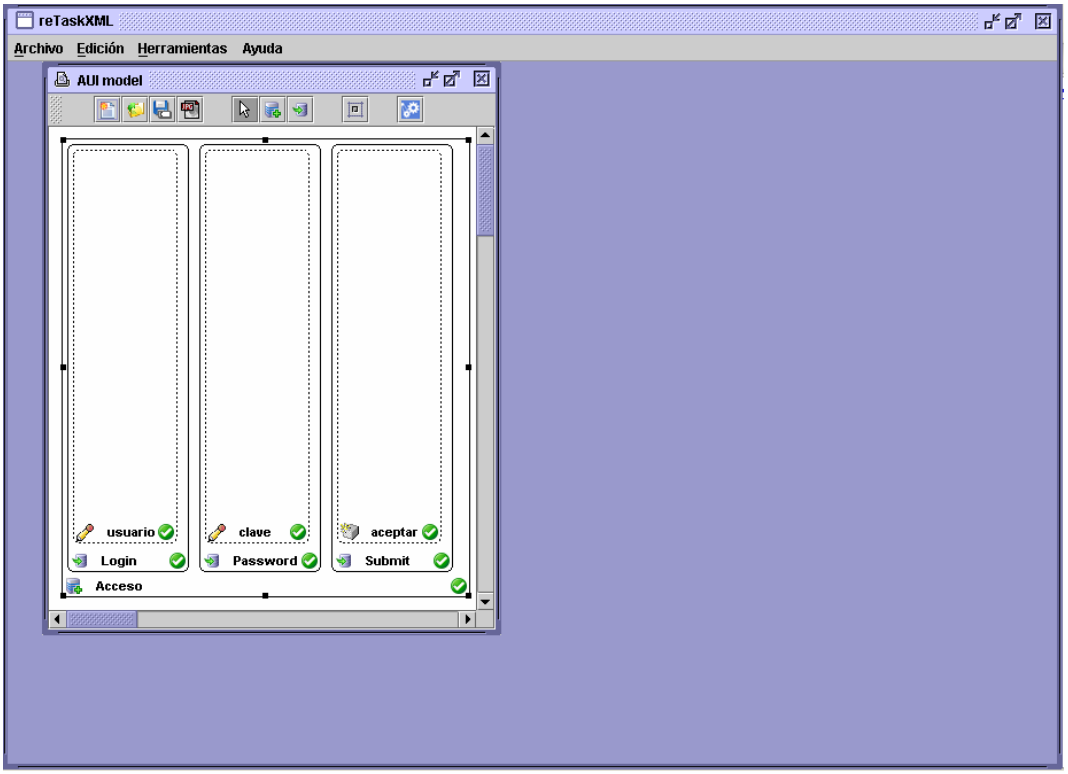

Figura 4-4. Interfaz de usuario abstracta de control de acceso.

```
- <auimodel>
  - <abstractContainer id="idaio0" name="Acceso" temporalOperator="enabling">
    - <abstractIndividualComponent id="idaio1" name="Login">
        <input id="idaio4" name="usuario"/>
      </abstractIndividualComponent>
    - <abstractIndividualComponent id="idaio2" name="Password">
        <input id="idaio5" name="clave"/>
      </abstractIndividualComponent>
    - <abstractIndividualComponent id="idaio3" name="Submit">
        <navigation id="idaio6" name="aceptar" actionType="interaction"/>
      </abstractIndividualComponent>
    </abstractContainer>
  </auimodel>
```

Figura 4-5. Código de especificación de control de acceso a nivel abstracto.

Al seleccionar pasar a ctt  se genera un fichero temporal con la especificación a nivel abstracto(Fig. 4-5). A este código xml se le aplicará la codificación xslt generándose otro archivo temporal con la especificación del modelo de tareas(Fig. 4-6).

```
- <taskmodel>
  - <task id="idaio0" name="Acceso">
      <task id="idaio1" name="Login"/>
      <task id="idaio3" name="Password"/>
      <task id="idaio5" name="Submit"/>
    </task>
  - <enabling>
      <source sourceId="idaio1"/>
      <target targetId="idaio3"/>
      <source sourceId="idaio3"/>
      <target targetId="idaio5"/>
    </enabling>
  </taskmodel>
```

Figura 4-6. Código de especificación de control de acceso en modelo de tareas.

A esta especificación le falta añadir los tipos de las tareas, que se añadirán mediante código java haciendo uso de la estructura de nodos que se genera al abrir este fichero desde el editor de tareas. Para establecer los tipos de cada tarea se seguirá los criterios indicados en el apartado 3.4.2.

Una vez aplicado el parseo aparecerá el editor de CTT con el diagrama correspondiente al modelo AUI especificado (Fig. 4-7).

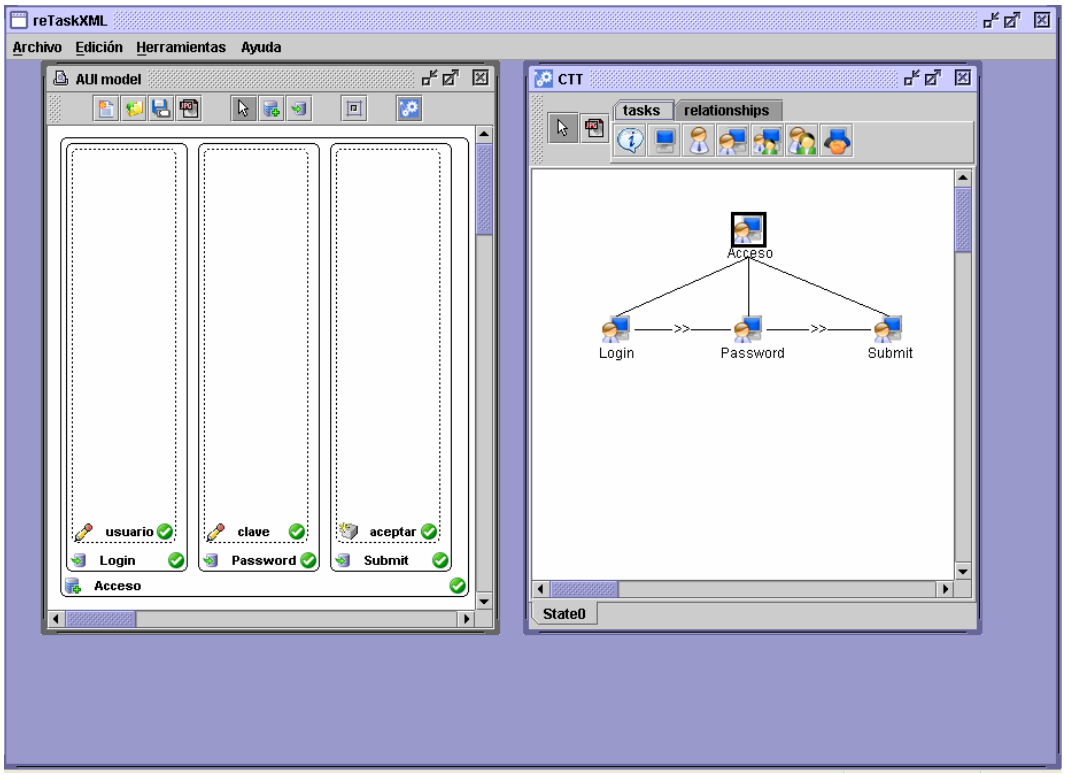

Figura 4-7. Modelo de tareas resultante de control de acceso.

Este diagrama está compuesto de una tarea Acceso que se corresponde con el contenedor, y que será de interacción al ser todas las tareas que cuelgan de ella de ese tipo. Los componentes Login y Password, al contar con una faceta de interacción queda reflejados en el modelo de tareas como dos tareas de interacción. La tarea *Submit* se obtiene del componente homónimo, su tipo depende de la faceta de control que contiene, que como se comentó anteriormente (Fig. 4-3) tiene la propiedad *actionType* igual a interacción. Todas estas tareas están enlazadas por relaciones temporales de habilitación, ya que según se van introduciendo datos se habilitan unas a las otras, y esa relación viene especificada en el contenedor.

4.2 Caso de estudio 2: Sistema de reserva de habitación de hotel

El segundo ejemplo consiste en una interfaz de usuario para la reserva de habitaciones de hotel(Fig. 4-8), como la que podríamos encontrar en cualquiera página dedicada a esta actividad. En ella el usuario introduciría todos los datos solicitados, como sus datos, datos del hotel y la reserva e información del pago, y por último el usuario confirmaría la reserva.

Figura 4-8. Interfaz de usuario concreta reserva de hotel.

Dicha interfaz se compone a nivel concreto, de un contenedor principal de tipo DesktopPane. Este contiene cinco contenedores de tipo Panel. El primero de ellos “Datos personales” cuenta con tres TextField, que deben ser completados con el nombre, apellidos y dirección de correo electrónico del cliente. Estos campos pueden rellenarse en cualquier orden, pero deben rellenarse todos para habilitar el siguiente Panel.

Un segundo contenedor de tipo Panel, para los datos del hotel, cuenta con tres componentes de tipo ComboBox, donde el usuario debe seleccionar Provincia, Localidad y hotel deseado, deberá realizarlo en esa secuencia para que se vayan habilitando los valores

correctos dependiendo de la selección en el ComboBox, una vez completada toda la secuencia el sistema mostrará en el TextArea etiquetado con “Precio/Noche”, el precio por noche de dicho hotel.

El siguiente contenedor es un Panel para las “Fechas de reserva”. Donde el usuario debe indicar la fecha de entrada y fecha de salida, para ello la interfaz cuenta con cuatro componentes de tipo TextField para días y años, para los meses son dos ComboBox. Al seleccionar correctamente las fechas se podrá pasar a la siguiente sección de la interfaz.

El cuarto contenedor “Datos habitación”. Cuenta con un TextField para indicar el número de habitaciones que se desea reservar, un List para seleccionar el tipo de habitaciones de esa reserva, y por último un CheckBox para indicar si los ocupantes de las habitaciones serán o no fumadores. El sistema según el usuario vaya completando correctamente los datos de cada componente, se irá habilitando el siguiente.

El último contenedor es de tipo Panel denominado “Datos de pago”, tiene en primer lugar un ComboBox donde se debe seleccionar el tipo de tarjeta de crédito. A continuación, un TextField donde indicar el número de tarjeta. En el siguiente TextField se indicará el nombre y apellidos del titular de la tarjeta. Dos ComboBox servirán para indicar el mes y año de caducidad de la tarjeta. Finalmente en un TextArea, se mostrará el precio total de la reserva.

Para confirmar la reserva, el cliente deberá pulsar el componente Button que aparece junto al total en la esquina inferior. Para habilitar este botón se han debido de completar correctamente todos los campos anteriores, ya que todos son obligatorios.

La interfaz abstracta correspondiente, se muestra en la figura siguiente(Fig. 4-9):

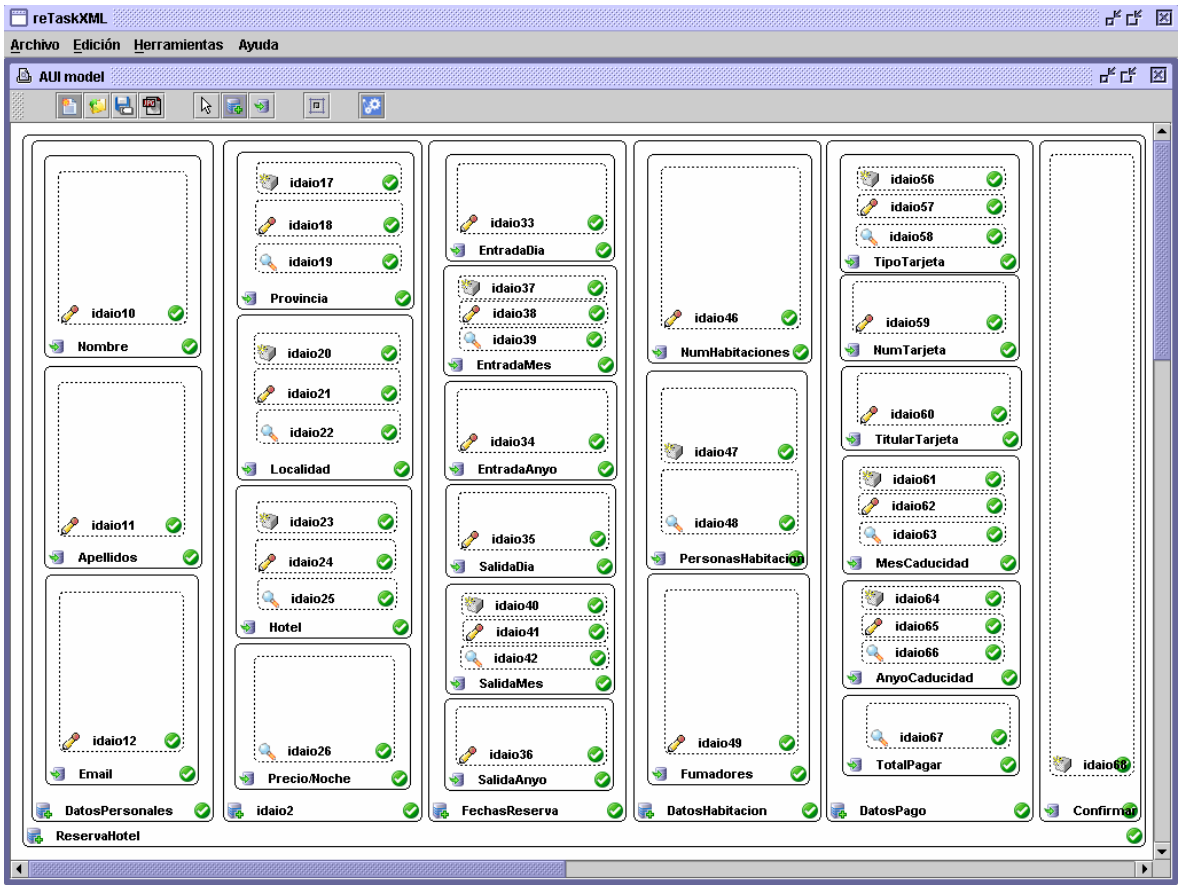

Figura 4-9. Interfaz de usuario abstrata de reserva de hotel.

Esta especificación cuenta con un container “ReservaHotel”, que se corresponde con el DesktopPane principal. Este container tiene configurado su atributo “temporalOperator” como enabling, ya que cada los AIO se van habilitando según el usuario va completando los campos.

Para el primer panel, se ha creado un container “DatosPersonales”, como es indiferente el orden en que se completen los campos, su operador temporal será concurrent. A su vez se han identificado tres componentes dentro de ese Panel que se corresponden con cada uno de los campos de texto que el usuario debe completar. Cada contenedor tiene una única faceta de tipo input.

Dentro del contenedor abstracto de nombre “DatosHotel” hay cuatro componentes abstractos de nombre “Provincia”, “Localidad”, “Hotel” y “Precio/Noche”, al habilitar cada uno al siguiente su atributo “temporalOperator” será de tipo enabling. El componente “Provincia” de tipo “ComboBox tiene tres facetas asociadas: una de control, otra de entrada y otra de salida. En este caso la faceta de control representa la posibilidad de elección de la provincia dentro del ComboBox, por lo que tendrá su propiedad “actionType” igual a interaction. La faceta de entrada indica la posibilidad de que el usuario introduzca algún dato, por lo que se trata de una faceta de interacción. La faceta de

salida indica que los ítems de dicho combo son mostrados por pantalla al usuario, su propiedad “actionType” será por tanto de application. El componente “Localidad” cuenta igualmente con tres facetas asociadas, al ser un ComboBox, al igual que ocurre con “Hotel”. El componente “Precio/Noche”, que a nivel concreto es un TextArea, tiene una faceta de salida y tiene su propiedad “actionType” igual a application, ya que con él, el sistema muestra el precio de estancia una noche en el hotel seleccionado.

El siguiente contenedor de nombre “FechasReserva”, dispone de cuatro componentes con facetas de tipo entrada, para introducir el día de entrada y de salida, así como el año de entrada y de salida. Para los meses al utilizarse ComboBox, se necesitan tres facetas de facetas, una de control, una de entrada y otra de salida, la faceta de control será de tipo interacción, mientras que la faceta de salida es de tipo aplicación. El orden en que se introduzcan los datos es indiferente, por lo que se dará concurrencia entre ellas, existiendo la obligatoriedad de que todos los campos se completen.

En el cuarto Panel denominado “DatosHabitación”, se han identificado tres componentes, al completar uno de ellos se habilita el siguiente, por lo que en el contenedor la propiedad “temporalOperator” se ha configurado como enabling. El componente “NumHabitaciones” tiene una faceta input, ya que el usuario debe introducir el número de habitaciones que desea reservar. El siguiente paso será indicar el número de personas que ocuparán las habitaciones anteriormente indicadas, para lo cual se ha utilizado un Componente de tipo List al que le corresponden una faceta de control cuya propiedad actionType será igual a interaction y otra de salida de tipo application. Para finalizar con este Panel, el usuario deberá indicar si las habitaciones serán para fumadores, esto se realiza con un componente, que al tratarse de un CheckBox cuenta con una única faceta de entrada.

El último Panel “DatosPago”, no existe ningún orden para realizar las operaciones, por lo que los componentes que contiene están asociados con relaciones de tipo concurrencia. El componente de tipo ComboBox de nombre “TipoTarjeta”, dispone de tres facetas control, entrada y salida. “NumTarjeta” es un componente con una faceta de entrada, donde el usuario debe introducir su número de tarjeta de crédito. El componente “TitularTarjeta” tiene una faceta de entrada para indicar el titular de la tarjeta de crédito. Existen dos componentes “MesCaducidad” y “AnyoCaducidad”, donde el usuario indicará el mes y año de caducidad de la tarjeta de crédito indicada, y cada una de ellas cuenta con tres facetas: una de control, una de entrada y otra de salida. Al completar todos los campos se mostrará en el TextArea “TotalPagar”, el precio de la reserva realizada, por lo que cuenta con una única faceta de salida.
El botón usado para confirmar , se habilitará al completar todos los Paneles y dispone de una faceta de control, ya que sirve para completar el proceso e informar al sistema de ello.

Al seleccionar “Pasar a CTT”, se generará la especificación de la interfaz abstracta(Fig. 4-10) en un fichero temporal con formato XML.

```
- <auimodel>
  - <abstractContainer id="idaio0" name="ReservaHotel" temporalOperator="enabling">
    - <abstractContainer id="idaio1" name="DatosPersonales" temporalOperator="concurrent">
      - <abstractIndividualComponent id="idaio7" name="Nombre">
          <input id="idaio10" name="idaio10"/>
        </abstractIndividualComponent>
      - <abstractIndividualComponent id="idaio8" name="Apellidos">
          <input id="idaio11" name="idaio11"/>
        </abstractIndividualComponent>
      - <abstractIndividualComponent id="idaio9" name="Email">
          <input id="idaio12" name="idaio12"/>
        </abstractIndividualComponent>
      </abstractContainer>
    - <abstractContainer id="idaio2" name="DatosHotel" temporalOperator="enabling">
      - <abstractIndividualComponent id="idaio13" name="Provincia">
          <navigation id="idaio17" name="idaio17" actionType="interaction"/>
          <input id="idaio18" name="idaio18"/>
          <output id="idaio19" name="idaio19" actionType="application"/>
        </abstractIndividualComponent>
      - <abstractIndividualComponent id="idaio14" name="Localidad">
          <navigation id="idaio20" name="idaio20" actionType="interaction"/>
          <input id="idaio21" name="idaio21"/>
          <output id="idaio22" name="idaio22" actionType="application"/>
        </abstractIndividualComponent>
      - <abstractIndividualComponent id="idaio15" name="Hotel">
          <navigation id="idaio23" name="idaio23" actionType="interaction"/>
          <input id="idaio24" name="idaio24"/>
          <output id="idaio25" name="idaio25" actionType="application"/>
        </abstractIndividualComponent>
      - <abstractIndividualComponent id="idaio16" name="Precio/Noche">
          <output id="idaio26" name="idaio26" actionType="application"/>
        </abstractIndividualComponent>
      </abstractContainer>
    - <abstractContainer id="idaio3" name="FechasReserva" temporalOperator="concurrent">
      - <abstractIndividualComponent id="idaio27" name="EntradaDia">
          <input id="idaio33" name="idaio33"/>
        </abstractIndividualComponent>
      - <abstractIndividualComponent id="idaio28" name="EntradaMes">
          <navigation id="idaio37" name="idaio37" actionType="interaction"/>
          <input id="idaio38" name="idaio38"/>
          <output id="idaio39" name="idaio39" actionType="application"/>
        </abstractIndividualComponent>
      - <abstractIndividualComponent id="idaio29" name="EntradaAnyo">
          <input id="idaio34" name="idaio34"/>
        </abstractIndividualComponent>
      - <abstractIndividualComponent id="idaio30" name="SalidaDia">
          <input id="idaio35" name="idaio35"/>
        </abstractIndividualComponent>
      - <abstractIndividualComponent id="idaio31" name="SalidaMes">
          <navigation id="idaio40" name="idaio40" actionType="interaction"/>
          <input id="idaio41" name="idaio41"/>
          <output id="idaio42" name="idaio42" actionType="application"/>
        </abstractIndividualComponent>
```

```
    - <abstractIndividualComponent id="idaio32" name="SalidaAnyo">
        <input id="idaio36" name="idaio36"/>
      </abstractIndividualComponent>
    </abstractContainer>
  - <abstractContainer id="idaio4" name="DatosHabitacion" temporalOperator="enabling">
    - <abstractIndividualComponent id="idaio43" name="NumHabitaciones">
        <input id="idaio46" name="idaio46"/>
      </abstractIndividualComponent>
    - <abstractIndividualComponent id="idaio44" name="PersonasHabitacion">
        <navigation id="idaio47" name="idaio47" actionType="interaction"/>
        <output id="idaio48" name="idaio48" actionType="application"/>
      </abstractIndividualComponent>
    - <abstractIndividualComponent id="idaio45" name="Fumadores">
        <input id="idaio49" name="idaio49"/>
      </abstractIndividualComponent>
    </abstractContainer>
  - <abstractContainer id="idaio5" name="DatosPago" temporalOperator="concurrent">
    - <abstractIndividualComponent id="idaio50" name="TipoTarjeta">
        <navigation id="idaio56" name="idaio56" actionType="interaction"/>
        <input id="idaio57" name="idaio57"/>
        <output id="idaio58" name="idaio58" actionType="application"/>
      </abstractIndividualComponent>
    - <abstractIndividualComponent id="idaio51" name="NumTarjeta">
        <input id="idaio59" name="idaio59"/>
      </abstractIndividualComponent>
    - <abstractIndividualComponent id="idaio52" name="TitularTarjeta">
        <input id="idaio60" name="idaio60"/>
      </abstractIndividualComponent>
    - <abstractIndividualComponent id="idaio53" name="MesCaducidad">
        <navigation id="idaio61" name="idaio61" actionType="interaction"/>
        <input id="idaio62" name="idaio62"/>
        <output id="idaio63" name="idaio63" actionType="application"/>
      </abstractIndividualComponent>
    - <abstractIndividualComponent id="idaio54" name="AnyoCaducidad">
        <navigation id="idaio64" name="idaio64" actionType="interaction"/>
        <input id="idaio65" name="idaio65"/>
        <output id="idaio66" name="idaio66" actionType="application"/>
      </abstractIndividualComponent>
    - <abstractIndividualComponent id="idaio55" name="TotalPagar">
        <output id="idaio67" name="idaio67" actionType="application"/>
      </abstractIndividualComponent>
    </abstractContainer>
    - <abstractIndividualComponent id="idaio6" name="Confirmar">
        <navigation id="idaio68" name="idaio68" actionType="interaction"/>
      </abstractIndividualComponent>
    </abstractContainer>
</auimodel>
```

Figura 4-10. Especificación fichero temporal AUI.

Al aplicar la transformación XSLT a esta especificación AUI, se genera un fichero temporal con la especificación del modelo de tareas(Fig. 4-11).

```
- <taskmodel>
  - <task id="idaio0" name="ReservaHotel">
    - <task id="idaio1" name="DatosPersonales">
        <task id="idaio7" name="Nombre"/>
        <task id="idaio8" name="Apellidos"/>
        <task id="idaio9" name="Email"/>
      </task>
    - <task id="idaio2" name="DatosHotel">
        <task id="idaio13" name="Provincia"/>
        <task id="idaio14" name="Localidad"/>
        <task id="idaio15" name="Hotel"/>
        <task id="idaio16" name="Precio/Noche"/>
      </task>
    - <task id="idaio3" name="FechasReserva">
        <task id="idaio27" name="EntradaDia"/>
        <task id="idaio28" name="EntradaMes"/>
        <task id="idaio29" name="EntradaAnyo"/>
        <task id="idaio30" name="SalidaDia"/>
        <task id="idaio31" name="SalidaMes"/>
        <task id="idaio32" name="SalidaAnyo"/>
      </task>
    - <task id="idaio4" name="DatosHabitacion">
        <task id="idaio43" name="NumHabitaciones"/>
        <task id="idaio44" name="PersonasHabitacion"/>
        <task id="idaio45" name="Fumadores"/>
      </task>
    - <task id="idaio5" name="DatosPago">
        <task id="idaio50" name="TipoTarjeta"/>
        <task id="idaio51" name="NumTarjeta"/>
        <task id="idaio52" name="TitularTarjeta"/>
        <task id="idaio53" name="MesCaducidad"/>
        <task id="idaio54" name="AnyoCaducidad"/>
        <task id="idaio55" name="TotalPagar"/>
      </task>
      <task id="idaio6" name="Confirmar"/>
    </task>
```

```
- <enabling>
    <source sourceId="idaio1"/>
    <target targetId="idaio2"/>
    <source sourceId="idaio2"/>
    <target targetId="idaio3"/>
    <source sourceId="idaio3"/>
    <target targetId="idaio4"/>
    <source sourceId="idaio4"/>
    <target targetId="idaio5"/>
    <source sourceId="idaio5"/>
    <target targetId="idaio6"/>
  </enabling>
- <concurrent>
    <source sourceId="idaio7"/>
    <target targetId="idaio8"/>
    <source sourceId="idaio8"/>
    <target targetId="idaio9"/>
  </concurrent>
- <enabling>
    <source sourceId="idaio13"/>
    <target targetId="idaio14"/>
    <source sourceId="idaio14"/>
    <target targetId="idaio15"/>
    <source sourceId="idaio15"/>
    <target targetId="idaio16"/>
  </enabling>
- <concurrent>
    <source sourceId="idaio27"/>
    <target targetId="idaio28"/>
    <source sourceId="idaio28"/>
    <target targetId="idaio29"/>
    <source sourceId="idaio29"/>
    <target targetId="idaio30"/>
    <source sourceId="idaio30"/>
    <target targetId="idaio31"/>
    <source sourceId="idaio31"/>
    <target targetId="idaio32"/>
  </concurrent>
- <enabling>
    <source sourceId="idaio43"/>
    <target targetId="idaio44"/>
    <source sourceId="idaio44"/>
    <target targetId="idaio45"/>
  </enabling>
- <concurrent>
    <source sourceId="idaio50"/>
    <target targetId="idaio51"/>
    <source sourceId="idaio51"/>
    <target targetId="idaio52"/>
    <source sourceId="idaio52"/>
    <target targetId="idaio53"/>
    <source sourceId="idaio53"/>
    <target targetId="idaio54"/>
    <source sourceId="idaio54"/>
    <target targetId="idaio55"/>
  </concurrent>
</taskmodel>
```

Figura 4-11. Especificación fichero temporal CTT.

Esta especificación generada contiene todas las tareas y relaciones del modelo de tareas equivalente, únicamente falta por indicar el tipo de las tareas, que como se indicó anteriormente, se añade posteriormente mediante código java a la estructura generada.

Una vez añadido el tipo a las tareas, este diagrama se abrirá abierto mediante el editor de CTT (Fig. 4-12).

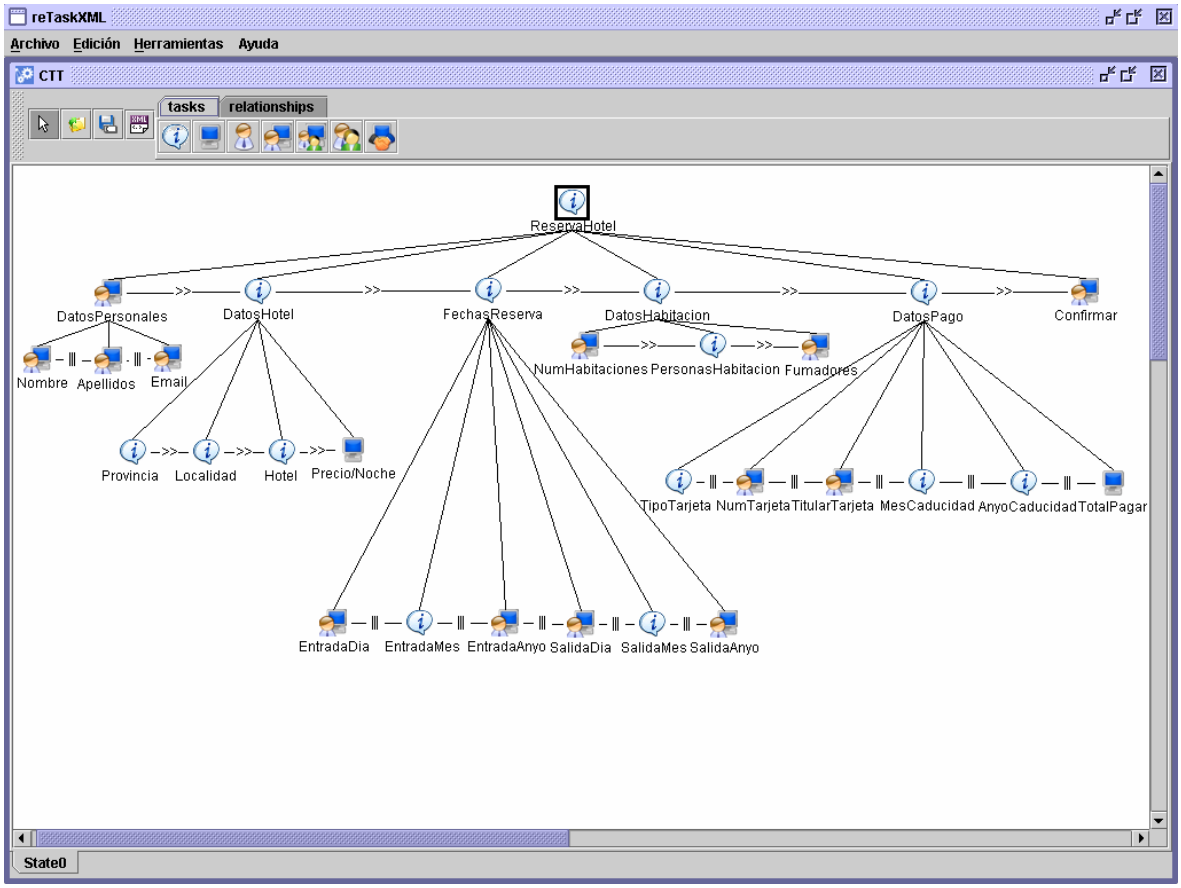

Figura 4-12. Modelo de tareas resultante de reserva de hotel.

Este diagrama cuenta con una tarea abstracta, que se corresponde con el contenedor principal “ReservaHotel”. Esta tarea cuenta con seis subtareas, que están asociadas mediante relaciones de habilitación. Esta tarea es de tipo abstracto, ya que sus tareas hijas son de tipo abstracto, excepto “DatosPersonales” y “Confirmar” que son tareas de interacción.

La tarea “DatosPersonales” tiene tres tareas hijas de tipo interacción ya que cuentas sólo con facetas de entrada que son siempre de interacción, relacionadas mediante una relación de concurrencia, por tanto el tipo de la tarea “DatosPersonales” será interacción.

“DatosHotel” tiene cuatro tareas hijas. “Provincia”, “Localidad” y “Hotel” son de tipo abstracto, ya que disponen de tres facetas: las facetas de entrada son siempre de tipo interacción, las facetas de control en este caso son de tipo interacción y las facetas de salida son para estos componentes de aplicación. La tarea “Precio/Noche” es de aplicación, ya que proviene de un componente con una faceta de salida con su propiedad “actionType” igual a application. Estas tareas se habilitan unas a las otras por lo que están asociadas mediante relaciones de habilitación.

En cuanto a “FechasReserva”, se trata de una tarea abstracta, debido a que sus tareas hijas son de tipo abstacto y de interacción, estas tareas están asociadas mediante relaciones de concurrencia. Como en el caso anterior, las tareas provenientes de un ComboBox serán de tipo abstracto al tener facetas de aplicación e interacción, mientras que las provenientes de un TextField son de tipo interacción al contar con facetas de entrada.

Para la tarea “DatosHabitacion” sus tareas hijas están asociadas mediante relaciones de habilitación y se trata de una tarea de tipo abstracto, ya que tiene dos tareas hijas de interacción y una de tipo abstracto. La tarea “NumHabitaciones” es de tipo interacción, al corresponderse con un componente con una única faceta de entrada. “PersonasHabitacion” es una tarea de tipo abstracto, al provenir de un componente con dos facetas, una de control de tipo interacción y otra de salida de tipo aplicación. Por último, la tarea “Fumadores” al provenir de un componente con una faceta de entrada, es una tarea de interacción.

La tarea “DatosPago”, es una tarea de tipo abstacto. Cuenta con tres tareas hijas de tipo abstracto, “TipoTarjeta”, “MesCaducidad” y “AnyoCaducidad”, las tres provienen de ComboBox. Dos de sus hijas, “NumTarjeta” y “TitularTarjeta”, son de tipo interacción al corresponderse con componentes con sólo tareas de entrada. La última tarea “TotalPagar” es de tipo aplicación, al provenir de un componente con una faceta de salida de tipo aplicación.

Finalmente, la última tarea “Confirmar”, es una tarea de interacción al provenir de un botón.

Capítulo 5. CONCLUSIONES Y TRABAJOS FUTUROS

Alcanzada esta sección de la memoria, llega el momento de presentar las conclusiones obtenidas fruto del trabajo realizado en la elaboración de este proyecto. También tendrá cabida en este capítulo la relación de este proyecto con otros ya desarrollados o que se encuentran en desarrollo. Para finalizar este capítulo se sugerirán una serie de trabajos futuros identificados durante la realización de este trabajo y que proponen nuevas direcciones de trabajo e investigación.

5.1 Conclusiones

Este proyecto se ha centrado en una serie de conceptos fundamentales que se han ido desarrollando en esta memoria: desarrollo basado en modelos, centrado en la propuesta UsiXML, prestando especial atención a IdealXML; generación de modelos de tareas (CTT) a partir de especificaciones de interfaces de usuario abstractas; familiarización con leguajes de especificación de transformaciones, especialmente XSLT para documentos XML.

Como ya se ha comentado en este documento, se ha seguido un desarrollo basado en modelos buscando una tendencia a la estandarización, y un lenguaje de representación común de los datos interactivos. Esta búsqueda, aporta una serie de beneficios, como que partiendo de la misma interfaz de usuario, ésta se pueda utilizar independientemente del contexto de uso y de las restricciones impuestas por un dispositivo o plataforma específicos. Otra aplicación de MDA, sería por ejemplo en Re-ingeniería de sistemas para hacer más fácil el re-estructurar o re-escribir parte o todo un sistema legado sin cambios en su funcionalidad, muy útil si el código fuente del sistema se ha extraviado.

El paso de interfaces abstractas a AUI, se realiza en sentido inverso a como lo hace UsiXML, de modelos más próximos a la interfaz a modelos más próximos a la especificación original de una interfaz, es decir teniendo una interfaz Concreta, poder realizar una abstracción de la misma y a partir de esta obtener el modelo de tareas correspondiente, esto nos permitirá obtener un modelo genérico más adaptable a otros dispositivos y plataformas.

La unión de XML y XSLT permite separar contenido y presentación por lo que se ve aumentada la productividad. Esta transformación se planteo como una manera optima realizar la transformación de documentos temporales en formato XML. Los problemas planteados por falta de compatibilidad completa de esta transformación XSLT con Java, hizo que la transformación tuviera que replantearse de modo que se optó por una solución combinada realizando la parte que requería una componente de programación en código Java.

5.2 Relación con otros proyectos

Como se comentó en apartados anteriores, existen varios proyectos ya concluidos que siguen el desarrollo de interfaces de usuario basada en modelos propuesta por UsiXML como son: TransformiXML KnowiXML, GrafiXML, VisiXML, SketchiXML, FormiXML, VisualiXML, ReversiXML e IdealXML. Destacar IdealXML que realiza el camino inverso al realizado en este trabajo.

Proyectos fin de carrera de otros compañeros relacionados con este son, y que ya han sido presentados el curso pasado o se presentarán en esta o futuras convocatorias son:

Título	Autor
"Abstracción de interfaces de usuario a partir de especificaciones concretas"	Francisco Javier Muñoz Márquez
"Diseño e implementación de un herramienta que permita facilitar un entorno colaborativo sobre el que especificar modelos de tareas."	Blanca Azahara Arribas Simarro
"Especificación gráfica de interfaces de usuario utilizando usixml".	Arturo Garcia Nuño
"Visualización en Java de Interfaces de Usuario Basadas en Modelos".	Miguel de los Reyes Bañon López
"Transformación de especificaciones concretas a finales de interfaces de usuario".	Juan Carlos Peña Rodríguez
"Simulación y animación gráfica de modelos de tareas usando la notación CTT".	Jesús Romero Hebrero

Tabla 5-1. Proyectos final de carrera relacionados.

Muy relacionado con este proyecto se encuentra, "Abstracción de interfaces de usuario a partir de especificaciones concretas", ya que este proyecto comienza en interfaces de usuario concretas y termina en interfaces de usuario abstractas, que es el punto donde se inicia este proyecto para llegar a modelo de tareas.

5.3 Trabajos Futuros

Los resultados desarrollados en este proyecto dejan un el camino abierto a nuevas líneas de trabajo que complementen el trabajo realizado. Los trabajos que se proponen para su posterior realización son los que a continuación se recogen.

Realizar una transformación automática de interfaces de usuario abstractas a interfaces de usuario concretas, y que automáticamente se adapte a la resolución típica de los dispositivos de interacción más utilizados actualmente como: ordenadores fijos y portátiles, teléfonos móviles de última generación, PDAs, navegadores GPS, teléfonos fijos, faxes, tabletPCs, etc.

Creación de un entorno integrado que permita la creación de interfaces de usuario desde una herramienta centralizada, es decir, debido a la gran cantidad de editores utilizados en UsiXML, el objetivo sería la integración de estos en una única herramienta.

El lenguaje UsiXML ha despertado un gran interés en la comunidad de interacción persona-ordenador. Sin embargo, todavía queda camino por andar en su aceptación y difusión definitiva, por lo que es necesario seguir aportando ideas relacionadas con cómo ayuda en el diseño e implementación de las herramientas que hagan de dicho lenguaje un lenguaje de amplia utilización en entornos industriales.

BIBLIOGRAFÍA

[Arnold, 2005] Arnold, C., Lucas D. y otros. Dom4j. MetaStuff Ltd. http://dom4j.org/index.html

[Bravo, 2004] Bravo, J. y otros (eCLUB). "Evolución de un Entorno de Enseñanza Basado en Escritorio hacia la Computación Ubicua. Aplicación a la Enseñanza de Materias Experimentales". Escuela Superior de Informática de Ciudad Real. Universidad de Castilla-la Mancha. Ciudad Real, Mayo 2004. chico.inf-cr.uclm.es/eCLUB/InformeTecnico.pdf

[Gea, 2002] Gea, M., Gutiérrez, F.L. Introducción a la Interacción Persona-Ordenador Capítulo 5, El diseño. Universidad de Granada, 2002. http://griho.udl.es/ipo

[González, 2005] González, O. XML Edición revisada y ampliada 2005. Anaya multimedia 2005.

[Griffiths, 2000] Griffiths, T., Barclay, F.J., Pinheiro, P. y otros. Teallach: a model-based user interface development environment for object databases. University of Manchester, 2000.

[Kay,2004] Kay, M. XSLT 2.0 Programmer's Reference, Third Edition. Wrox Press. 2004.

[Limbourg,2004] Limbourg, Q. et al. UsiXML v1.8.0 Documentation. Université catholique de Louvain, Bélgica 2004-2007. http://www.usixml.org/v2

[López, 2005] López Jaquero, V. M. Tesis doctoral: Interfaces de usuario adaptativas basadas en modelos y agentes software. Universidad de Castilla-la Mancha. Departamento de sistemas informáticos. Albacete Julio de 2005.

[Mangano,2002] Mangano, S. XSLT Cookbook. O'Reilly. USA, 2002.

[Millar,2003] Millar, J., Mukerji, J. MDA Guide Version 1.0.1. OMG 2003. http://www.omg.org/docs/omg/03-06-01.pdf

[Montero, 2005] Montero Simarro, F. Tesis doctoral: Integración de calidad y experiencia en el desarrollo de interfaces de usuario dirigido por modelos. Universidad de Castilla-la Mancha. Departamento de sistemas informáticos. Albacete Julio de 2005.

[Montero,2005] Montero Simarro, F. IdealXML. Escuela Politécnica Superior de Albacete. España.2005-2007. http://www.dsi.uclm.es/personal/FranciscoMonteroSimarro/IdealXML.htm

[Mori, 2004] Mori, G., Paternò, F., et al. Teresa.2004-2007 http://giove.isti.cnr.it/teresa.html

[Paternò,1999] Paternò, F. Model-based design and evaluation of interactive applications. Springer, 1999.

[Puerta, 2004] Puerta, A. R., Macias, J. A., Catells, P. MBUI para procesos de ingeniería

inversa.V Congreso interacción persona ordenador, Lleida 2004.

[Ramos et al. ,2002] Ramos, I., Lozano, M. D., González, P., Montero, F., Molina, J. P. Desarrollo y generación de interfaces de usuario a partir de técnicas de análisis de tareas y casos de uso. Escuela Politécnica Superior, Universidad de Castilla-La Mancha y Universidad Politécnica de Valencia, 2002.

[Sommerville, 2000] Sommerville, I. Legacy Systems. School of Computer Science en St Andrews University, 2000. http://www.comp.lancs.ac.uk/computing/resources/IanS/SE7/ElectronicSupplements/LegacySys.pdf

[UML,1999] Booch, G., Rumbaugh, J., Jacobson, I. El Lenguaje Unificado de Modelado. Addison-Wesley. 1999.

[Vanderdonckt, 2005] Vanderdonckt, J. A MDA-Compliant Environment for Developing User Interfaces of Information Systems, Proc. of 17th Conf. on Advanced Information Systems Engineering CAiSE'05 (Porto, 13-17 June 2005). Université catholique de Louvain, 2005.