# UNIVERSIDAD DE CASTILLA-LA MANCHA
# ESCUELA POLITÉCNICA SUPERIOR

## INGENIERÍA
## EN INFORMÁTICA

## PROYECTO FIN DE CARRERA

"EgiuXML: Editor gráfico de Interfaces de Usuario a nivel concreto utilizando UsiXML"

Arturo García Nuño

**Diciembre, 2007**

# UNIVERSIDAD DE CASTILLA-LA MANCHA
# ESCUELA POLITÉCNICA SUPERIOR

DEPARTAMENTO DE SISTEMAS INFORMÁTICOS

# PROYECTO FIN DE CARRERA

"EgiuXML: Editor gráfico de Interfaces de Usuario a nivel concreto utilizando UsiXML"

Autor: Arturo García Nuño

Director: Francisco Montero Simarro

**Diciembre, 2007**

## Resumen

En los últimos años se ha realizado un gran esfuerzo en la investigación de métodos que permitan la inclusión del diseño de la interfaz de usuario dentro de un proceso de desarrollo basado en modelos. Intentando obtener beneficios tales como la automatización de la generación de la interfaz de usuario, la generación de dicha interfaz para distintos dispositivos o lenguajes a partir de unos modelos comunes o la mejora de las propiedades de usabilidad del sistema.

Los entornos de desarrollo de interfaces de usuario basado en modelos (Mb-UID) (*Model-based User Interfaces Development*) proponen una serie de modelos de distinta temática y diferente nivel de abstracción. El diseñador utiliza notaciones de mayor nivel de abstracción para especificar estos modelos o descripciones declarativas de la interfaz. El problema que se encuentra es que no existe una notación estándar para la descripción de los diferentes modelos.

Por este motivo para realizar la abstracción de la interfaz de usuario se propuso el desarrollo basado en modelos impuesto por UsiXML, buscando una tendencia a la estandarización, y un lenguaje común de los datos interactivos. UsiXML propone cuatro niveles de abstracción como marco de desarrollo de interfaces de usuario: Tareas & Conceptos, UI abstracta (AUI), UI Concreta (CUI) y UI Final (FUI). En este trabajo se profundizará en el modelo de interfaz de usuario CUI, en el cuál permite la especificación de la apariencia y el comportamiento de una interfaz de usuario con elementos que pueden ser percibidos por los usuarios, consiguiendo una representación de las interfaces de usuario independientemente de cuál sea la plataforma y el dispositivo en el que se visualice dicha interfaz de usuario.

En este contexto, el proyecto consiste en el análisis y diseño de una aplicación software que permita **especificar interfaces de usuario de forma gráfica a nivel concreto**, utilizando objetos de interacción concreta, y **gestionar dichas especificaciones utilizando el lenguaje UsiXML,** almacenando las especificaciones en un fichero que sigue el estándar impuesto**.**

Esta herramienta contribuirá en el paradigma de diseño basado en modelos intentando atajar dificultades que afloran durante el diseño de una interfaz de usuario y a su vez aportará su esfuerzo en la indagación de beneficios como la semiautomatización de la creación de la interfaz de usuario, y reutilización de experiencia gracias a la recuperación y almacenamiento de las especificaciones a nivel concreto en ficheros.

## Agradecimientos

Quiero dejar constancia de la gratitud que me merecen todos aquellos que han hecho posible la elaboración de este proyecto y la culminación de una carrera que a veces se puso cuesta arriba.

Agradezco a mi familia, que durante estos años de carrera, tuvieron plena confianza en mí, y siguen siendo el apoyo para que nunca me rinda en las metas que quedan por delante en esta vida.

También agradezco a los frailes franciscanos y a toda la familia franciscana que me dieron ejemplo de cómo orar con Fe, y tener confianza a que fuesen escuchadas mis plegarias. A todos ellos, gracias por hacer la vida más feliz en estos últimos años de carrera.

Por último agradezco a mi tutor Francisco Montero Simarro, por ser un buen tutor y amigo, siempre disponible para todas las consultas y por ser siempre tan optimista con el trabajo realizado.

A todos ellos y a Dios Padre, muchas gracias.

*Se lo dedico a mi madre, a mi padre,*
*a mis hermanos,*
*y a la familia franciscana,*
*que me han dado todo lo que soy.*

# Índice General

# Índice de figuras

# Índice de tablas

# Capítulo 1: Introducción

Este proyecto se encuadra en el hueco o gap existente entre dos disciplinas que focalizan interés y gran volumen de investigación y publicaciones en estos momentos. Dichas disciplinas son la Ingeniería del Software (IS) y la Interacción Persona-Ordenador (IPO). Ambas comparten, cuando menos, interés y contribuciones en lo que a desarrollo de software se refiere y tienen un nexo en común la *interfaz de usuario* que es especialmente importante, ya que es la parte de cualquier sistema con la que los usuarios u otros sistemas tienen que interactuar e intercambiar órdenes o información

## 1.1 CONTEXTO

El desarrollo de software involucra diversos aspectos, tales como la definición de actividades, utilización de metodologías, gestión de requerimientos, pruebas, etc. Siempre en busca de un proceso más rentable y la generación de un software de calidad. La experiencia confirma que, el desarrollo de software es un proceso difícil. Entregar un producto de calidad y que cumpla con las expectativas del cliente, es aún más complicado.

Hay que tener en cuenta cuáles son las necesidades y cualidades de los usuarios actualmente, ya que éste no sólo se conforma con que el software cumpla con los requisitos funcionales sino la importancia que le hace recaer en la interfaz de usuario. Dicha interfaz debe ofrecer unas características subjetivas y de satisfacción que el usuario demanda de manera implícita en muchas ocasiones. Para corroborar la importancia de la interfaz de usuario sólo hay que ver la consideración que le confieren las empresas a la apariencia de sus plataformas Web publicitarias para atraer más clientes, lo que luego se traduce en ingresos económicos.

Por lo tanto, las aplicaciones interactivas como el uso de éstas en distintos tipos de dispositivos (ordenadores personales, teléfonos móviles, PDAs), han hecho que el diseño de la interfaz de usuario sea un ingrediente esencial en la calidad final de la aplicación. La filosofía *Model-based User Interfaces Development* (Mb-UID) estudia el desarrollo de interfaces de usuario basado en modelos. Propone una serie de modelos de distinta temática y diferente nivel de abstracción, que incluyen, por ejemplo, modelos de dominio, de presentación, de tareas, de diálogo, de usuario y de la plataforma. El diseñador utiliza notaciones de mayor nivel de abstracción para especificar estos modelos o descripciones declarativas de la interfaz.

En el ámbito de la IPO, no existe una notación estándar para la descripción de los diferentes modelos dentro de los entornos de desarrollo de **Interfaces de Usuario** basados en modelos (Mb-UIDE) desarrollados. Tradicionalmente, cada Mb-UIDE ha utilizado una notación diferente, ya sea gráfica o textual, e incluso un mismo Mb-UIDE utiliza diferentes notaciones o lenguajes de modelado para los diferentes modelos declarativos de los que se compone el modelo de interfaz. Debido a ello hay una tendencia que busca cierta estandarización en este ámbito.

En este sentido se ha propuesto un lenguaje de especificación de interfaces de usuario denominado UsiXML (Limbourg et al., 2004) que utiliza y se aprovecha de las ventajas que ofrece XML y con él documenta los diferentes modelos asociados a la propuesta UsiXML. La tecnología XML, como estándar de representación común, permite la especificación del modelo de interfaz abstracto, la descripción de las características específicas de los diferentes dispositivos, así como la especificación del proceso de transformación de los objetos de interacción abstractos en objetos de interacción concretos.

El objetivo con la propuesta de UsiXML es facilitar un mecanismo que permita la representación de las interfaces de usuario sea la misma, independientemente de cuál sea la plataforma y el dispositivo en el que se visualice dicha interfaz de usuario. En este sentido, UsiXML no es la única propuesta en esta dirección, hay otros lenguajes basados en XML para la especificación de interfaces como, por ejemplo, XIML (Puerta, 2001), UIML (Abrams et al., 1999) o XUL (Mozilla, 2003) los comentaremos y describiremos en profundidad en un capítulo posterior.

## 1.2 MOTIVACIÓN

En este proyecto final de carrera lo que se pretende es **contribuir a la especificación declarativa de los modelos asociados con una interfaz de usuario**. En este proyecto nos centraremos en la especificación de interfaces de usuario a nivel concreto tal y como la definida en UsiXML.

De forma algo más precisa, el proyecto cristalizará en el desarrollo de una herramienta software capaz de **especificar interfaces de usuario de forma gráfica y permitir gestionar dichas especificaciones utilizando el lenguaje UsiXML**.

Los resultados derivados de la realización de este proyecto final de carrera, contribuyen al proceso de Ingeniería de Software para la elaboración de software de

calidad que cumpla con las expectativas del cliente y donde se considere el desarrollo de la interfaz de usuario de una forma sistemática.

## 1.3 OBJETIVOS DEL PROYECTO

Los principales objetivos asociados a este proyecto son los siguientes:

- Familiarizarnos con el lenguaje de marcado extensible de interfaces de usuario (UsiXML), que nos dará las pautas que deben seguir los ficheros generados por la aplicación resultante de la elaboración del proyecto.

- Estudiar las herramientas disponibles en el ámbito del desarrollo basado en modelos. Especial consideración tendrán las herramientas afines a la principal motivación de este proyecto.

- Profundizar y poner en práctica conocimientos relacionados con la programación Orientada a Objetos, para ello habrá que determinar qué lenguaje y entorno utilizar para abordar el desarrollo de la herramienta asociada a este proyecto.

- Desarrollar un producto software que permita la generación de una interfaz de usuario a nivel concreto y guardarlo en formato UsiXML o una variante de este lenguaje.

- Conseguir desarrollar dicho producto software de calidad, es decir, que cumpla los requisitos establecidos (tanto funcionales como no funcionales) y que cumpla los estándares asociados.

- Desarrollar un producto software usable, es decir, que los usuarios aprendan pronto a manejarlo, e intuyan el funcionamiento de cada elemento de la interfaz de usuario.

- Obtener un producto software fácil de mantener y bien documentado, para que posteriormente pueda ser modificado sin dificultad.

- Obtener un producto software que incorpore novedades importantes respecto a programas con los que pueda lograrse una funcionalidad parecida.

Determinando funcionalidad adicional y características deseables respecto a las que dichos entornos disponibles ofrezcan.

- Comprender la importancia de un modelo de Interfaz de Usuario Concreta (CUI) para modelar o abstraer una Interfaz de Usuario Gráfica (GUI**)** generada por un diseñador de IU.

- Comprender el lugar que ocupa la aplicación desarrollada por el proyecto dentro la estructura de las herramientas de UsiXML según la clasificación de MDA (Modelo Dirigido por Arquitectura)

## 1.4 ESTRUCTURA DEL DOCUMENTO

En este primer capítulo se puede encontrar una introducción a la materia del desarrollo de interfaces de usuario basado en modelos así como la motivación que ha provocado el desarrollo del trabajo y los objetivos del trabajo dadas las motivaciones expuestas.

En el segundo capítulo se realiza una descripción del estado actual de los trabajos relacionados con este proyecto como: el diseño de interfaces de usuario basado en modelos, propuestas metodológicas para el desarrollo de interfaces, lenguajes de descripción de interfaces de usuario basados en XML, herramientas de modelado que sirven a los desarrolladores para construir modelos o descripciones declarativas de la interfaz, entre ellas GrafiXML, IdealXML, CTTE.

En el tercer capítulo se muestra el proceso de Ingeniería del software seguido a lo largo del desarrollo del proyecto, para conseguir un software de calidad. Las distintas fases consideradas en este apartado son: Identificación de objetivos, de requisitos funcionales y no funcionales, análisis de requisitos, diseño e implementación. Además se muestra una descripción extensa de la herramienta. Y se hace mención de algún detalle de implementación que se considera importante.

En el cuarto capítulo para corroborar la explicación del capítulo anterior sobre el funcionamiento del producto software desarrollado, se realizarán dos casos de estudios con los que demostrar la conversión de una interfaz de usuario gráfica generada por la misma aplicación a un modelo de nivel concreto en formato UsiXML.

En el quinto capítulo se recogerán las conclusiones obtenidas tras la realización del proyecto, así como otros trabajos futuros que pueden partir de este proyecto.

## 1.5 COMENTARIOS PREVIOS

Antes de empezar con el desarrollo de los capítulos sugeridos, es conveniente la definición aunque sea ligera, de una serie de conceptos, que por su relación con el tema que se está tratando aparecerán con de forma reiterada a lo largo de todo esta memoria, de hecho algunos términos ya han aparecido en este primer capítulo. Muchos de esos términos serán tratados con mayor detalle en los capítulos específicos donde su aparición sea estelar, y sirva, por tanto, lo siguiente como una mera presentación propia de un capítulo con pretensiones meramente introductorias y descriptivas.

UsiXML 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. Se puede definir UsiXML como una ontología para descripción de interfaces de usuario. Varios son los modelos incluidos en UsiXML, se habla un poco en el proyecto de cada uno de ellos, pero se hace más énfasis en el Modelo de interfaz concreta (CUI). UsiXML tiene un sitio Web oficial: http://www.usixml.org, donde se puede obtener información sobre este lenguaje.

Hay herramientas gráficas como GrafiXML en el cual se puede definir un modelo CUI. GrafiXML dibuja interfaces de usuario para múltiples plataformas de computación. Se puede guardar una interfaz de usuario en varios formatos como Java o XHTML, pero la principal forma de guardarlas es en UsiXML (un lenguaje de descripción de interfaces de usuario en XML) y exportarlo a formato de código. GrafiXML es similar a alguna interfaz de usuario de algún editor de construcción, excepto que manipula más propiedades de objetos que otros editores y guarda la interfaz de usuario en UsiXML en lugar de en otro particular formato de código. De esta manera, es posible mantener múltiples versiones localizadas de la misma interfaz de usuario y atribuirlas a contextos particulares de uso. (UsiXML, 2007).

La herramienta desarrollada en el proyecto ha tomado como referencia la herramienta GrafiXML aunque las pretensiones de cada una de ellas son diferentes.

# Capítulo 2: Estudio del estado del arte

En el presente capítulo se realiza una descripción del estado actual de los trabajos, principalmente en el ámbito académico, relacionados con este proyecto. En dicho capitulo se destaca la importancia de las interfaces de usuario en una aplicación software, el gran esfuerzo en la investigación de métodos que permitan la inclusión del diseño de la interfaz de usuario dentro de un proceso de desarrollo basado en modelos, describiendo algunas de estas propuestas metodológicas en detalle junto con algunas notaciones de mayor nivel de abstracción para especificar estos modelos, se hace mención con más o menos profundidad de aquellas herramientas que a priori parecen más representativas destinadas a dar soporte a la especificación de los distintos modelos, finalizando con una comparativa de las diferentes propuestas, enriqueciendo de esta manera la motivación por la que se ha desarrollado este trabajo.

## 2.1 DESARROLLO DE INTERFACES DE USUARIO

La interfaz de usuario es el vínculo entre el usuario y el programa que se ejecuta en la computadora. Una interfaz es un conjunto de comandos, menús o de cualquier otro mecanismo, a través del cual el usuario se comunica con el programa. Entre sus principales funciones podemos destacar que son por ejemplo *la 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.*

Las interfaces de usuario pueden adoptar muchas formas, la más simple sería la línea de comandos. Consiste en una ventana que espera comandos introducidos por el usuario con el teclado, los interpreta y los entrega al programa para su ejecución de las acciones oportunas. La respuesta del programa es mostrada al usuario en la misma ventana, se interactúa con la información de la manera más simple posible, sin gráficas, solo el texto.

La interfaces que proporcionan la mayoría de las aplicaciones hoy en día son las interfaces gráficas. La interfaz gráfica de usuario (en inglés *Graphical User Interface*, GUI) es un tipo de interfaz de usuario que utiliza un conjunto de imágenes y objetos gráficos, por ejemplo ventanas, iconos, etiquetas, menús para representar la

información y acciones disponibles en la interfaz. Habitualmente las acciones se realizan mediante manipulación directa para facilitar la interacción del usuario con la computadora. Las interfaces que utilizan este tipo de elementos de diseño se han convertido en un estándar, al mismo tiempo han surgido otras formas de interactuar, nuevos dispositivos y cantidades ingentes de información disponibles para cualquier colectivo de usuarios.

Uno de los ejemplos más utilizados de interfaces gráficas son las llamadas WIMP (*Windows Icons Menus Pointer*), como la interfaz gráfica de Microsoft Windows o los escritorios para el sistema operativo Linux KDE o GNOME, que están basados en la interfaz gráfica presentada por Apple para sus MacIntosh. Dicha interfaz fue basada en los trabajos en el Xerox Parc, que a su vez están basados en los trabajos iniciales realizados en el *Stanford Research Laboratory* y el MIT (*Massachusetts Institute of Technology*). Las investigaciones que llevan a las interfaces de usuario que tendremos en el futuro están siendo realizadas en las universidades y unos pocos laboratorios comerciales (Jaquero, 2005).

Existen muchos tipos de software para la creación de interfaces de usuario. Pero un hecho comprobable es que prácticamente todas estas herramientas tienen en común que para generar interfaces gráficas usan un sistema de ventanas la cual permite la división de la pantalla en diferentes regiones rectangulares, llamadas “ventanas” cuya parte central es el conjunto de herramientas (*toolkit*). El sistema de ventanas provee de procedimientos que permiten dibujar figuras en la pantalla y sirve como medio de entrada de las acciones del usuario. Mientras que el *toolkit* contiene los objetos gráficos (*widgets*[1]) más empleados tales como menús, botones, etiquetas, barras de *scroll*, y campos para entrada de texto. El *toolkit* generalmente se conecta a los programas de aplicación a través de una serie de procedimientos definidos por el programador. La función de estos procedimientos es el decidir la forma en que se comportarán los objetos gráficos.

No se puede separar lo que es la evolución del desarrollo de interfaces de usuario de la propia evolución de la IS, de la que la disciplina IPO se nutre de metodologías, lenguajes y herramientas añadiendo consideraciones relacionadas con la interacción entre persona y ordenador. En el ámbito del desarrollo de software surgió la propuesta *Model Driven Architecture* (MDA) (OMG, 2003) del *Object Management Group* (OMG), en la cual reafirma el concepto de modelo y, a partir de él, asegura mejorar la calidad del software desarrollado, facilitando portabilidad, interoperabilidad y reusabilidad (Montero, 2005).

[1] Es un componente gráfico, o control, con el cual el usuario interactúa.

Ahora bien, en los últimos años se ha realizado un gran esfuerzo en la investigación de métodos que permitan la inclusión del diseño de la interfaz de usuario dentro de un proceso de desarrollo basado en modelos. De hecho en el ámbito académico, aunque no en el industrial, la propuesta metodológica más extendida para el desarrollo de interfaces de usuario está basada en la utilización de modelos. Estos son descripciones del sistema en diferentes direcciones y niveles de abstracción recogiendo características, identificando tareas, perfiles de usuario y datos que manipular. Después, por compilación de dichas descripciones, se obtiene una aplicación software.

El desarrollo de interfaces de usuario basado en modelos tiene como fin obtener beneficios tales como la automatización de la generación de la interfaz de usuario, la generación de dicha interfaz para distintos dispositivos o lenguajes a partir de unos modelos comunes o la mejora de las propiedades de usabilidad del sistema. Todo ello conlleva a una reducción de costes en el desarrollo de software.

En lo que se refiere modelar interfaces de usuario, aunque resulta sencillo modelar las componentes habituales de una interfaz de usuario, no es tan directo modelar interfaces de usuario a tal nivel de abstracción que cubran todos los aspectos que deben tenerse en cuenta, como pueden ser: diseño visual, asistencia, manejo de errores, facilidades de ayuda, etc. (Montero, 2005).

En el apartado siguiente se verán distintas metodologías que han sido propuestas para el desarrollo de interfaces de usuario basada en modelos.

## 2.2 PROPUESTAS METODOLÓGICAS

El proceso que determinaba la realización de propuestas metodológicas en IS, con el paso del tiempo, ha llegado a ser un Proceso Unificado, mientras en IPO lo que marca la realización de propuestas similares es la puesta en práctica de metodologías y técnicas de Diseño Centrado en el Usuario (DCU). Por Diseño Centrado en el Usuario se entiende aquel proceso por el que al usuario y a sus datos se les establece como criterio con el que evaluar los diseños, y como la fuente de ideas de diseño (Wixon, 1996). Las propuestas de (Gould et al., 1985) han sido utilizadas por (Vredenburg et al., 2002) para definir el DCU estableciendo que es el proceso por el que se involucra activamente al usuario, con el fin de entender sus requisitos y tareas, en un desarrollo

multidisciplinar basado en el diseño iterativo y en la evaluación. Por DCU se entiende, por tanto, una filosofía de desarrollo de software, que frente a la tradicional que se centra en la tecnología, pone al usuario como centro y objetivo de todo (Montero, 2005).

Los elementos más destacados para la puesta en práctica de una metodología de DCU han sido utilizados en los Mb-UIDEs, y son los siguientes:

**El modelado de usuario**: El modelo de usuario recoge las características del usuario final. Unas características que deben ser tenidas en cuenta para abordar la personalización final de la interfaz de usuario en sus posibles vertientes; adaptación y/o adaptabilidad, así como para permitir proporcionar ayuda al usuario con la que facilitar el uso del sistema software.

**El modelo de tareas:** el desarrollador recoge una especificación de las tareas que el usuario llevará a cabo en el sistema para lograr sus objetivos. Dichas tareas, que pueden ser de diferentes tipos, se representan utilizando una descomposición jerárquica de las acciones involucradas en su desarrollo, haciéndose uso de notaciones que serán presentadas en un apartado posterior.

**La presentación de interfaces de usuario**: Una representación común, que permita la especificación de un modelo de interfaz abstracto, la descripción de las características específicas de los diferentes dispositivos y de las unidades de información.

Llegados a este punto se ha conseguido aprender que la idea que subyace debajo de la propuesta de los Entornos de Desarrollo de Interfaces de Usuario basados en modelos es la especificación de todos aquellos aspectos relacionados con la interfaz de usuario y que influyen en la misma utilizando modelos. Se pueden revisar (Szekely, 1996; Schlungnaum, 1996; Puerta, 1997; Griffiths et al, 1998; Silva, 2000; Limbourg et al., 2004) para una discusión de la naturaleza de los modelos que involucra el modelo de interfaz de usuario utilizando una metodología basada en modelos.

Al no existir consenso en la notación o lenguaje utilizado, no todas las propuestas presentan los mismos modelos, ni utilizan las mismas notaciones para su especificación. Prácticamente, la única característica común que en la que se coincide es que son apuestas de carácter académico más que empresarial, y donde el modelado es el mecanismo utilizado. En cualquier caso, aparecen reiteradamente diferentes modelos característicos, algunos de forma constante y otros se han ido añadiendo conforme han ido sucediéndose las diferentes propuestas y los nuevos retos con los que se ha ido encontrando el desarrollo de interfaces de usuario. Los modelos de dominio,

de tareas, de usuario, de diálogo, de presentación, de plataforma, de contexto o de comportamiento son los modelos utilizados en las diferentes propuestas, y con ellos se recogen los elementos que tienen influencia para el posterior desarrollo de una aplicación y de su interfaz de usuario.

La verdadera eficacia y versatilidad del desarrollo basado en modelos para abordar el diseño de interfaces de usuarios no se encuentra en la propia especificación de modelos utilizando una notación u otra, y recogiendo información más o menos significativa en cada uno de esos modelos. La potencia de los Mb-UIDEs está en las diferentes relaciones que se establecen entre los modelos especificados.

A continuación se hablará de algunas de las propuestas académicas que ofrecen mecanismos que permiten el modelado de interfaces de usuario y posteriormente se hará hincapié en las diferentes notaciones propuestas que permiten abordar la tarea de modelar la presentación y de, indirectamente, generar prototipos de interfaz.

### 2.2.1 CAMELEON

El objetivo del proyecto de CAMELEON (*Context Aware Modelling for Enabling and Leveraging Effective interactiON*) (Calvary et al. 2003) es construir métodos y entornos de apoyo para el diseño y desarrollo de sistemas informáticos interactivos multi-contexto.

En este trabajo lo que se destaca del proyecto Europeo CAMELEON es que propone que el modelo de interfaz se estructure en cuatro capas de abstracción y estos están definidos desde la más abstracta a la más concreta. Dichos niveles de abstracción son los ilustrados en la Figura 2.1.


**Figura 2.1 Marco de desarrollo de interfaces de usuario definido en Cameleon**

Las **tareas y los conceptos** representan las tareas que el usuario podrá realizar a través de la interfaz de usuario y los objetos del dominio que dichas tareas deben manipular para ser llevadas a cabo por el usuario.

La **interfaz de usuario 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 interfaz de usuario.

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

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

Se puede considerar que el marco de desarrollo de interfaces que ofrece Cameleon constituyó el punto de partida de otros muchos métodos, entre ellos TERESA o UsiXML que se tratarán posteriormente en este mismo capítulo.

### 2.2.2 IDEAS

IDEAS (*Interface Development Environment within OASIS*) (Lozano, 2001) es una metodología de desarrollo de interfaces de usuario basada en modelos que cubre todo el ciclo de vida del desarrollo de una interfaz de usuario. El método permite la especificación de la interfaz de usuario tanto a través de la especificación de una serie de diagramas basados en UML, como formalmente usando el lenguaje de especificación OASIS. En IDEAS se propone un ciclo de vida iterativo centrado en el usuario.

La Figura 2.2 muestra los diagramas usados en cada una de las fases del desarrollo de una interfaz de usuario en IDEAS.

El modelo de tareas en el caso de IDEAS es modelado a través de dos tipos de diagramas de secuencia. El modelo de dominio es modelado usando diagramas de clases. Los diagramas de clases son usados además en la especificación del diagrama de roles (usado para la personalización estática de la interfaz de usuario para cada uno de los roles definidos).

El modelo de diálogo es modelado usando dos tipos de diagramas basados en los diagramas de estados. Por una parte se modela las transiciones que pueden ocurrir entre las distintas ventanas (diagrama de interacción de diálogos), y por otra parte se modela cuáles son las transiciones internas que se producen dentro de cada ventana. En IDEAS propone este modelo de diálogo para la descripción grafica de la GUI que lleve a la implementación final de la UI. Este Modelo de Dialogo es una descripción abstracta de las acciones, y sus posibles relaciones temporales, que los usuarios y los sistemas pueden llevar a cabo a nivel de UI durante una sesión interactiva.

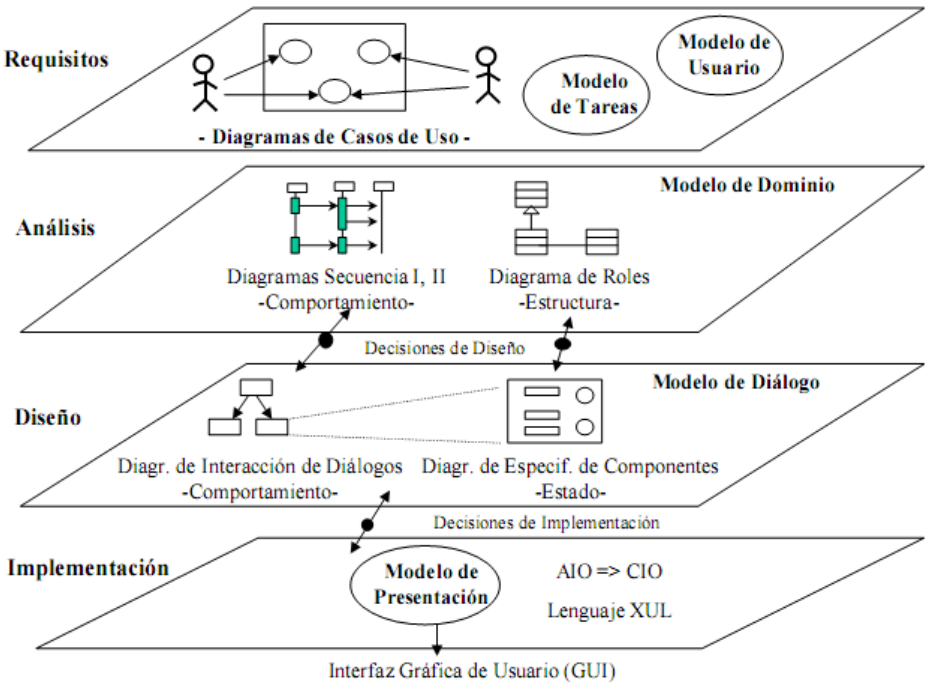


**Figura 2.2 Etapas de diseño en IDEAS.**

Concretamente, en IDEAS el Modelo de Diálogo, establece cuando el usuario puede invocar comandos, seleccionar o especificar datos de entrada y cuando el sistema puede requerir datos del usuario o mostrar los datos de salida. Para todo ello se hace uso de los Objetos de Interacción Abstractos, del inglés *Abstract Interaction Objects* (AIOs).

El Modelo de Dialogo se aborda desde cuatro diagramas: Diagrama de Interacción de Diálogos, Diagrama de Estados Internos, Diagrama de Especificación de Componentes y Tabla de Definición de Componentes.

Es en el diagrama de especificación de componentes donde se plantea una especificación abstracta de la interfaz de usuario). Esta especificación abstracta es traducida al lenguaje XUL para su visualización. (Lozano, 2001).

### 2.2.3 TERESA

TERESA (*Transformation Environment for inteRactivE Systems representAtions*) es una herramienta diseñada para la generación de interfaces de usuario para distintas plataformas (portátiles, ordenadores de sobremesa, PDA, etc.) a partir de modelos de tareas creados utilizando la notación de ConcurTaskTrees (CTT). La notación CTT (Paternò, 1999) es una notación basada en el lenguaje formal LOTOS para el análisis y generación de interfaces de usuario a partir de modelos de tareas. Esta notación está siendo utilizada ampliamente en distintas metodologías como método para la especificación del modelo de tareas.

*CTT* se centra en las actividades, presentando una estructura jerárquica de éstas. Para facilitar su especificación *CTT* posee una notación gráfica. Las relaciones temporales y de concurrencia entre las distintas tareas pueden ser especificadas utilizando los operadores correspondientes. La notación *CTT* puede ser editada utilizando el editor CTTE, herramienta que describiremos más adelante.

En *CTT* existen cuatro tipos de tareas:

- Tareas de usuario: son las tareas realizadas por el usuario; normalmente son actividades cognitivas importantes, como por ejemplo decidir cuál es la mejor estrategia para resolver un problema.

- Tareas de aplicación: son las tareas ejecutadas completamente por la aplicación. Las tareas de aplicación suministran información al usuario, como por ejemplo presentar los resultados de una consulta a una base de datos.

- Tareas de interacción: son las tareas que realiza el usuario interactuando con el sistema, por ejemplo pulsar un botón.

- Tareas abstractas: son las tareas que necesitan actividades complejas para su ejecución (y que normalmente se descompondrán en tareas más simples), como por ejemplo una sesión del usuario con el sistema.

En cuanto a los operadores temporales en *CTT*: una tarea podrá ser descompuesta en distintas subtareas de distintos tipos. Estas subtareas podrán estar relacionadas mediante los operadores temporales de *CTT*.

Actualmente, la notación *CTT* se ha convertido en la notación para especificación de tareas más utilizada, y va camino de convertirse en el estándar en la especificación de modelos de tareas. (Jaquero, 2005)

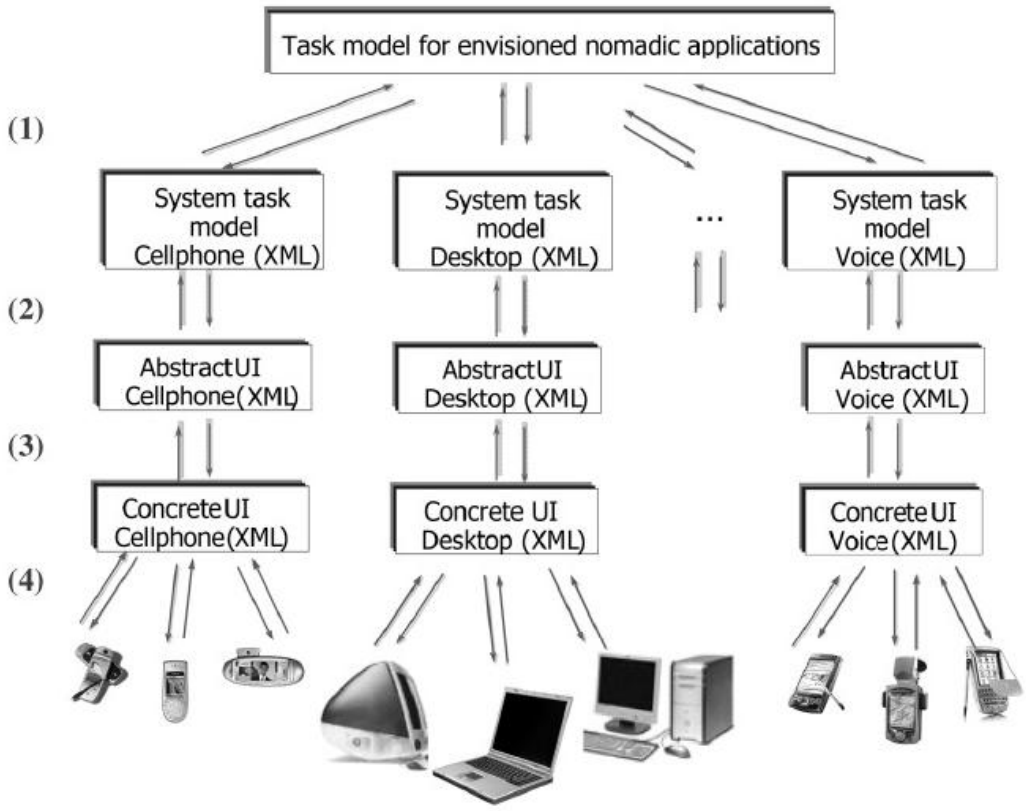


**Figura 2.3 En Multimodal Teresa se pueden diseñar diferentes interfaces de usuario, para diferentes plataformas, a partir de un mismo modelo de tareas.**

Uno de los trabajos más importantes y recientes de TERESA es Multimodal Teresa (Mori et al., 2004; Paternò et al., 2006), un entorno para el diseño de interfaces multimodal. Maneja diferentes niveles de abstracción partiendo del modelo de tareas de

CTT para crear las UIs mediante una serie de transformaciones semiautomáticas. Interfaces que se generaran para la o las plataformas especificas a partir del mismo modelo inicial. La Figura 2.3 muestra un esquema del funcionamiento de TERESA. En primer lugar se construye un modelo de tareas de alto nivel para aplicaciones independientemente de la plataforma. El primer paso del método consiste en la elaboración de un modelo de tareas del sistema para las diferentes plataformas consideradas (telefonía, escritorio, voz, etc.). Entonces se genera una interfaz de usuario abstracta, AbstractUI, según la plataforma. En el siguiente paso, según propiedades especificas de cada plataforma, se generan las interfaces de usuario concretas: ConcretUI. El cuarto y último paso consiste en la generación del código a partir de la descripción de la interfaz concreta. (Mori et al., 2004).

Por los cuatro pasos del método seguido puede apreciarse que Multimodal Teresa se basa en el *Cameleon Reference Framework* (Calvary et al., 2003) que define los pasos para el desarrollo de UI para aplicaciones interactivas multi-contexto.

### 2.2.4 OVID

La metodología OVID (*Object, View and Interaction Design*) es un conjunto de técnicas para el diseño de interfaces de usuario orientadas a objetos desarrolladas por IBM. La metodología se centra en tres elementos del diseño de la interfaz de usuario: los *objetos* que el usuario percibe, las *vistas* que se proporcionan de esos objetos, y las *interacciones* que el usuario tiene con los objetos. (Roberts et al., 1998)

El proceso básico de esta metodología es el siguiente:

1. Se genera un conjunto inicial de objetos examinando el análisis de tareas.

2. Se definen las vistas para permitir al usuario ver los aspectos correspondientes de cada objeto.

3. Se describen las tareas en términos de los nuevos objetos y vistas.

4. Se describen en detalle las interacciones del usuario con los objetos.

En la Figura 2.4 se puede observar este ciclo de desarrollo. Para diseñar los objetos, las vistas y las interacciones, OVID toma como entrada los requisitos y el

análisis de las tareas del usuario. OVID genera a partir de estas entradas una salida estructurada que puede ser integrada fácilmente en las metodologías para el diseño de programas. Para ello, OVID se sirve de UML como lenguaje de especificación. (Jaquero, 2005)

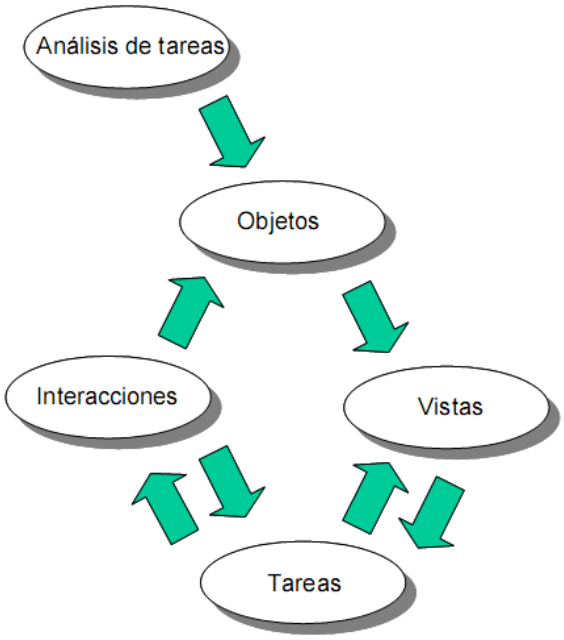


**Figura 2.4 Ciclo de desarrollo en OVID**

### 2.2.5 JUST-UI

El propósito de Just-UI (Molina, 2003) es la generación automática de interfaces de usuario para distintas plataformas utilizando para ello modelado conceptual. Este método está orientado a la generación de aplicaciones de gestión principalmente.

La principal aportación de Just-UI es la inclusión de *patrones* conceptuales para la captura de los requisitos necesarios para la generación de la interfaz de usuario de forma automática extendiendo OO-Method, un método para la generación automática de aplicaciones basado en el lenguaje formal OASIS.

Dentro de este método se introducen dos tipos de *patrones*. Los *patrones simples* que constituyen las primitivas necesarias para la creación de la interfaz de usuario (como por ejemplo, un filtro), y los *patrones de presentación*, que permiten la agrupación de los AIO que componen las unidades de presentación en un *patrón*. Por ejemplo, el patrón Instancia de Presentación permite modelar la presentación e interacción con una instancia.

Además, de los dos tipos de *patrones* anteriormente comentados, también se introduce el concepto de acceso al sistema para modelar los puntos a través de los cuales el usuario puede comenzar su interacción con el sistema.

Los patrones en Just-UI se organizan en tres niveles:

**Nivel 1**: Árbol de jerarquía de acciones donde se define el acceso a la funcionalidad del sistema.

**Nivel 2**: Unidades de interacción que presentan al usuario escenarios donde se produce el diálogo con el sistema, pueden ser de servicio, de instancia de población y maestro/detalle.

**Nivel 3**: Patrones auxiliares que funcionan como piezas base para crear las unidades de interacción y complementan a estas aportando semántica precisa relacionada con actividades ligadas a la interacción como introducción, selección, información complementaria, dependencia, agrupación de argumentos, filtrado, visualización, navegación, acción y ordenación.

Con la finalidad de despejar alguna duda del uso de *patrones* conceptuales para la captura de los requisitos en la generación de la interfaz de usuario se mostrará el siguiente ejemplo:

El nivel 1 está constituido por sólo un patrón: el *Árbol de jerarquía de acciones (AJA)*. Este patrón se emplea para definir el acceso a la funcionalidad de la aplicación por parte del usuario. Dependiendo del tipo de usuario, los permisos establecidos, las tareas que deba llevar a cabo o el dispositivo a emplear, el acceso a presentar puede ser radicalmente distinto. Por ese motivo, se sugiere disponer de análisis previo de tareas (Paternò, 2000) o árboles de descomposición funcionales (ARF) como ayuda al diseño de AJA efectivos. En la Figura 2.5 se muestra un meta-modelo asociado a este patrón. El meta-modelo ayuda a explicar la relación del patrón con otros elementos del modelo conceptual y constituye la base para construir herramientas de soporte al modelo como editores, representaciones del modelo y traductores del modelo.

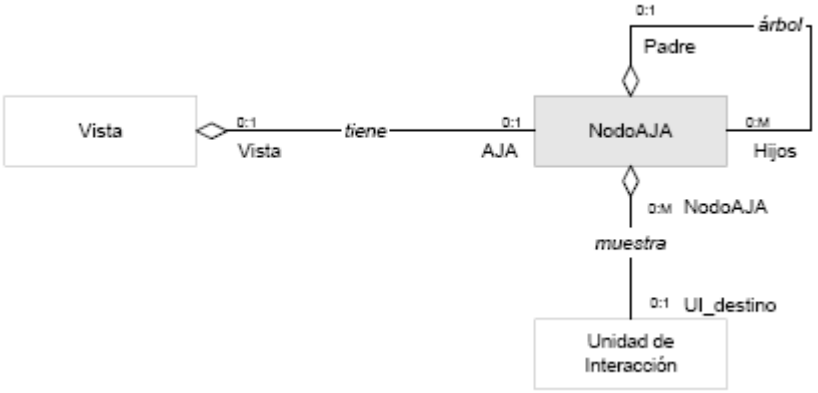


**Figura 2.5 Meta-modelo del árbol de jerarquía de acciones**

Cada patrón, de los propuestos por esta metodología, está descrito por una plantilla de descripción adaptada al análisis de interfaces de usuario en un marco de modelado conceptual, en dicha descripción se encuentra el meta-modelo.

Para modelar la navegación en JUST-UI se crean diagramas de navegación, los cuales consisten en un grafo, cuyos nodos son los *patrones* identificados.

El método propuesto en esta aproximación es soportado por la herramienta Just-UI/Visio. (Molina, 2003).

### 2.2.6 MOBI-ID

MOBI-D (*Model-Based Interface Designer*) es un conjunto de herramientas para el desarrollo de interfaces de usuario desarrollado en la Universidad de Stanford por Ángel Puerta y su equipo. Se trata de un sistema que soporta el diseño de la interfaz de usuario por medio de modelos de Tareas, Dominio, Usuario, Dialogo y Presentación.

**Las fases de diseño de una interfaz de usuario en MOBI-D**

MOBI-D está conformado por cuatro herramientas complementarias que cubren todo el ciclo de vida del desarrollo de una interfaz de usuario (Puerta, 1997):

- Elicitación de las tareas del usuario: esta fase es soportada por la herramienta U-Tel.

- Definición de los modelos de tareas, usuario y dominio. La edición de estos modelos es creada con la herramienta MOBI-D.

- Integración de los modelos de tareas del usuario, usuario y dominio. La integración de estos modelos es soportada por la herramienta MOBI-D.

- Análisis de los modelos de diseño de forma interactiva para derivar los potenciales diseños de la presentación y los diálogos. Esta actividad es soportada por la herramienta TIMM.

- Creación de la presentación y los diálogos basados en las tareas. Esta fase es soportada por la herramienta MOBILE.

El lenguaje de modelado asociado, MIMIC, proporciona además la posibilidad de realizar un Modelo de Diseño. Una de las principales ventajas de MOBI-D se produce en su Modelo de Diseño, donde se describen las relaciones dinámicas entre las entidades del resto de modelos.

El lenguaje de especificación XIML surgió como evolución de la propuesta MIMIC, de dicho lenguaje se hablará en un apartado posterior de este trabajo.

**La arquitectura de MOBI-D**

Como se puede ver en la Figura 2.6 el modelo de interfaz actúa como repositorio central para el conocimiento sobre el diseño de la interfaz. En este repositorio tenemos una representación declarativa de todos los aspectos relevantes de la interfaz de usuario, incluyendo sus componentes y su diseño.

Los desarrolladores acceden y modifican el modelo de interfaz a través de las herramientas proporcionadas, las cuales aportan un cierto grado de automatización. Obsérvese como se mantiene una separación entre el desarrollo de la parte funcional de la aplicación y la interfaz de usuario.

MOBI-D ha sido otra de las propuestas basadas en modelos más influyentes en las propuestas actuales, como evolución a su antecesor MECANO, proponiendo un ciclo completo dentro del desarrollo de interfaces de usuario cubierto por un conjunto de herramientas. (Puerta et al., 1999)

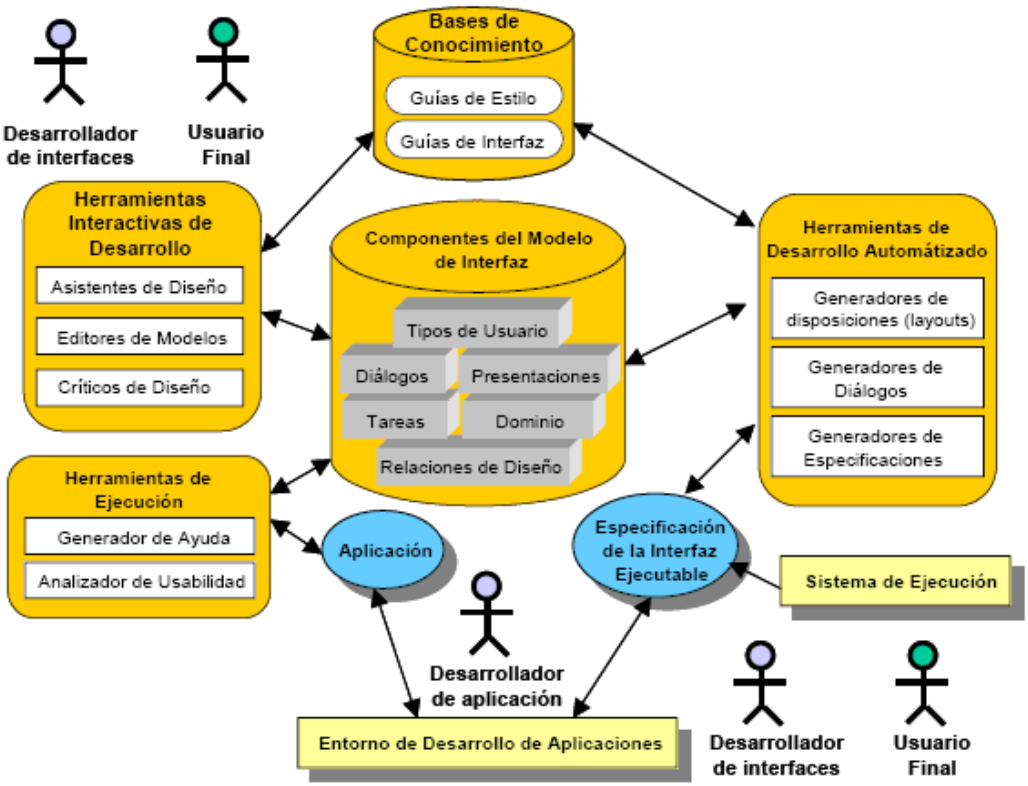


**Figura 2.6 Arquitectura de MOBI-D**

### 2.2.7 OO-H (Object Oriented Hypermedia Method)

OO-H (Cachero, 2003) es un método diseñado para la creación de aplicaciones Web. Para ello se basa en tres modelos principales: el *Diagrama de Acceso Navegacional* (DAN), el *Diagrama de Presentación Abstracta* (DPA) y el *Diagrama de Diseño Visual* (DDV).

El método utiliza diagramas de clases para modelar el dominio del problema, el cual es completado añadiéndole una serie de estereotipos.

El *diagrama de acceso navegacional* se crea usando cuatro constructores básicos:

- Destino navegacional: es usado para agrupar constructores facilitando el diseño, de forma análoga a como funcionan los paquetes en UML.

- Clase navegacional: son vistas de las clases identificadas en el modelo de dominio. A través de estas vistas se puede especificar la visibilidad de los distintos atributos de las clases.

- Enlace navegacional: permiten describir los caminos que puede seguir el usuario para navegar a través de las vistas creadas con las clases navegacionales.

- Colección: es un agrupamiento de enlaces navegacionales (por ejemplo para crear menús).

El *diagrama de presentación abstracta* refleja cómo será la estructura y cuáles son las relaciones de cada página, de forma análoga a como sucede en la definición de **interfaz de usuario abstracta**.

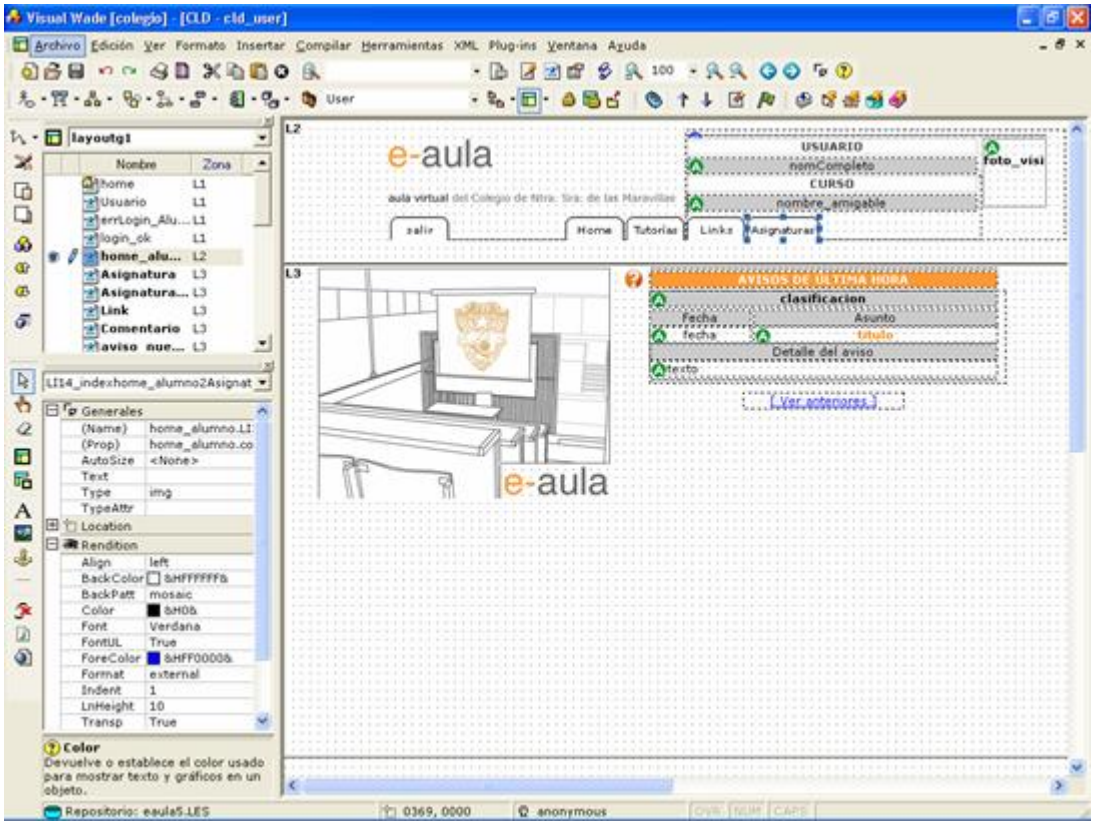


**Figura 2.7. Entorno de trabajo VisualWade que está asociado a la propuesta OO-H**

Finalmente, el *diagrama de diseño visual* permite crear una presentación más concreta basada en el *diagrama de presentación abstracta*, donde se puede definir principalmente, cómo se va a realizar la visualización de los distintos constructores usados en *diagrama de acceso navegacional*.

OO-H permite una personalización de la interfaz de usuario Web generada basándose en un modelo de usuario definido a través de un diagrama de clases. Dentro de dicho diagrama de clases también es posible la inclusión de perfiles que pueden ser usados por los usuarios particulares siguiendo relaciones de herencia.

Todo el proceso de diseño puede ser realizado a través de la herramienta VisualWADE. (Cachero, 2003).

### 2.2.8 UWE

UWE (Koch, 2006) es una metodología basada en modelos proveniente del ámbito hipermedial. Dicha metodología propone un método que cubre todo el proceso de desarrollo del software. La principal ventaja de UWE es que sólo usa *modelos basados en UML* para el modelado de la aplicación, introduciendo un nuevo perfil para ello. De esta forma facilitando el acceso a dicho método de los desarrolladores acostumbrados al uso de *UML*. Además, también es un método conservativo con respecto a los *modelos UML* que utiliza. Para completar la especificación del sistema, UWE hace uso del lenguaje OCL, lenguaje habitualmente utilizado en la especificación de restricciones dentro de los distintos diagramas UML.

UWE propone un método de diseño para el diseño de aplicaciones hipermediales adaptativas, por lo que enfatiza en gran medida el concepto de personalización dentro del meta-modelo utilizado.

La metodología propone un proceso compuesto por cuatro actividades, cada una de las cuales determina la especificación de una serie de modelos.

1. Etapa de análisis: dentro de esta etapa se realiza un análisis del sistema construyendo para ello diagramas de casos de uso.

2. Etapa de diseño conceptual: dentro de esta etapa se modela el universo de la aplicación, creando para ello el modelo de dominio.

3. Etapa de diseño navegacional: dentro de esta etapa se define la navegación entre los distintos objetos del dominio. Para ello se construyen los modelos de Espacio de navegación y Estructura de navegación.

4. Etapa de diseño de la presentación: la presentación se describe en función de distintos modelos estándares *UML*.

El proceso de desarrollo propuesto es soportado por la aplicación AgoUWE, herramienta que se apoya en ArgoUML para su implementación (Knapp et al., 2004)

### 2.2.9 UML*i*

En UML*i* de Pinheiro da Silva se propone una *ampliación* de la notación propuesta dentro de *UML* para intentar atajar las limitaciones que dicha notación presenta para el diseño de interfaces de usuario**.**

UML*i* es, al igual que UWE, una *ampliación de UML* conservativa, facilitando su utilización por parte de desarrolladores familiarizados con *UML*. El método está centrado en la especificación de interfaces de usuario, y no en la generación de una interfaz de usuario ejecutable a partir de su especificación. Al igual que muchas de las propuestas para el desarrollo de interfaces de usuario basados en modelos, centra su radio de acción principalmente a las interfaces de usuario basadas en formularios. El método se apoya en la notación formal del lenguaje LOTOS.

Pinheiro ofrece una serie de primitivas para la especificación abstracta de la interfaz de usuario:

- *Freecontainer*: representan contenedores raíz, que no pueden ser contenidos por ningún otro contenedor.

- *Container*: representan contenedores que podrán contener o ser contenidos por otros contenedores. Estos contenedores también podrán contener a todo el resto de primitivas abstractas (excepto a las Freecontainer).

- *Inputter*: representan objetos de interacción abstractos a través de los cuales el usuario puede hacer una operación de entrada (como por ejemplo, obtener un dato a través del teclado).

- *Displayer*: representan objetos de interacción abstractos usados para realizar operaciones de salida (como por ejemplo, mostrar un mensaje al usuario).

- *Editor*: representan objetos de interacción abstracta que son a la vez Inputter y Displayer.

- *ActionInvoker*: representan aquellos objetos de interacción abstractos capaces de recibir eventos y desencadenar una acción (como por ejemplo, un botón).

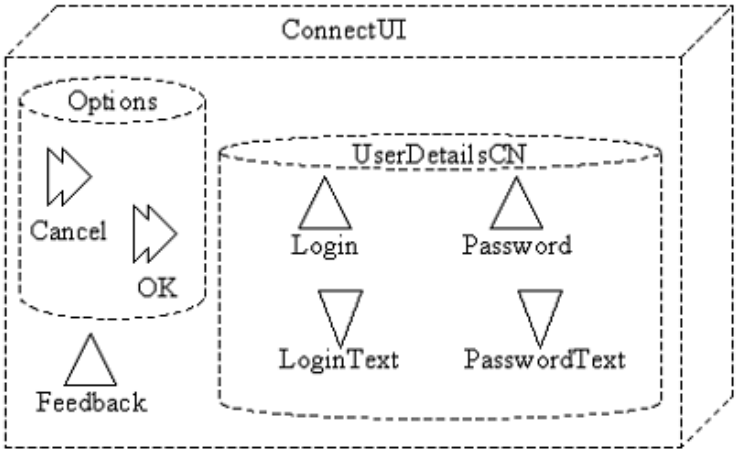


**Figura 2.8 Especificación abstracta de una interfaz de usuario en UMLi**

El modelo de tareas en UML*i* es especificado utilizando una extensión de los diagramas de actividad. Una de las extensiones aplicadas sobre los diagramas de secuencia usados en la inclusión de nuevos operadores para la selección de estados que permiten especificar los conceptos de repetición, opcional u orden aleatorio.

El método es soportado por ARGO*i*[2], una extensión del entorno ArgoUML de código abierto. (Silva et al., 2003)

UMLi En el modelado de la presentación concreta se describe la interfaz en términos de dependencia del entorno, de manera que se presentan directamente los objetos que los usuarios manipulan en la UI. Por ejemplo, la Figura 2.9 muestra los elementos del modelo de presentación concreto para los componentes de Java Swing, uno de los toolkits mas empleados para el desarrollo de todo tipo de aplicaciones. (Penichet, 2007)

[2] http://www.cs.man.ac.uk/img/umli/download2.html

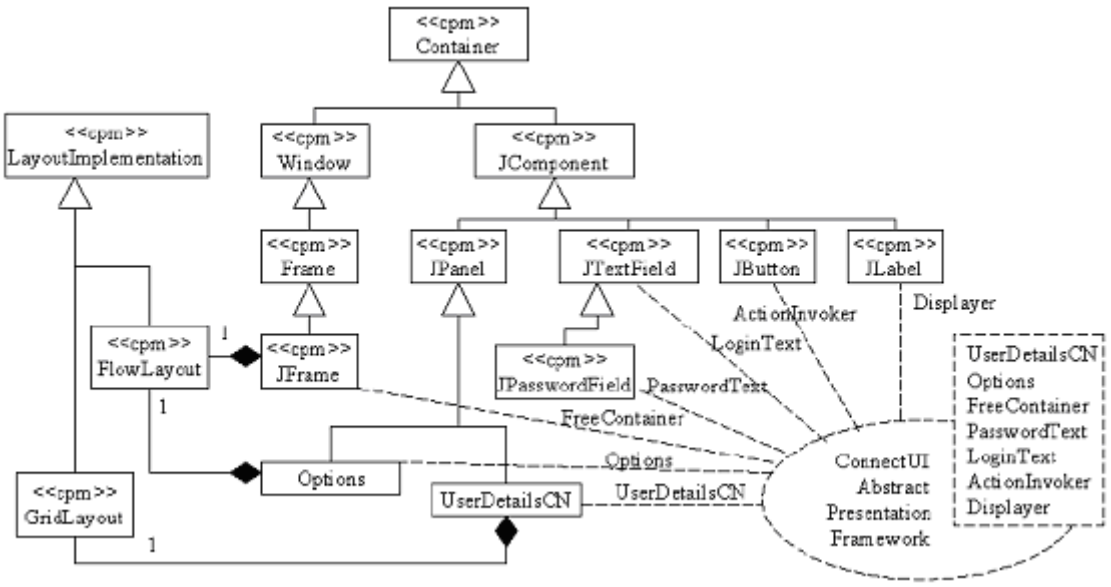


**Figura 2.9 Modelo de presentación concreto para elementos del toolkit Java Swing en UMLi**

## 2.2.10 USIXML

UsiXML (*User Interface eXtensible Markup Language*)[3] (Limbourg et al., 2004) propone un lenguaje de descripción de interfaces de usuario (*User Interface Description Language*, UIDL), basado en XML, con el que se asegura poder trabajar a diferentes niveles de abstracción (tareas, dominio, interfaz abstracta, interfaz concreta, interfaz final). Puede describir interfaces para múltiples contextos de uso, con independencia de instrumento, de plataforma y de modalidad. Esta propuesta garantiza la trazabilidad, tanto hacia arriba como hacia abajo, y tiene como referente el nivel de abstracción. También se proponen mecanismos para gestionar las asociaciones que se establecen entre los diferentes modelos contemplados.

Está basado en los principios de modelado MDA, así como en el marco de referencia Cameleon, un proceso MBUI ya explicado en este trabajo que define los pasos y tipos de modelos en el desarrollo de interfaces de usuario para aplicaciones interactivas multi-contexto. Como recordatorio, se proponen cuatro tipos de modelos, cuya jerarquía se muestra en la Figura 2.10, junto con los roles que participan en el desarrollo de una interfaz.

[3] http://www.usixml.org

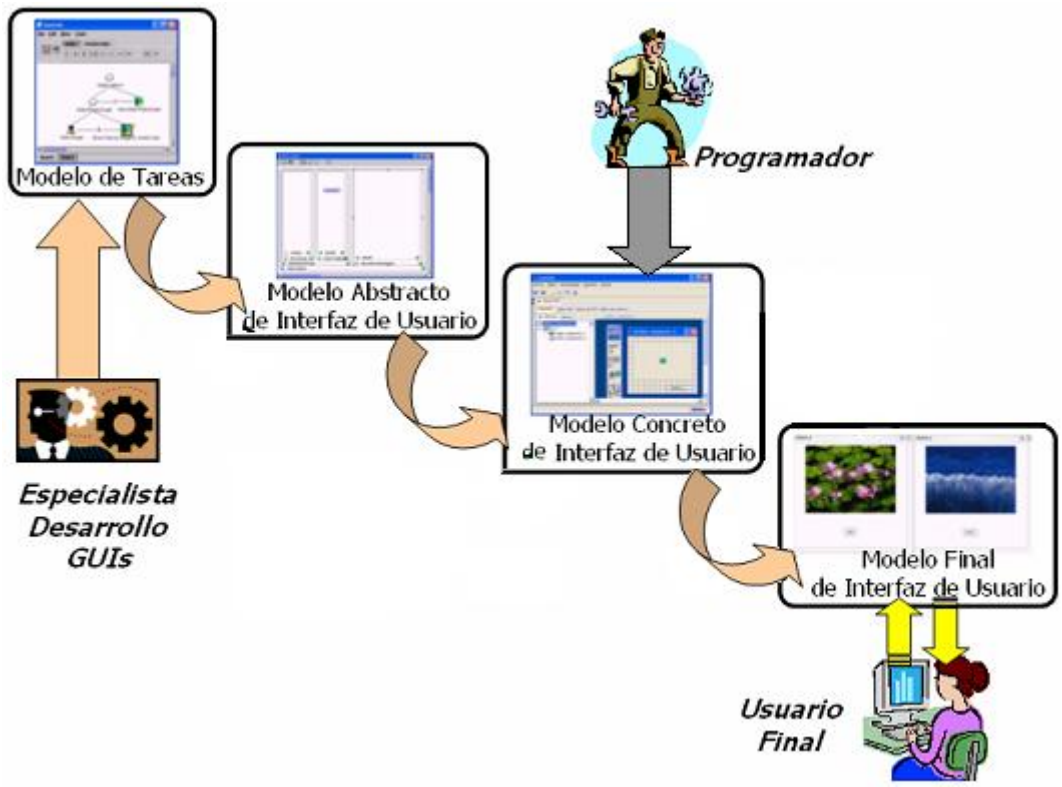


**Figura 2.10 Jerarquía de modelos de UsiXML y procesos en su construcción.**

Una interfaz de usuario en UsiXML 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. A continuación se describen los modelos incluidos en UsiXML.

### 2.2.10.1 Modelo de Interfaz de Usuario

El modelo de interfaz de usuario representa toda la información disponible de la interfaz de usuario, y actúa por lo tanto como contenedor del resto de modelos.

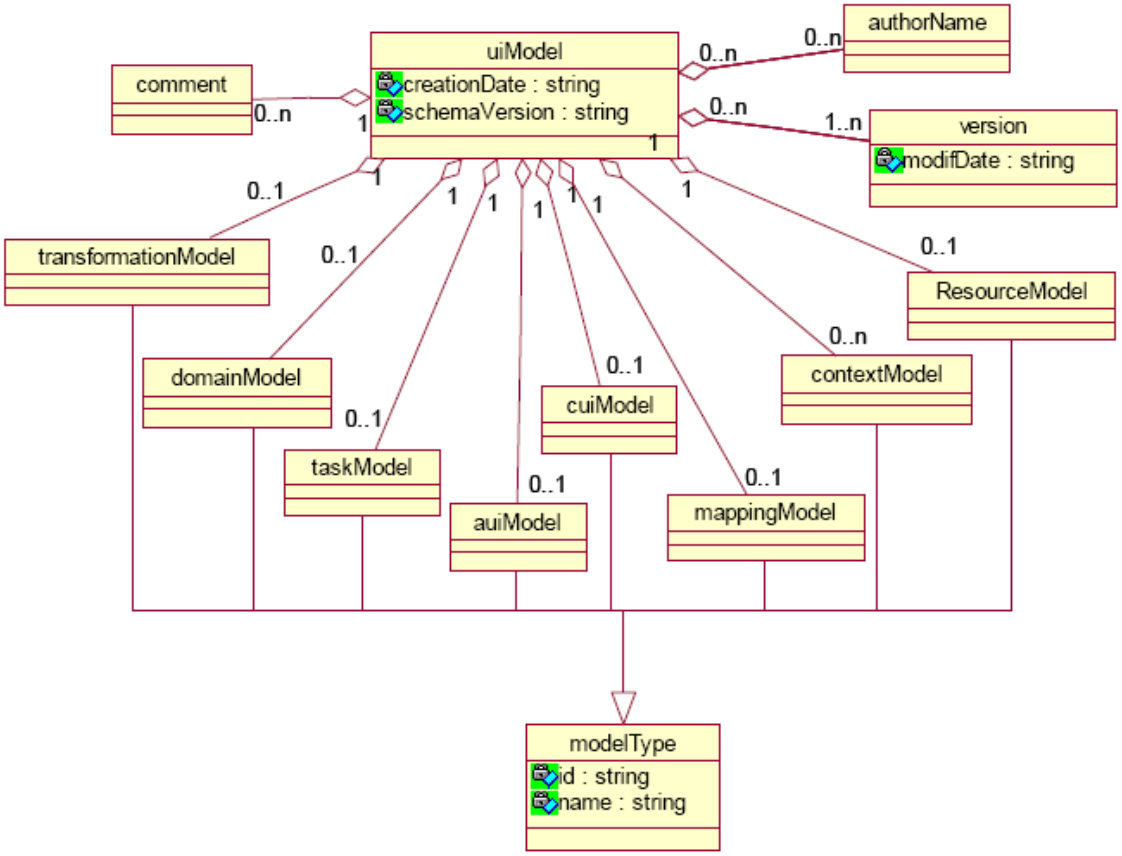


**Figura 2.11 Modelo de interfaz de usuario de UsiXML.**

### 2.2.10.2 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 interfaz de usuario. Las tareas se descomponen en acciones atómicas que representan los pasos necesarios para alcanzar los objetivos de la tarea. En UsiXML permite representar un modelo de tareas en forma de árbol, siguiendo una notación muy similar a la propuesta en CTT, con una representación en XML.

El modelo de tareas es el pilar principal de una aproximación basada en modelos. Dicho modelo proporciona una descripción de las tareas que el usuario podrá realizar a través de la interfaz de usuario, así como una especificación de las relaciones temporales que existen entre dichas tareas. Por ejemplo, dichas relaciones especifican si dos tareas pueden realizarse al mismo tiempo, o si por el contrario deben realizarse siguiendo una secuencia temporal predeterminada.

Habitualmente la captura de los requisitos de las distintas tareas que se realizarán con la interfaz de usuario suele realizarse utilizando diagramas de casos de uso. Por ello el proceso de diseño del modelo de tareas es guiado lógicamente por el análisis de casos de uso realizado en la fase de requisitos. Cada caso de uso representa una tarea abstracta que el usuario debe llevar a cabo.

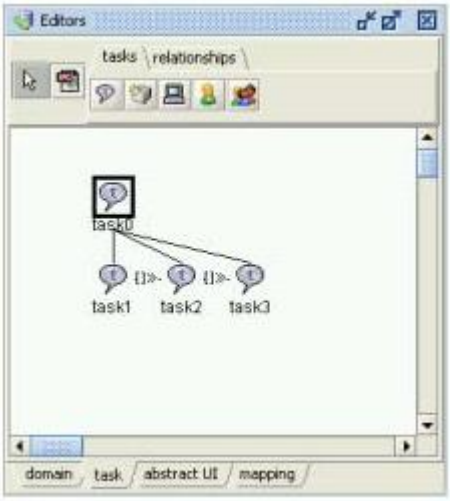


**Figura 2.12 Ejemplo de un modelo de tareas**

A partir del modelo de tareas y los objetos de interacción asociados a las acciones de dicho modelo se deriva una interfaz de usuario abstracta que representa, de una manera independiente de la modalidad y la plataforma, la futura interfaz de usuario. (Jaquero, 2005).

### 2.2.10.3 Modelo de Dominio

El modelo de dominio representa los objetos del dominio con los que interactúan las tareas que el usuario podría realizar con el sistema. UsiXML permite expresar un diagrama de clases y objetos de UML con una representación XML, incluidas las relaciones entre las distintas clases y objetos. Adicionalmente, también permite especificar ciertas propiedades de los atributos de las clases y objetos que permiten generar la interfaz de usuario de una forma más precisa.

#### 2.2.10.4 Modelo de Interfaz de Usuario Abstracta (AUI)

El modelo de interfaz abstracta permite representar la interfaz de usuario utilizando un alto nivel de abstracción. Esta descripción abstracta es independiente tanto de la plataforma destino como de la modalidad (es decir el modo de interacción del usuario) y de cualquier contexto de uso.

UsiXML proporciona un modelo de interfaz de usuario abstracta que representa una expresión canónica de prestaciones y manipulación de conceptos del dominio y funciones que como ya se ha mencionado son tan independientes como sea posible de modalidades y plataforma de computación específica.

La interfaz de usuario abstracta está formada por objetos de interacción abstracta (AIO) que pueden ser de dos tipos: componentes de interacción abstractos (AIC) y contenedores abstractos (AC). Un componente de interacción abstracta es una abstracción que permite la descripción de objetos interactivos que son independientes de la modalidad por la que son proporcionados en el mundo físico. Un componente de interacción abstracta puede estar compuesto de múltiples facetas. Cada faceta describe una función particular del componente de interacción abstracta que puede estar endosada en el mundo real.

Hay cuatro tipos de facetas: la faceta de entrada describe una acción de entrada soportada por un componente de interacción abstracto (AIC); la faceta de salida describe los datos que pueden ser mostrados al usuario por el componente de interacción abstracto (AIC); la faceta de navegación describe la posibilidad de transición del componente; y la faceta de control describe el enlace entre un componente de interacción abstracto (AIC) y las funciones del sistema. Un contenedor abstracto (AC) es una entidad que permite la agrupación lógica de otros ACs u otros AICs. Los ACs dan soporte a la ejecución de conjuntos de tareas conectadas lógica o semánticamente. Los contenedores abstractos (AC) pueden ser transformados de nivel abstracto a concreto, dentro de uno o más contenedores gráficos como ventanas o cuadros de diálogo, cuadros de esquemas (*layout boxes*) o recuadros de emisión en el caso de auditoria de interfaces, mientras un componente de interacción abstracto (AIC) podría asumir diferentes funcionalidades puesto que puede asumir diferentes facetas. Esta transformación de nivel abstracto a concreto se denomina reificación y en sentido contrario sería abstracción.

En este modelo es posible establecer relaciones. Una relación importante es la relación del control de diálogo. Esta relación permite una especificación de un flujo de

control entre los objetos de interacción abstracta y pueden ser derivados de relaciones del modelo de tareas. (Montero, 2005)

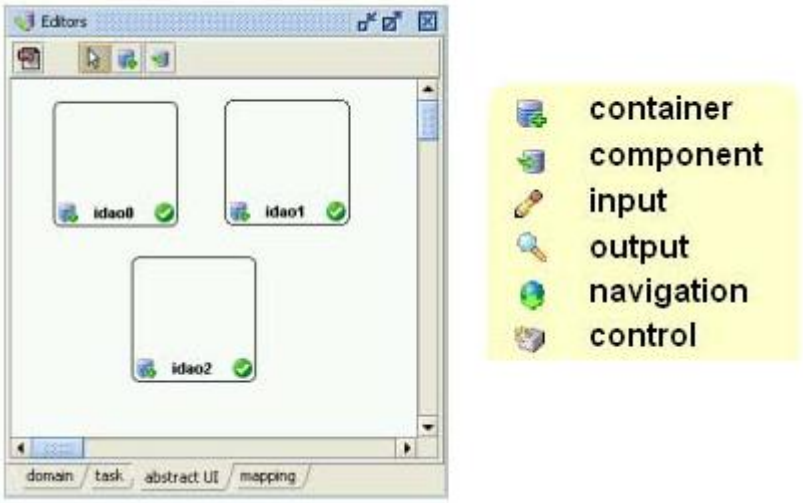


**Figura 2. 13 Ejemplo de una Interfaz de Usuario Abstracta y la asociación de una notación gráfica para cada elemento de AUI establecida por la tesis de Montero (Montero, 2005).**

La Figura 2.14 muestra en detalle la relación de los conceptos empleados en esta ontología en el modelado de la IU abstracta.

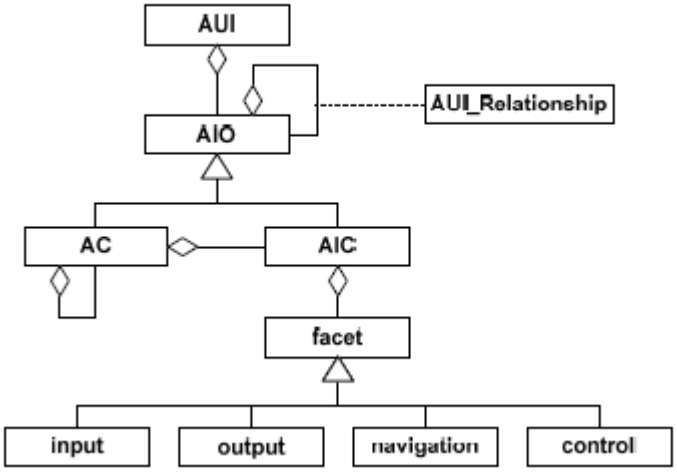


**Figura 2.14 Detalle de la relación de los conceptos empleados en UsiXML en el modelado de la UI abstracta**

### 2.2.10.5 Modelo de Interfaz de Usuario Concreta

El modelo de interfaz concreta permite una descripción o especificación detallada de la apariencia e interactividad de las interfaces de usuario a un sistema interactivo, incluyendo las representaciones de objetos principales y el comportamiento del sistema de interfaz de usuario. La especificación de la interfaz de usuario concreta se describe de forma independiente de la posterior implementación. (Montero, 2005).

Un modelo de Interfaz de Usuario Concreta **es un modelo de interfaz de usuario que permite la especificación de la apariencia y el comportamiento de una interfaz de usuario con elementos que pueden ser percibidos por los usuarios**. Un modelo de interfaz de usuario concreta es compuesto de Objetos de Interacción Concreta (CIO) y relaciones concretas. Un objeto de interacción concreta (CIO) es definido como una entidad que los usuarios pueden percibir como texto, gráficos, animación y/o manipular (por ejemplo un botón, list box o un check box). Un CIO se caracteriza por atributos (aunque no siempre) tales como por ejemplo: id, name, icon, content, defaultContent, defaultValue. Cada CIO puede clasificarse en subcategorías CIOs dependiendo de la modalidad de interacción de la interfaz de usuario final (FUI): graphicalCIO para GUIs, AuditoryCIO para las interfaces vocales, 3DCIO para interfaces de usuario 3D, etc. En este trabajo se centra en CIO gráfica, en esta subcategoría se encuentran los elementos básicos de una tradicional interfaz gráfica 2D o 3D (IU Virtual). Cada graphicalCIO hereda las propiedades de los objetos CIO más generales, a su vez cada graphicalCIO puede encontrase en una de las dos categorías posibles, como se puede observar en la Figura 2.15:

1. Los elementos *Graphical Container* (GC) que son aquellos que contienen otros widgets, estos elementos son por ejemplo ventanas, esquemas (layout), cuadros de diálogo, paneles.
2. Los elementos *Graphical Individual Component* (GIC) que son aquellos elementos tradicionales que se encuentran típicamente en los anteriores Contenedores. Es decir elementos tales como textComponent, videoComponent, imageComponent, imageZone, radioButton, toggleButton, icon, checkbox, item, comboBox, button, tree, menu, menuItem, drawingCanvas, colorPicker, hourPicker, datePicker, filePicker, progressionBar, slider, y cursor.

Gracias a este mecanismo de la herencia progresiva, todos los elementos finales de la CUI heredan de la parte superior aquellas propiedades de la categoría a la que pertenecen. Las propiedades que han sido escogidas para asignárselas al conjunto de elementos del nivel CUI están basadas en las propiedades comunes establecidas en la mayoría de las herramientas de diseño de ventanas de interfaz como Windows GDI,

Java AWT y Swing, HTML. Estas propiedades pertenecen a la intersección de la propiedad de los principales conjuntos de herramientas y gestores de ventanas, como Windows GDI, AWT y Swing de Java, HTML. Por supuesto, sólo las propiedades que se mantienen son de alto interés común. De esta manera, un CIO puede especificarse independiente del lenguaje de programación de alto nivel (HTML, VRML o Java) que se utilice para desarrollar el software final. (Vanderdonckt, 2005).

Los conceptos importantes de CIO que se deben quedar claros son:

- Un CIO es una entidad que el usuario puede percibir o manipular. Un CIO es una abstracción de un conjunto de *widgets* encontrados en los *toolkits* más populares y estándares.

- En otras palabras, un CIO trata de generalizar elementos que aparecen en diferentes herramientas de desarrollo que el usuario final manipula o percibe. Una etiqueta puede aparecer en diferentes *toolkits*, pero la esencia es la misma en todos ellos.

- Los CIOs son dependientes de la modalidad, pero todavía son independientes de la plataforma, de los *toolkits* concretos que se vayan a emplear en su implementación.

- Limbourg propone el modelo CUI (Figura 2.15) como un conjunto de CIOs y sus relaciones. Además distingue entre CIOs gráficos y auditivos según la modalidad.

- A su vez, estos se pueden ser contenedores o componentes individuales. Así, y respetando la nomenclatura anglosajona, hay cuatro tipos de CIO:

    - *Graphical Container* (GC),

    - *Graphical Individual Component* (GIC),

    - *Auditory Container* (AC), y

    - *Auditory Individual Component* (AudIC).

Los CIOs gráficos son descritos con más detalle por su frecuencia y complejidad, frente a los auditivos. (Penichet 2007)

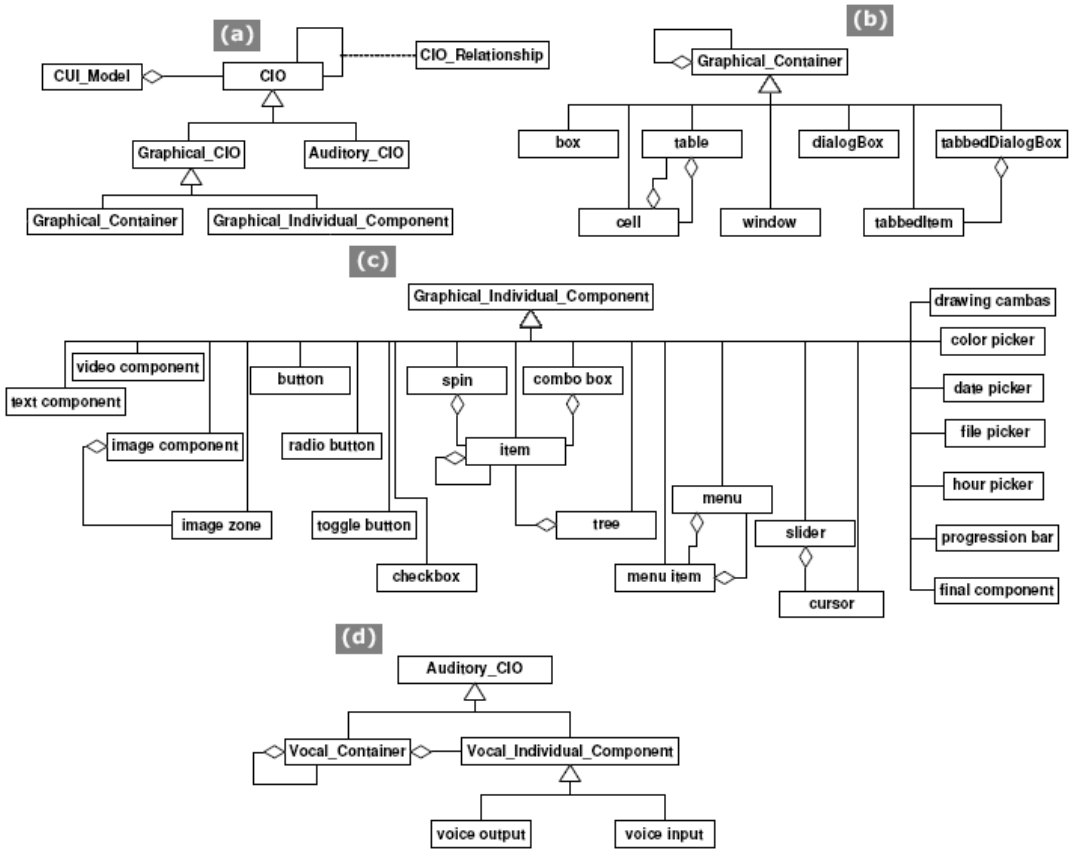


**Figura 2.15 (a) Modelo CUI, (b) CUIs del tipo Graphical Container, (c) CUIs del tipo Graphical Individual Component, (d) CUIs del tipo auditivo**

Los controles de diálogo definidos en modelos de interfaz de usuario abstractos permiten la especificación del flujo de control entre los objetos de interacción concretos. (Montero, 2005).

Las entradas de interfaces de usuario concretas incluyen:

- Perfiles de usuario
- Escenarios
- Modelos de tareas
- Diagramas de secuencia, si son usados
- El modelo principal
- El modelo de la visión: estático y dinámico
- Estilo de interacción y opciones de tecnología de implementación

Las salidas incluyen:

- Ventanas, cuadros de diálogo, y esquemas (*layouts*) ágiles; las técnicas de interacción usadas en estos esquemas (*layouts*); y las acciones del sistema resultante durante y después de usar las técnicas de interacción
- Representaciones de objetos principales, interacción con representaciones de objetos principales incluyendo alguna selección y dinámica retroalimentación
- Menú y esquemas de paleta, acciones del sistema en los elementos de menú y opciones de botón, accesos directos, y facilidades para deshacer

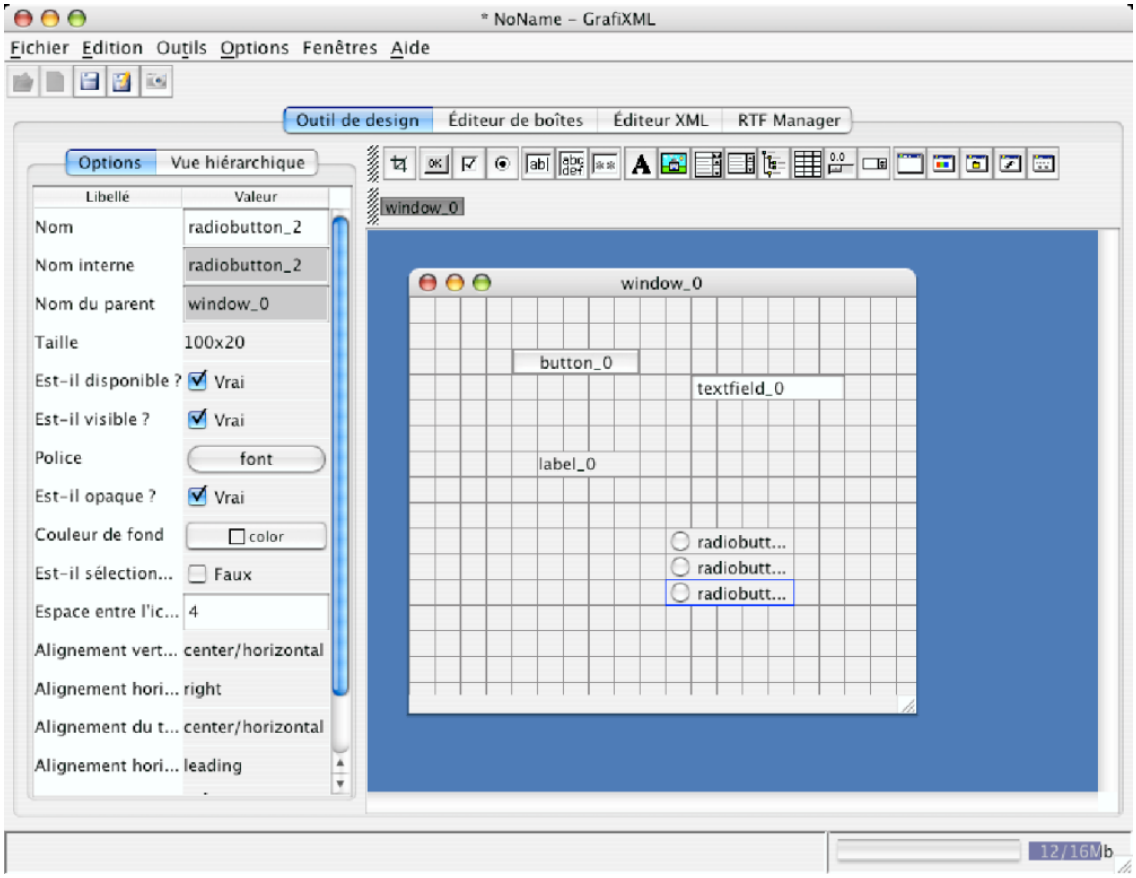


**Figura 2.16 Ejemplo de una Interfaz de Usuario Concreta**

Las salidas también incluyen:

- Facilidades de ayuda
- Documentación de usuario
- Accesorios de entrenamiento

Estas salidas pueden ser representadas usando

- Texto y diagramas

- Bocetos y tablero de historial
- Secuencias de interacción
- Esquemas generados por herramientas
- Prototipos

Un aspecto de las salidas es mostrar cómo las interacciones de los usuarios afectan al estado principal y modelos de visión. (ConcreteUserInterface, 2007)

### 2.2.10.6 Modelo de Relaciones Intermodelo (mapping)

El problema del mapping (Puerta et al., 1999; Limbourg et al., 2004) consiste en la identificación, establecimiento manual o derivación automática de relaciones entre los modelos recogidos en un entorno de desarrollo basado en modelos. En este sentido, son de singular importancia las relaciones que se establecen entre los modelos independientes de la plataforma (usuario, tarea, dominio y presentación abstracta), y los dependientes de la misma (presentación concreta y diálogo). El conjunto de relaciones de mapping da lugar a una serie de transformaciones con las que es posible aspirar al desarrollo automático de interfaces de usuario.

En UsiXML (Limbourg et al., 2004) se ha definido un modelo de mapping en el que se identifican distintas relaciones entre los elementos integrantes de los modelos de dominio, tareas y presentación El modelo de relaciones intermodelo permite almacenar las relaciones que aparecen entre unos modelos y otros. Por ejemplo, se puede representar en qué contenedor se visualizará una tarea concreta. Las relaciones intermodelo pueden ser creadas usando IdealXML[4] (Montero, 2005), herramienta desarrollada por Francisco Montero que permite además crear los modelos de tareas, dominio y abstracto de UsiXML, dicha herramienta se describirá algo más detalladamente en un apartado más adelante de este trabajo.

Las relaciones entre modelos facilitan posibles caminos para abordar la generación automática de interfaces de usuario, permitiendo aumentar la productividad en el proceso de desarrollo desde el punto de vista de la IS, y la generación rápida de prototipos de interfaz de usuario desde una visión IPO. (Montero, 2005)

[4] http://www.dsi.uclm.es/personal/FranciscoMSimarro/idealXML.htm

### 2.2.10.7 Modelo de Transformación

El modelo de transformación en UsiXML describe el proceso de transformación entre los distintos modelos de UsiXML. La idea principal detrás de este modelo es permitir la especificación de la transformación de modelos de UsiXML en otros, de forma que se pueda generar una interfaz de usuario a partir de una serie de modelos base. Para ello en la propuesta usiXML, define un conjunto de reglas que determinan la existencia de unos elementos de especificación en función de la existencia de otros elementos pertenecientes a otros modelos. En usiXML hay definidas 15 reglas de transformación con las que es posible, primero, obtener, a partir de los modelos de dominio y de tareas, el modelo de presentación especificado a nivel abstracto y, en un segundo lugar, a partir de él es posible obtener una especificación concreta de la interfaz de usuario. Una de las novedades del modelo de transformación propuesto por AB-UIDE (*Agent-Based User Interface Development Environment*) (Jaquero, 2005) es la posibilidad de comenzar el proceso de desarrollo por distintos puntos de entrada, siguiendo lo que se ha denominado un desarrollo multi-ruta de interfaces de usuario.

### 2.2.10.8 Modelo de Contexto

El modelo de contexto de usiXML describe las componentes del contexto consideradas habitualmente en el desarrollo de interfaces de usuario, es decir debe capturar cualquier atributo importante del contexto de uso en el que la interfaz va destinada, dicho modelo está a su vez compuesto por otros modelos: El modelo de usuario, que clasifica a los usuarios de la aplicación en diferentes perfiles (estereotipos). El modelo de plataforma, que recoge la información relevante de la combinación de software-hardware que se va a utilizar. El modelo de entorno que define las propiedades interesantes del ambiente físico donde la aplicación está siendo ejecutada. UsiXML ofrece un detallado modelo para la plataforma basado en el estándar definido por el World Wide Web Consortium (W3C) denominado "*Composite Capabilities/Preferente Profiles*" (CC/PP[5]), para la descripción de perfiles de plataformas.

[5] http://www.w3.org/mobile/CCPP/

### 2.2.10.9 Modelo de Recursos

El modelo de recursos permite el almacenamiento de los recursos de forma separada a los elementos que los utilizan, permitiendo de forma sencilla la internacionalización de las aplicaciones.

### 2.2.10.10 Interfaz de usuario final – Final User Interface (FUI)

La interfaz de usuario final (FUI – Final User Interface) representa el código que será ejecutado en la plataforma destino para mostrar la interfaz de usuario. Por lo tanto, esta versión de la interfaz de usuario es dependiente tanto de la plataforma donde se ejecutará la interfaz como de la modalidad que será usada para interactuar con ella. (Jaquero, 2005).

El proceso de reificación de objetos de interacción concreta (CIO) a elementos FUI si se ha de tener en cuenta el sistema operativo, el dispositivo final, lenguaje y la plataforma. Es entonces cuando se seleccionan los elementos físicos tangibles por el usuario para poder concluir el desarrollo del a UI de la aplicación.

La interfaz de usuario final contiene la interfaz de usuario operacional, generada a partir de las capas anteriores: modelo de tareas, interfaz de usuario abstracta e interfaz de usuario concreta. (Romero, 2007).

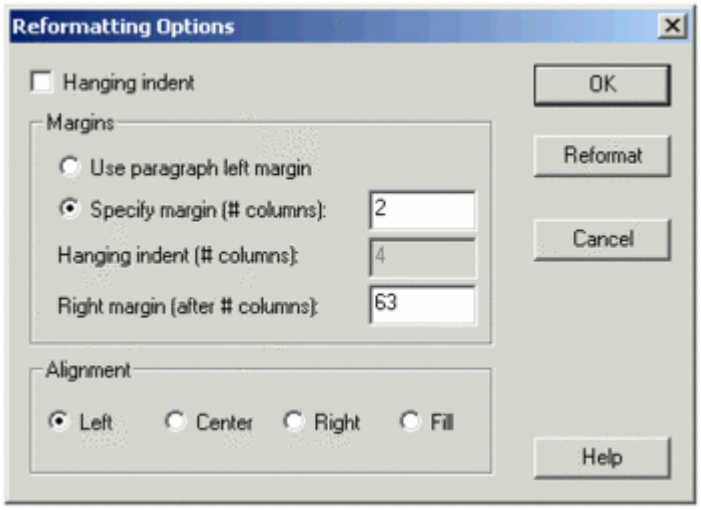


**Figura 2.17 Ejemplo de Interfaz de Usuario Final**

En este apartado se ha permitido el beneplácito de extender el nombre dado al lenguaje asociado a la metodología para hacer referencia a la propuesta metodológica, en un apartado posterior de este trabajo se concluirá todos los aspectos y características de UsiXML pero ya desde un punto de vista de lenguaje XML de marcado que describe la interfaz de usuario desde varias perspectivas. Estás perspectivas ya han sido más que explicadas.

## 2.3 LENGUAJES Y NOTACIONES

En el ámbito de la disciplina IPO, como ya se mencionó anteriormente, no se dispone de un lenguaje unificado, como es UML en Ingeniería del Software, pero si es cierto que la tendencia que en la actualidad sigue IPO es la misma que la IS, ambas disciplinas apuestan por lo declarativo es decir están abandonando o dejando en un segundo plano la utilización de lenguajes de programación. La utilización de lenguajes declarativos para definición de interfaces de usuario proporciona varias ventajas, como la facilidad de aprendizaje y la posibilidad de definir la interfaz de forma independiente de la definición de la lógica de la aplicación y el contenido. De esta manera se permite que diferentes especialistas trabajen independientemente en el desarrollo de la aplicación.

En IPO el Lenguaje de Marcado eXtensible (XML) ha cobrado una relevancia importante. Con él es posible describir a diferentes niveles de abstracción la interfaz de usuario sin dejar de asociar dicha descripción con la funcionalidad que debe acompañar cualquier producto software. XML no está relacionado con ningún lenguaje de programación, sistema operativo o proveedor de software. De hecho, es bastante sencillo producir o hacer uso de XML utilizando diferentes lenguajes de programación. La independencia de la plataforma hace que XML sea muy útil para lograr la interoperabilidad entre diferentes plataformas de programación y sistemas operativos.

XML no es más que un conjunto de reglas para definir etiquetas semánticas que nos organizan un documento en diferentes partes. En este sentido, XML es un metalenguaje que establece la sintaxis utilizada para la elaboración de otros lenguajes de etiquetas estructurados. En XML se usan hojas de estilos, como el lenguaje XSL (*Extensible Stylesheet Language*, lenguaje de hojas de estilos extensible) y CSS (*Cascading Style Sheets*, hojas de estilos en cascada), para mostrar los datos en un explorador. En XML se separan los datos de la presentación y el proceso, de forma que es posible mostrar y procesar los datos según se desee, aplicando distintas aplicaciones y hojas de estilos. (Montero, 2005)

Gracias a estas características ofrecidas por XML, han surgidos varios lenguajes basados XML para su utilización en la especificación y generación de interfaces de usuario. Estos lenguajes podemos clasificarlos en dos grupos: aquellos que son destinados a cubrir la mera descripción de interfaces de interfaces de usuario como es el caso de UIML, XUL, Xforms, y aquellos lenguajes que se pueden llegar considerar propuestas metodológicas ya que no se limitan a describir la interfaz sino que facilitan mecanismos para describir dentro de un ciclo de vida de diseño de interfaces de usuario, dependiendo de los casos, del usuario, sus tareas, su dominio, etc, es el caso de UsiXML o XIML.

### 2.3.1 USIXML

UsiXML (*User Interface eXtensible Markup Language*), es un lenguaje de descripción de interfaces de usuario (*User Interface Description Language*, UIDL) basado en XML. Puede describir interfaces para múltiples contextos de uso, con independencia de instrumento, de plataforma y de modalidad.

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 el UI 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. Es decir, 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.

Para completar el estudio realizado en este trabajo de la propuesta UsiXML y despejar alguna posible duda de lo que puede abarcar y no, se definen a continuación las principales características de UsiXML:

- UsiXML es entendido por no desarrolladores, como analistas, especificadotes, diseñadores, expertos de los factores humanos, jefes de proyecto, programadores novatos,… Por supuesto, UsiXML puede ser usado por desarrolladores con experiencia. Gracias a UsiXML, los no desarrolladores pueden configurar la interfaz de usuario de alguna aplicación interactiva nueva por especificación, describiéndola en UsiXML, sin requerir habilidades de programación usualmente encontradas en lenguajes de marcado (por ejemplo HTML) y lenguajes de programación (por ejemplo Java o C++).

- UsiXML consiste en un Lenguaje de Descripción de Interfaces de Usuario (UIDL), que es un lenguaje declarativo que captura la esencia de lo que las interfaces de usuario son o deben ser independientemente de las características físicas.

- UsiXML describe un alto nivel de abstracción de los elementos constituyentes del interfaz de usuario de la aplicación: controles, contenedores, modalidades, técnicas de interacción,… UsiXML permite mezclar conjuntos de herramientas de desarrollo de aplicaciones interactivas. Una interfaz de usuario de alguna aplicación de UsiXML adaptable se ejecuta en todos los conjuntos de herramientas que lo implementan: compiladores e intérpretes.

- UsiXML es independiente de los dispositivos: una interfaz de usuario puede ser descrita de una manera que despoje autonomía con respecto a los dispositivos usados en las interacciones tales como ratón, pantalla, teclado, sistemas de reconocimiento de voz,…

- UsiXML es independiente de la plataforma: una interfaz de usuario puede ser descrita de una manera que despoje autonomía con respecto a varias plataformas de computación, tales como teléfonos móviles, Pocket PC, Tabled PC, ordenadores portátiles, ordenadores de escritorio,… En caso de necesidad, una referencia a una modalidad particular puede ser incorporada.

- UsiXML es independiente de la modalidad: una interfaz de usuario puede ser descrito de manera que despoje autonomía con respecto a alguna modalidad de interacción tal como interacción gráfica, interacción vocal, interacción 3D.

- UsiXML permite rehusar elementos previamente descritos en anteriores interfaces de usuario para componer una interfaz de usuario en nuevas aplicaciones.

La cobertura de UsiXML en términos de objetivos de interfaces de usuario es grande. Sin embargo, no soporta la cobertura de todas las características de todos los tipos de interfaces de usuario. Por tanto:

- UsiXML no quiere insertar todavía otros lenguajes para implementación de interfaces de usuario. En lugar de eso, propone la integración de alguno de estos formatos: CHTML, WML, HTML, XHTML, VoiceXML, VRML, Java, C++,....

- UsiXML no describe detalles de bajo nivel de los elementos involucrados en las distintas modalidades, tales como atributos de sistemas operativos, eventos y primitivas.

- UsiXML no quiere soportar todos los atributos, eventos, y primitivas de todos los objetos de interacción existentes próximos a todos los conjuntos de herramientas. En su lugar, tiene la intención de soportar subconjuntos comunes de ellos. (UsiXML, 2007)

### 2.3.2 XIML *(eXtensible Interface Markup Language)*

XIML (*eXtensible Interface Markup Language*)[6] (Puerta et al., 2001), es un lenguaje de especificación basado en XML es un lenguaje para la creación de interfaces de usuario independientes de plataforma en el que se hace un especial hincapié en desacoplar la representación gráfica de la interacción.

El lenguaje XIML se presenta como una propuesta de representación común para datos interactivos, ya que contempla los principales requisitos que debe cumplir un lenguaje de este tipo: soporta funcionalidad a lo largo del ciclo de vida completo del desarrollo de interfaces de usuario, es capaz de relacionar los elementos abstractos y concretos de una interfaz, permite a los sistemas basados en conocimiento tratar los datos capturados.

Los elementos del lenguaje y sus relaciones se muestran en la siguiente figura.

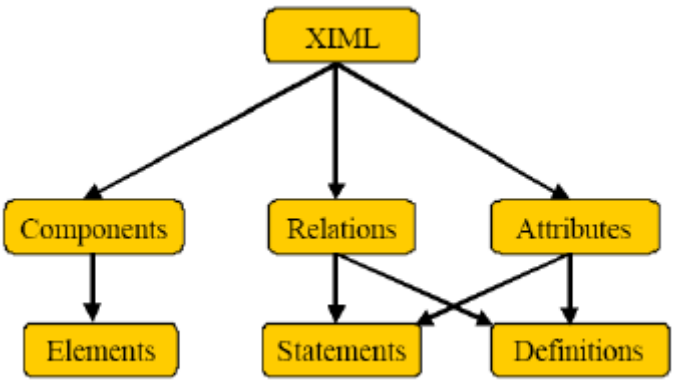


**Figura 2.18 Elementos de XIML**

[6] http://www.ximl.org

En XIML se definen 5 componentes principales: tarea, dominio, usuario, diálogo, y presentación. Estos componentes agrupan a elementos más básicos. Los tres primeros componentes se pueden considerar contextuales y abstractos, mientras que los dos últimos son implementaciones concretas. Visto con más detalle:

- Tarea. Representa los procesos de negocio en los que interviene un usuario humano y/o las tareas que puede realizar el usuario. Este componente define una descomposición jerárquica de tareas y subtareas que definen el *workflow* de estas tareas y los datos que utilizan.

- Dominio. Es una colección organizada de instancias y clases organizadas en una jerarquía. Esta jerarquía es conceptualmente como una ontología pero mucho más sencilla. Los objetos se definen mediante pares atributo-valor y sólo pueden ser objetos que el usuario visualiza o modifica. Estos objetos pueden ser simples o complejos. Ejemplos de estos objetos pueden ser Fecha, Mapa, Contrato.

- Usuario. Este elemento define una jerarquía de usuarios. Un nodo de esta jerarquía puede representar a un grupo de usuarios o a un único usuario.

- Presentación. Este elemento define una jerarquía de elementos de interacción que se le mostrarán al usuario. No se especifica el aspecto final de estos elementos.

- Diálogo. Es la implementación de las tareas. Este elemento define una colección estructurada de elementos que determinan las acciones de interacción que puede realizar del usuario. Ejemplos pueden ser: *Click*, RespuestaDeVoz, etc. En este elemento también se especifica la navegación de la aplicación.

Estos componentes son el resultado de numerosas investigaciones de modelos de interfaces de usuario, de hecho se observa una cierta semejanza con la propuesta del conjunto de herramientas Mobi-D, puesto surgió como evolución de la propuesta MIMIC (Puerta, 1997). Aunque se pueden definir otros componentes, u otros elementos dentro de los componentes, la experiencia ha demostrado que se pueden acabar englobando en los cinco definidos. En cualquier caso, si se desea añadir nuevos elementos, el lenguaje está diseñado para soportarlos.

En las relaciones de XIML se engloba toda la información acerca de las relaciones entre los elementos de los componentes. Por ejemplo, permite ligar

elementos de dominio y elementos visuales en relaciones del estilo “Date es visualizado con presentaciónA o presentaciónB”. Las relaciones pueden ser sentencias o definiciones. Las definiciones ligan clases, mientras que las sentencias ligan instancias.

Los atributos son propiedades de los elementos a los que se les puede asignar un valor. Estos valores pueden ser tipos básicos o instancias de otros elementos. Al igual que las relaciones, estos atributos pueden ser sentencias o definiciones.

Otras características que presenta XIML es la separación entre definición del interfaz y su visualización se consigue mediante la creación de “presentaciones”. En XIML se pueden definir varias presentaciones, cada una para cada tipo de dispositivo (Web, PDA, teléfonos móviles, etc.), e indicar en cada una cómo se muestran los elementos visuales. Cuando el sistema detecte el tipo de dispositivo que usa el usuario, usará la presentación adecuada.

Soporta una modificación dinámica del aspecto del interfaz de usuario para adaptarse a los cambios de las características del dispositivo de visualización. Por ejemplo, si se reduce el tamaño de la pantalla del dispositivo de visualización, los elementos gráficos podrán adaptarse al tamaño.

XIML permite múltiples formas de visualización de los elementos gráficos, es decir permiten que el usuario elija es aspecto más conveniente. Esto supone que empresas externas pueden ofrecer múltiples visualizaciones que pueden usar los usuarios sin tener que descargar ningún tipo de software. (Rico, 2004)

### 2.3.3 XUL *(XML-based User-Interface Language)*

El lenguaje XUL[7], es el lenguaje multi-plataforma que permite especificar interfaz de usuario usando para ello una sintaxis basada en XML con un nivel de abstracción muy cercano al nivel concreto de interfaces de usuario propuesto para UsiXML, específicamente diseñado para aplicaciones en red como navegadores, programas de correo. Está integrado dentro de la arquitectura de Mozilla[8] para el desarrollo de Interfaces Web multiplataforma, dentro de la cuál se hace uso de tecnologías W3C ya existentes.

[7] http://www.xulplanet.com
[8] http://www.mozilla.org

La arquitectura se basa en el uso de paquetes que pueden ser abordados desde una perspectiva abstracta o concreta. Los paquetes se componen de contenido, apariencia, comportamiento, localización, plataforma. En cada uno de ellos se hace uso de diferentes tecnologías.

Su ejecución deberá realizarse bajo el entorno de Mozilla y en los Sistemas Operativos en los cuáles Mozilla se ejecute, pro ejemplo en cualquier navegador basado en el motor *Gecko* del proyecto Mozilla. Entre dichos navegadores se encuentran: Netscape 6 en adelante, Mozilla 6 en adelante, Camino, Firefox, IBM Web Browser o Galeon, entre otros. Ello permite que una interfaz de usuario diseñada en XUL pueda ser ejecutada en casi cualquier plataforma, desde un PC con el sistema operativo Microsoft Windows (Netscape, Mozilla, Firefox), con el sistema operativo Linux (Netscape, Mozilla, Firefox, Galeón), con el sistema operativo OS/2 de IBM (IBM Web Browser) a un Apple ejecutando Mac Os (Camino), una máquina Sun con el sistema operativo Solaris (Netscape), e incluso dispositivos móviles ejecutando la distribución de Linux Familiar y la versión para dicha distribución de Mozilla: Minimo.

Tiene la capacidad de separar la interfaz de la lógica de la aplicación, lo cuál facilita el mantenimiento de la interfaz sin necesidad de alterar la lógica de la aplicación.

El lenguaje XUL se haya englobado dentro de un conjunto de facilidades que se agrupan dentro del XPToolKit31. Dentro de XPToolKit se proporcionan las herramientas, basadas en los estándares más extendidos para la creación de aplicaciones basadas en tecnología Web (JavaScript, CSS, SOAP, DOM, RDF, etc), necesarias para la creación de aplicaciones independientes, pero que se ejecutan utilizando el motor del navegador Mozilla.

La arquitectura de XPToolKit permite la especificación de la interfaz de usuario utilizando el lenguaje basado en XML XUL, definiendo el aspecto de los componentes a través de la utilización de estilos, normalmente aplicándolo mediante hojas de estilo en cascada (CSS32). El comportamiento de los elementos es creado usando JavaScript33, el cual permite acceder a la estructura de la interfaz de usuario representada en forma de árbol DOM34 (*Document Object Model*), tal y como ocurre habitualmente en el desarrollo de aplicaciones Web, y manipular la interfaz. Para aquellas ocasiones en que sea necesario invocar funcionalidades no escritas en JavaScript XPToolkit suministra una serie de objetos XPCOM. Estos objetos son una versión multiplataforma de los conocidos objetos COM de Microsoft y proporcionan independencia respecto del sistema operativo concreto. Ahora bien, para poder utilizar estos objetos desde JavaScript, es necesario un lenguaje intermedio de conexión:

XPConnect (Cross-Platform Connect) y otro que permita definir API's adecuadas para los objetos XPCOM. Este último es el XPIDL (*Cross-Platform Interface Definition Language*), el cual permite describir la signatura de los objetos de una manera independiente de la plataforma (de una forma análoga a lo que sucede con IDL en el modelo de CORBA). (Jaquero, 2005)

En Java existen gran cantidad de APIs para la generación de GUIs basadas en XML, muchas de ellas basadas en XUL, ejemplos de ellas son: Luxor XUL, XWT, Thinlets o SwingML. XUL se utiliza en IDEAS (Lozano, 2001) para describir el aspecto final de la interfaz que sería creada por aplicación de la metodología.

El código XUL puede ser visualizado en cualquier navegador compatible con los navegadores basados en el motor desarrollado por la fundación Mozilla. La Figura 2.19 muestra la interfaz de usuario visualizada en el navegador Mozilla.

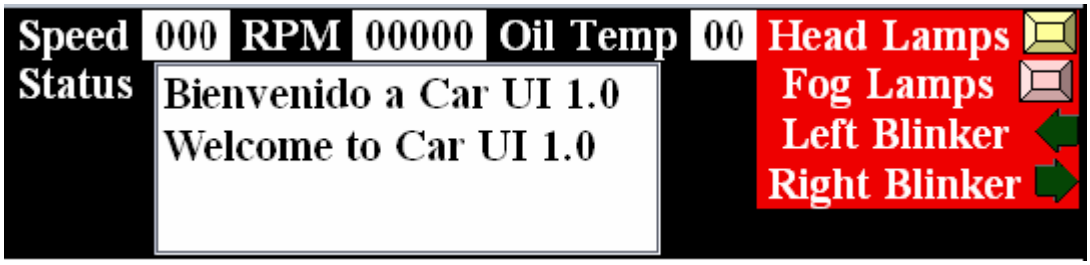


**Figura 2.19 Interfaz de usuario visualizada en el navegador Mozilla (Jaquero, 2005)**

### 2.3.4 UIML *(User Interface Markup Language)*

El lenguaje UIML[9], es un sencillo lenguaje basado en XML que permite realizar una descripción declarativa de la interfaz de usuario de un modo independiente del dispositivo. Uno de los objetivos de UIML es reducir el tiempo que los desarrolladores invierten en describir interfaces para múltiples familias de dispositivos.

Para describir una interfaz de usuario en UIML se debe realizar, por un lado la definición de la interfaz genérica, y por otro un documento UIML que representa el estilo de presentación apropiado para el dispositivo en el cuál la interfaz de usuario se va a ejecutar. De este modo, una misma aplicación solamente necesitará un único documento UIML de especificación válido para cualquier dispositivo y un documento de estilo propio para cada dispositivo.

[9] http://www.uiml.org

Podemos decir que UIML se podría utilizar en los niveles 2 y 3 del marco de desarrollo de interfaces de usuario definido en Cameleon y parcialmente en el nivel 1. La definición de la interfaz se enmarcaría en los niveles de tareas-dominio y AUI y la definición del estilo en el nivel CUI. Estas tareas incluso podrían ser llevadas a cabo por equipos de desarrollo diferentes.

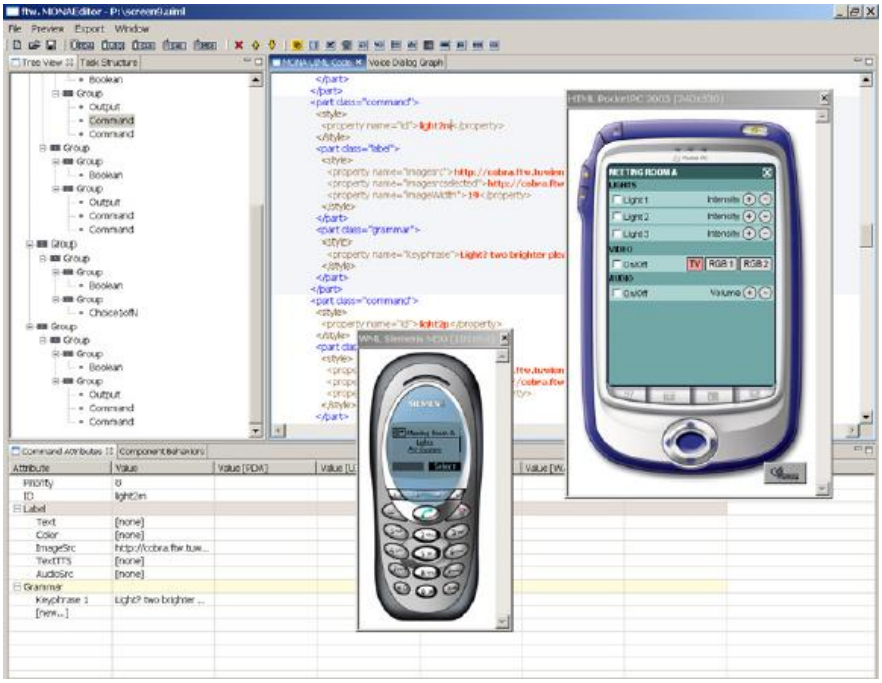

**Figura 2.20 Entorno que facilita la utilización del lenguaje UIML**

La flexibilidad para la selección de dispositivos finales es limitada, ya que aunque la parte correspondiente a la interfaz genérica puede mantenerse independientemente de que aumente el número de dispositivos finales, no sucede lo mismo con la parte que mapea a los dispositivos específicos, que crece cuando este número aumenta, aumentando también el coste de mantenimiento.

El lenguaje UIML permite la traducción automática al lenguaje utilizado por el dispositivo final. El proceso de traducción se realiza en el propio dispositivo o en el servidor de la interfaz dependiendo del dispositivo del que se trate.

### 2.3.5 XFORMS (the nesXt generation of web FORMS)

XForms[10] es una propuesta del consorcio W3C para la especificación de formularios para la Web que puedan ser usados en una amplia variedad de plataformas. Su versión 1.0 ha llegado a ser recientemente una recomendación de W3C.

El desarrollo de XForms pretende cubrir las limitaciones de los formularios HTML tradicionales que no disponen de una separación entre el propósito y la presentación de un formulario. XForms se compone de secciones separadas que describen lo que el formulario hace y cómo se presenta.

La especificación de XForms está compuesta de dos módulos, el Modelo XForms y el soporte de la interfaz de usuario. El modelo XForms representa las diferentes partes del formulario desde el punto de vista de los datos genéricos. La interfaz de usuario XForms define la parte del formulario que representa los elementos de la presentación, proporcionando flexibilidad y control sobre la misma a través de las diferentes presentaciones que pueden estar relacionadas con un mismo modelo XForms, dependiendo de la plataforma final.

XForms se podría englobar fundamentalmente en el nivel 2, ya que es un lenguaje orientado a la implementación de formularios Web en diferentes dispositivos pero no a la descripción de la interfaz en un nivel alto de abstracción.

### 2.3.6 OpenLASZLO

OpenLaszlo[11] es una plataforma basada en XML, de software libre, para construir aplicaciones RIA (*Rich Internet Applications*) son aquellas aplicaciones que combinan las funcionalidades de una aplicación de escritorio con las funcionalidades de una aplicación Web. Son mucho más ricas en cuanto a presentación, interacción con el usuario y mensajería. Este tipo de aplicaciones siempre se ejecutan en el entorno de un navegador. Soluciones actuales de RIA son por ejemplo applets, flash, activeX.

[10] http://www.w3.org/MarkUp/Forms/
[11] http://www.laszlosystem.com

La arquitectura de esta plataforma es similar a la de XUL[12] de Mozilla (Ginda, 2004), XAML[13] de Microsoft o Flex[14] de Macromedia. El entorno de ejecución es Macromedia Flash dentro de un navegador.

La plataforma Laszlo está formada por dos piezas: un lenguaje declarativo basado en XML (con JavasScript empotrado) y un servidor de aplicaciones (llamado LPS) que se llama Servidor de Presentación.

Lsz es el lenguaje asociado a la propuesta openlaszlo. Lsz es un lenguaje orientado a objetos basado en etiquetas que usa XML y sintaxis JavaScript para crear la capa de presentación de las aplicaciones de Internet ricas. Normalmente estas aplicaciones son compiladas por un compilador OpenLaszlo[15]. Las aplicaciones se pueden servir como ficheros ya compilados o bien se pueden compilar y servir dinámicamente con el servidor de presentación.

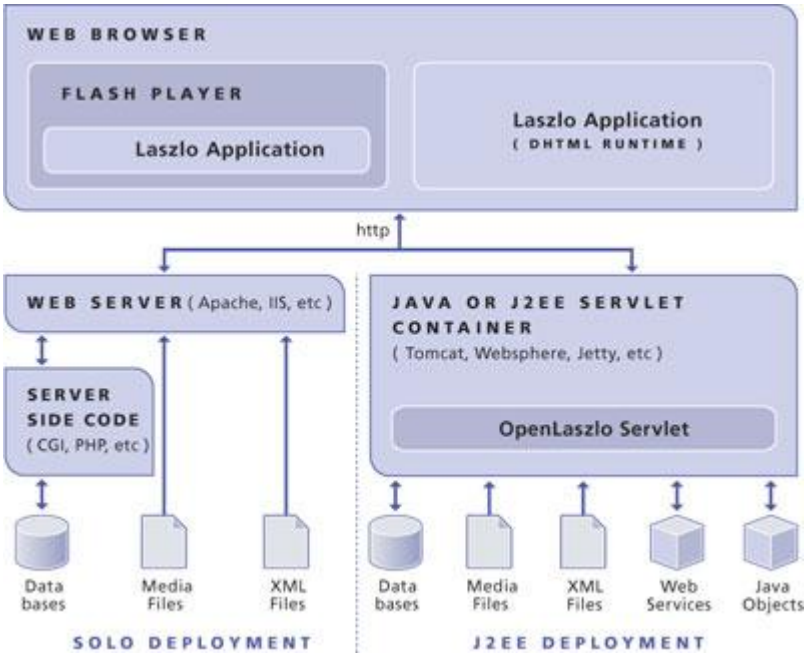


**Figura 2.21 Arquitectura de OpenLaszlo**

En la parte del cliente, una aplicación OpenLaszlo se ejecuta como película Flash (con extensión swf) dentro de un plugin Flash en el entorno de un navegador, usando el modo seguro *sandbox* del reproductor de Flash, el cual garantiza que no se pueda acceder al sistema de ficheros del cliente o en general al entorno nativo del cliente.

[12] http://www.mozilla.org/projects/xul/
[13] http://www.xaml.org/index.html
[14] http://www.adobe.com/products/flex/
[15] http://www.openlaslo.org

Esta plataforma fue fundada en el año 2000 por la compañía Laszlo System y a mediados del 2004 la hizo de fuente abierta.

Una característica valiosa de Laszlo como ya se ha mencionado se basa en el uso de etiquetas extensible XML, que permite a los desarrolladores avanzados crear nuevos objetos interactivos enlazados a datos o personalizar los existentes poniéndolos a disposición de otros desarrolladores creando nuevas etiquetas. Los desarrolladores pueden definir sus propias etiquetas y extender las ya creadas en Laszlo utilizando programación orientada a objetos.

El sistema dinámico de animación y fuerza de Laszlo permite a los elementos en pantalla ajustarse automáticamente a la actividad del usuario, dotando a las aplicaciones de un sentido simpático y fluido.

## 2.4 HERRAMIENTAS

Los MB-UIDE proporcionan herramientas de modelado que sirven a los desarrolladores construir modelos o descripciones declarativas de la interfaz. El principal objetivo de estas herramientas es ocultar a los desarrolladores la sintaxis de los lenguajes de modelado y proporcionarles una interfaz que les permita especificar adecuadamente el modelo de interfaz. Muchas de ellas disponen de editores, interpretes, generadores y otras aplicaciones útiles para llevar a cabo tareas relacionadas con la elaboración, manipulación o generación de modelos. Se ha desarrollado una amplia gama de herramientas de modelado, a menudo especializadas en diferentes niveles del modelo. Gran parte de estas de herramientas proceden del ámbito académico e incluso se encuentran en desarrollo.

En este apartado se presentarán aquellas que a priori parecen más representativas destinadas a dar soporte a la especificación de los distintos modelos. Se verán con más o menos detalle en función de la motivación que impulsó la generación de este proyecto.

### 2.4.1 El editor CTTE

CTTE (*ConcurTaskTrees Environment*)[16] es una herramienta para la especificación de tareas utilizando la notación CTT, en otras palabras es una aplicación

[16] http://giove.cnuce.cnr.it/ctte.html

para la creación de modelos CTT, su validación y simulación. Dicha simulación permite evaluar la usabilidad del modelo propuesto. La Figura 2.23 muestra el mismo modelo de tareas que el mostrado en la Figura 2.22, durante el proceso de simulación.

La herramienta CTTE también permite obtener una especificación LOTOS equivalente al modelo de tareas especificado e incluso ciertas tareas de chequeo de modelos como la comprobación de alcanzabilidad, para comprobar que todas las tareas pueden ser alcanzadas en alguna ocasión.

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.

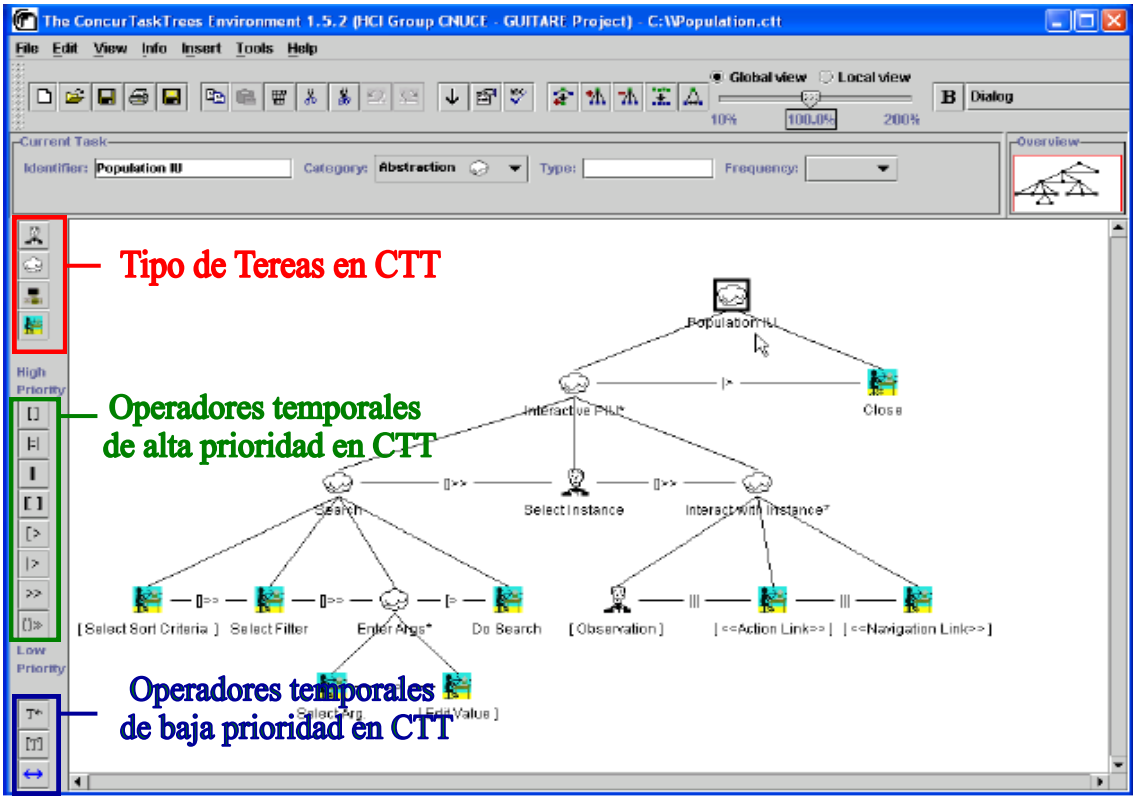


**Figura 2.22 Herramienta CTTE**

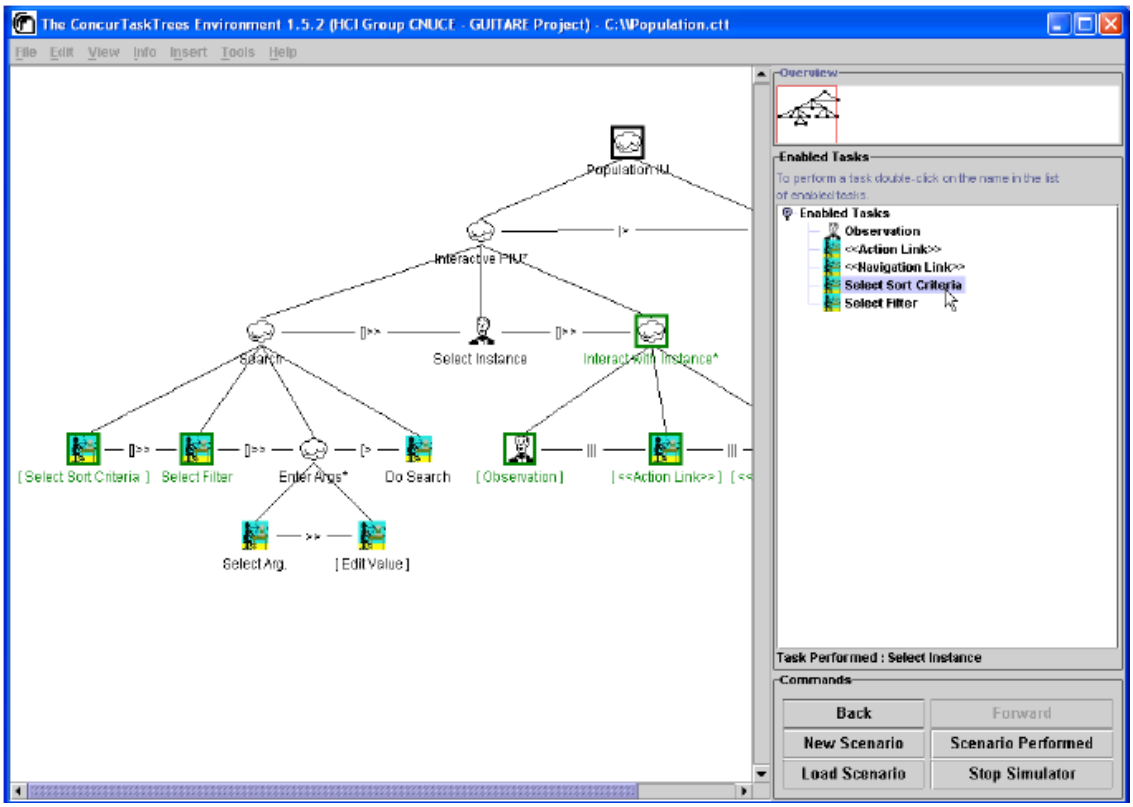


**Figura 2.23 Animación del modelo de tareas presentado en la Figura 2.22**

En la Figura 2.22 se remarca como CTTE utiliza cuatro de los cinco tipos diferentes de tareas que hay en la notación CTT en función del actor o actores que la lleven a cabo. También hace disponible al usuario de una serie de operadores para la descripción de relaciones temporales entre las tareas especificadas.

La herramienta ha sido utilizada en diversos proyectos para varias áreas de aplicación por parte de muchos grupos en el ámbito académico aunque han sido poco probados en la construcción de sistemas reales. Se ha implementado en Java 1.3 beta y se ha comprobado que es útil para aclarar las cuestiones y diseño soportar el análisis y evaluación de las opciones del diseño. Es una herramienta libre disponible en http://giove.cnuce.cnr.it/ctte.html.

En el futuro se dedicará a la ampliación de la herramienta para obtener un entorno para el diseño de aplicaciones multi-contexto que puedan ser accedidas desde una amplia variedad de dispositivos (que van desde los teléfonos celulares a las grandes y flat-screens) y entornos. También se integrar esta herramienta en entornos UML (*Rational Rose*), a fin de ofrecer más apoyo en el diseño interactivo de sistemas software. (Molina, 2003)

### 2.4.2 IdealXML

IDEALXML[17] es un entorno de desarrollo de interfaces de usuario basado en modelos y en usiXML. La herramienta ofrece dos facilidades por un lado es un editor/gestor de patrones de cualquier tipo (interacción, colaboración, diseño, etc.) que almacena utilizando el formato PLML[18] (*Pattern Language Markup Language*) (Fincher, 2003) y, por otro, es un entorno de desarrollo que facilita al ingeniero experiencia basada en patrones para la especificación de productos software en los que la usabilidad está considerada.

Es una herramienta desarrollada 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. Una parte importante de la descripción la constituyen los ejemplos asociados a cada patrón donde puede verse de forma gráfica una posible implementación de la solución que el patrón documenta.

Por lo tanto IdealXML sirve como entorno de especificación de un producto software en el que puede considerarse la experiencia modelada y asociada a los patrones disponibles en el repositorio asociado en cada momento al entorno. Los patrones se consideran modelos en IdealXML, pudiendo ser enriquecidos y utilizados para la especificación de distintos productos software.

En la Figura 2.24 se muestran las diferentes secciones que describen un patrón en IDEALXML, las mismas aparecen recogidas en la ventana titulada *Patterns properties*. Las diferentes flechas que parten de algunas de las secciones de dicha ventana muestran, a su vez, los diálogos asociados a la edición del correspondiente valor afín a cada una de ellas. Una de las secciones cuya descripción no aparece reflejada en la figura mostrada arriba es la sección *diagram*, con ella es posible asociar diferentes modelos, concretamente los modelos de dominio, tareas, presentación y mapping, a cada patrón.

[17] http://www.dsi.uclm.es/personal/FranciscoMSimarro/idealXML.htm
[18] http://www.cs.kent.ac.uk/people/staff/saf/patterns/plml.html

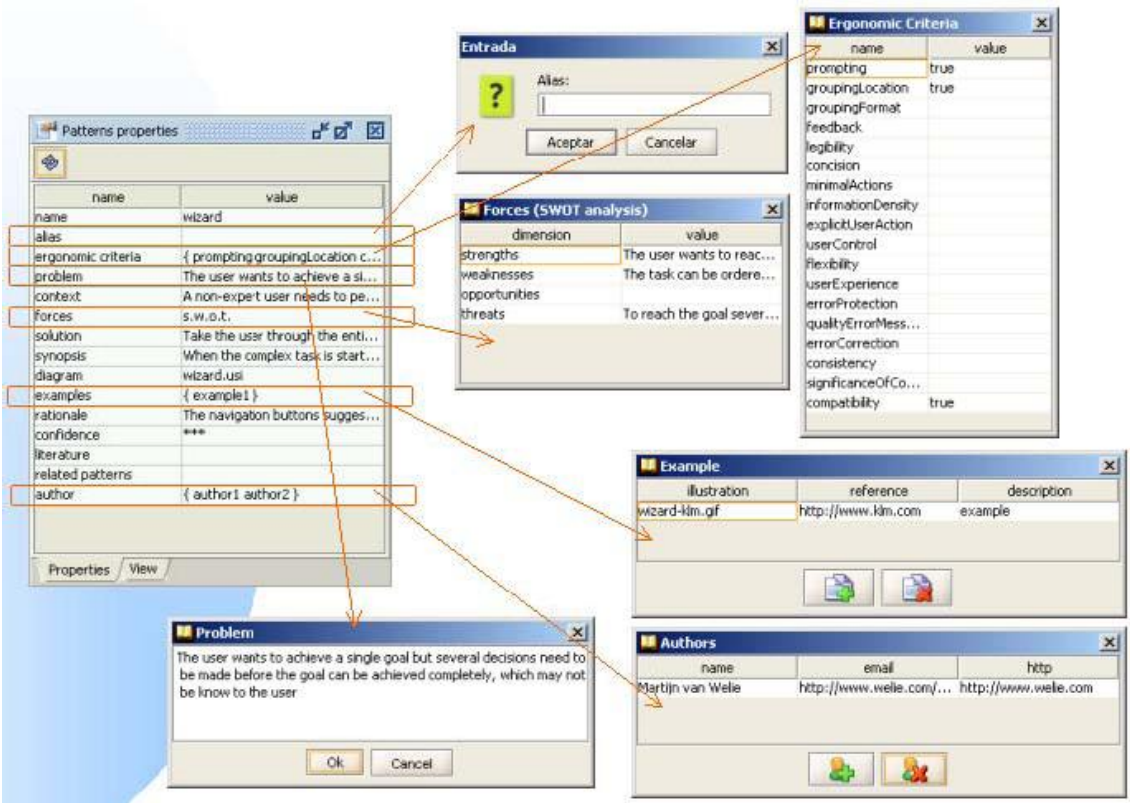


**Figura 2.24 Secciones que describen un patrón en IDEALXML**

Los diferentes editores implementados en IDEALXML para permitir la especificación de los modelos asociados se muestran en la Figura 2.25, junto con cada editor aparecen reflejados los posibles elementos de modelado que pueden utilizarse en cada uno de ellos y su representación gráfica.

Los patrones editados y almacenados utilizando IDEALXML se guardan utilizando usiXML. El diseño y posterior implementación de IDEALXML se ha hecho contemplando la posibilidad de que otras notaciones pudieran utilizarse para describir un patrón de interacción. Además, en su diseño e implementación se han utilizado diferentes patrones de diseño.

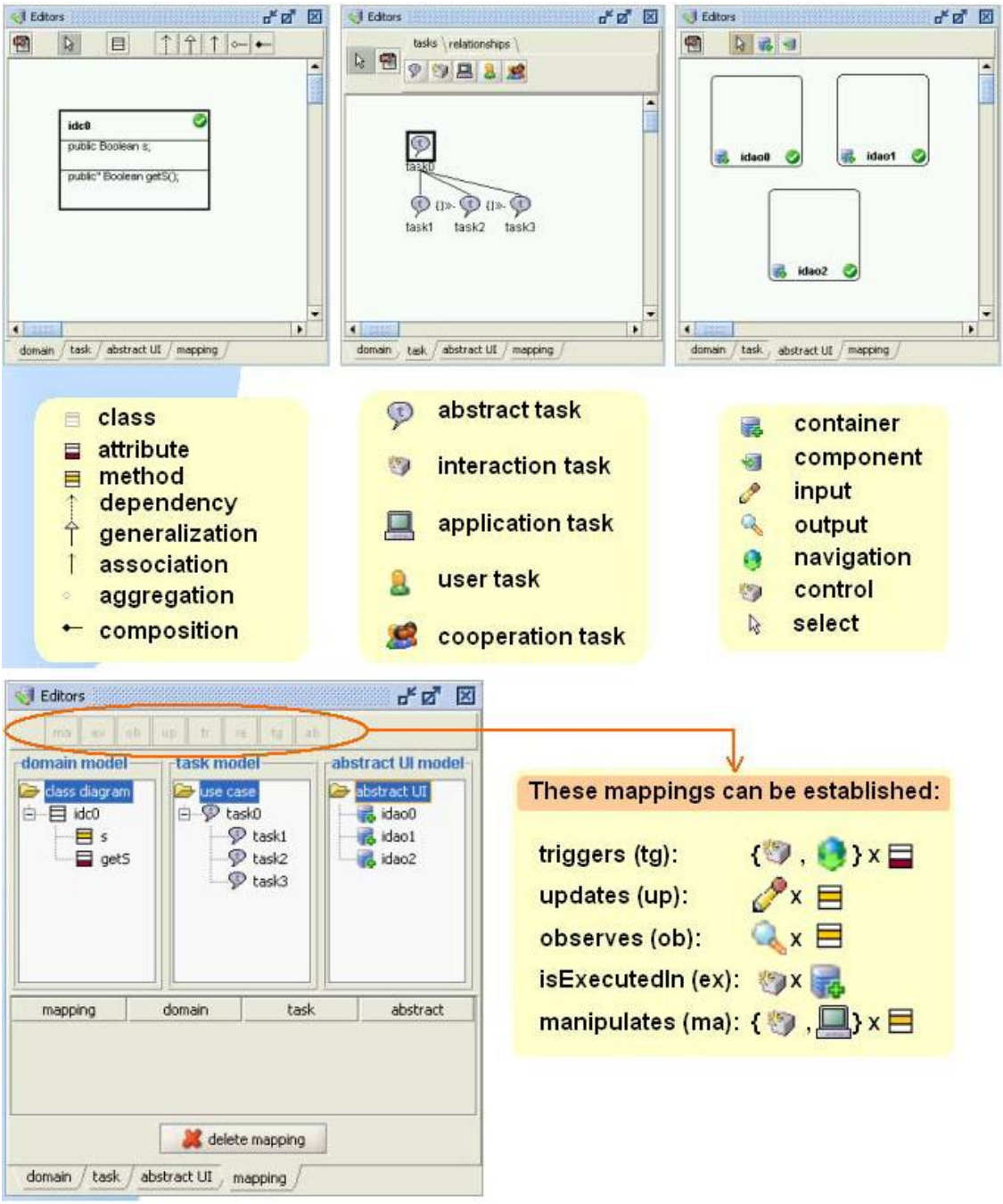


**Figura 2.25 Modelos, y elementos de modelado que pueden se utilizados en IDEALXML**

IDEALXML incorpora su propio editor CTT en el que, al igual que en el original, se dispone de los diferentes tipos de tareas y de las relaciones temporales identificadas en la propuesta original de Paternò. Las diferencias entre la herramienta vista antes CTTE e IDEALXML, en lo que al uso de la notación CTT se refiere, están en el aspecto visual que ofrecen los iconos asociados a las tareas y en la posibilidad, más digna de mención, de almacenar la especificación realizada utilizando el lenguaje usiXML.

Para concluir IdealXML es una herramienta que permite la gestión, manipulación y transferencia de experiencia modelada en forma de patrones y está disponible para su descarga en un sitio Web propio (http://www.dsi.uclm.es/personal/FranciscoMSimarro/idealXML.htm) .La principal aportación de la mencionada herramienta es que convierte a los patrones (interacción, colaboración, diseño), en elementos abiertos y no en cajas negras propias y únicamente manipulables internamente por su propietario. Otro sitio Web donde tiene especial cabida la herramienta IDEALXML es en UsiXML (http://www.usixml.org/index.php?mod=pages&id=15). Además de incluir una referencia al entorno en dicho sitio Web, también es considerada plenamente integrada en el marco propuesto con usiXML debido a la funcionalidad ofrecida por ella para editar cualquiera de los modelos relacionados con el dominio, las tareas, la presentación abstracta y el mapping entre ellos (Montero, 2005).

### 2.4.3 GrafiXML

GrafiXML[19] es una herramienta gráfica que dibuja interfaces de usuario para múltiples plataformas de computación. Es considerado como el editor mas elaborado de UsiXML para el modelado de interfaces de usuario concreta (CUI), y modelado de contexto. Se puede guardar una interfaz de usuario en varios formatos como Java o XHTML, HTML, e incluso XUL, gracias a una serie de plugins, pero la principal u original forma de guardarlas es en UsiXML.

La herramienta GrafiXML es similar a alguna interfaz de usuario de algún editor de construcción, excepto que manipula más propiedades de objetos que otros editores y guarda la interfaz de usuario en UsiXML en lugar de en otro particular formato de código. De esta manera, es posible mantener múltiples versiones localizadas de la misma interfaz de usuario y atribuirlas a contextos particulares de uso. (UsiXML, 2007)

[19] http://www.usixml.org

A continuación se muestra una imagen del entorno GrafiXML:

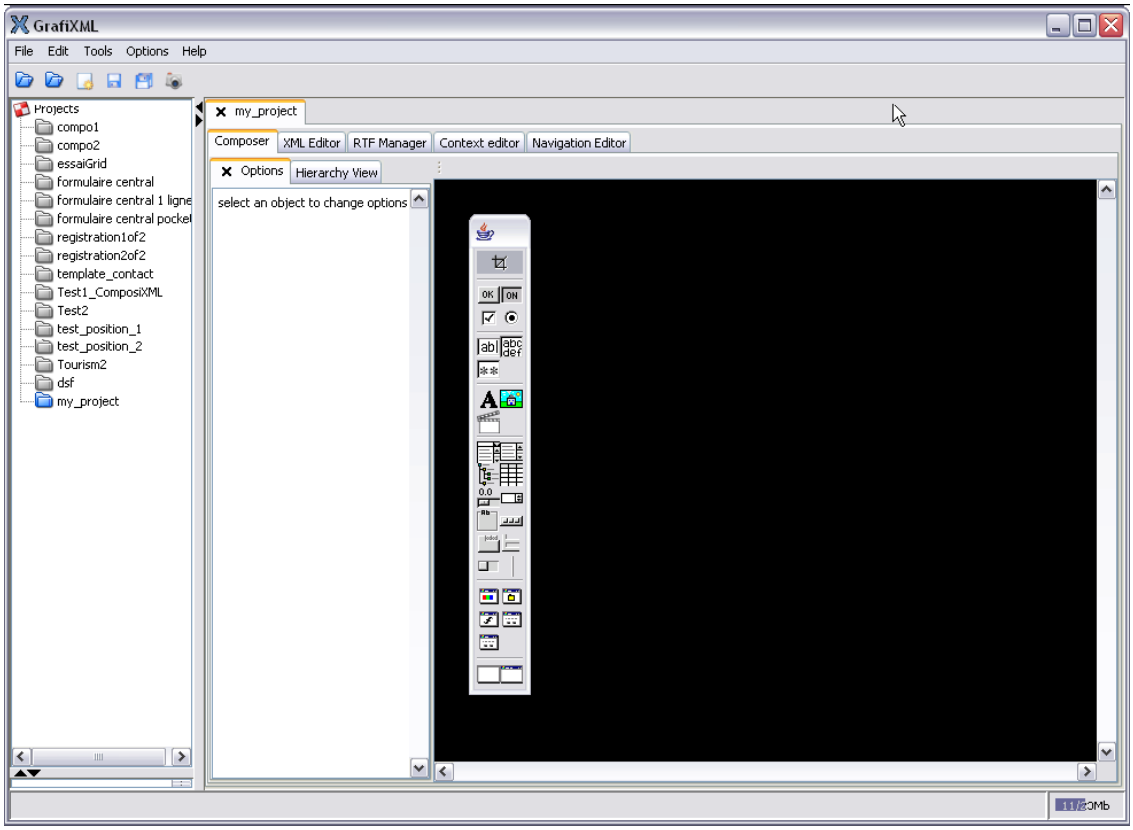


**Figura 2.26 Herramienta GrafiXML**

En la imagen se observa que hay dos secciones separadas, a la izquierda hay una ventana contiene un árbol con los proyectos, mientras que en la sección derecha está todos los elementos para editar el proyecto actual, recibe el nombre de área de trabajo. En esta área de trabajo hay una barra de herramientas de componentes para generar la interfaz gráfica.

Antes de añadir cualquier componente sobre la ventana gráfica decir la interfaz generada por el usuario, la herramienta solicita que el usuario se decida por un layout para la distribución de los componentes en la ventana. Los *layouts* que ofrece son un subconjunto de los layout que ofrece cualquier entorno de desarrollo en Java y son los siguientes:

- *FlowBox*: Posiciona un componente a continuación de otro. El usuario puede decidir donde se coloca el primer componente según la alienación.

- *GridBox*: Divide la ventana en regiones estableciendo el usuario el numero de columnas y filas, dejando que introducir un único componente por celda.

- *GridBagBox*: Divide la ventana en casillas de un cierto tamaño fijo, de tal manera que el usuario pueda situar los componentes ocupando estos casillas completas.

- *BorderBox*: Divide la ventana en un grupo de cinco secciones (*boxes*) colocados arriba, abajo, derecha izquierda y en medio de la ventana. El usuario puede seleccionar los *boxes* que desea establecer, no es obligatorio añadir los cinco *boxes*. A su vez cada *box* contendrá su propio layout, pudiendo seleccionar para aquella sección cualquier división de las comentadas.

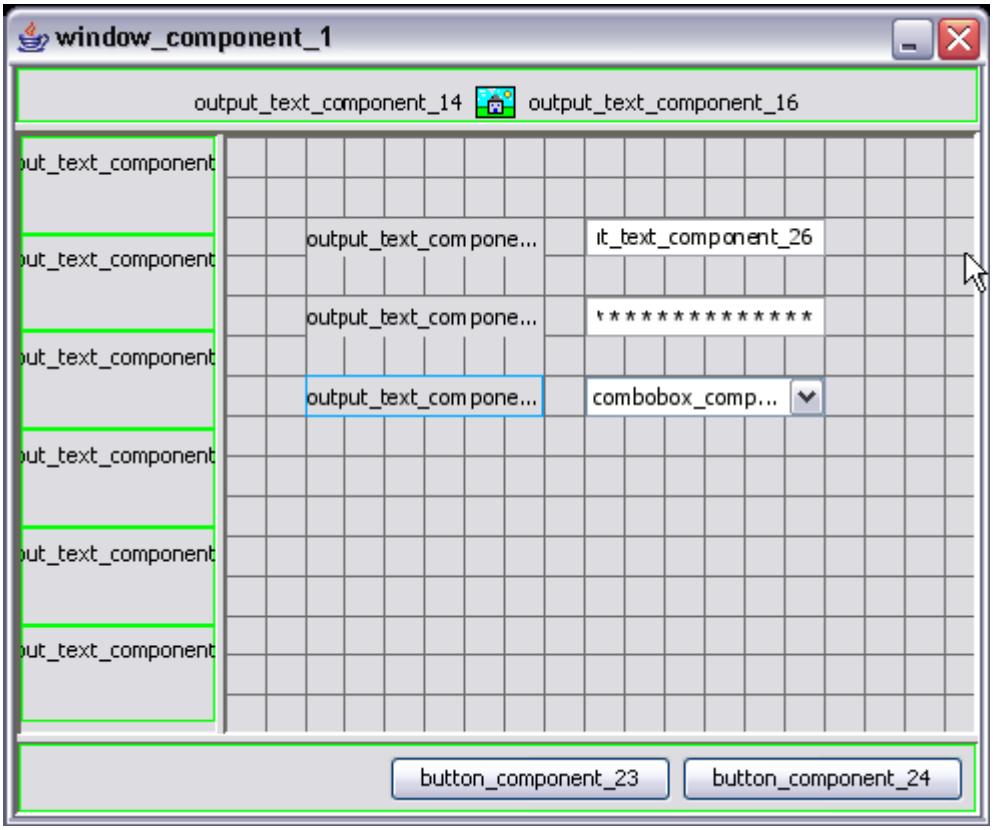


**Figura 2.27 Diseño de una ventana con layout BorderBox**

Una vez establecido el *layout*, el usuario ya puede insertar los diferentes componentes sobre la ventana, pero el *layout* no se podrá cambiar, se hace hincapié en ello ya que sería un factor que la herramienta desarrollada en este proyecto innova respecto a GrafiXML.

GrafiXML permite añadir contenido a los componentes, por ejemplo un texto o una imagen, te permite además elegir a que idioma irá asociado el contenido, en otras palabras el contenido puede asociarse a una de las diferentes plataformas donde se vaya

a utilizar la interfaz final. Este contenido será almacenado en el modelo de recursos, si el usuario exporta la GUI a otro lenguaje como Java o XHTML, varios ficheros serán salvados para cada plataforma (*language*) establecida.

GrafiXML tiene la opción de mostrar una visión previa de la interfaz diseñada por el usuario, en esta visión previa sería una muestra de cómo sería el aspecto de la interfaz final, deja al usuario la opción de escoger el lenguaje que muestre en la visión previa.

El usuario puede modificar los atributos de los componentes, aunque la configuración de varios de éstos no se hacen presentes en la visión previa, de hecho en la ventana de diseño no se hace presente la configuración de ningún atributo de los componentes incluidos los contenidos establecidos.

Se ha mencionado que la principal aportación de GrafiXML era guardar la interfaz en UsiXML, por ello contiene un editor XML que permite al usuario mostrar y editar texto UsiXML correspondiente.

```
                <outputText id="output_text_component_2"
                    name="output_text_component_2"
                    content="/uiModel/resourceModel/cioRef[@cioId='output_text_component_2']/
                    defaultContent="Recherche multi-marques"
                    isVisible="true" isEnabled="true"
                    bgColor="#0099ff" isBold="true" textColor="#000000"/>
            </constraint>
            <constraint gridx="1" gridy="1" gridwidth="5"
                gridheight="1" weightx="1.0" weighty="1.0"
                fill="both" insets="0,0,0,0">
                <checkBox id="checkbox_component_3"
```

**Figura 2.28 XML Editor: Especificación UsiXML**

GrafiXML también incluye un editor de contexto en el cual se puede añadir, eliminar o editar contextos para la aplicación del usuario.

GrafiXML trabaja con los *plugins*. Hay diferentes tipos de *plugins*.

- *Plugin* de exportación: añade la función de exportar la especificación de UsiXML a otro lenguaje.

- *Plugin* de importación: añade la función de importar la descripción de una interfaz de usuario escrito en un idioma distinto UsiXML. Algunos de los *plugins* de importación están en desarrollo.

- *Plugin* de menú: agregar una entrada en un menú para añadir algunas funcionalidades a GrafiXML.

- *Plugins* de *popmenu*: añadir entradas en el menú (en los widgets de botón derecho del ratón) para añadir algunas funcionalidades a este componente.

- *Plugin* de proyecto: permite al usuario iniciar un proyecto con algunos valores por defecto o para eliminar funcionalidades de GrafiXML.

Los *plugins* disponibles para GrafiXML son:

- Para exportar a fuentes XHTML, XUL y Java ™.
- *ComposiXML*: Todo componente compuesto o individual de una interfaz gráfica de usuario se somete a una serie de operaciones para componer una nueva interfaz de componentes existentes y para descomponer una de las interfaces existentes en piezas pequeñas para que pueden utilizarse a su vez para otra interfaz.
- Asesor de usabilidad: Mejorar la usabilidad y accesibilidad de las interfaces de usuario
- *Graceful Degradation*: *Plugins* utilizados para transformar la interfaz de usuario de una plataforma a otra.

Como conclusión GrafiXML es una aplicación que permite editar especificaciones concretas de interfaz de usuario y del modelo de contexto almacenándolo en formato UsiXML, 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 *plugins*.

Es una herramienta en desarrollo, cada pocos meses en la página web www.usixml.org hay una nueva versión de esta aplicación, y ello también explica el porqué de la carencia de funcionalidad de algunos aspectos que en principio debía de ofrecer. Una de estas carencias ya ha sido explicada y consiste en que no se aprecia ningún cambio en los componentes cuando sus atributos son configurados de diferentes maneras cuando éstos son mostrados en la visión previa que se supone que debe de dar una apariencia lo más real a lo que luego será la interfaz gráfica final. Otra carencia es la no disponibilidad de añadir contenido a un componente de video, cuando se observa que es necesaria. Grafixml carece la posibilidad de cambiar de layout a mitad de desarrollo, además hay cierta limitación en el posicionamiento de los componentes en la ventana con los *layouts* disponibles. Es un programa algo inestable en el aspecto que

suele fallar a menudo, y algunos componentes funcionan en algunos *layouts*. Además es un programa algo complejo o poco intuitivo a la hora de generar la interfaz gráfica.

Ahora bien, hay que comprender que no es una herramienta que vaya destinada a mejorar aspectos de usabilidad o a innovar aspectos visuales en la construcción de interfaces, sino que es una herramienta que va enfocada a editar especificaciones concretas de interfaz de usuario dentro del marco propuesto por usiXML. Aunque no vendrían mal algunas aportaciones dirigidas a los intereses del diseñador gráfico, sin desviarse de la finalidad por la que está siendo construida GrafiXML.

## 2.5 TABLA COMPARATIVA DE LAS PROPUESTAS METODOLÓGICAS

Los MB-UIDEs se han sucedido históricamente desde finales de los ochenta hasta nuestros días. La pervivencia de los mismos ha venido sustentada desde la IS y desde la IPO gracias a que aseguran productividad en el desarrollo de interfaces de usuario y posibilitan la puesta en práctica de metodologías centradas en el usuario.

En este trabajo se han hablado de diferentes apuestas académicas que ofrecen mecanismos que permiten el modelado de Interfaces de Usuario. No todas las propuestas presentan los mismos modelos, ni utilizan las mismas notaciones para su especificación. Cada propuesta posee sus propias herramientas visuales donde el usuario hace uso de una notación que permite la especificación de los distintos modelos.

La idea en todos los entornos propuestos es la misma: elaboración de modelos conceptuales en los que se recogen las características estáticas y dinámicas de una aplicación, incluida la parte de la interfaz de usuario, e interpretación, compilación o transformación de dichos modelos hasta obtener una interfaz de usuario que esté ligada, o que pueda ligarse, con la funcionalidad de la aplicación. A continuación se muestra una tabla que resume las características de las diferentes propuestas metodológicas, entre ellas las herramientas, que hay mayormente en el ámbito académico, que han sido creadas para soportar dichos métodos.

**Tabla 2.1 Tabla comparativa de las diferentes metodologías**

| | Modelos | Notación | Soporte Software | Traza | Considera Usabilidad | Considera Experiencia | Generación de IU automática | Ciclo Completo de IU Desarrollo | Referencia |
|---|---|---|---|---|---|---|---|---|---|
| Trident | Tareas Dominio | Diag. Ent/Rel | SEGUIA | X | X | Únicamente modelo Presentación | v | X | (Vanderdonckt, 95) |
| MOBI- D | T , Do, Usuario, Dialogo, Presentación | MIMIC (precursor XIML) | MOBI-D UTEL TIMM, MOBILE | X | v | Únicamente modelo Presentación | v (no completa) | v | (Puerta, 97) |
| OWID | UML | OO | OlivaNova | X | v | X | X | X | (Roberts et al., 98) |
| IDEAS | T, Do, Di, P | OASIS, XUL | | v | v | Únicamente modelo Presentación | X | v | (Lozano, 01) |
| JUST-UI | T, Do, U, Di, P | OO, (OASIS) | Just-UI/Visio | X | X | v | v | v | (Molina, 03) |
| OO-H | Do, Nav, P | OO, (OASIS) | Visual WADE | X | X | v | v | v | (Cachero, 03) |
| TERESA | T, P | CTT, AUI, LOTOS | CTTE | v | X | X | v (no completa) | X | (Mori et al., 04) |
| UsiXML | T, Do, AUI, CUI, Contexto, Recursos, Trans. | UsiXML | GrafiXML, IdealXML, KnowiXML ReversXML TransformiXML | v | v (gracias a la aportación de IdealXML) | v (gracias a la aportación de IdealXML) | v | v | (Limbourg et al., 04) |

En este capítulo no se han recogido todas las metodologías que se han desarrollado para la contribución del modelado de Interfaces de Usuario. Pero si es cierto que todas estas propuestas es un subconjunto de todas aquellas importantes o de gran relevancia en el perímetro de los Mb-UIDE.

La tabla muestra un resumen de las nociones básicas de cada propuesta, cuales son las herramientas o software de soporte de los mecanismos que proponen cada una de ellas. Gracias a estas herramientas ponen en práctica las conceptos teóricos y verifican la utilidad de dichas investigaciones académicas.

La tabla también refleja las notaciones utilizadas para especificar los diferentes modelos, se observa que en la mayoría de las ocasiones hacen uso de notaciones propias, un problema de estandarización reflejado en este trabajo desde comienzos del mismo. Se puede prestar atención que conforme fueron apareciendo notaciones relacionadas con el análisis y diseño orientado a objetos (OOA/OOD, OSE, OMT, etc.), y se extendieron su uso, en el desarrollo de interfaces de usuario. Por se buscó un estándar internacional como es el Lenguaje Unificado de Modelado (UML) el ejemplo de esta iniciativa para especificación y desarrollo de interfaces de usuario en ese sentido son OVID (Roberts et al., 1998), Wisdom (Nunes, 2001), UMLi (Silva et al., 2003), e IDEAS (Lozano, 2001). Las insuficiencias más destacables que se le achacan a UML desde el terreno de IPO, y que impiden su acogida plena por parte de esta disciplina, es que UML no incorporación de técnicas de Diseño Centrado en el Usuario de forma paralela con la elaboración de un producto software. Debido a esto se buscaron otras tendencias de lenguajes declarativos de modelado basados en XML (XUL (Mozilla, 2003), XIML (Puerta et al., 2001), UsiXML (Limbourg et al., 2004)).

La columna traza hace referencia si las propuestas mantiene la trazabilidad a lo largo del proceso de desarrollo de la interfaz., ya que no se trata de la especificación modelos inconexos, sino que están relacionados unos con otros y existe una trazabilidad entre ellos, de manera que un cambio en uno podría suponer un cambio en algún otro, lo que esta dentro de la lógica de la especificación del sistema desde diferentes puntos de vista. Esa trazabilidad asegura la coherencia de dicha especificación.

La columna usabilidad se refiere a si la propuesta metodológica considera la usabilidad de la interfaz que se pretende desarrollar de cara al usuario final y si está considerada desde los primeros pasos en el proceso de desarrollo. Está cualidad de un producto software está poco contemplado por estas metodologías, y en caso de aparecer el proceso de elaboración ya está avanzado y se dispone de una versión preliminar del producto software que se pretende desarrollar. El prototipado y evaluación de prototipos son las principales armas utilizadas, ya se esté en el ámbito de la IS, ya de

propuestas presentadas desde el terreno de la IPO. Una de las metodologías cuya herramienta ha sido vista en detalle en este proyecto, IdealXML, otorga a UsiXML mecanismos para desarrollar una IU donde esté presente la usabilidad y la experiencia del desarrollador.

La automatización disminuye el trabajo en el desarrollo del software, y por lo tanto también en el ámbito de las interfaces de usuario, se ha visto la posibilidad de generar algunos diagramas e incluso la propia interfaz para distintos dispositivos o lenguajes a partir de unos modelos comunes o incluso desde la propia etapa de elicitación de requisitos de forma automática o semiautomática. Se observa que hay metodologías que van consiguiendo este propósito gracias a que establecen operaciones o un conjunto de pasos de transformación entre los diferentes modelos. Esto en concreto ya se ha mencionado en este trabajo, gracias a la identificación de *mappings* entre los distintos modelos y de la entrada en escena de las correspondientes transformaciones de modelos, habituales en un entorno de desarrollo basado en modelos. Algunas de dichas transformaciones están disponibles en UsiXML.

La penúltima columna describe si las metodologías cubren todo el ciclo de vida del desarrollo de una interfaz de usuario o bien solo cubren algunas fases, suele ser un punto a favor aquellas que proporcionan una solución completa al diseño e implementación de interfaces de usuario que cubra todo el ciclo de desarrollo de la interfaz de usuario, que abarca desde las fases de adquisición de requisitos a la fase de implementación y despliegue. Ahora bien mejor resultaría si además hay una herramienta que pueda integrar todas las fases, y no una herramienta por cada etapa donde no haya integración ni un repositorio ya que estos son requerimientos mínimos para que una herramienta CASE sea eficaz y útil, ya sea en el desarrollo de software como en el de interfaces.

## 2.6 ANÁLISIS Y CONCLUSIONES

Las interfaces de usuario son el vínculo entre el usuario y el programa que se ejecuta en la computadora, han evolucionado desde interfaces textuales donde cada orden debía ser escrita usando el teclado, hasta interfaces de usuario gráficas de gran complejidad. La interfaz de usuario es el aspecto más importante de cualquier aplicación. Una aplicación sin una interfaz cómoda e ineficaz, impide que los usuarios saquen el máximo rendimiento de un programa, un programa muy poderoso con una interfaz pobremente elaborada tiene poco valor para un usuario no experto.

Los requisitos que el usuario impone a la interfaz de usuario de los sistemas software actualmente hacen especialmente importante la participación de especialistas en Interacción Hombre-Máquina en el proceso de desarrollo.

La disciplina IPO se ocupa del análisis y diseño de interfaces entre el hombre y la máquina, se nutre de metodologías, lenguajes y herramientas. En los últimos años se ha realizado un gran esfuerzo en la investigación de métodos que permitan la inclusión del diseño de la interfaz de usuario dentro de un proceso de desarrollo basado en modelos. Intentando obtener beneficios tales como la automatización de la generación de la interfaz de usuario, la generación de dicha interfaz para distintos dispositivos o lenguajes a partir de unos modelos comunes o la mejora de las propiedades de usabilidad del sistema.

Los entornos de desarrollo de interfaces de usuario basado en modelos (Mb-UID) propone una serie de modelos de distinta temática y diferente nivel de abstracción, que incluyen, por ejemplo, modelos de dominio, de presentación, de tareas, de diálogo, de usuario y de la plataforma. El diseñador utiliza notaciones de mayor nivel de abstracción para especificar estos modelos o descripciones declarativas de la interfaz. Debido a ello el Lenguaje de Marcado eXtensible (XML) ha cobrado una relevancia importante puesto que con él es posible describir a diferentes niveles de abstracción la interfaz de usuario sin dejar de asociar dicha descripción con la funcionalidad que debe acompañar cualquier producto software.

Han surgido muchos lenguajes basados en XML para su utilización en la descripción y generación de interfaces de usuario como los mencionados en este trabajo UIML, XUL, XForms, y aquellos lenguajes que han sido desarrollados no sólo para describir la interfaz sino también para dar soporte al desarrollo basado en modelos como es el caso de XIML, UsiXML.

UsiXML (Limbourg, 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. Basado en el proyecto europeo Cameleon (Calgary et al., 2003) que propone cuatro niveles de abstracción como marco de desarrollo de interfaces de usuario: Tareas & Conceptos, UI abstracta (AUI), UI Concreta (CUI) y UI Final (FUI). CUI es un modelo de interfaz de usuario que permite la especificación de la apariencia y el comportamiento de una interfaz de usuario con elementos que pueden ser percibidos por los usuarios, estos elementos a este nivel reciben el nombre de CIO (Objeto de Interacción Concreta).

Existe una herramienta gráfica GrafiXML en el cual el diseñador puede definir un modelo CUI generando una interfaz gráfica como cualquier editor de construcción, con algunas limitaciones pequeñas ya mencionadas en este capítulo, y almacenarlo en lenguaje UsiXML, de manera que una interfaz de usuario puede especificarse independiente del lenguaje de programación de alto nivel (HTML, VRML o Java). Consiguiendo un nivel de abstracción que ayuda al objetivo de reutilización de desarrollos de interfaces anteriores que hacen posible el desarrollo de una nueva interfaz de usuario de forma sistemática, disminuyendo el trabajo lo que supone obtención de beneficios económicos, nunca olvidando los intereses del usuario final con el cometido de obtener software de calidad.

En este proyecto tiene la intención de contribuir a la especificación declarativa del modelo de interfaz concreta (CUI) incluido en UsiXML. Se ha desarrollado una herramienta software capaz de especificar interfaces de usuario de forma gráfica y permitir gestionar dichas especificaciones utilizando el lenguaje UsiXML**.** Esta aplicación incorpora novedades importantes respecto a programas con los que pueda lograrse una funcionalidad parecida como es el caso de GrafiXML. En toda fase desarrollo de esta herramienta ha estado presente el factor de usabilidad, de manera que el desarrollador de interfaces, para el cuál va destinado esta herramienta, satisfaga sus necesidades de diseñar una interfaz a nivel concreto de la manera más intuitiva y eficaz en el menor tiempo posible.

En el capítulo siguiente se describirá el proceso de desarrollo de la aplicación, y posteriormente habrá una descripción extensa de la herramienta. En el capitulo 4 se mostrarán dos ejemplos donde se demostrará la potencia, utilidad y la aportación a la generación de interfaces de usuario que ofrece la herramienta desarrollada en este proyecto bautizada con el nombre de EgiuXML.

# Capítulo 3: EGIUXML: Detalles de implementación

Este capítulo está organizado en dos partes. En la primera parte del capítulo se recoge una descripción de la herramienta desarrollada desde un punto de vista ingenieríl y utilizando los diagramas y recursos que el proyectando dispone al haber cursado asignaturas relacionadas con Ingeniería del Software. En este sentido, los distintos apartados considerados en esta sección serán: Identificación de objetivos, requisitos funcionales y no funcionales, fase de análisis de requisitos y diseño, y fase de implementación. En la segunda parte del capítulo se describirá en detalle la herramienta implementada.

## 3.1 DESCRIPCIÓN DEL SISTEMA

**EgiuXML** (**E**ditor **g**ráfico de **i**nterfaces de **u**suario a nivel concreto utilizando Usi**XML**) **es una herramienta nueva desarrollada en este proyecto, que permite especificar interfaces de usuario de forma gráfica y gestionar dichas especificaciones utilizando el lenguaje UsiXML.**

EgiuXML es una herramienta gráfica que permite al usuario dibujar interfaces de usuario a nivel concreto utilizando elementos *Graphical Container* (GC) y elementos *Graphical Individual Component* (GIC), más adelante se verán cuales son los elementos de interacción disponibles en esta herramienta, y permite transformar la interfaz editada gráficamente en un modelo de interfaz concreta con sólo pulsar el botón correspondiente y almacenar la especificación en un fichero con extensión ".uxml" que sigue el estándar impuesto por UsiXML**.** Los ficheros generados .uxml siempre podrán volver a ser abiertos por esta aplicación para continuar construyendo gráficamente la interfaz de usuario.

El usuario puede crear una interfaz de usuario concreta (CUI) a su medida, pudiendo configurar los elementos CIO con las propiedades establecidas en UsiXML que como ya se explicó en el capítulo 2 son propiedades comunes establecidas en la mayoría de las herramientas de diseño de ventanas de interfaz.

Una característica importante de EgiuXML que se debe tener en cuenta desde el principio es que esta herramienta cuando almacena la especificación en un fichero, no solo almacena el modelo CUI sino también el modelo de recursos donde estarán

presentes algunos valores de las propiedades de los componentes y un modelo de contexto definido por defecto.

Durante el desarrollo de la herramienta siempre se ha buscado que ésta sea entendida, aprendida, útil y atractiva para el usuario. Además que pueda contribuir en el paradigma de diseño basado en modelos intentando atajar las dificultades que afloran durante el diseño de una interfaz de usuario y a su vez que pueda aportar nuevos beneficios: semiautomatización de la creación de la interfaz de usuario, reutilización de las especificaciones a nivel concretas almacenadas en ficheros es decir reutilización de la experiencia, así como incorporar ventajas encontradas en cualquier editor de construcción de ámbito empresarial, etc.

El proceso de desarrollo comienza con la etapa de adquisición de requisitos, guiada por los casos de uso identificados, donde se capturan los requisitos tanto funcionales como no funcionales para la futura aplicación. En la etapa de análisis de requisitos y diseño se estudian los requisitos descritos en la etapa anterior y se realizan los diagramas de clase. Además debe quedar claro como van a interaccionar las diferentes partes de la aplicación, la realización de diagramas de secuencia aclararán dichas interacciones. Finalmente, la etapa de implementación genera el código de la aplicación que será ejecutado dentro de la arquitectura.

### 3.1.1 Descripción de Objetivos del Sistema

A continuación se van a describir claramente cuál es el principal y único objetivo de esta aplicación y sobre el que se ha focalizado todo el esfuerzo. El objetivo es: **“Gestionar especificaciones a nivel concreto de las Interfaces de Usuario gráficas generadas”**. Se muestra la tabla con la información más relevante del objetivo citado.

**Tabla 3. 1 Objetivo 1 del sistema**

| | |
|---|---|
| **OBJ–01** | Gestionar especificaciones a nivel concreto de las Interfaces de Usuario gráficas generadas |
| **Versión** | 1.0 |
| **Autor** | Arturo García Nuño |
| **Descripción** | El sistema debe permitir especificar gráficamente interfaces de usuario Concreta, transformar la interfaz editada en un modelo de interfaz concreta siguiendo el estándar impuesto por UsiXML y almacenar la especificación en un fichero. En la especificación debe tener además el modelo de recursos correspondiente con la interfaz editada. |
| **Importancia** | Vital |
| **Urgencia** | Inmediata |
| **Estado** | Construido |
| **Estabilidad** | Media |
| **Comentarios** | Ninguno |

Una vez ya comprendido el objetivo que se desea conseguir, en el siguiente subapartado se darán comienzo las fases de desarrollo anteriormente citadas con el fin de lograr alcanzar los objetivos descritos.

### 3.1.2 Fase de adquisición de requisitos

Una vez identificados los objetivos del sistema se va realizar la adquisición de requisitos tan importante y esencial para un correcto diseño de la aplicación. Una adquisición deficiente de los requisitos conducirá al diseño de sistemas incorrectos. Unos requisitos erróneos detectados en las últimas fases del desarrollo introducirán un aumento importante en los costes y los plazos de entrega.

La elicitación de requisitos es la actividad que se considera como el primer paso en un proceso de ingeniería de requisitos trata los objetivos del mundo real, las funcionalidades y las restricciones de los sistemas software. También trata las relaciones entre estos factores para precisar las especificaciones del comportamiento del sistema, y su evolución a lo largo del tiempo y en distintas familias de software.

A continuación se identifican los requisitos del sistema.

#### 3.1.2.1 Identificación de Requisitos de Información

Los requisitos de información son aquellos que describen qué información relevante para el cliente deberá gestionar y almacenar el sistema software a desarrollar así como qué restricciones o reglas de negocio debe cumplir dicha información. (UML, 2007)

En esta aplicación se ha identificado un requisito de información ya que tendremos que almacenar o gestionar las propias especificaciones de los elementos CIO que finalmente se vuelcan a fichero.

**Tabla 3. 2 Requisito de Información 1**

| | |
|---|---|
| **IRQ–01** | Información sobre la especificación de la Interfaz de Usuario Concreta |
| **Versión** | 1.0 |
| **Autor** | Arturo García Nuño |
| **Objetivos asociados** | OBJ-01. Gestionar especificaciones a nivel concreto de las Interfaces de Usuario gráficas generadas |
| **Requisitos asociados** | --- |
| **Descripción** | El sistema deberá almacenar o gestionar las propias especificaciones de los elementos CIO de la interfaz de usuario concreta generada, según las configuraciones realizadas por el usuario. Todas las especificaciones de la interfaz diseñada deberán ser volcadas en un fichero. |
| **Importancia** | Vital |
| **Urgencia** | Inmediata |
| **Estado** | Construido |
| **Estabilidad** | Alta |
| **Comentarios** | Ninguno |

#### 3.1.2.2 Identificación de Requisitos Funciones: Casos de Uso del Sistema

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.

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*?".

Los objetivos del usuario cuando se utiliza la aplicación son descritos mediante la utilización de los 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.

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.

Un caso de uso esta muy relacionado con lo que pudiera ser considerado un escenario en el sistema, esto es, lo que ocurre cuando alguien interactúa con el sistema.

Este tipo de diagramas facilita la comunicación con el cliente, debido a su sencillez. Además, ayudan a determinar nuevos requisitos del sistema conforme es analizado y diseñado el sistema. (UML, 2007)

En este subapartado la funcionalidad que debe ofrecer el sistema al usuario debe quedar clara con el diagrama de casos de uso, siempre con el propósito de lograr desarrollar una herramienta que cumpla los objetivos para los cuales ha sido desarrollada. A un nivel de abstracción alto el siguiente diagrama de casos de uso muestra que hay un actor "Usuario" que interactúa con el sistema con cuatro fines generales.

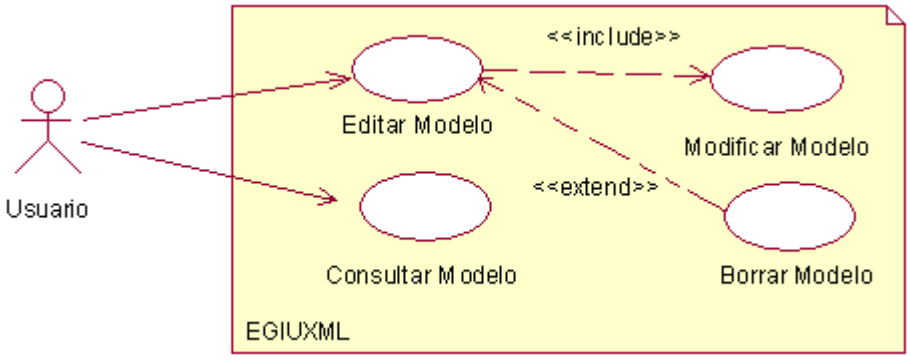


**Figura 3.1 Diagrama de casos de uso del sistema**

Como se puede observar el diagrama anterior se compone de un único actor que interactúa con el sistema, denominado "Usuario", y por cuatro casos de uso

denominados: “Editar modelo”, “Consultar modelo”, “Modificar modelo”, este caso de uso cuyo comportamiento se ha querido destacar aunque está incluido en UC-01 Editar Modelo, y “Borrar Modelo” que es un caso de uso adicional a UC-01. En las tablas se describe el actor y los casos de uso de la Figura 3.1.

A continuación se define el actor “Usuario” utilizado en el diagrama de casos de uso anterior:

**Tabla 3.3 Actor 1: Usuario**

| | |
|---|---|
| **ACT–01** | Usuario |
| **Versión** | 1.0 |
| **Autor** | Arturo García Nuño |
| **Descripción** | Este actor representa a cualquier usuario que interactúe con la aplicación de abstracción creada. |
| **Comentarios** | Este usuario posiblemente sea un programador o diseñador de interfaces de usuario, o bien un usuario con unos pequeños conocimientos en el ámbito del desarrollo de software, no requerido por la dificultad de la aplicación pero si para comprender los conceptos en los que se basa la herramienta |

A continuación se definen los casos de uso del diagrama anterior.

**Tabla 3.4 Caso de uso 1: Editar modelo**

| | | |
|---|---|---|
| **UC–01** | Editar modelo | |
| **Versión** | 1.0 | |
| **Autor** | Arturo García Nuño | |
| **Objetivos asociados** | OBJ-01 Gestionar especificaciones a nivel concreto de las Interfaces de Usuario gráficas generadas | |
| **Requisitos asociados** | IRQ-01 Información sobre la especificación de la Interfaz de Usuario Concreta | |
| **Descripción** | El usuario puede realizar una serie de operaciones para editar gráficamente el modelo de una interfaz concreta añadiendo contenedores o componentes concretos al modelo, configurando sus atributos o eliminándolos, o bien generar un modelo vacío o recuperar uno anteriormente guardado para su posterior edición. | |
| **Precondición** | Ninguna | |
| **Secuencia normal** | **Paso** | **Acción** |
| | 1 | El usuario solicita al sistema la realización de una operación sobre el modelo concreto. |
| | 2 | El sistema realiza la operación solicitada |
| **Poscondición** | 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 | 1 segundo |
| **Frecuencia** | Alta | |
| **Estabilidad** | Alta | |
| **Comentarios** | Ninguno | |

**Tabla 3.5 Caso de uso 2: Consultar modelo**

| | | |
|---|---|---|
| **UC–02** | Consultar modelo | |
| **Versión** | 1.0 | |
| **Autor** | Arturo García Nuño | |
| **Objetivos asociados** | OBJ-01 Gestionar especificaciones a nivel concreto de las Interfaces de Usuario gráficas generadas | |
| **Requisitos asociados** | IRQ-01 Información sobre la especificación de la Interfaz de Usuario Concreta | |
| **Descripción** | El usuario debe saber en cada momento como es modelo que va generando, ya sea la estructura del modelo, las propiedades de sus componentes o la especificación en UsiXML equivalente al modelo. Incluso el usuario puede consultar la especificación de un modelo previamente guardado por la herramienta | |
| **Precondición** | Ninguna | |
| **Secuencia normal** | **Paso** | **Acción** |
| | 1 | El usuario solicita al sistema que muestre la estructura, propiedades o la especificación en UsiXML. |
| | 2 | El sistema realiza la operación solicitada |
| **Poscondición** | 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 | 0.2 segundos |
| **Frecuencia** | Alta | |
| **Estabilidad** | Alta | |
| **Comentarios** | Ninguno | |

**Tabla 3.6 Caso de uso 3: Modificar modelo**

| | | |
|---|---|---|
| **UC–03** | Modificar modelo | |
| **Versión** | 1.0 | |
| **Autor** | Arturo García Nuño | |
| **Objetivos asociados** | OBJ-01 Gestionar especificaciones a nivel concreto de las Interfaces de Usuario gráficas generadas | |
| **Requisitos asociados** | IRQ-01 Información sobre la especificación de la Interfaz de Usuario Concreta | |
| **Descripción** | El usuario puede configurar las propiedades y el contenido de los contenedores y componentes del modelo, así como cambiarlos de posición, quedándose reflejado cada cambio en la interfaz gráfica que observa y manipula el usuario | |
| **Precondición** | El componente debe estar agregado al modelo para configurarlo | |
| **Secuencia normal** | **Paso** | **Acción** |
| | 1 | El usuario solicita al sistema el cambio de una propiedad de un componente o contendor del sistema |
| | 2 | El sistema realiza el cambio solicitado. |
| **Poscondición** | 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 | 0.8 segundos |
| **Frecuencia** | Alta | |
| **Estabilidad** | Alta | |
| **Comentarios** | Ninguno | |

**Tabla 3.7 Caso de uso 4: Borrar Modelo**

| | | |
|---|---|---|
| **UC–04** | Borrar Modelo | |
| **Versión** | 1.0 | |
| **Autor** | Arturo García Nuño | |
| **Objetivos asociados** | OBJ-01 Gestionar especificaciones a nivel concreto de las Interfaces de Usuario gráficas generadas | |
| **Requisitos asociados** | IRQ-01 Información sobre la especificación de la Interfaz de Usuario Concreta | |
| **Descripción** | El usuario puede deshacer cualquier elemento que forme parte del modelo CUI editado, quedándose reflejado cada borrado en la interfaz gráfica que observa y manipula el usuario, y en su correspondiente especificación en UsiXML. | |
| **Precondición** | Ninguna | |
| **Secuencia normal** | **Paso** | **Acción** |
| | 1 | El usuario solicita al sistema eliminar elemento CIO |
| | 2 | El sistema realiza la operación solicitada |
| **Poscondición** | 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 | 0.2 segundos |
| **Frecuencia** | Alta | |
| **Estabilidad** | Alta | |
| **Comentarios** | Ninguno | |

#### 3.1.2.3 Identificación de Requisitos No Funcionales

Los requisitos no funcionales (RNF) se conocen como un conjunto de características de calidad, que es necesario tener en cuenta al diseñar e implementar software. Describe no lo que el software hará, sino la manera de hacerlo.

Se han identificado tres requisitos no funcionales: *usabilidad*, *tiempo de respuesta corto y fiabilidad*. Estos requisitos no funcionales tienen una importancia vital en las interfaces de usuario. Si esta herramienta pretende ayudar en el paradigma del desarrollo de interfaces basado en modelos para generar interfaces donde la satisfacción del usuario es un objetivo primordial, sería contradictorio que la herramienta destinada al desarrollador no sea usable, y más aún en nuestro proyecto, ya que una de las innovaciones que pretende aportar respecto al resto de herramientas de funcionalidad parecida es una interfaz mucho más amigable y de fácil uso.

También es importante el rendimiento y la fiabilidad de una aplicación, ya que si tarda mucho en procesar las órdenes indicadas por los usuarios, y las procesa mal no se habrá conseguido nada más que un programa que será desinstalado y sustituido por otro más eficiente. La siguiente tres tablas describen los requisitos no funcionales citados anteriormente:

**Tabla 3.8 Requisito No Funcional 1: Usabilidad**

| | |
|---|---|
| RNF–01 | Usabilidad |
| Versión | 1.0 |
| Autor | Arturo García Nuño |
| Objetivos asociados | OBJ-01 Gestionar especificaciones a nivel concreto de las Interfaces de Usuario gráficas generadas |
| Requisitos asociados | Ninguno |
| Descripción | La aplicación ha de ser sencilla, intuitiva, de fácil aprendizaje, fácil de recordar y atractiva. El usuario debería tener control sobre las operaciones que en cada momento se estén realizando y ser informado de situaciones de error de forma adecuada con mensajes de error significativos. |
| Importancia | Vital |
| Urgencia | Inmediata |
| Estado | Construido |
| Estabilidad | Alta |
| Comentarios | Ninguno |

**Tabla 3.9 Requisito No Funcional 2: Rendimiento de las operaciones**

| | |
|---|---|
| RNF–02 | Rendimiento de las operaciones |
| Versión | 1.0 |
| Autor | Arturo García Nuño |
| Objetivos asociados | OBJ-01 Gestionar especificaciones a nivel concreto de las Interfaces de Usuario gráficas generadas |
| Requisitos asociados | Ninguno |
| Descripción | El sistema debe ofrecer un buen rendimiento a la hora de ejecutar la aplicación, con tiempos de respuesta lógicos, haciendo más hincapié en las transformaciones de la interfaz gráfica a la especificación del modelo CUI, y en el proceso contrario, recuperar todo el modelo a partir de la especificación |
| Importancia | Vital |
| Urgencia | Inmediata |
| Estado | Construido |
| Estabilidad | Media |
| Comentarios | Ninguno |

**Tabla 3.10 Requisito No Funcional 3: Fiabilidad**

| | |
|---|---|
| RNF–03 | Fiabilidad |
| Versión | 1.0 |
| Autor | Arturo García Nuño |
| Objetivos asociados | OBJ-01 Gestionar especificaciones a nivel concreto de las Interfaces de Usuario gráficas generadas |
| Requisitos asociados | Ninguno |
| Descripción | El sistema ha de ser robusto, que ocurra el mínimo error y recuperarse en caso de que lo hubiese de manera que no influya en el funcionamiento de la aplicación en adelante. |
| Importancia | Vital |
| Urgencia | Inmediata |
| Estado | Construido |
| Estabilidad | Alta |
| Comentarios | Ninguno |

### 3.1.3 Fase de Análisis de Requisitos y Diseño

En la fase de análisis de requisitos se trata de modelar los requisitos capturados, tanto funcionales como no funcionales, en un lenguaje sin ambigüedades y que pueda permitir un diseño detallado de las necesidades del sistema. En el diseño se define la estructura del sistema y responder a las preguntas ¿qué componentes existen?, ¿qué papel juega cada componente?, ¿cómo se relacionan los componentes?, empleando diagramas y notaciones formales.

Esta sección se divide en tres subapartados. El primero consiste en determinar cuales van a ser los elementos CIO que van a estar disponibles en la herramienta. Posteriormente en el segundo subapartado se mostrará la estructura del sistema utilizando dos diagramas de clases, en uno se modelará las clases del sistema que permite la implementación de los elementos CIO, en el otro diagrama se modelará las clases que hacen posible el buen funcionamiento global de la herramienta y de la comunicación y sincronización entre las distintas áreas de trabajo que hacen posible por parte del usuario un control absoluto del modelo que diseña. Puesto que los diagramas de clase son por definición estáticos, en el tercer subapartado se mostrarán diagramas de secuencia con el fin de corroborar el modo de interactuar entre las clases modeladas por el segundo diagrama para realizar los servicios que ha de ofrecer la herramienta.

#### 3.1.3.1 Estudio e identificación de los contenedores y componentes concretos

Al analizar el CU-1 se observa que el usuario edita gráficamente el modelo de una interfaz concreta añadiendo contenedores o componentes concretos al modelo, configurando sus atributos o eliminándolo dichos elementos CIO. Hay que determinar cuáles son los elementos CIO que la herramienta va a tener a disposición, y cuales son las propiedades de estos elementos que puede configurar el usuario. Una vez determinados que elementos se van a usar en el programa podemos diseñar la estructura del sistema para su posterior implementación.

Analizando el CU-2 extraemos que el sistema muestre o almacene la especificación del modelo CUI editado junto con el modelo de recursos en un fichero en UsiXML, de manera que hay que instruirse en el modelo CUI de esta metodología ya explicada más que detalladamente en el capitulo 2.

Los elementos extraídos del modelo CUI de UsiXML (UsiXML, 2007) que van a ser útiles en nuestra herramienta se muestran en la siguiente tabla junto con las propiedades características de cada una de ellas. Las propiedades comunes que están presentes en los elementos CIO son: *"name", "width", "height"*, la propiedad común para los contenedores es *"layout*", las propiedades comunes para los componentes concretos son *"ToolTipText", "is visible", "is enable","background", "foreground" y "font"* (con la excepción de VideoComponent). Los contendores escogidos son Window y Box, el resto son componentes concretos.

**Tabla 3.11 Contendores y componentes concretos utilizados en EgiuXML**

| CIO | Icono | Propiedades |
|---|---|---|
| Window |  | "autoscroll" |
| Box |  | "background" |
| Button |   | "text", "icon" |
| TextField (InputText) |  | "text", "is editable", "numberOfColumns", "isPassword" |
| Label (OutputText) |   | "text", "horizontalAlignment", "verticalAlignment", "icon" |
| ToggleButton |   | "text", "icon" |
| RadioButton |  | "text","deafultState", "icon", "groupName" |
| CheckBox |  | "text", "deafultState", "icon", "groupName" |
| ComboBox |  | "name", "item" |
| TextArea (InputText) |   | "defaultext", "numberOfColumns", "numbersOfLines", "isEditable", "isWordWrapper", "forceWordWrapped" |
| List |  | "item" |
| Tree |  | |
| Table |  | "xsize", "ysize" |
| Spin |  | "orientation" |

| | | |
|---|---|---|
| ProgressBar |  | "orientation", "minValue", "maxValue", "indeterminate" |
| Slider |  | "orientation", "minValue", "maxValue", "step", "defaultPosition" |
| DatePicker |  | "day", "month", "year" (defaultContent) |
| HourPicker |  | "hour", "minute", "seconds" (defaultContent) |
| ColorPicker |  | |
| FilePicker |  | "type" |
| ImageComponent |  | "Icon" |
| VideoComponent |  | "file" |

Merece la pena realizar algún comentario de la tabla anterior para empezar a comprender, en esta fase del desarrollo, las posibilidades que debe ofrecer la herramienta.

Hay que destacar que el componente *Window* es un contenedor o elemento CIO especial con el resto de elementos, ya que todo elemento CIO ha de colocarse dentro de un contenedor excepto *Window*, y éste no puede colocarse en el interior de otro contenedor.

No hay que olvidar que un componente CIO es una abstracción de los *widget*s que se encuentran en las herramientas de diseño de interfaces en los lenguajes de programación de alto nivel. El componente concreto definido en UsiXML *"InputText"* es una abstracción de varios de *widgets*, estos son *Textfield*, *TextArea*, y un *Passwordtextfield*. Para facilitar el diseño al desarrollador de interfaces, usuario de nuestra herramienta, se han definido dos componentes CIO *TextField* (equivaldrá también a un *Passwordtextfield*) y TextArea para editarlos gráficamente y posteriormente en la especificación en UsiXML ya serán designados como "*InputText"*.

Los atributos como por ejemplo *"day", "month", "year"* del componente *DatePicker* no se van a encontrar en la especificación de UsiXML sino mas bien se encontrará el atributo *"defaultcontent"*, pero el usuario va a configurar los primeros

aunque en la especificación almacenada en el fichero aparezca de la segunda manera, el fin de este cambio es ir satisfaciendo los requisitos no funcionales analizados

Para el diseño de la herramienta se ha tenido en cuenta, que el valor de algunos de los atributos mostrados en la tabla, han de aparecer en el modelo de recursos en la especificación almacenada.

#### 3.1.3.2 Diagramas de clases

El Diagrama de Clases es el diagrama principal para el análisis y diseño. Un diagrama de clases presenta las clases del sistema con sus relaciones estructurales y de herencia.

Los diagramas de Clases por definición son estáticos, esto es, representan que partes interactúan entre sí, no lo que ocurre en un instante dado. (UML, 2007)

Antes de empezar describiendo los diagramas que estructuran el sistema, sería aconsejable avisar al lector que la aplicación será implementada en el lenguaje de programación Java, por ello el uso de clases que proporcionan las librerías de este lenguaje que facilitan el diseño y su posterior implementación de la aplicación.

El primer diagrama de clases que se describe es aquel que modelará las clases del sistema que permite la implementación de los elementos CIO, debido a la herencia de las clases que extienden de "*JComponent*" de la librería *Java.Swing*, ha resultado mucho más sencilla la creación y modificación de elementos en la interfaz concreta.

Como se puede observar para facilitar la implementación de los distintos componentes y contenedores usados en este módulo, se han reutilizado otros similares que proporciona Java. La clase "*JComponent*" tiene una serie de atributos y métodos (tiene muchos más, pero no se han utilizado) que facilitan mucho la implementación de las clases de niveles inferiores y hacen posible que la configuración del usuario influya en el aspecto de los elementos CIO.

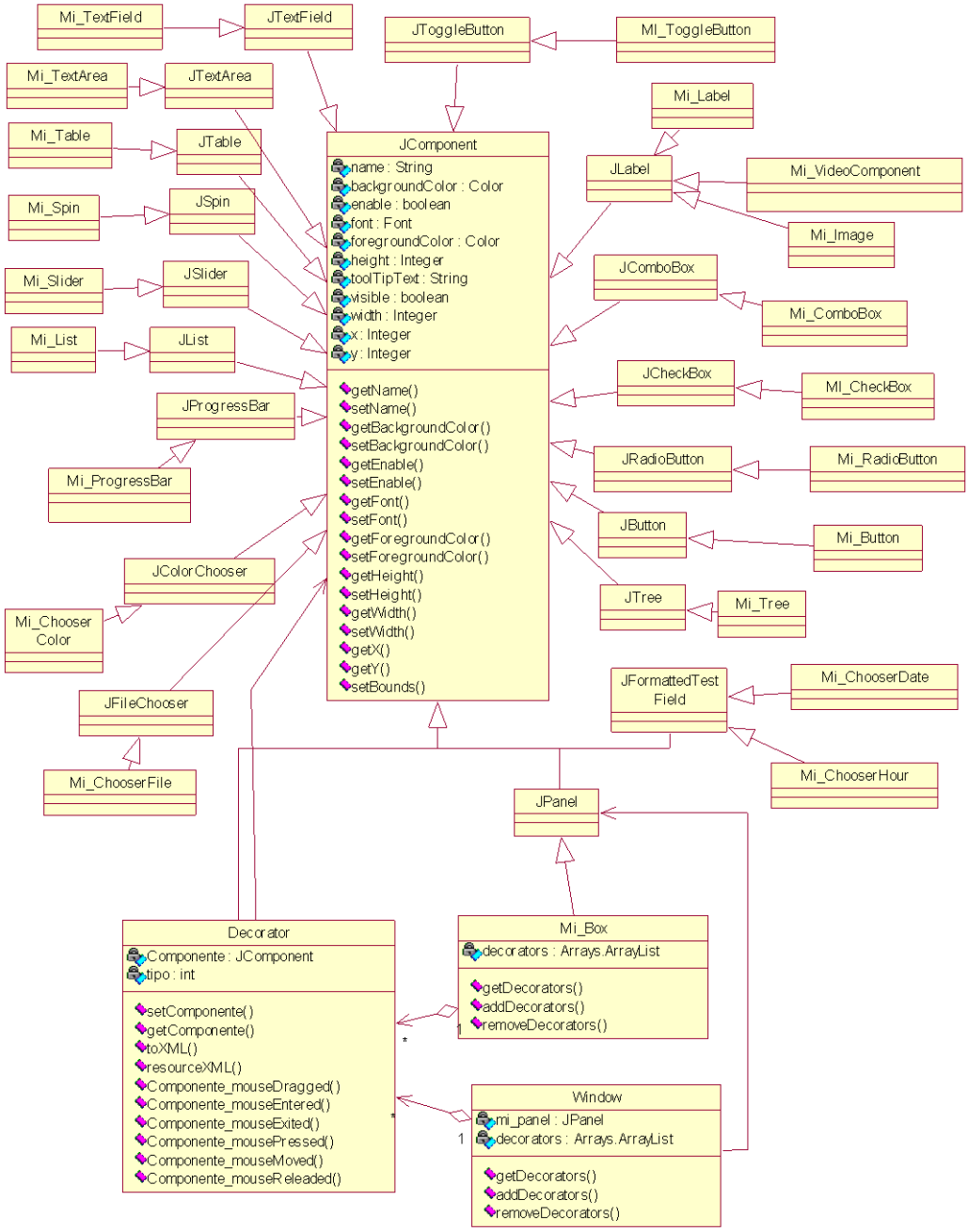


**Figura 3.2 Diagrama de clases para modelar los elementos CIO**

Los dos primeros niveles los proporciona Java, y el tercero se ha añadido para adecuar las clases del segundo nivel a elementos CIO aumentando la adaptabilidad de estos componentes a las necesidades del usuario. Por ejemplo, la clase "*Button*" hereda las propiedades de la clase "*JButton*", y de esta manera se ahorra mucho trabajo, ya que se aprovecha la implementación de dichas clases. Otro ejemplo es "*Mi_Image*", "*Mi_Label*", "*Mi_VideoComponent*" que heredan las propiedades de "*JLabel*", luego

cada uno de ellas añadirá atributos y operaciones para hacerlas más especificas y abstraer el elemento CIO que representan.

En las clases del tercer nivel se ha añadido alguna funcionalidad de tratamiento distinto para cada elemento como: retornar los atributos y valores del elemento CIO que representa, recoger las modificaciones realizadas al elemento actualizando tanto los valores de sus atributos como su apariencia, marcar el elemento seleccionado o no.

También ha visto la necesidad de añadir algunas funcionalidades que son comunes para todos los elementos: posibilidad de mover del objeto, posibilidad de redimensionar el objeto, eliminar objetos, recoger la acción al *clickear* sobre el componente, cambio del puntero del ratón si el objeto puede redimensionarse o moverse, transformar a código UsiXML cada uno de los elementos que forman parte de la interfaz gráfica.

Ante la necesidad de aumentar funcionalidad dinámica a los objetos que representan los CIOs y al ver que no es posible que las clases hereden de dos clases al mismo tiempo se utiliza el patrón *Decorator* (Gamma et al., 1994) que es aplicable cuando hay necesidad de extender la funcionalidad de una clase, pero no hay razones para extenderlo a través de la herencia. Por eso la utilización de la clase "*Decorator*" que mantiene una referencia a un objeto "*JComponent*" y añade responsabilidades al componente. De esta manera evitamos que las clases que representan los elementos CIO estén cargadas de funciones.

Las clases "*Window*" y "*Mi_Box*", es decir, dentro de ellas se pueden añadir otros componentes. Además "*Window*" representa el área de trabajo donde el usuario va a diseñar la interfaz gráfica, en otras palabras el tapiz donde el diseñador desarrolla gráficamente la interfaz, en adelante para referirnos a este contenedor lo nombraremos como panel principal. "*Mi_Box*" incluso también se permite el anidamiento, es decir añadir un contenedor dentro de otro. El resto de las clases son componentes que se pueden agregar en el panel principal o en los contenedores de tipo "*Window*" o "*Box*". La clase "*Window*" mantiene una referencia a un objeto "*JPanel*" y la clase "*Mi_Box*" hereda de "*JPanel*" y ésta a su vez de la clase "*java.awt.Container*" a través de "*JComponent*". Por lo tanto estas clases tienen la capacidad de gestionar un vector con los objetos que se agregan a los contenedores, y además se les ha añadido otro vector para gestionar los "*Decorator*" asociados a dichos objetos, y así facilitar la gestión del modelo. De esta manera resultará mucho más fácil la modificación de objetos del modelo, y el posterior almacenamiento del modelo en un fichero.

El segundo diagrama modela la estructura de la herramienta para que pueda ofrecer todos sus servicios, ya sea desde configurar un componente o transformar la interfaz gráfica en la especificación UsiXML correspondiente. Es importantísimo tal como se vio en los requisitos del sistema que el usuario tenga la misma imagen del modelo que está construyendo ya sea en el árbol de jerarquía de componentes, en el panel principal, o en el instante que se almacena la especificación en un fichero. Del mismo modo que cuando se realice un cambio de una propiedad de un componente se actualice aquellas partes del sistema para que exista coherencia en todo el modelo.

Cuando muchos objetos interactúan con otros, se puede formar una estructura muy compleja, con objetos con muchas conexiones con otros. En un caso extremo cada objeto puede conocer a todos los demás objetos. Para evitar eso se ha utilizado el patrón *Mediator* (Gamma et al., 1994) de esta manera simplifica la comunicación entre objetos: Los objetos que se comunican de la forma "muchos a muchos" puede ser remplazada por una forma "uno a muchos" que es menos compleja y más elegante. Además esta forma de comunicación es más fácil de entender al centralizar el control. La clase “Escritorio” es quien hace de mediador que implementa el comportamiento cooperativo coordinando el resto de objetos. El resto de las clases tienen un atributo de tipo “*Escritorio*” ya que han de conocer a su mediador.

A continuación se describirá lo que representa cada clase, y junto con los diagramas de secuencia se va a entender mejor la comunicación establecida entre estas clases. Además esta estructura supone el pilar del diseño de la interfaz de usuario de la herramienta ya que modela a un nivel alto de abstracción la distribución y papel de las diferentes áreas de trabajo.

La clase “*Interfaz*” supone la clase principal de la aplicación ya que inicia el resto de instancias. Responsable de recoger las solicitudes del usuario de parte de la interfaz (por ejemplo barras de menú). La clase “*Escritorio*” además de la principal función ya explicada, es responsable de definir las diferentes áreas de trabajo que dispone el usuario para desarrollar el modelo. Las demás clases van a representar un área o ventana con una funcionalidad concreta de cara al usuario.

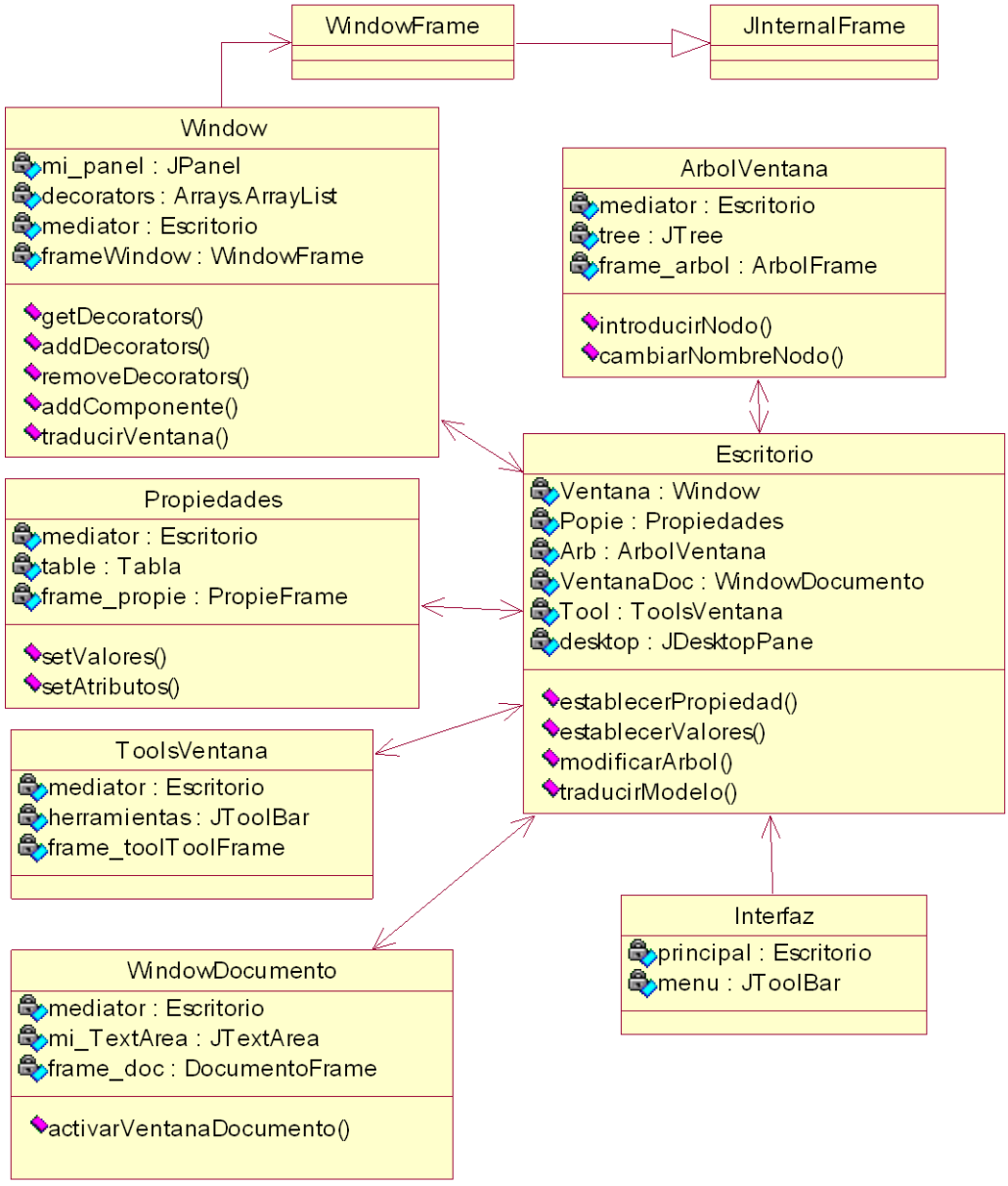


**Figura 3.3 Diagrama de clases para modelar la estructura del sistema**

La clase "*Propiedades*" modela el área donde el usuario puede consultar y modificar los atributos de los elementos CIO, por ello contiene un elemento "*Tabla*" que heredará de "*JTable*" para mostrar los atributos y sus valores asociados. El modelo de interfaz concreta que esté modelando el usuario puede ser representado por una jerarquía de contendores y componentes concretos facilitando la visión global del modelo, dicha funcionalidad se encarga se encarga "*ArbolVentana*" que contendrá un elemento "*JTree*" para representar dicha jerarquía en forma de árbol. La clase "*ToolsVentana*" modela el área que sólo contendrá una barra de herramientas donde

estará toda la paleta de componentes y contenedores que puede utilizar el usuario. La especificación del modelo puede ser consultada por el usuario en cualquier momento sin necesidad de ser almacenada, de manera que la clase "*WindowDocumento*" tiene la responsabilidad de mostrar el texto en UsiXML equivalente al modelo gráfico construido por el usuario, por ello la necesidad de contener un elemento "*JTextArea*".

En el anterior diagrama ya se mencionó el papel importante de la clase "*Window*" ya que no sólo representa un contenedor CIO especial sino que además modela el área de trabajo donde el usuario va a diseñar la interfaz gráfica. Contendrá un elemento "*JPanel*" para contener el resto de elementos CIOs que se agreguen al modelo, y además un elemento "*WindowFrame*" que heredará de "*JInternalFrame*" de la librería Javax.swing para que el usuario trabaje por medio de ventanas, el resto de clases (excepto "*Escritorio*") contienen un elemento que hereda de "JInternaFrame" aunque no se ha puesto en el diagrama para liar menos. "*Window*" es responsable de recoger la acción del ratón cuando se pulsa sobre la ventana y avisar a la clase "Escritorio" de cualquier información que necesiten tener el resto de clases para mantener la coherencia en la herramienta.

Con la finalidad de comprender y ver una visión global de toda la estructura del sistema cuyas partes ya han sido profundamente explicadas, se ha desarrollado un diagrama de paquetes.

Los paquetes ofrecen un mecanismo general para la organización de los modelo/subsistema agrupando elementos de modelado. Cada paquete corresponde a un subsistema del sistema general. Los paquetes son unidades de organización de uso de los modelo de UML (UML, 2007)

Este diagrama de paquetes desarrollado pretende encapsular y ocultar toda la complejidad del sistema, y para despejar cualquier duda, modela las relaciones entre la parte que ofrece los servicios de la herramienta con los elementos CIO que pueden formar parte del modelo y que están a disposición del usuario. El diagrama lo componen los dos paquetes principales "*ObjetosInteracción*" y "*EditorGráfico*"

De cara a la implementación, comprender este encapsulamiento es de gran importancia, ya que se puede ir desarrollando todas las funcionalidades que ofrece el software sin llegar a tener el paquete "*ObjetosInteracción*" completo. Esto permite que se enfatice el trabajo del desarrollo para que la herramienta funcione correctamente y sea usable. Y en el momento que la funcionalidad y usabilidad estén aseguradas se irán añadiendo al sistema la mayor parte de los elementos CIO que se han escogido.

El diagrama de paquetes planteado para la implementación de la aplicación es el siguiente:

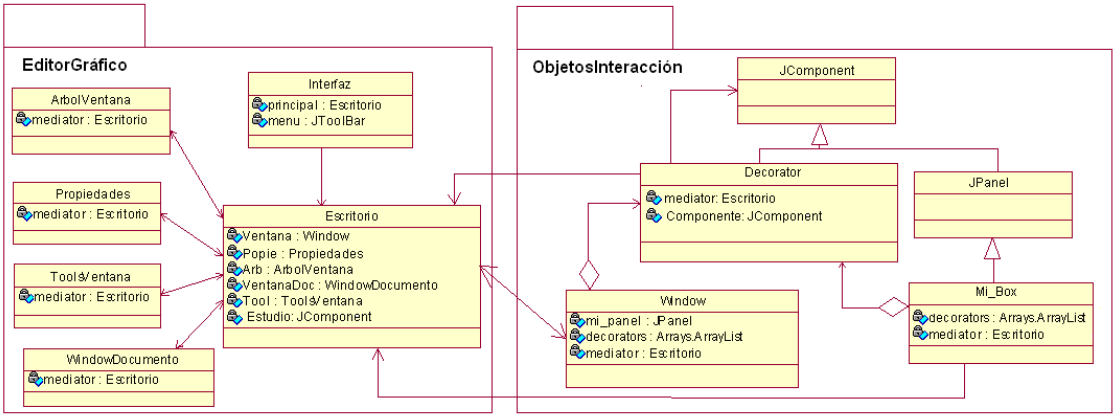


**Figura 3.4 Diagrama de paquetes del sistema desarrollado**

Como puede observarse en la Figura 3.4, la clase Escritorio es importantísima, pues es la única clase conocedora de todos los demás objetos, conoce todas las instancias que representa las áreas de trabajo, inclusive "*Window*", que además es el contenedor "padre" de cualquier modelo. Escritorio conoce cuál es el elemento CIO seleccionado, de modo que ante cualquier modificación de algún atributo de algún componente en la tabla de propiedades sabe cuál es el componente que ha de actualizarse.

La clase "*Window*" como la clase "*Mi_Box*" tienen la capacidad de gestionar un vector con los objetos que se van agregando, y otro vector para gestionar los "*Decorator*" asociados a dichos objetos, esta situación facilita la gestión y el almacenamiento de las especificaciones de los elementos. Sólo hace falta que Escritorio tenga una referencia al nodo raíz de la estructura, o dicho en otras palabras, que contenga una referencia al contenedor "*padre*" "*Window*".

Por último hay que decir que toda instancia de "*Decorator*" (Gamma et al., 1994) debe conocer a Escritorio para comunicarse con el resto de áreas de trabajo, y así se muestre la tabla de propiedades correcta y un árbol de jerarquía actualizado.

### 3.1.3.3 Diagramas de de secuencia

Los diagramas de secuencia son adecuados para observar la perspectiva cronológica de las interacciones entre los objetos durante un escenario concreto. En este proyecto se han hecho uso para enfatizar la comunicación establecida entre las clases

descritas del segundo diagrama. Estas clases representan las diferentes áreas de trabajo que el usuario dispone para modelar de una manera organizada y sencilla. Un buen entendimiento del mecanismo del paso información entre las distintas instancias facilitan las tareas de implementación.

El primer diagrama describe la interacción entre los objetos cuando el usuario agrega un componente, en este caso concreto un elemento CIO *Button*, al contendor *Window*, por lo tanto la estructura árbol también cambia, y el nuevo elemento pasa ser seleccionado en el instante que es agregado y la tabla de propiedades debe mostrar los atributos asociados al elemento con sus valores por defecto.

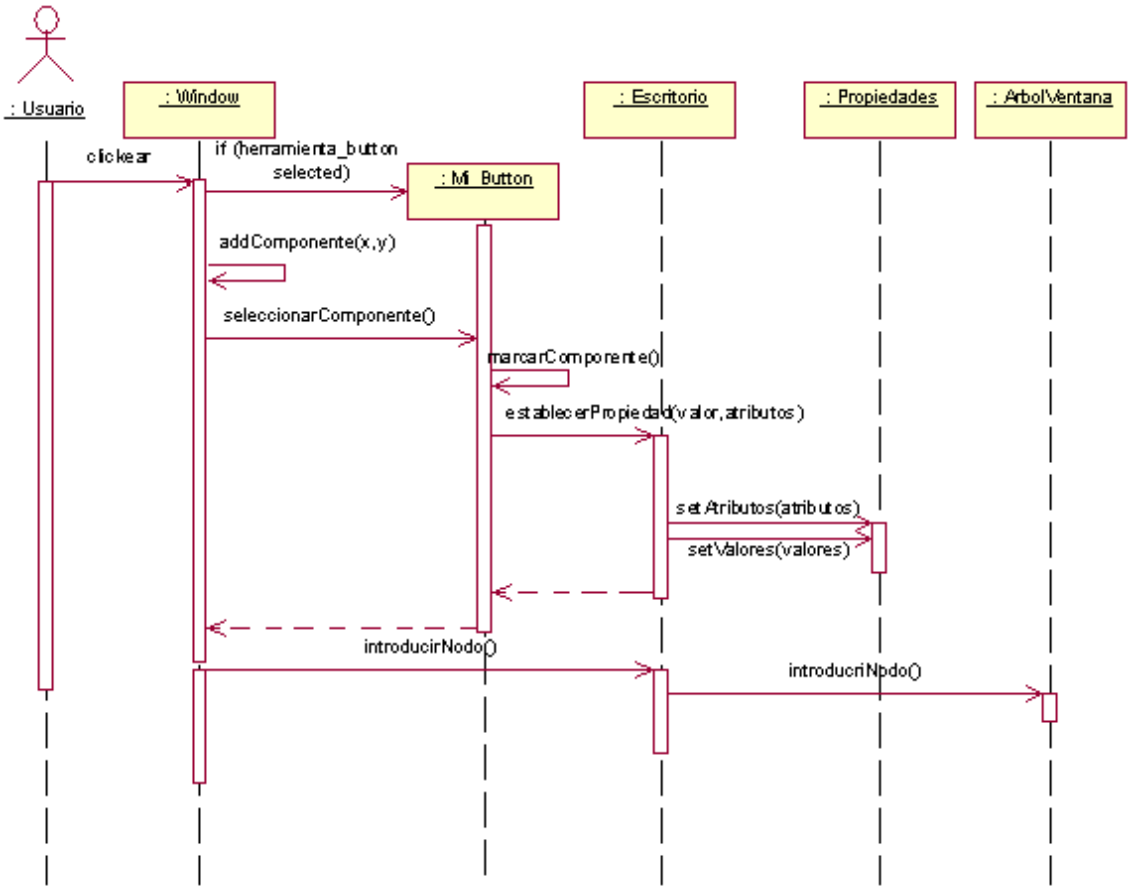


**Figura 3.5 Diagrama de secuencia "Agregar un componente Button"**

Tal como se observa en el diagrama anterior el objeto de "*Window*" es responsable de crear la instancia de "*Mi_Button*", añade el componente tanto en el panel que se visualiza como el decorador asociado a dicho componente en la cola de *decorators* aunque esto último no se haya puesto por redundancia. El componente se marca como seleccionado para distinguirlo del resto de elementos agregados. La tabla de propiedades y el árbol que simboliza el modelo se actualizan gracias al "*Escritorio*" que hace de mediador.

El segundo diagrama de secuencia pretende dar la perspectiva cronológica de la interacción cuando el usuario modifica algún valor en la tabla de propiedades, y el componente tiene que recoger el dato para que esté actualizado y que se refleje su configuración en la interfaz gráfica tal como se definió en los requisitos. La instancia de propiedades proporciona al "Escritorio", mediador, los datos introducidos por el usuario, este sabe cual es el componente que se está configurando (el objeto seleccionado) y reconfigura el componente. En el caso de que se modifique el identificador del componente el mediador avisará a "*ArbolVentana*" para actualizar el árbol.

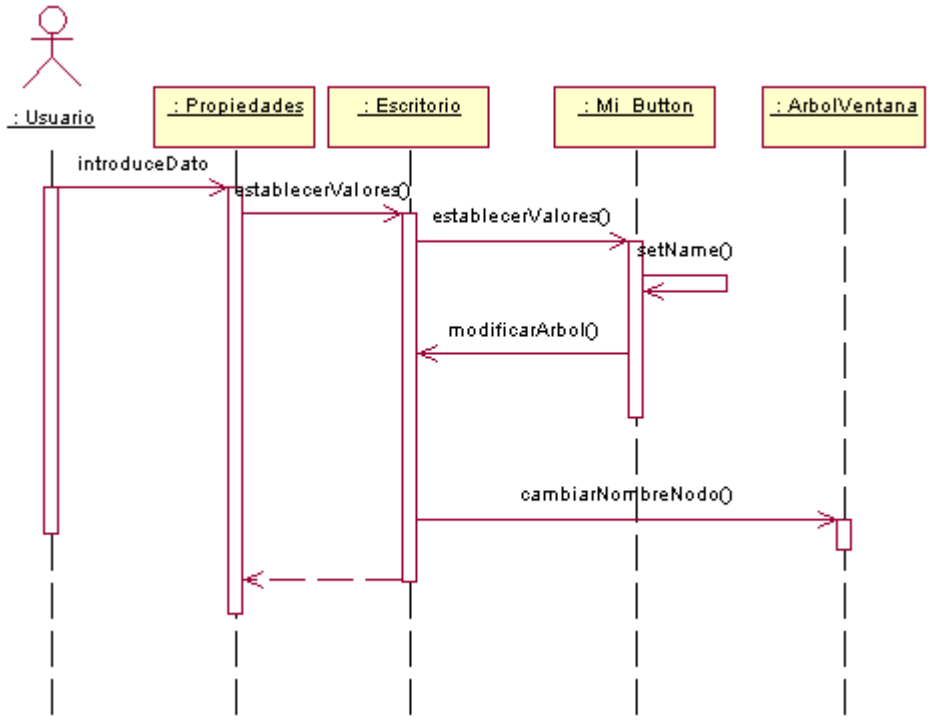


**Figura 3.6 Diagrama de secuencia "Configurar el atributo *name* a un componente Button"**

El tercer escenario estudiado en detalle es cuando el usuario desea consultar la especificación en UsiXML del modelo CUI de la interfaz gráfica que lleva desarrollada hasta el momento. El objeto "Interfaz" manda hacer la tarea al mediador que conoce el resto de objetos, este objeto traduce la información correspondiente a la cabecera del archivo XML o modelo de contexto, para la especificación CUI de la ventana solicita que haga el trabajo el propio contendor "*Window*" y por cada componente que contenga este contendor manda traducir el componente al objeto "*Decorator*" que tenga asociado dicho componente. En implementación se aprovechará este proceso recursivo, favoreciendo el cumplimiento de las restricciones temporales establecidas en los requisitos.

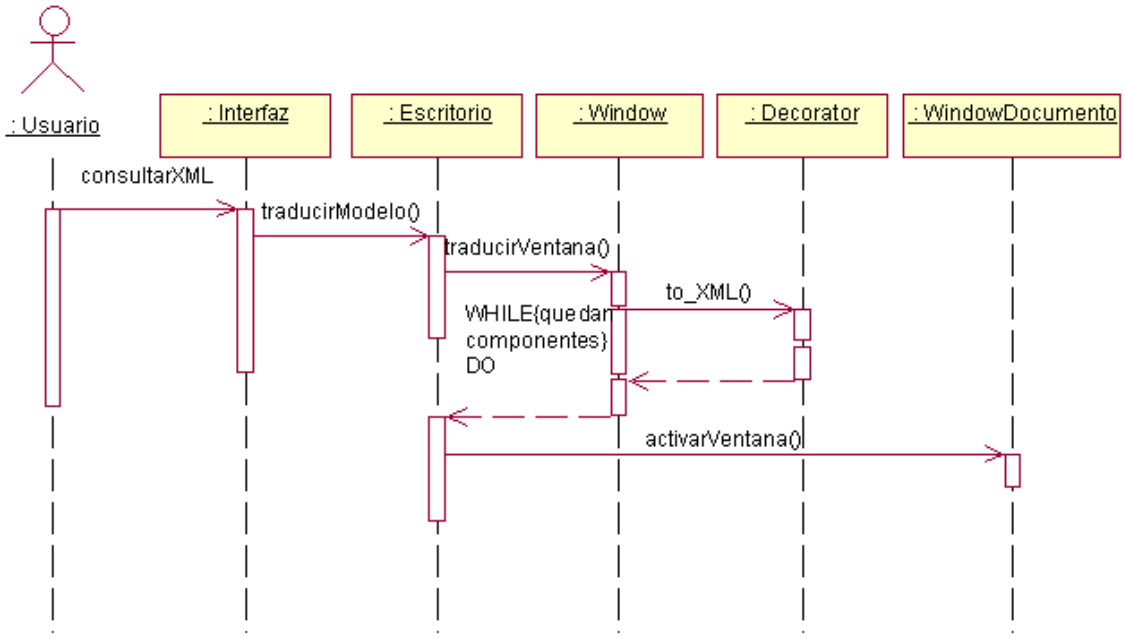


**Figura 3.7 Diagrama de secuencia "Consultar especificación CUI en UsiXML del modelo"**

### 3.1.4 Entorno de programación y librerías utilizadas en la implementación

Una vez realizado el diseño de la aplicación en desarrollo, pasamos a la fase de implementación de dicha aplicación. Esta aplicación se ha implementado en Java, y en concreto con la herramienta Borland JBuilder. La elección de Borland JBuilder se ha debido a la experiencia que se tiene de la herramienta obtenida a lo largo de la carrera, ya que se ha trabajado con ella en distintas asignaturas, y se ha considerado que era la herramienta que mejor se adaptaba a las necesidades del proyecto.

Un detalle importante que merece la pena señalar, es que se ha utilizado la librería de código abierto "Dom4j" (Modelo de Objetos del Documento para Java) 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.

"Dom4j" permite ver el mismo documento de otra manera, describiendo el contenido del documento como un conjunto de objetos, y un programa Java puede actuar sobre ellos. Dicha librería permite recorrer un documento XML de manera sencilla, generar un documento XML asegurando que se respeta el formato declarativo del lenguaje XML, por ello ha sido de mucha utilidad para que los modelos CUI sean almacenados con extensión ".uxml" siguiendo el estándar UsiXML. Su página oficial

es: www.dom4j.org, y en ella se ha podido encontrar mucha información relevante sobre esta librería.

Otro segundo detalle que se debe comentar, es que con el objetivo de implementar el componente *VideoComponent*. Además este no debía estar implementado con una imagen simbólica para diseñar la interfaz ya que se tenía el requisito de que el usuario pueda hacerse una idea bastante buena a como podrá ser la interfaz final. Así que el usuario debe saber como es la reproducción de un video en la interfaz grafica que diseña y así añadir contenido a dicho elemento CIO. Para ello se ha tenido que utilizar la librería Javax.media del conjunto de herramientas que ofrece la extensión de Java JMF *(Java Media Framework)* con el fin de reproducir archivos con extensión ".avi" y ".mpg" y utilizar un panel de controles para controlar la reproducción.

JMF es una expansión de Java que permite la programación de tareas multimedia en este lenguaje de programación. Permite la captura, procesamiento y almacenamiento de datos multimedia. Permite entre otras muchas funciones, la transmisión y recepción a través de Internet, reproducir ficheros multimedia en applets y aplicaciones y reproducir flujos multimedia recibidos en tiempo real a través de la red. Este complemento de java tiene que estar instalado en la máquina del usuario para la ejecución correcta de la aplicación desarrollada. Se puede descargar de manera gratuita desde: http://java.sun.com/products/java-media/jmf/2.1.1/download.html.

En la implementación de la aplicación se ha tenido en cuenta todo el modelado de las fases anteriores obteniendo todas las ventajas del uso de los patrones de diseño, en el siguiente apartado se demostrará como la interfaz ha sido fiel a la estructura modelada del sistema y como en todas las secciones de la interfaz se mantiene la coherencia del modelo CUI que diseña el usuario cumpliendo las restricciones temporales establecidas.

Se ha hecho un gran uso de cuadros de diálogos para notificar la imposibilidad de realizar ciertas operaciones, no debidas por el hecho de que ha ocurrido un fallo sino más bien para que no ocurran en un futuro por el bien del usuario y el de su modelo. Puesto que es una herramienta que deja mucha libertad a la hora de diseñar la interfaz, no puede dejar que una manos de malas intenciones pueda realizar una especificación en UsiXML ambigua e inútil, por ejemplo con el uso de ciertos caracteres para el contenido de ciertos atributos que pueden romper el formato de un documento en XML, todos estos detalles y otros muchos que podrán darse en los casos de estudio del capítulo siguiente han sido contemplados por la herramienta EgiuXML para que sea una herramienta muy fiable.

## 3.2 DESCRIPCIÓN DE LA HERRAMIENTA

En el apartado anterior se ha descrito el proceso de Ingeniería de Software para desarrollar la herramienta, en el apartado presente describiremos el fruto de tal proceso demostrando, junto con el capítulo siguiente, la utilidad y calidad de esta herramienta.

EgiuXML (**E**ditor **g**ráfico de **I**nterfaces de **U**suario a nivel concreto utilizando Usi**XML**) es una herramienta que permite especificar interfaces de usuario de forma gráfica y almacenar dichas especificaciones utilizando el lenguaje UsiXML**.** Gran parte del esfuerzo en el desarrollo de esta aplicación ha sido para que el usuario cree **una interfaz de usuario concreta (CUI) con la máxima flexibilidad y usabilidad**, para hacerlo innovador con respecto a otras herramientas semejantes, pero la gran utilidad y potencia de esta herramienta se esconde en **la transformación a UsiXML a nivel concreto**, aunque a primera vista parezca lo menos llamativo.

El modo de interactuar con la aplicación y todas las posibilidades que ofrece es lo que se va a ver a continuación, viendo ya ejemplos prácticos de uso en el capítulo siguiente. Este apartado se va a dividir en tres subapartados, en el primero se explicará el aspecto general de la interfaz, las áreas de trabajo o secciones que está dividido la interfaz de la herramienta cuyas funcionalidades ya se ha mencionado en este capítulo pero ahora se va a enfatizar el aspecto visual y la manera de utilizarlos por el usuario. En el segundo subapartado será ver en más detalle la sección de la barra de herramientas o botones correspondientes a los contenedores y componentes concretos que el usuario puede insertar en el panel principal. La otra sección vista en detalle en el último subapartado es el menú y una barra de botones de atajo para que el usuario pueda realizar varias acciones (guardar, abrir, convertiraXML,..).

### 3.2.1 Áreas de Trabajo

En aplicaciones, suele llamarse al área de trabajo como la parte principal del programa, es donde el usuario realiza el trabajo o edición. El área de trabajo suele estar rodeada de diferentes accesos directos, herramientas y barras que permiten trabajar a esta. En nuestra aplicación ese concepto sería aplicable al panel principal, aunque se ha querido extender ese concepto al resto de secciones de la interfaz ya que son igual de importantes de cara al usuario.

El entorno de la herramienta EgiuXML (véase Figura 3.8) es un entorno típico con un menú, una barra de botones de atajo, una barra de contenedores y componentes

concretos, un árbol representado la jerarquía de contendores y componentes concretos del modelo, un panel principal y un panel que contiene la tabla de propiedades.

La serie de botones de atajo y el menú que le permiten realizar diversas operaciones útiles al usuario es una sección estática. Recibe este apelativo ya que se encuentra en el entorno pero no en una ventana interna y no puede desplazarse, mientras que el resto de las secciones si se encuentran cada una de ellas en una ventana interna, así que pueden colocarse de la manera más satisfactoria para el usuario, e incluso estas ventanas internas pueden ser minimizadas o modificadas en tamaño por el usuario. En conclusión la interfaz puede ser reconfigurada de la manera que más le antoje al usuario. La Figura 3.8 muestra el entorno de la aplicación, exactamente muestra el aspecto por defecto que va a tener nada más iniciar el programa.

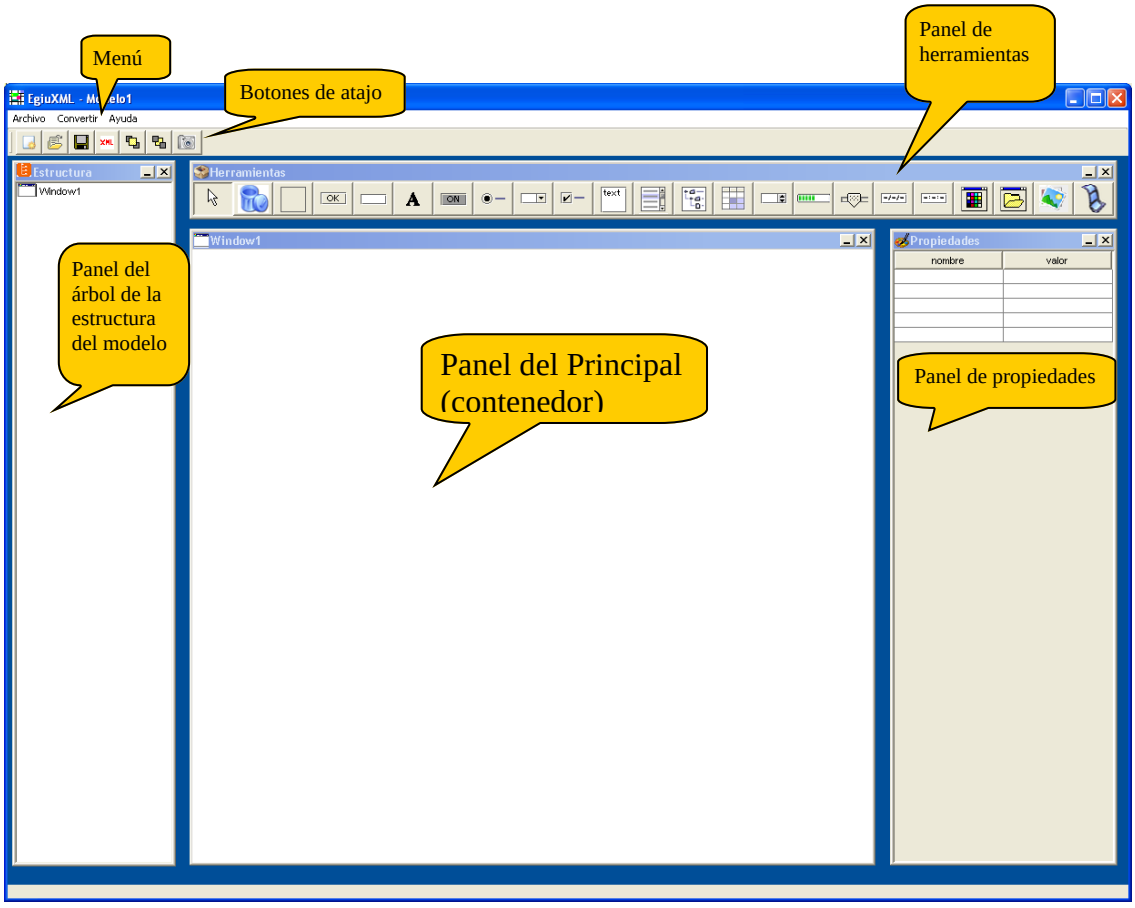


**Figura 3.8 Entorno de EgiuXML**

En la imagen se observa que hay cinco secciones separadas, en la parte de arriba, es lo que se ha denominado como parte estática y es el menú y los botones de atajo. Entre las ventanas o secciones movibles, la situada más alta es una barra de herramientas que le facilitará al usuario la creación de la interfaz gráfica, se utilizará para agregar contenedores Box o componentes concretos. De dichas secciones se

profundizará más adelante. A continuación se centrará en el resto de ventanas que puede manejar el usuario.

#### 3.2.1.1 Panel principal

La ventana central del entorno es lo que recibe el nombre de panel principal, o mejor dicho contenedor Window ya que es considerado elemento CIO en el modelo que diseña el usuario. Por lo tanto los cambios de tamaño que el usuario haga a la ventana se reflejarán en los atributos del elemento. Simboliza la ventana principal de la interfaz que está desarrollando esta herramienta.

Al seleccionar algún botón de la barra de componentes concretos y *clickear* posteriormente sobre el panel principal, el elemento CIO será agregado y dibujado sobre la ventana y a partir de ese elemento se puede trabajar sobre el componente.

El usuario debe seleccionar un *layout* para la distribución de los componentes en la ventana. Los *layouts* que se pueden seleccionar son los siguientes:

- **LibreLayout**: Tan simple como útil. Posiciona un componente donde pulsa el usuario el ratón, en otras palabras lo posiciona justo donde se desea. Es el layout por defecto.

- **Horizontal**: Equivale al FlowLayout de Java. Posiciona un componente a continuación de otro de manera horizontal

- **Vertical**: Posiciona un componente a continuación de otro verticalmente.

EgiuXML permite la alineación de componentes cuando el *layout* seleccionado es LibreLayout, de manera que el usuario no se tiene que preocupar de desplazar los componentes para que estén alineados y no estén descolocados. Hay cuatro maneras de alinear: derecha, izquierda, arriba y abajo. En función de la alineación seleccionada se escogerá un componente u otro como referencia para colocar el resto. Esta operación se permite cuando hay componentes seleccionados.

Se puede seleccionar componentes haciendo *click* en “Seleccionar”  en la barra de herramientas de componentes, luego haciendo *click* en un punto de la ventana y por último arrastrando el rectángulo de selección. Los objetos seleccionados aparecerán con un rectángulo de color rojo alrededor.

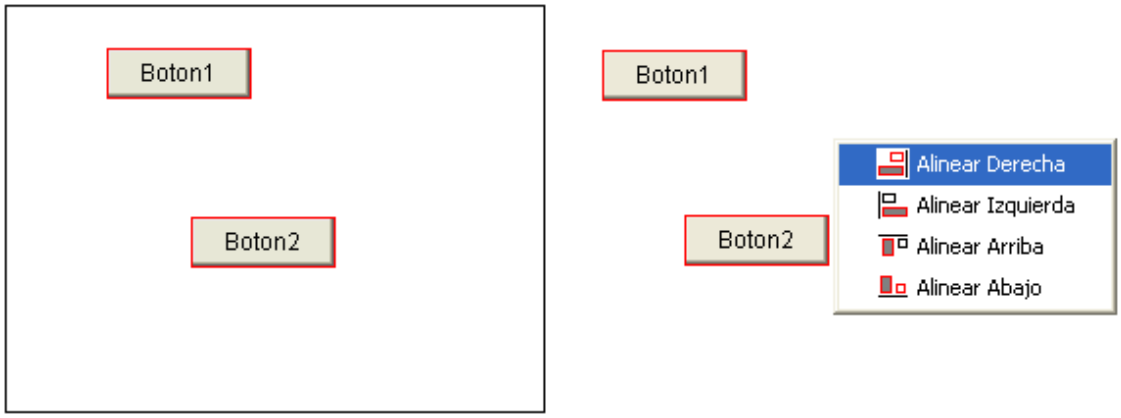


**Figura 3.9 Selección de componentes y alineación**

La ventana, por defecto, tiene el atributo *autoscroll* a "false", el usuario puede ponerlo siempre que desee a trae entonces aparecerá un *scrollbar* horizontal y vertical en la ventana, ahora bien el área establecido para colocar un componente sobre el panel sigue siendo el mismo, que es el establecido por los valores de los atributos *"width", "height"*. La modificación de este atributo como la selección del layout puede ser realizada en cualquier momento, aunque esté a medio construir el modelo.

#### 3.2.1.2 Panel del árbol de la estructura del modelo

En la parte izquierda del entorno por defecto de la herramienta, se puede observar que hay un panel con fondo blanco con una serie de nodos. En dicho panel se irá creando un árbol cuando el usuario vaya creando su interfaz, con lo que le será de gran ayuda para realizar algunas operaciones sobre los nodos, y también le servirá como una buena guía para observar la estructura de la interfaz creada. El nombre del nodo coincide con el valor *name* del elemento CIO que representa.

A continuación se muestra un ejemplo del estado del árbol una vez creada una interfaz de usuario concreta:

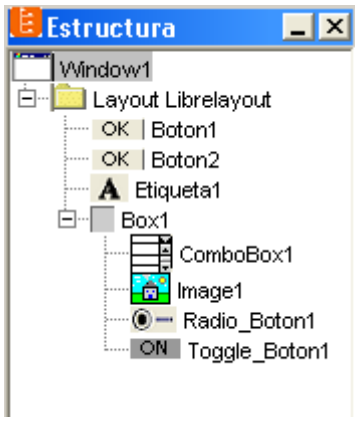


**Figura 3.10 Árbol de estructura del modelo mostrado en EgiuXML**

En la Figura 3.10 se puede observar que cada nodo contiene un simbolito asociado con el elemento CIO que representa. Además informa del *layout* configurado de la ventana. Para seleccionar un elemento agregado al modelo hay dos maneras. La primera es *clickeando* el ratón sobre el componente (siempre que la opción de seleccionar de la barra de herramientas esté marcada).Y la segunda opción es *clickeando* sobre el nodo correspondiente en este panel del árbol de la jerarquía de componentes.

Este panel es una clara demostración de que se ha desarrollado una herramienta centrada en el usuario, permitiendo que genere un modelo CUI de la manera más sencilla, ahorrándole tiempo y equivocaciones por desorganización.

#### 3.2.1.3 Panel de propiedades

En la parte derecha del entorno se puede observar que hay un panel de propiedades, en el que en todo momento se podrán observar los valores que tienen los atributos del contenedor o componente seleccionado, y además se podrán modificar aquí dichos componentes y contenedores, y automáticamente se actualizan en el panel principal, en la tabla y en incluso árbol cuando es necesario. A continuación se muestra un ejemplo del estado del panel de propiedades una vez seleccionado un componente de tipo *TextField*:

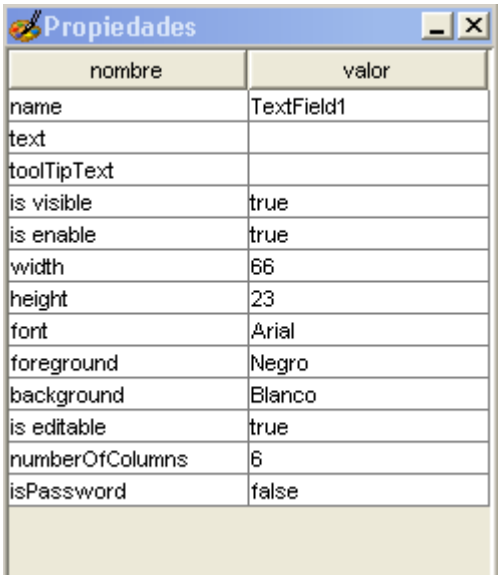


**Figura 3.11 Panel de propiedades**

Algunos valores, el usuario puede rellenarlos introduciendo un dato, algunos de ellos obligan introducir sólo valores numéricos por ejemplo *"width"* o *"height"*, otros no te dejan introducir ciertos caracteres como: ", /, <, >, ñ, siempre para evitar problemas posteriormente con la especificación. En estos casos la aplicación siempre avisa al usuario que no deja modificar el atributo con el nuevo dato (véase Figura 3.12).

Otros valores son escogidos de una lista de opciones que ofrece la aplicación como puede ser para propiedades como *"foreground", "background",* que deja la selección de 11 colores muy utilizados, *"fuente",* que deja la selección de 5 fuentes muy utilizadas, *"layout"* cuyas opciones ya se han visto, y otras tantas propiedades como *"is enable"* o *"is visible"* con las opciones de *"true"* o *"false"*.

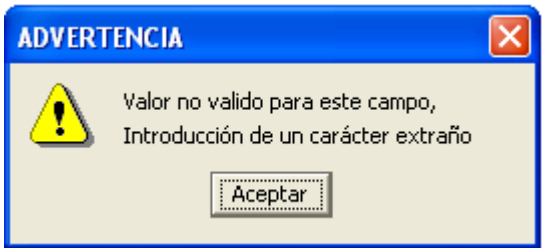


**Figura 3.12 Cuadro de advertencia**

En ocasiones, el valor de una propiedad te abre un cuadro de diálogo que te permite escoger un fichero y así agregar al contenido del componente una imagen. Por ejemplo asignarle una imagen a un componente Image.

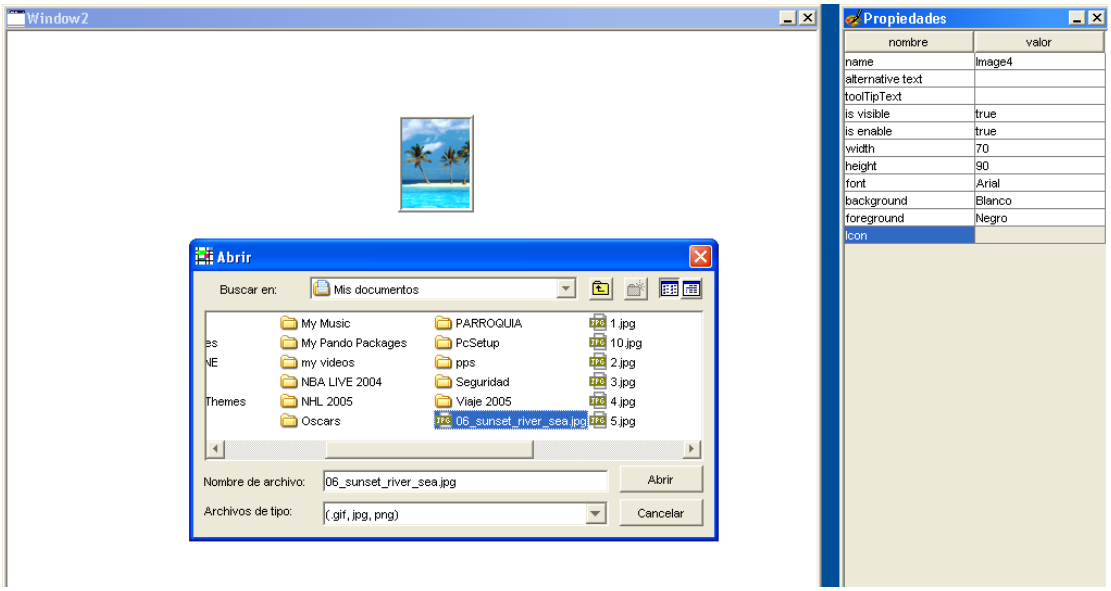


**Figura 3.13 Cuadro de dialogo de selección de un archivo de imagen**

Hay componentes como el ComboBox o List que se le pueden añadir ítems. Pulsando sobre el botón "*añadir/borrar item*" en el valor de la propiedad *item* aparece una ventanita interna para configurar los *item* que se desea tener, dicha ventana tiene el siguiente aspecto:

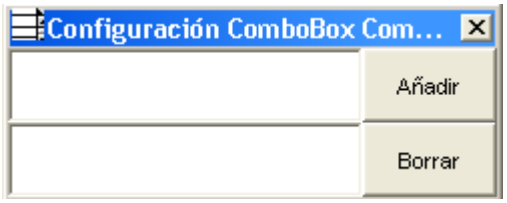


**Figura 3.14 Ventana de configuración de Ítems de un *ComboBox* o *List***

Antes de ver en más detalle la barra de elementos CIO, y algunas características de configuración de algún componente concreto, hay que mencionar que la aplicación antes de mostrar el entorno de la Figura 3.15 muestra un cuadro de diálogo solicitando el nombre del modelo que va a desarrollar como el que se muestra en la siguiente figura.

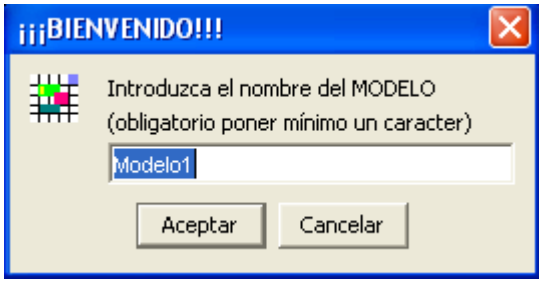


**Figura 3.15 Cuadro de solicitud nombre del modelo**

### 3.2.2 Barra de Herramientas

El usuario dispone de una serie de botones que le facilitarán la creación de la interfaz de usuario concreta. Al seleccionar un botón de un componente y posteriormente haciendo *click* en el panel principal, una instancia del componente es añadido al modelo, presentando su aspecto en el interior de la ventana cuyo lugar de posicionamiento depende del layout que esté configurado en el contendor (*Window* o *Box*) que lo posee.

El usuario puede añadir tantos componentes o contenedores *Box* como él considere, e incluso puede añadir contenedores Box dentro de otros contenedores. El contendor *Window* es especial, y solo hay uno por modelo(al menos eso es una limitación de esta primera versión de la herramienta).

Los componentes concretos y contenedores que puede usar el usuario ya fueron expuestos en la Tabla 3.11, que además se mostraban las propiedades características de cada uno de ellos:

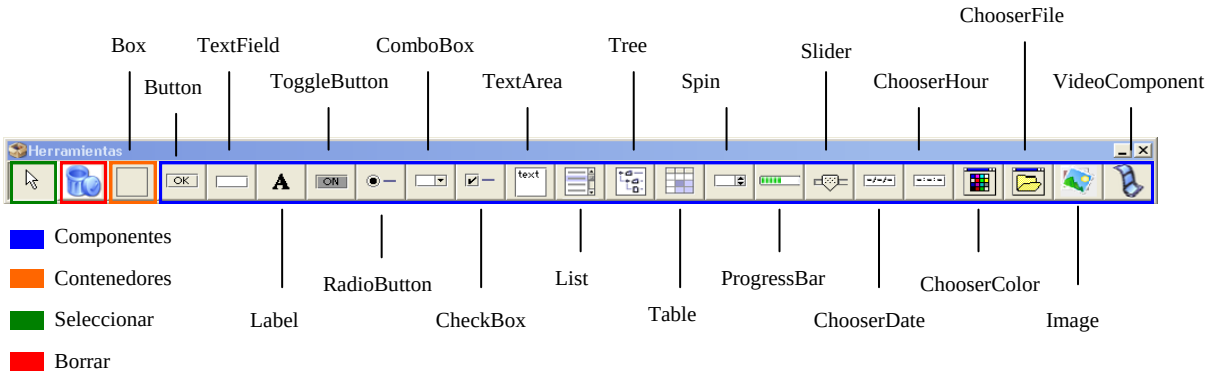


**Figura 3.16 Barra de contendores y componentes concretos**

Junto con los componentes concretos y el contendor, se encuentra el botón seleccionar, que permite seleccionar los elementos cuando se les *clickea* con el ratón, así como seleccionar varios componentes con un cuadro de selección con la finalidad de alinearlos.

Para modificar el tamaño de los elementos se puede hacer de dos maneras: La primera ya ha sido vista, es configurando el atributo *"width"* o *"height"* introduciendo un valor numérico en el campo valor en la tabla de propiedades. Y la segunda manera es pulsando en el botón de seleccionar, después colocar el puntero sobre la esquina inferior derecha del componente y cuando se transforma en la flecha de dos puntas ,

se arrastra el ratón apareciendo un recuadro de configuración de tamaño que indica el tamaño que va a tener el componente cuando se deje de pulsar el ratón. Es decir de una manera muy cómoda y facilitando la tarea al usuario.

Para eliminar un elemento también hay dos maneras, la primeras es pulsando el botón de borrar  y posteriormente pulsar sobre el componente que se desea eliminar. La segunda manera es *clickear* con el segundo botón del ratón y pulsar en la operación correspondiente del menú que aparece en pantalla, tal como se muestra en la Figura 3.17.


**Figura 3.17 Menú contextual para borrar un elemento**

Al eliminar un contenedor Box ya supone la eliminación de todos los componentes que éste contiene. Un Box también tiene la propiedad layout para configurar la distribución de los componentes, y los *layouts* posibles son los mismos que los vistos anteriormente para Window.

### 3.2.3 Barra de Menú y Botones de Atajo

En este subapartado se va a describir las operaciones que ofrece la herramienta por medio de la barra de menú o los botones de atajo. La serie de botones de atajo, Figura 3.18, permite al usuario realizar las siguientes operaciones:

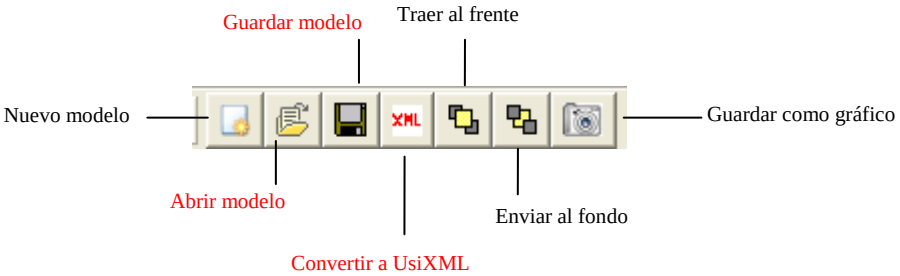


**Figura 3.18 Botones de atajo**

- Guardar como gráfico: Realiza una captura de pantalla y permite al usuario guardarla donde desee en formato .jpg

- Enviar al fondo / Traer al Frente: Cuando se agregan componentes a un mismo contendor, el componente agregado primero va a tapar al que se agregue después si éstos se superponen. Para facilitarle la tarea al usuario y no tenga que

planear, antes de añadir elementos, el orden de agregación de componentes, la herramienta permite al usuario decidir que componente debe sobreponerse al resto, gracias al uso de estos dos botones.

- Convertir a UsiXML: Consiste en mostrar en otra ventana interna la especificación del modelo CUI utilizando el lenguaje de modelado UsiXML. Ver la Figura 3.19.

- Guardar el modelo: Guarda en el directorio que desea el usuario, la especificación en UsiXML del modelo en un fichero de extensión ".uxml". Cuando guarda la especificación, la herramienta vuelve a traducir el modelo, de manera que se almacena el modelo CUI tal como se encuentra en el instante de pulsar el botón Guardar. Al guardar el fichero, puedes guardarlo por defecto con el nombre del modelo o con otro nombre de manera que el identificador del modelo se actualizará.

- Abrir modelo: Abre el fichero con extensión ".uxml" y se recupera todo el modelo, generando la interfaz gráfica correspondiente al modelo CUI del fichero. Antes de abrir un fichero se pregunta al usuario si desea guardar el modelo actual abierto.

- Cabe destacar la importancia de estos botones ya que ofrece una funcionalidad esencial a la herramienta por la que fue desarrollada y por la que se encuentra dentro del paradigma de desarrollo de interfaces basado en modelos.

- Nuevo modelo: Reinicia un modelo desde el principio. Antes de abrir un fichero vacío se pregunta al usuario si desea guardar el modelo actual abierto y posteriormente solicita el nombre que se desea poner al nuevo modelo.

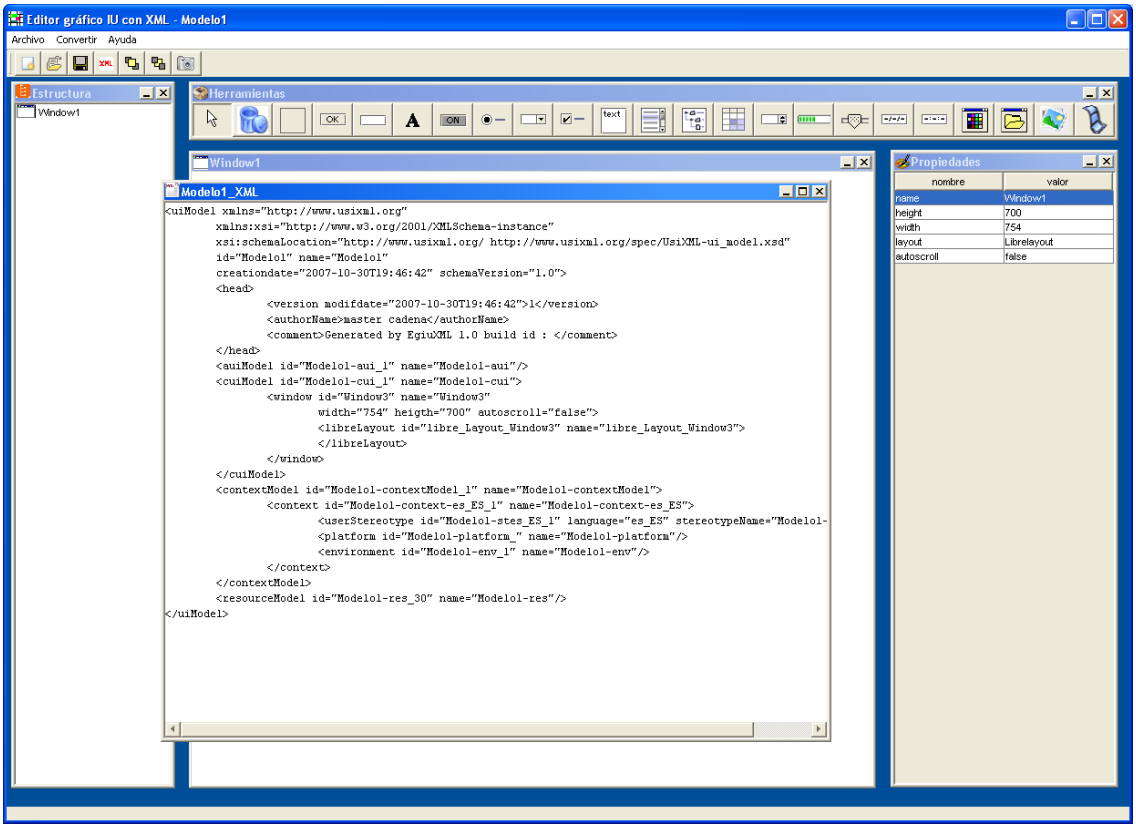


**Figura 3.19 Ventana de especificación del modelo en UsiXML en EgiuXML**

En el capítulo siguiente veremos en más detalle el contenido UsiXML generado, pero con en la Figura 3.19 se puede observar como la herramienta que también es responsable de generar el modelo por defecto establecido de contexto y el modelo recursos de la interfaz gráfica generada por el usuario.

De hecho si se observa bien el código en UsiXML generado por la herramienta abarca todo el modelo de interfaz de usuario. Estableciendo una cabecera al documento XML donde se encuentra información del sistema donde se ha desarrollado el modelo, se muestra la posibilidad para trabajos futuros de especificar el modelo a nivel abstracto. Muestra la especificación a nivel concreto para lo que ha sido diseñada la herramienta. Muestra un modelo de contexto por defecto en el cual el modelo de usuario, clasifica al usuario con un perfil español, y deja la posibilidad para versiones posteriores de especificar de manera más profunda el modelo de plataforma y de entorno.

Por último destacar que en el entorno facilitado se permite modelar el modelo de recursos y, por lo tanto, el almacenamiento de los recursos de forma separada a los elementos que los utilizan, dejando un lugar fijo de almacenamiento dentro del sistema donde se ejecuta el software. De esa manera el contenido de los elementos CIO será

almacenado y si el usuario exporta la GUI diseñada a otro lenguaje como Java o XHTML, varios ficheros serán salvados para la plataforma establecida. Demostrando de esa manera parte de la capacidad de reutilización que ofrece el modelado de interfaces de usuario.

La barra de menú ofrece las mismas operaciones que los botones de atajo, excepto la función de "Traer al frente" o "Enviar al fondo" el componente seleccionado. Sin embargo añade la opción de salir del programa y la operación de mostrar el manual de ayuda de la herramienta. Para cada una de las operaciones de la barra del menú hay una tecla de acceso rápido asociada para activar esa acción.

A continuación se muestra el menú para consultar el manual de ayuda:


**Figura 3.20 Menú para abrir la ayuda**

A veces un programa es muy bueno, pero carece de un manual de ayuda para ayudarte a comprender su manejo, y entonces decides utilizar una herramienta más sencilla o con mejor manual de ayuda. Por este motivo se ha considerado necesaria la realización de un manual de ayuda.

Para que los usuarios que utilicen esta herramienta no tengan problemas a la hora de aprender su funcionamiento, se ha realizado un manual de ayuda en castellano. De esta manera se conseguirá que los usuarios no tengan muchos problemas para familiarizarse con dicha herramienta.

En este manual se explica la funcionalidad de cada uno de los elementos que tiene la interfaz de la herramienta, todo aquello que ofrece la herramienta para la configuración de los elementos CIO, así como las opciones de los menús y botones de atajo.

Además, se han realizado algún ejemplo práctico, que sirva como tutorial que explique paso a paso cómo se utiliza la herramienta.

A continuación se muestra una instantánea del manual de ayuda

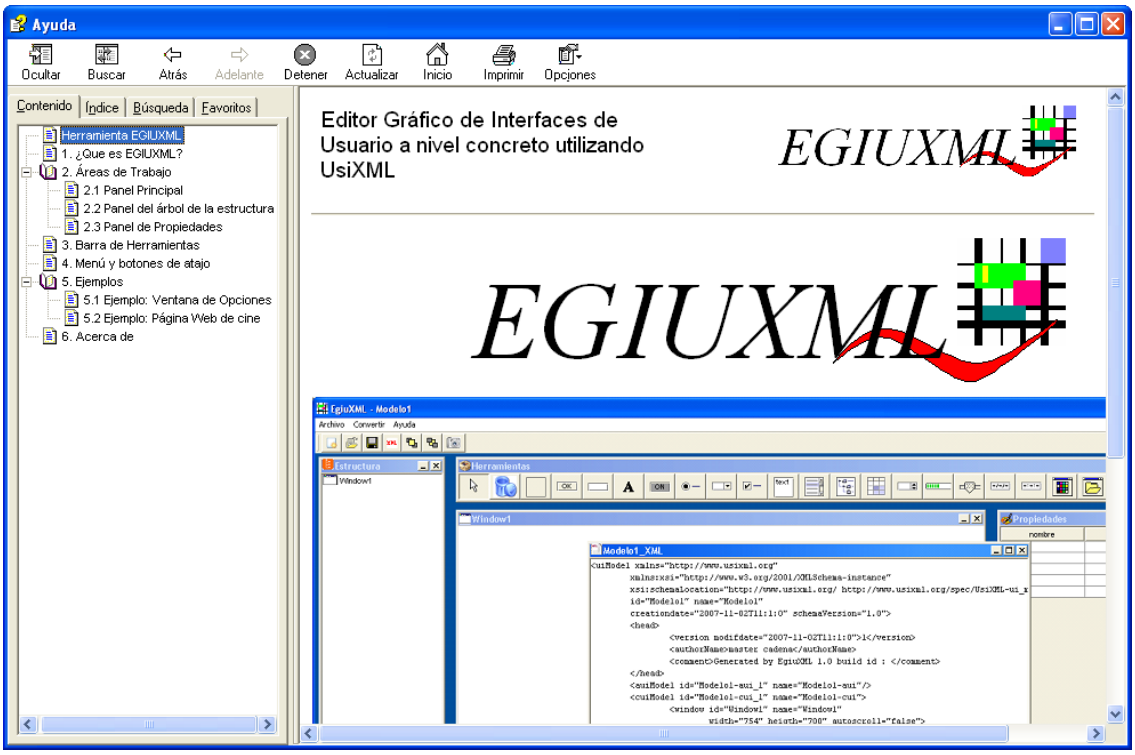


**Figura 3.21 Manual de ayuda**

## 3.3 CONCLUSIONES

Una vez llegado a este punto del trabajo, se puede concluir que la herramienta satisface tanto en funcionalidad perseguida como en modo de facilitarla. En el primer apartado de este capítulo se describió el proceso de desarrollo del software, cuyos mecanismos del proceso fueron recogidos de la metodología El Proceso Unificado utilizando UML para expresar gráficamente todos los esquemas del sistema software. En el diseño del sistema se apostó por el concepto de patrón (Gamma et al., 1994). Los patrones aportan experiencia, documentación de soluciones y todo ello conlleva a que aportan mejora en la calidad del producto y, en este sentido, dos patrones GoF han sido especialmente útiles: el *Decorator* y el *Mediator*..

Desde las primeras fases del desarrollo de la aplicación se identificaron los requisitos no funcionales de manera muy clara. Había una exigencia en que la herramienta fuese pensada al máximo detalle para los intereses del usuario, esta debía ser usable, de fácil uso y aprendizaje, cómoda para ahorrar tiempo de diseño y atractiva para todo el tiempo que la utilice. EgiuXML es una herramienta desarrollada para hacer interfaces de usuario de calidad, no se ha querido caer en el dicho de *en casa del*

*herrero, cuchillo de palo*, y en este sentido esta herramienta debía ofrecer aquellas cualidades que se desean aplicar a las interfaces modeladas por la aplicación.

En el segundo apartado de este capítulo, se han descrito detalladamente todos los detalles implementados que han sido pensados para la satisfacción del usuario, sin olvidar el principal objetivo para la que ha sido desarrollada, la especificación de interfaces de usuario a nivel concreto. De manera que contribuye, o mejor dicho **es una herramienta más del grupo del paradigma de desarrollo de interfaces basado en modelos, útil para la reutilización y automatización en el proceso de la generación de interfaces, donde se ha considerado los intereses del diseñador de interfaces**. La consideración en el desarrollador es una importante innovación con respecto a otras herramientas de funcionalidad parecida.

La herramienta facilita la especificación del modelo CUI en UsiXML correspondiente a la interfaz gráfica de manera inmediata, permitiendo almacenar un modelo complejísimo con gran variedad de elementos concretos y estos configurados con un alto número de atributos, en un fichero XML. Pero el diseño gráfico "complejo", complejo dentro de lo que cabe ya que este software reduce cualquier complejidad, nunca se va a perder puesto que la aplicación recupera la interfaz gráfica tal como la almacenó el usuario para que éste la edite o modifique en lo que estime oportuno.

Hay un tema que falta por tratar de la herramienta, es el relativo a las características o tipo de interfaces que se pueden diseñar con el entorno desarrollado. Hemos presentado en entorno desarrollado de forma estática y diferentes consideraciones relacionadas con su especificación, ahora falta demostrar la funcionalidad del entorno desarrollado para ello, en el capítulo siguiente, se utilizarán dos casos de estudio de dos interfaces que podemos encontrar en la red o en un típico software y cómo éstas pueden ser diseñadas con EgiuXML obteniendo posteriormente su especificación CUI en el lenguaje UsiXML.

# Capítulo 4: Caso de estudio

En el presente capítulo para corroborar la explicación del capítulo anterior sobre el funcionamiento de la herramienta EgiuXML desarrollada, se realizarán dos casos de estudios con los que demostrar la conversión de una interfaz de usuario gráfica generada por la misma aplicación a un modelo de nivel concreto en formato UsiXML.

Los dos ejemplos desarrollados son: en primer lugar una típica ventana de configuración de opciones de un programa de Chat, y el segundo consiste en una interfaz web que nos podemos encontrar en cualquier página web de cine.

A continuación se va a explicar el proceso de desarrollo de la generación de estos dos ejemplos y el código UsiXML correspondiente generado por la aplicación.

## 4.1 CASO DE ESTUDIO 1: Ventana de Opciones

Este primer ejemplo consiste en la creación de una interfaz de una ventana de configuración de opciones de un típico programa de Chat para charlar con los contactos que tienen direcciones procedentes de un proveedor de correo electrónico, se ha cogido como referencia las opciones que ofrece el software *Windows Live Messenger*. Aunque la aplicación desarrollada es capaz de especificar cualquier interfaz real del programa de referencia, la interfaz especificada en este caso de estudio es algo distinta ya que nuestra intención es presentar la mayoría de las posibilidades que ofrece la herramienta desarrollada en una ventana sin que ésta pierda un poco el sentido de realismo.

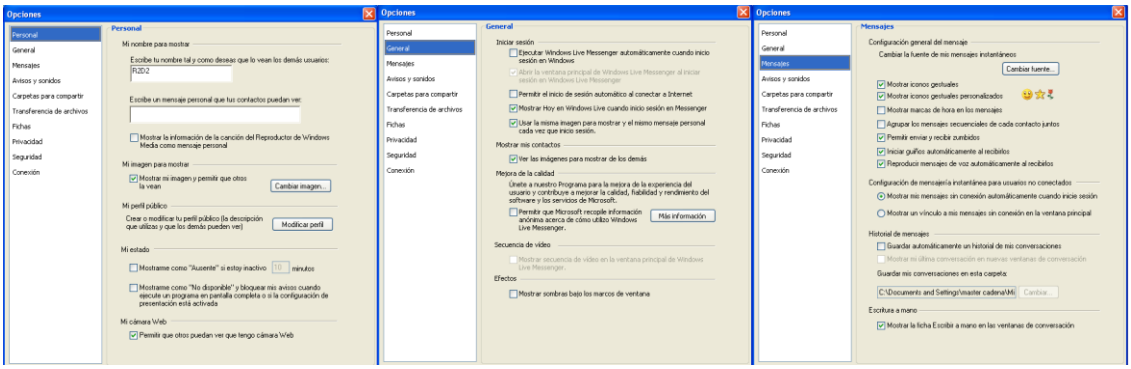


**Figura 4.1 Ventanas de Opciones del programa de referencia *Windows Live Messenger***

Tal como se observa en la Figura 4.1, las distintas ventanas originales muestran muy pocos tipos de *widgets* y de un aspecto muy repetitivo mientras que nuestra

interfaz utiliza muchos componentes, aún con eso es una interfaz muy creíble que puede ser utilizada por cualquier software con la misma funcionalidad.

La interfaz diseñada se compone, como siempre, de un único contenedor *Window*, que en este caso representa la ventana de Opciones, por ello el identificador de este contenedor es cambiado y de esa manera se muestra el título de ventana que se desea. Este contenedor contiene otros tres contenedores Box, cada uno representa o simula un aspecto de configuración de las posibles opciones de la ventana que se diseña.

Antes de seguir explicando las partes que forman la interfaz y cómo estas han sido configuradas, es mejor ver la Figura 4.2 que se muestra la interfaz de usuario concreta creada para este ejemplo.

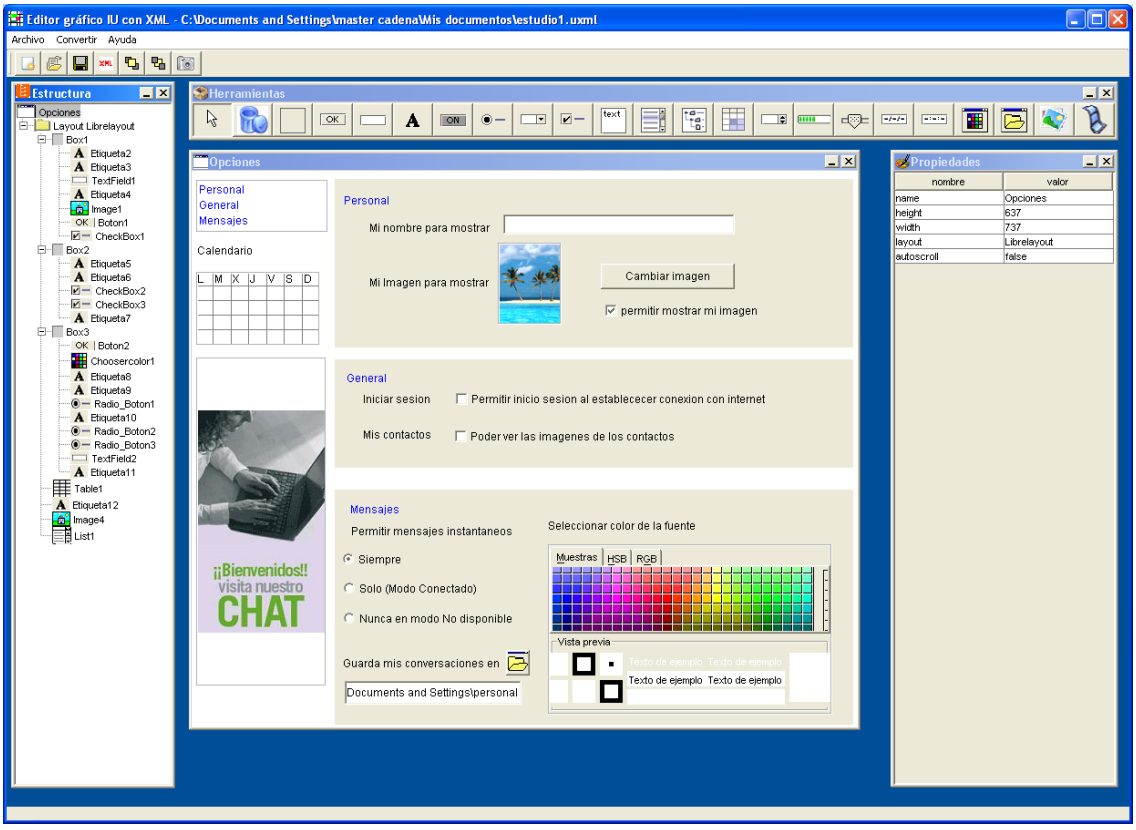


**Figura 4.2 Interfaz de usuario concreta de una Ventana de Opciones**

En el panel principal, empezando por la parte de la izquierda, hay un componente List que contiene tres ítem, los ítem son agregados de la manera que se vio en la Figura 3.14 en el capítulo anterior. Debajo hay un componente Label y un componente Table que ha sido configurado para que tenga siete columnas y cinco filas, este componente representa un calendario, por ello en la primera fila cada celda

contiene un contenido o texto, es un ejemplo de las posibilidades que ofrece la herramienta al manejar un elemento Table. Al final de esta parte de la interfaz se encuentra un componente Image, por ejemplo para representar publicidad, cuyo atributo *"icon"* ha sido configurado para agregarle al contenido de este componente una imagen, esta imagen forma parte del conjunto de recursos y será almacenada por la herramienta sin que tenga que hacer algo más el usuario. Estos componentes concretos están contenidos directamente dentro del contenedor Window denominado "Opciones", este contenedor está configurado con *layout* "*LibreLayout"*, quedándose todo muy bien alineado gracias a las opciones de selección y alineación que permite la herramienta.

En la parte del centro, ya se ha mencionado que hay tres contendores Box, cada uno de ellos ha sido configurado con *layout* "*LibreLayout"* debido a su flexibilidad y facilidad de uso. Empezando por el contenedor de arriba, contiene tres componentes Label, dos de ellos configurados para que el texto esté alineado horizontalmente a la izquierda. Una ventaja que ofrece *layout* "*LibreLayout"* es que podemos insertar el texto tan largo como se desee sin influir en la posición del resto de componentes. Hay un componente TextField, un Button configurados con los valores por defecto excepto el texto de este último. También hay un CheckBox con el atributo *"defaultstate"* a *"true"*, y un componente Image sin ninguna imagen asociada, la imagen de palmeras ,que se observa, forma parte del componente Image por defecto para que el usuario sepa desde el primer instante que es un elemento Image y no se confunde con otro elemento.

En el segundo Box, se encuentran tres componentes de tipo Panel, dos de ellos también configurados con *"horizontalAlignment"* a la izquierda, y dos CheckBox. Los tamaños de los componentes pueden ser configurados de las dos maneras vistas en el capitulo anterior, y gracias a las opciones de alineación que ofrece la herramienta, el formulario se queda muy bien estructurado.

En el tercer Box, hay tres componentes Label configurados de manera parecida a los anteriores, siempre el componente Label que titula el sector de configuración de opciones es configurado con color de fuente "Azul". Hay tres elementos RadioButton, han sido configurados de tal manera que pertenecen al mismo grupo usando el atributo "*groupname"*, uno de ellos el denominado *"Radio_Boton1"* tiene como *defaultstate* a "true". Además hay un selector de color conocido como elemento CIO *ColorPicker*. Un elemento Button cuyo texto ha sido borrado y se le ha agregado una imagen y por último un TextField configurado con un texto por defecto y el campo *"editable a false"*. De esta manera se muestra la gran variedad de posibilidades de configuración de componentes que permite hacer esta herramienta.

Antes de ver el código en UsiXML correspondiente a la interfaz gráfica equivalente se puede echar un vistazo más detallado al árbol de la jerarquía de componentes del modelo que ofrece la herramienta para consolidar la explicación de las partes que forman la interfaz generada. De hecho esta jerarquía es muy similar a la estructura del árbol DOM del modelo CUI en UsiXML pero muchísimo más resumida y más atractiva de cara al usuario.

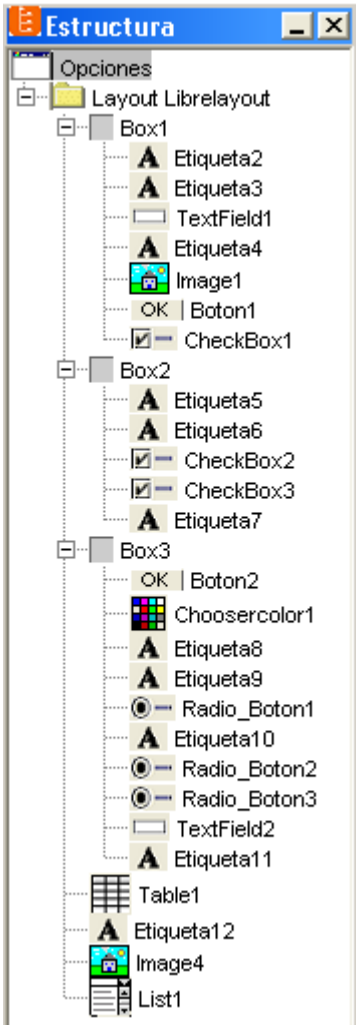


**Figura 4.3 Árbol de Elementos CIO de la interfaz generada en el Caso de Estudio 1**

En cualquier momento se puede ver la especificación de la interfaz en desarrollo en UsiXML en la ventana interna designada para ello, la manera más rápida es dándole al botón atajo  o pulsando la tecla F9.

Para no aburrir al lector, en vez de mostrar todo el código de una, se va ir mostrando por partes, destacando aquello que es interesante para el buen entendimiento de un modelo a nivel concreto.

En primer lugar se va a mostrar la cabecera del código UsiXML que se ha generado a partir de la interfaz grafica diseñada. En ella recoge fecha y la hora del sistema en la cual se está desarrollando la interfaz en el campo *"creationdate"*, concretamente justo en el instante que hace la traducción al ser pulsado dicho botón o es almacenada la especificación en un fichero. En el subnodo *"<authorname>"* se muestra el nombre del usuario del sistema que está desarrollando la interfaz y con el subnodo "<*comment*>" se comenta la herramienta utilizada para la generación del modelo de interfaz de usuario. Se mencionó en capítulo 3, que el modelo AUI es un posible trabajo futuro para EgiuXML, pero que por ahora no abarca tal interés y por ello sólo se encuentran el identificador y nombre del modelo AUI.

```
<uiModel xmlns="http://www.usixml.org"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.usixml.org/ http://www.usixml.org/spec/UsiXML-ui_model.xsd"
        id="Estudio 1" name="Estudio 1"
        creationdate="2007-10-29T14:6:52" schemaVersion="1.0">
        <head>
                <version modifdate="2007-10-29T14:6:52">1</version>
                <authorName>master cadena</authorName>
                <comment>Generated by EgiuXML 1.0 build id : </comment>
        </head>
        <auiModel id="Estudio 1-aui_1" name="Estudio 1-aui"/>
```

**Figura 4.4 Cabecera del código UsiXML generado en el Caso de Estudio 1**

La Figura 4.5 es el código del modelo CUI, pero hasta el final del primer contenedor Box. Se observa en dicha figura, que el primer nodo es etiquetado con la etiqueta *cuimodel*, y abarca gran parte del código generado. Dentro de dicho nodo hay otro nodo que es el contenedor concreto de tipo *Window*, en este caso denominado *"Opciones*", la configuración de las propiedades de este de contenedor son reflejados en los atributos del nodo correspondiente. Dentro de dicho contenedor hay un nodo que simboliza el layout configurado, en este caso es "*LibreLayout"*. Este layout tiene la singularidad que permite situar los elementos CIO donde desea el usuario. Para especificar las posiciones "x" e "y" que sitúa a cada elemento dentro del contendor al cuál pertenece, se define un nodo denominado *constraint* que tiene como campos *"locationx"* y *"locationy"*. Solamente cuando el layout configurado del contendor padre es *LibreLayout*, ya sea el contenedor padre *Window* o Box, el nodo que especifica un elemento CIO va a tener como padre al nodo *constraint*, que éste es a su vez es hijo de *LibreLayout*.

A continuación se muestra una parte del código generado, junto con imágenes de los nodos del árbol de la jerarquía del modelo, para facilitar el entendimiento de la correspondencia entre la interfaz gráfica diseñada y los nodos en UsiXML.

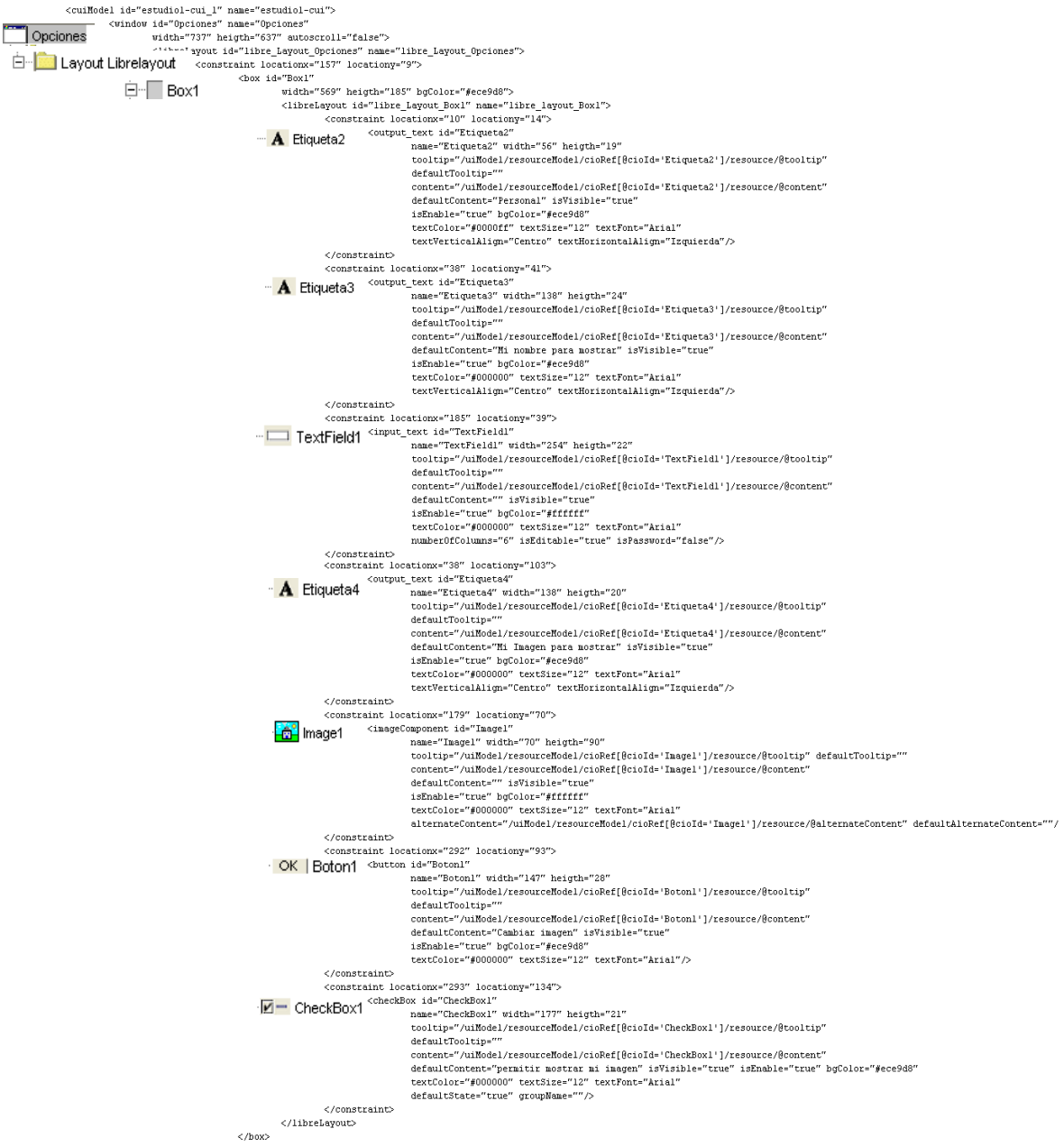


**Figura 4.5 Código UsiXML a nivel concreto correspondiente al contendor denominado Box1**

El primer elemento CIO que contiene *Window*, es el contenedor concreto de tipo Box denominado *"Box1"*, con una serie de atributos. Este nodo tiene siete nodos hijos, uno por componentes concretos que contiene: tres de tipo de tipo *Box*, uno de tipo *TextField*, uno de tipo *CheckBox*, uno de tipo *Button*, y otro de tipo *Image*. El nodo que especifica una *Label* es etiquetada como *output_text*, y el nodo que especifica un *TextField* es *input_text*. Cada uno de estos componentes concretos tiene una serie de atributos asociados. Los valores de los atributos son aquellos configurados por el usuario.

Hay alguna peculiaridad en los atributos que deber ser explicada. Primero que el valor del color, ya sea para el color de la fuente (*"foreground"*) o el color de fondo ("*background"*), está expresado en forma hexadecimal. Y la segunda peculiaridad es el valor del atributo *"content", "tooltip" o "alternateContent"* que como valor contienen una referencia al atributo y nodo del modelo de recursos donde se encuentra el verdadero valor configurado por el usuario.

El resto del código es similar dentro del modelo CUI, siendo fiel a la jerarquía de contenedores y componentes. Y cada componente es fiel a la configuración establecida por el usuario. Se puede destacar que el nodo, que especifica el componente concreto Table, tiene tantos nodos hijos *cell* como celdas tengan contenido, en este caso era un calendario y se configuró solamente los días de la semana, así que sólo hay siete nodos hijos etiquetados *cell*, cuyos campos *"xIndex"* e *"yIndex"* son los índices para identificar cada celda. Algo parecido ocurre con los nodos etiquetados "*listBox"* o "*comboBox"* que tiene tantos nodos hijo *item* como *items* agregados por el usuario al elemento List o ComboBox.

A continuación se muestra el resto de código para el modelo CUI.

```
:constraint locationx="157" locationy="207">
        <box id="Box2"
                width="569" heigth="119" bgColor="#ece9d8">
                <libreLayout id="libre_Layout_Box2" name="libre_layout_Box2">
                        <constraint locationx="7" locationy="11">
                                <output_text id="Etiqueta5"
                                        name="Etiqueta5" width="56" heigth="19"
                                        tooltip="/uiModel/resourceModel/cioRef[@cioId='Etiqueta5']/resource/@tooltip"
                                        defaultTooltip=""
                                        content="/uiModel/resourceModel/cioRef[@cioId='Etiqueta5']/resource/@content"
                                        defaultContent="General" isVisible="true"
                                        isEnable="true" bgColor="#ece9d8"
                                        textColor="#0000ff" textSize="12" textFont="Arial"
                                        textVerticalAlign="Centro" textHorizontalAlign="Centro"/>
                        </constraint>
                        <constraint locationx="31" locationy="32">
                                <output_text id="Etiqueta6"
                                        name="Etiqueta6" width="100" heigth="23"
                                        tooltip="/uiModel/resourceModel/cioRef[@cioId='Etiqueta6']/resource/@tooltip"
                                        defaultTooltip=""
                                        content="/uiModel/resourceModel/cioRef[@cioId='Etiqueta6']/resource/@content"
                                        defaultContent="Iniciar sesion" isVisible="true"
                                        isEnable="true" bgColor="#ece9d8"
                                        textColor="#000000" textSize="12" textFont="Arial"
                                        textVerticalAlign="Centro" textHorizontalAlign="Izquierda"/>
                        </constraint>
                        <constraint locationx="129" locationy="32">
                                <checkBox id="CheckBox2"
                                        name="CheckBox2" width="349" heigth="23"
                                        tooltip="/uiModel/resourceModel/cioRef[@cioId='CheckBox2']/resource/@tooltip"
                                        defaultTooltip=""
                                        content="/uiModel/resourceModel/cioRef[@cioId='CheckBox2']/resource/@content"
                                        defaultContent="Permitir inicio sesion al establecer conexion con internet" isVisible="true" isEnable="true" bgColor="#ece9d8"
                                        textColor="#000000" textSize="12" textFont="Arial"
                                        defaultState="False" groupName=""/>
                        </constraint>
                        <constraint locationx="128" locationy="73">
                                <checkBox id="CheckBox3"
                                        name="CheckBox3" width="340" heigth="23"
                                        tooltip="/uiModel/resourceModel/cioRef[@cioId='CheckBox3']/resource/@tooltip"
                                        defaultTooltip=""
                                        content="/uiModel/resourceModel/cioRef[@cioId='CheckBox3']/resource/@content"
                                        defaultContent="Poder ver las imagenes de los contactos" isVisible="true" isEnable="true" bgColor="#ece9d8"
                                        textColor="#000000" textSize="12" textFont="Arial"
                                        defaultState="false" groupName=""/>
                        </constraint>
                        <constraint locationx="31" locationy="73">
                                <output_text id="Etiqueta7"
                                        name="Etiqueta7" width="104" heigth="19"
                                        tooltip="/uiModel/resourceModel/cioRef[@cioId='Etiqueta7']/resource/@tooltip"
                                        defaultTooltip=""
                                        content="/uiModel/resourceModel/cioRef[@cioId='Etiqueta7']/resource/@content"
                                        defaultContent="Mis contactos" isVisible="true"
                                        isEnable="true" bgColor="#ece9d8"
                                        textColor="#000000" textSize="12" textFont="Arial"
                                        textVerticalAlign="Centro" textHorizontalAlign="Izquierda"/>
                        </constraint>
                </libreLayout>
        </box>
```

**Figura 4.6 Código UsiXML a nivel concreto correspondiente al contendor denominado Box2**

```
<constraint locationx="157" locationy="350">
        <box id="Box3"
                width="569" heigth="258" bgColor="#ece9d8">
                <libreLayout id="libre_Layout_Box3" name="libre_layout_Box3">
                        <constraint locationx="188" locationy="176">
                                <button id="Boton2"
                                        name="Boton2" width="26" heigth="28"
                                        tooltip="/uiModel/resourceModel/cioRef[@cioId='Boton2']/resource/@tooltip"
                                        defaultTooltip=""
                                        content="/uiModel/resourceModel/cioRef[@cioId='Boton2']/resource/@content"
                                        defaultContent="" isVisible="true"
                                        isEnable="true" bgColor="#ece9d8"
                                        textColor="#000000" textSize="12" textFont="Arial"/>
                        </constraint>
                        <constraint locationx="234" locationy="60">
                                <colorPicker id="Choosercolor1"
                                        name="Choosercolor1" width="312" heigth="187"
                                        tooltip="/uiModel/resourceModel/cioRef[@cioId='Choosercolor1']/resource/@tooltip"
                                        defaultTooltip="" isVisible="true"
                                        isEnable="true" bgColor="#ffffff"
                                        textColor="#000000" textSize="12" textFont="Arial"/>
                        </constraint>
                        <constraint locationx="234" locationy="29">
                                <output_text id="Etiqueta8"
                                        name="Etiqueta8" width="195" heigth="21"
                                        tooltip="/uiModel/resourceModel/cioRef[@cioId='Etiqueta8']/resource/@tooltip"
                                        defaultTooltip=""
                                        content="/uiModel/resourceModel/cioRef[@cioId='Etiqueta8']/resource/@content"
                                        defaultContent="Seleccionar color de la fuente" isVisible="true"
                                        isEnable="true" bgColor="#ece9d8"
                                        textColor="#000000" textSize="12" textFont="Arial"
                                        textVerticalAlign="Centro" textHorizontalAlign="Izquierda"/>
                        </constraint>
                        <constraint locationx="6" locationy="11">
                                <output_text id="Etiqueta9"
                                        name="Etiqueta9" width="76" heigth="21"
                                        tooltip="/uiModel/resourceModel/cioRef[@cioId='Etiqueta9']/resource/@tooltip"
                                        defaultTooltip=""
                                        content="/uiModel/resourceModel/cioRef[@cioId='Etiqueta9']/resource/@content"
                                        defaultContent="Mensajes" isVisible="true"
                                        isEnable="true" bgColor="#ece9d8"
                                        textColor="#0000ff" textSize="12" textFont="Arial"
                                        textVerticalAlign="Centro" textHorizontalAlign="Centro"/>
                        </constraint>
                        <constraint locationx="5" locationy="65">
                                <radioButton id="Radio_Boton1"
                                        name="Radio_Boton1" width="105" heigth="23"
                                        tooltip="/uiModel/resourceModel/cioRef[@cioId='Radio_Boton1']/resource/@tooltip"
                                        defaultTooltip=""
                                        content="/uiModel/resourceModel/cioRef[@cioId='Radio_Boton1']/resource/@content"
                                        defaultContent="Siempre" isVisible="true" isEnable="true" bgColor="#ece9d8"
                                        textColor="#000000" textSize="12" textFont="Arial"
                                        defaultState="true" groupName="mensaje"/>
                        </constraint>
                        <constraint locationx="7" locationy="36">
                                <output_text id="Etiqueta10"
                                        name="Etiqueta10" width="198" heigth="19"
                                        tooltip="/uiModel/resourceModel/cioRef[@cioId='Etiqueta10']/resource/@tooltip"
                                        defaultTooltip=""
                                        content="/uiModel/resourceModel/cioRef[@cioId='Etiqueta10']/resource/@content"
                                        defaultContent="Permitir mensajes instantaneos" isVisible="true"
                                        isEnable="true" bgColor="#ece9d8"
                                        textColor="#000000" textSize="12" textFont="Arial"
                                        textVerticalAlign="Centro" textHorizontalAlign="Centro"/>
                        </constraint>
                        <constraint locationx="6" locationy="96">
                                <radioButton id="Radio_Boton2"
                                        name="Radio_Boton2" width="160" heigth="24"
                                        tooltip="/uiModel/resourceModel/cioRef[@cioId='Radio_Boton2']/resource/@tooltip"
                                        defaultTooltip=""
                                        content="/uiModel/resourceModel/cioRef[@cioId='Radio_Boton2']/resource/@content"
                                        defaultContent="Solo (Modo Conectado)" isVisible="true" isEnable="true" bgColor="#ece9d8"
                                        textColor="#000000" textSize="12" textFont="Arial"
                                        defaultState="False" groupName="mensaje"/>
                        </constraint>
                        <constraint locationx="6" locationy="126">
                                <radioButton id="Radio_Boton3"
                                        name="Radio_Boton3" width="250" heigth="31"
                                        tooltip="/uiModel/resourceModel/cioRef[@cioId='Radio_Boton3']/resource/@tooltip"
                                        defaultTooltip=""
                                        content="/uiModel/resourceModel/cioRef[@cioId='Radio_Boton3']/resource/@content"
                                        defaultContent="Nunca en modo No disponible" isVisible="true" isEnable="true" bgColor="#ece9d8"
                                        textColor="#000000" textSize="12" textFont="Arial"
                                        defaultState="False" groupName="mensaje"/>
                        </constraint>
                        <constraint locationx="11" locationy="210">
                                <input_text id="TextField2"
                                        name="TextField2" width="194" heigth="26"
                                        tooltip="/uiModel/resourceModel/cioRef[@cioId='TextField2']/resource/@tooltip"
                                        defaultTooltip=""
                                        content="/uiModel/resourceModel/cioRef[@cioId='TextField2']/resource/@content"
                                        defaultContent="C.\ Documents and Settings\personal" isVisible="true"
                                        isEnable="true" bgColor="#ffffff"
                                        textColor="#000000" textSize="12" textFont="Arial"
                                        numberOfColumns="6" isEditable="false" isPassword="false"/>
                        </constraint>
                        <constraint locationx="9" locationy="182">
                                <output_text id="Etiquetall"
                                        name="Etiquetall" width="199" heigth="17"
                                        tooltip="/uiModel/resourceModel/cioRef[@cioId='Etiquetall']/resource/@tooltip"
                                        defaultTooltip=""
                                        content="/uiModel/resourceModel/cioRef[@cioId='Etiquetall']/resource/@content"
                                        defaultContent="Guarda mis conversaciones en" isVisible="true"
                                        isEnable="true" bgColor="#ece9d8"
                                        textColor="#000000" textSize="12" textFont="Arial"
                                        textVerticalAlign="Centro" textHorizontalAlign="Izquierda"/>
                        </constraint>
                </libreLayout>
        </box>
</constraint>
```

**Figura 4.7 Código UsiXML a nivel concreto correspondiente al contendor denominado Box3**

```
            <constraint locationx="5" locationy="111">
                    <table id="Table1"
                            name="Table1" width="136" heigth="81"
                            tooltip="/uiModel/resourceModel/cioRef[@cioId='Table1']/resource/@tooltip"
                            defaultTooltip="" isVisible="true"
                            isEnable="true" bgColor="#ffffff"
                            textColor="#000000" textSize="12" textFont="Arial" xsize="5" ysize="7">
                            <cell id="f0c0" name="f0c0" defaultContent="L"
                                    xIndex="0" yIndex="0"/>
                            <cell id="f0c1" name="f0c1" defaultContent="M"
                                    xIndex="0" yIndex="1"/>
                            <cell id="f0c2" name="f0c2" defaultContent="X"
                                    xIndex="0" yIndex="2"/>
                            <cell id="f0c3" name="f0c3" defaultContent="J"
                                    xIndex="0" yIndex="3"/>
                            <cell id="f0c4" name="f0c4" defaultContent="V"
                                    xIndex="0" yIndex="4"/>
                            <cell id="f0c5" name="f0c5" defaultContent="S"
                                    xIndex="0" yIndex="5"/>
                            <cell id="f0c6" name="f0c6" defaultContent="D"
                                    xIndex="0" yIndex="6"/>
                    </table>
            </constraint>
            <constraint locationx="5" locationy="78">
                    <output_text id="Etiqueta12"
                            name="Etiqueta12" width="63" heigth="19"
                            tooltip="/uiModel/resourceModel/cioRef[@cioId='Etiqueta12']/resource/@tooltip"
                            defaultTooltip=""
                            content="/uiModel/resourceModel/cioRef[@cioId='Etiqueta12']/resource/@content"
                            defaultContent="Calendario" isVisible="true"
                            isEnable="true" bgColor="#ffffff"
                            textColor="#000000" textSize="12" textFont="Arial"
                            textVerticalAlign="Centro" textHorizontalAlign="Centro"/>
            </constraint>
            <constraint locationx="5" locationy="78">
                    <output_text id="Etiqueta12"
                            name="Etiqueta12" width="63" heigth="19"
                            tooltip="/uiModel/resourceModel/cioRef[@cioId='Etiqueta12']/resource/@tooltip"
                            defaultTooltip=""
                            content="/uiModel/resourceModel/cioRef[@cioId='Etiqueta12']/resource/@content"
                            defaultContent="Calendario" isVisible="true"
                            isEnable="true" bgColor="#ffffff"
                            textColor="#000000" textSize="12" textFont="Arial"
                            textVerticalAlign="Centro" textHorizontalAlign="Centro"/>
            </constraint>
            <constraint locationx="5" locationy="205">
                    <imageComponent id="Image4"
                            name="Image4" width="143" heigth="361"
                            tooltip="/uiModel/resourceModel/cioRef[@cioId='Image4']/resource/@tooltip" defaultTooltip=""
                            content="/uiModel/resourceModel/cioRef[@cioId='Image4']/resource/@content"
                            defaultContent="chat1.JPG" isVisible="true"
                            isEnable="true" bgColor="#ffffff"
                            textColor="#000000" textSize="12" textFont="Arial"
                            alternateContent="/uiModel/resourceModel/cioRef[@cioId='Image4']/resource/@alternateContent" defaultAlternateContent=""/>
            </constraint>
            <constraint locationx="5" locationy="10">
                    <listBox id="List1"
                            name="List1" width="145" heigth="58"
                            tooltip="/uiModel/resourceModel/cioRef[@cioId='List1']/resource/@tooltip"
                            defaultTooltip=""
                            isVisible="true"
                            isEnable="true" bgColor="#ffffff"
                            textColor="#0000ff" textSize="12" textFont="Arial">
                            <item id="List1_item1"
                                    content="/uiModel/resourceModel/cioRef[@cioId='List1_item1']/resource/@content"/>
                            <item id="List1_item2"
                                    content="/uiModel/resourceModel/cioRef[@cioId='List1_item2']/resource/@content"/>
                            <item id="List1_item3"
                                    content="/uiModel/resourceModel/cioRef[@cioId='List1_item3']/resource/@content"/>
                    </listBox>
            </constraint>
    </libreLayout>
</window>
```

**Figura 4.8 Código UsiXML a nivel concreto correspondiente a los componentes de Opciones**

El único código que falta por explicar es código del modelo de recursos. En el capítulo anterior se mencionó que aunque el código generado muestra un modelo de contexto, éste es siempre el mismo y por defecto se ha establecido que el modelo de usuario, nodo *"userStereotype"*, especifique al usuario con un perfil español. Al igual que el modelo AUI, EgiuXML deja la posibilidad para versiones posteriores especificar de manera más profunda el modelo de contexto.

El modelo de recursos tiene tantos nodos hijos como elementos CIO con contenido agregado; los atributos o propiedades de los elementos CIO que se pueden considerar como recurso son *text, tooltipText, icon, file, alternateContent*. En el momento que se le asigna un valor a estos atributos debe aparecer un nodo etiquetado cioRef en el modelo de recursos con el campo *cioId* cuyo valor sería el identificador del elemento que se le ha agregado el contenido. Los *item* añadidos a *List* y *ComboBox* generan nodos *cioref* independientemente del elemento *List* o *ComboBox* que los contiene. De hecho si se hace un estudio exhaustivo al modelo CUI definido en UsiXML se consideran componentes concretos.

```
        </cuiModel>
        <contextModel id="estudiol-contextModel_1" name="estudiol-contextModel">
                <context id="estudiol-context-es_ES_1" name="estudiol-context-es_ES">
                        <userStereotype id="estudiol-stes_ES_1" language="es_ES" stereotypeName="estudiol-stes_ES"/>
                        <platform id="estudiol-platform_" name="estudiol-platform"/>
                        <environment id="estudiol-env_1" name="estudiol-env"/>
                </context>
        </contextModel>
        <resourceModel id="estudiol-res_29" name="estudiol-res">
                <cioRef cioId="Etiqueta2">
                        <resource content="Personal" tooltip="" contextId="estudiol-context-es_ES_29"/>
                </cioRef>
                <cioRef cioId="Etiqueta3">
                        <resource content="Mi nombre para mostrar" tooltip="" contextId="estudiol-context-es_ES_29"/>
                </cioRef>
                <cioRef cioId="Etiqueta4">
                        <resource content="Mi Imagen para mostrar" tooltip="" contextId="estudiol-context-es_ES_29"/>
                </cioRef>
                <cioRef cioId="Botonl">
                        <resource content="Cambiar imagen" tooltip="" contextId="estudiol-context-es_ES_29"/>
                </cioRef>
                <cioRef cioId="CheckBoxl">
                        <resource content="permitir mostrar mi imagen" tooltip="" contextId="estudiol-context-es_ES_29"/>
                </cioRef>
                <cioRef cioId="Etiqueta5">
                        <resource content="General" tooltip="" contextId="estudiol-context-es_ES_29"/>
                </cioRef>
                <cioRef cioId="Etiqueta6">
                        <resource content="Iniciar sesion" tooltip="" contextId="estudiol-context-es_ES_29"/>
                </cioRef>
                <cioRef cioId="CheckBox2">
                        <resource content="Permitir inicio sesion al establececer conexion con internet" tooltip="" contextId="estudiol-context-es_ES
                </cioRef>
                <cioRef cioId="CheckBox3">
                        <resource content="Poder ver las imagenes de los contactos" tooltip="" contextId="estudiol-context-es_ES_29"/>
                </cioRef>
                <cioRef cioId="Etiqueta7">
                        <resource content="Mis contactos" tooltip="" contextId="estudiol-context-es_ES_29"/>
                </cioRef>
                <cioRef cioId="Boton2">
                        <resource tooltip="" Icon="fichero.PNG" contextId="estudiol-context-es_ES_29"/>
                </cioRef>
                <cioRef cioId="Etiqueta8">
                        <resource content="Seleccionar color de la fuente" tooltip="" contextId="estudiol-context-es_ES_29"/>
                </cioRef>
                <cioRef cioId="Etiqueta9">
                        <resource content="Mensajes" tooltip="" contextId="estudiol-context-es_ES_29"/>
                </cioRef>
                <cioRef cioId="Radio_Botonl">
                        <resource content="Siempre" tooltip="" contextId="estudiol-context-es_ES_29"/>
                </cioRef>
                <cioRef cioId="Etiquetal0">
                        <resource content="Permitir mensajes instantaneos" tooltip="" contextId="estudiol-context-es_ES_29"/>
                </cioRef>
                <cioRef cioId="Radio_Boton2">
                        <resource content="Solo (Modo Conectado)" tooltip="" contextId="estudiol-context-es_ES_29"/>
                </cioRef>
                <cioRef cioId="Radio_Boton3">
                        <resource content="Nunca en modo No disponible" tooltip="" contextId="estudiol-context-es_ES_29"/>
                </cioRef>
                <cioRef cioId="TextField2">
                        <resource content="C.\ Documents and Settings\personal" tooltip="" contextId="estudiol-context-es_ES_29"/>
                </cioRef>
                <cioRef cioId="Etiquetall">
                        <resource content="Guarda mis conversaciones en" tooltip="" contextId="estudiol-context-es_ES_29"/>
                </cioRef>
                <cioRef cioId="Etiquetal2">
                        <resource content="Calendario" tooltip="" contextId="estudiol-context-es_ES_29"/>
                </cioRef>
                <cioRef cioId="Image4">
                        <resource content="chatl.JPG" tooltip="" contextId="estudiol-context-es_ES_29"/>
                </cioRef>
                <cioRef cioId="Listl_iteml">
                        <resource content="Personal" contextId="estudiol-context-es_ES_29"/>
                </cioRef>
                <cioRef cioId="Listl_item2">
                        <resource content="General" contextId="estudiol-context-es_ES_29"/>
                </cioRef>
                <cioRef cioId="Listl_item3">
                        <resource content="Mensajes" contextId="estudiol-context-es_ES_29"/>
                </cioRef>
        </resourceModel>
</uiModel>
```

**Figura 4.9 Código UsiXML del modelo de recursos correspondiente al Caso de Estudio1**

Como se observa en la Figura 4.9, el contenido del componente Image, o el valor del campo *Icon* del Button denominado "Boton2" es el nombre del fichero de la imagen (.jpg, .png o .gif), no especifica una dirección absoluta ya que estas imágenes se encuentran en la carpeta .src/resourceModel del directorio de la herramienta. Durante la configuración de los elementos, el usuario selecciona una imagen para agregársela como icono o como simple imagen al componente, el fichero de imagen es copiado a dicho directorio. De esta manera al trabajar en otra máquina y recuperar el modelo, sólo hay que preocuparse en colocar los recursos en esta carpeta de la otra máquina, sin más dificultad.

Es cierto que el código generado podía ser algo más sencillo o más intuitivo, pero para que EgiuXML sea lo más fiel posible al estándar de UsiXML, y para que las innovaciones de esta herramienta ayuden a otras herramientas de funcionalidad parecida, y no eche por tierra el trabajo de otros, se ha decidido que el código sea exactamente de esta manera.

## 4.2 CASO DE ESTUDIO 2

En el apartado presente se va a mostrar un ejemplo del potencial que tiene este programa. En el capítulo 3 se describió detalladamente cómo funciona la herramienta, en el apartado 4.1 se describió la creación de una interfaz de configuración típica que se puede encontrar en cualquier programa, en concreto fue una pantalla de opciones de un software de Chat. Ahora bien, esta herramienta puede ir aún algo más lejos que hacer interfaces propias de formularios. En concreto va a consistir en una interfaz de una posible página Web de cine. En estás páginas, en ocasiones muestran *trailers* de películas, dejan un espacio para la publicidad, además este espacio suele ir variando a lo largo del tiempo, hay catálogos, buscadores, un menú de registro, etc. Son páginas donde el aspecto visual es prioritario para captar el mayor número de internautas. Se va a demostrar que EgiuXML es capaz de desarrollar las interfaces de este tipo de páginas a nivel concreto y obtener la especificación en UsiXML correspondiente.

La interfaz desarrollada no se ha basado en ninguna página en concreto sino en muchas, ya que parte de la finalidad del desarrollo de este caso de estudio es completar el conocimiento sobre la herramienta usando elementos CIO y propiedades todavía no vistos en profundidad. De modo que para no utilizar siempre los mismos componentes, se ha hecho una especie de mezcla de diferentes web, sin dejar un ápice de realismo.

A continuación se muestra la interfaz de usuario concreta creada para este caso de estudio:

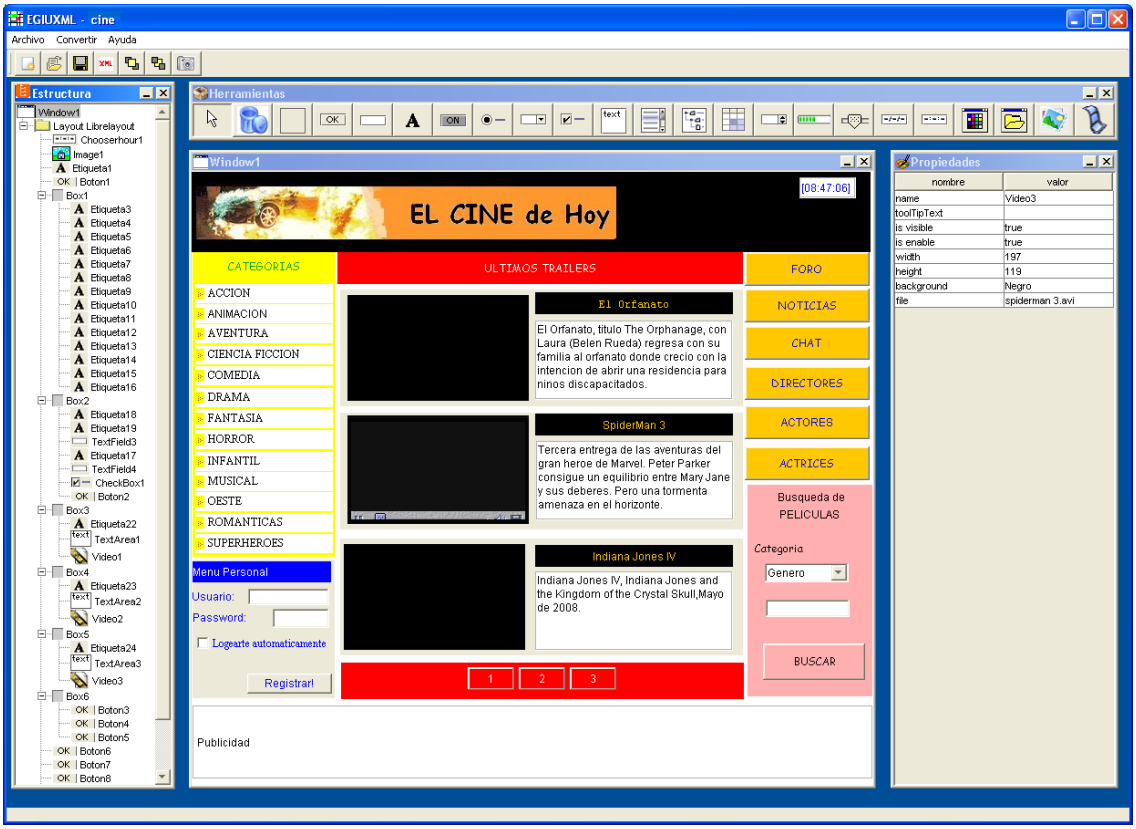


**Figura 4.10 Interfaz de usuario concreta de una página Web de cine**

La interfaz se compone, como es lógico de un contenedor Window, configurado con *LibreLayout*, que gracias a su flexibilidad permite cualquier distribución de los componentes, en este caso la interfaz tiene una distribución muy parecida a la que se establecería con un layout *BorderLayout*, layout típico que aparece en algunos editores de interfaces. Pero incluso con *LibreLayout* es más fácil configurar cada sección a gusto del usuario.

Empezando desde la parte de arriba de la interfaz, hay un componente *Image*, que se le ha agregado una imagen de un título inventado, que simboliza el titulo de la página o la marca de la empresa asociada. En la misma zona, de manera superpuesta hay un componente *ChooserHour*, que muestra por defecto la hora en la cual se agregó el componente a la interfaz, pero este contenido se puede modificar como desee el usuario, quedándose almacenado o reflejado en la especificación.

En la parte derecha hay dos contenedores Box, el de más arriba contiene catorce *Labels*, una para el titulo de la sección, con el texto centrado y fondo amarillo. Las otras

trece para cada categoría de películas, con letra *"times new roman"* y el texto alineado a la izquierda, cada una de estas *Label* se le ha añadido el mismo icono a la izquierda del texto. En la carpeta de recursos del directorio de la herramienta, dicha imagen sólo va ser guardada una vez, evitando tener tantas copias de un mismo archivo. Este contenedor podía configurarse con *layout Vertical*, o *LibreLayout*, según sean los márgenes que desea el usuario que tenga el Box con respecto a los componentes, se puede decir que con layout Vertical quedaría mejor.

En el Box de abajo, es un pequeño formulario que suele aparecer para que un usuario se logué, este contendor contiene tres componentes *Label*, todos ellos configurados con el texto alienado a la izquierda. Dos *TextField*, uno de ellos con el atributo *"isPassword"* a *"true*" de manera que si se escribe en el casillero del *TextField* aparecerán '*' por cada carácter que se escriba, de esta manera esta herramienta también permite utilizar el típico *widget PasswordField* pero aprovechando el componente *TextField*. Además se utiliza un *Button* y un *CheckBox* sin ninguna peculiaridad especial.

En la zona de debajo de la interfaz hay un componente *Image* que suelen utilizarlo para la publicidad. Como en el momento de desarrollo de una interfaz no se sabe que publicidad se va a mostrar, en el nivel concreto hay una propiedad para este componente que es *"alternative text"* (*alternateContent* en la especificación) que muestra un texto en el momento que no tenga agregado ningún archivo de imagen, y dejar de mostrarse cuando ya hay una imagen añadida. Esto facilita las tareas al diseñador responsable de finiquitar la interfaz de usuario final (FUI) cuando sepa la publicidad que ha de poner.

En la parte de la derecha hay cinco componentes *Button* que muestra los servicios que ofrece la Web. Abajo un *Box*, que es un panel de búsqueda de películas, contiene dos *Label*, un *ComboBox* cuyos ítem agregados son las distintas modalidades de búsqueda, un *TextField* para que introduzca la palabra clave de búsqueda, y un *Button* cuyo color de fondo se ha configurado igual que el de su contenedor.

Únicamente queda por explicar la parte central, en la parte de arriba del centro hay una *Label* grande de color de fondo rojo, y la alineación del texto configurada es central tanto vertical como horizontalmente. En la parte de abajo del centro hay un rectángulo muy parecido al anterior, pero se trata de un contenedor *Box*, con layout *Horizotal*, por eso se han quedado tan bien centrados los tres *Button* añadidos al contendor, estos botones serían útiles para acceder a otras galerías de *trailers*.

Lo que supone el centro de la interfaz son contendores *Box*, configurados con *layout LibreLayout*. Los tres contendores contienen un *Label*, el atributo texto tiene como valor el titulo de la película. Un componente *TextArea* donde hay una breve sinopsis de la película, para configurar el texto por defecto que se debe mostrar en la interfaz hay que pulsar en el campo de valor de la tabla de propiedades para el atributo *defaultText* y se muestra por pantalla la ventana interna de la Figura 4.11, además para que el texto aparezca dentro del recuadro y aproveche las líneas posibles de *TextArea* independientemente de lo saltos de línea, el atributo "*isWordWrapped*" se pone "*true*" y para evitar que aparezcan palabras entre dos líneas el atributo "*forceWordWrapped*" también se pone a "*true*", y como no interesa que el usuario escriba en esta área el valor de "*is editable*" se pone a "*false*".


**Figura 4.11 Ventana de configuración del texto de un componente TextArea**

El último componente añadido es un *VideoComponet*, ya se mencionó que un componente poco utilizado en las herramientas de diseño de interfaces, mientras que EgiuXML está disposición del usuario como cualquier otro elemento CIO. En él se van a reproducir los videos de los *trailers* de las películas, y a cada *VideoComponente* se le va a añadir como contenido el fichero de video que ha de reproducir. Para evitar que el programa se ralentice lo mínimo posible, el video solo se reproduce cuando el componente es seleccionado, en ese instante aparece el panel de controles de reproducción dentro de las dimensiones del componente. Hay que entender que esta herramienta no es un reproductor de video, es una aplicación para diseñar interfaces y uno de sus fines es que el diseñador sepa como es la reproducción de un video en la interfaz grafica que diseña.

A continuación se muestra el código en UsiXML de la especificación de uno de estos contenedores, que a primera vista parece ser la parte más interesante del código del modelo concreto generado, ya que sería del mismo estilo que el explicado en el apartado anterior pero más extenso por el número de componentes añadidos.

En la Figura 4.12, se observa como el nodo que especifica el elemento CIO TextArea está etiquetado como *"input_text"* igual que si se tratase de un elemento

*TextField*, la manera de distinguirlos por el código es porque aparecen tres atributos *"numberOfLines", "isWordWrapped"* y *"forcewordWrapped*" en el componente *TextArea* mientras que dichos atributos no se encuentran en *TextField*, por eso cuando se genera un modelo al abrir una especificación a partir de un fichero ".uxml", se genera el componente correcto gracias a estos tres atributos.

```
<constraint locationx="163" locationy="266">
      <box id="Box4" width="442" heigth="128" bgColor="#ece9d8">
            <libreLayout id="libre_Layout_Box4" name="libre_layout_Box4">
                  <constraint locationx="214" locationy="7">
                        <output_text id="Etiqueta23" name="Etiqueta23" width="217" heigth="24"
                              tooltip="/uiModel/resourceModel/cioRef[@cioId='Etiqueta23']/resource/@tooltip"
                              defaultTooltip=""
                              content="/uiModel/resourceModel/cioRef[@cioId='Etiqueta23']/resource/@content"
                              defaultContent="Indiana Jones IV" isVisible="true" isEnable="true" bgColor="#000000"
                              textColor="#ffc800" textSize="12" textFont="Arial"
                              textVerticalAlign="Centro" textHorizontalAlign="Centro"/>
                  </constraint>
                  <constraint locationx="214" locationy="36">
                        <input_text id="TextArea2" name="TextArea2" width="217" heigth="87"
                              tooltip="/uiModel/resourceModel/cioRef[@cioId='TextArea2']/resource/@tooltip"
                              defaultTooltip=""
                              content="/uiModel/resourceModel/cioRef[@cioId='TextArea2']/resource/@content"
                              defaultContent="Indiana Jones IV, Indiana Jones and the Kingdom of the Crystal Skull,Mayo de 2008. "
                              isVisible="true" isEnable="true" bgColor="#ffffff" textColor="#000000" textSize="12" textFont="Arial"
                              numberOfColumns="12" numberOfLines="5"
                              isWordWrapped="true" forcewordWrapped="true" isEditable="false"/>
                  </constraint>
                  <constraint locationx="8" locationy="8">
                        <videoComponent id="Video2" name="Video2" width="199" heigth="116"
                              tooltip="/uiModel/resourceModel/cioRef[@cioId='Video2']/resource/@tooltip"
                              defaultTooltip="" isVisible="true" isEnable="true" bgColor="#000000"/>
                  </constraint>
            </libreLayout>
      </box>
</constraint>
```

**Figura 4.12 Código UsiXML a nivel concreto correspondiente al contenedor denominado Box4**

Para culminar este caso de estudio la especificación en UsiXML se puede almacenar en un fichero ".uxml" que no supera los 65KB, cuyo nombre será el asignado al modelo.

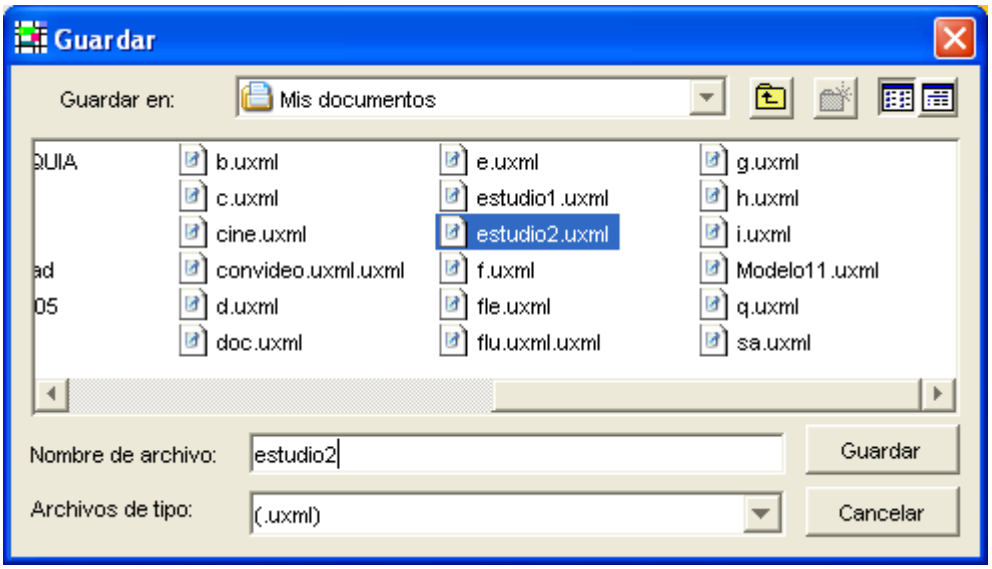


**Figura 4.13 Cuadro de diálogo para almacenar el código UsiXML del modelo del caso de estudio 2**

## 4.3 CONCLUSIONES

Tras la realización de los dos ejemplos explicados en este capítulo se puede destacar tres aspectos claves de la herramienta EgiuXML.

El primero, es que se trata de una herramienta para desarrollar y diseñar interfaces de usuario actuales, satisfaciendo las necesidades de los usuarios de hoy en día. Las interfaces que permite generar no son de programas antiguos y desfasados, son interfaces atractivas, completas, y fáciles de asimilar y de aprender, es decir las interfaces generadas ofrecen todo aquello que es exigido en la industria de la informática y en el mercado.

El segundo aspecto importante a destacar, es que el diseñador que utiliza EgiuXML tiene una idea muy clara de lo que posteriormente puede ser la interfaz final. El entorno facilitado es WYSIWYG (*what you see it what you get*), es decir, sin necesidad de pulsar un botón de atajo para mostrar una visión previa de la interfaz diseñada el usuario ve lo que está especificando grafica y continuamente, ya que el aspecto de la interfaz final se le muestra en el mismo panel principal mientras está diseñándolo a nivel concreto. Esto supone para el diseñador un ahorro de tiempo considerable.

El tercer e importante aspecto, es que se ha demostrado cómo la especificación del modelo en UsiXML es completamente fiel a la configuración gráfica establecida por el usuario. Además es indiferente el punto en el que se encuentra el desarrollo de la interfaz para poder almacenar su especificación cuando se desee, y además va a tardar lo mismo, se generará un fichero más o menos extenso en función del número de componentes añadidos, pero el tiempo en generarlo es inmediato, satisfaciendo los tiempos exigidos en los requisitos del sistema. Pudiendo volver a editar y reutilizar el modelo desarrollado con solo tener el fichero almacenado previamente.

En el capitulo siguiente se hará un extracto de estas conclusiones junto con las demás obtenidas del resto del trabajo realizado en este proyecto final de carrera.

# Capítulo 5: Conclusiones y trabajos futuros

Una vez terminada la presentación del proyecto realizado, llega el momento de hacer un compendio de las conclusiones obtenidas del trabajo realizado, evaluando si los objetivos propuestos en el Capítulo 1 han sido alcanzados. En el segundo apartado de este capítulo se mencionarán posibles trabajos futuros que tengan como punto de partida el trabajo y estudio realizado en este proyecto

## 5.1 CONCLUSIONES

La principal pretensión de este proyecto ha sido contribuir a la especificación declarativa de los modelos asociados con una interfaz de usuario, centrándose en la especificación de interfaces de usuario a nivel concreto tal y como estaba definida en UsiXML.

Para hacer factible esta contribución al paradigma de desarrollo de interfaces basado en modelos, se planificaron unos objetivos que había que abordarlos detallada y secuencialmente.

A continuación repasamos si los objetivos conseguidos con la realización de este proyecto final de carrera:

En este proyecto nos hemos familiarizado con el lenguaje de marcado extensible de interfaces de usuario UsiXML, éste marcó las pautas que deben seguir los ficheros generados por la aplicación resultante de la elaboración del proyecto. Una descripción de dicho lenguaje, especialmente de aquello directamente ligado a este proyecto puede encontrarse en el capítulo segundo de este documento, véanse los apartados 2.2.9 y 2.3.1.

En este proyecto se han estudiado las herramientas disponibles en el ámbito del desarrollo basado en modelos. Especial consideración se ha prestado a las herramientas afines a la principal motivación de este proyecto. En este sentido, se realizó un estudio de las herramientas más representativas destinadas a dar soporte a la especificación de los distintos modelos, en especial se profundizó en IdealXML y en GrafiXML (Véase en el apartado 2.4 del capítulo 2).

En este proyecto se ha profundizado y se han puesto en práctica conocimientos relacionados con la programación Orientada a Objetos, para ello se ha determinado qué

lenguaje y entorno utilizar para abordar el desarrollo de la herramienta asociada a este proyecto  Finalmente se opto por elegir Java y el entorno Borland JBuilder para el desarrollo de la aplicación. La justificación de la elección se debe a la experiencia que se tiene de la herramienta obtenida a lo largo de la carrera, y a la agilidad de programar y aprender el lenguaje Java.

En este proyecto final de carrera se ha desarrollado un producto software que permite la generación de una interfaz de usuario a nivel concreto y guardarlo en formato UsiXML o una variante de este lenguaje  El resultado más tangible de este proyecto es una herramienta software EGIUXML que permite la especificación de interfaces de usuario a nivel concreto siguiendo las recomendaciones propuestas en UsiXML.

El procedimiento seguido para la elaboración del entorno anterior ha tenido en cuenta criterios de calidad, es decir, el producto desarrollado cumple los requisitos establecidos (tanto funcionales como no funcionales) y los estándares asociados. En este sentido, EGIUXML satisface tanto la funcionalidad perseguida como el modo de facilitarla. (Véase capítulo 3 y 4)

En este proyecto se ha desarrollado un producto software fácil de usar, es decir, que los usuarios aprendan pronto a manejarlo, e intuyan el funcionamiento de cada elemento de la interfaz de usuario. En los capítulos tercero y cuarto se ha presentado la herramienta de forma estática y se han utilizado diferentes supuestos prácticos que demuestran su uso.

Además, se ha proporcionado un producto software fácil de mantener y bien documentado, para que posteriormente pueda ser modificado sin dificultad.

En la realización de la herramienta se siguió un proceso de desarrollo del software desde un punto de vista ingenieríl, cuyos mecanismos de proceso están dictados por el Proceso Unificado (Véase apartado 3.1 del capítulo 3). Además los ficheros fuentes han sido comentados meticulosamente.

En función del trabajo realizado, se entiende, que se ha facilitado un producto software que incorpora novedades  importantes respecto a programas con los que pueda lograrse una funcionalidad parecida, especialmente GrafiXML. Determinando funcionalidad adicional y características deseables respecto a las que dichos entornos disponibles ofrecen

EGIUXML aporta beneficios como la semiautomatización de la creación de la interfaz de usuario, o la reutilización de las especificaciones a nivel concreto almacenadas en ficheros. En este entorno destaca la obstinación por considerar los intereses del diseñador de interfaces. En este sentido, el proyectando ha volcado sus esfuerzos en el desarrollo de una aplicación para que el usuario (el diseñador de interfaces) cree una interfaz de usuario concreta (CUI) con la máxima flexibilidad y usabilidad.

Fruto de la realización de este proyecto el proyectando ha comprendido la importancia de un modelo de Interfaz de Usuario Concreta (CUI) para modelar o abstraer una Interfaz de Usuario Gráfica (GUI) generada por un diseñador de IU.

El entorno desarrollado facilita la labor de especificación de interfaces, Consiguiendo y facilitando un nivel de abstracción que ayuda al objetivo de reutilización de desarrollos de interfaces anteriores y que hacen posible el desarrollo de una nueva interfaz de usuario de forma sistemática, disminuyendo el trabajo lo que significa la obtención de beneficios económicos.

Gracias a la realización de este proyecto se ha comprendido el lugar que ocupa el concepto de modelo en la especificación de productos software. Se ha estudiado y trabajado con UsiXML y se han entendido mejor otras propuestas más propias de ingeniería del software como es MDA.

Hay diversos proyectos, que al igual que ocurre con el que se se presenta en este documento, siguen el desarrollo de interfaces de usuario basadas en modelos propuestos en UsiXML. Se pueden mencionar algunas de estas herramientas o proyectos ya finalizados: IdealXML, TransformiXML, FlashiXML, VisualiXML, KnowiXML, GrafiXML, VisiXML, SketchiXML, FormiXML y ReversiXML. En este proyecto se ha descrito algunas de ellas profundamente debido por la relevancia que tenían respecto a nuestra herramienta, pero se puede obtener información de todas estas aplicaciones en la página www.usixml.org.

En la Figura 5.1 puede observarse donde se situaría cada una de las aplicaciones anteriores en la jerarquía seguida en UsiXML, incluida la aplicación desarrollada por el proyecto dedicada específicamente al nivel concreto:

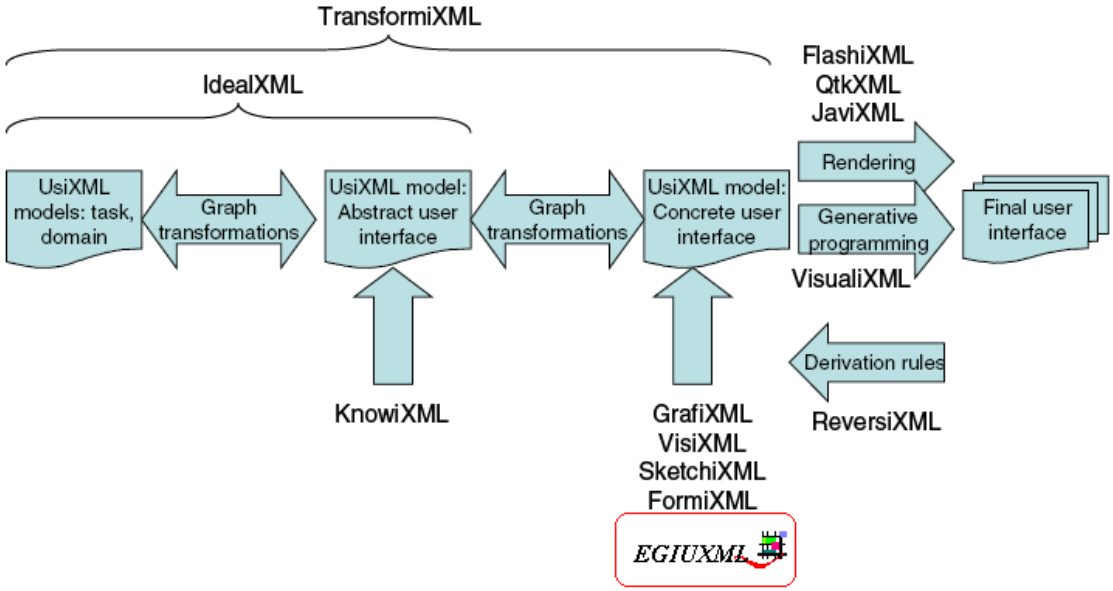


**Figura 5.1 Estructura de las herramientas de UsiXML según la clasificación de MDA**

El trabajo realizado en este proyecto siempre ha sido enfocado a colaborar en el proceso de elaboración de software de calidad donde se considere el desarrollo de la interfaz de usuario de una forma sistemática, uniendo tiempo y todo el esfuerzo posible al conjunto de herramientas ya construidas anteriormente que batallan con el mismo fin.

## 5.2 TRABAJOS FUTUROS

Durante la elaboración del proyecto se han ido identificando posibles trabajos futuros relacionados con este proyecto. Dichos trabajos se proponen para que posteriormente puedan ser realizados, y así ir progresando en el desarrollo de interfaces de usuario. Se proponen los siguientes trabajos:

- La realización de una aplicación que partiendo del entorno desarrollado, pueda realizar la transformación automática de la interfaz de usuario editada gráficamente a una interfaz de usuario abstracta, tal como se ha conseguido la especificación de la interfaz de usuario concreta, guardando en un fichero las especificaciones de los diferentes modelos descritos en UsiXML. También la mejora de la aplicación puede incluir un editor de contexto en el cual se puede añadir, eliminar o editar contextos para la aplicación del usuario. Aunque es un aspecto desmesuradamente ambicioso, EgiuXML puede ser el origen, o formar

parte, de ese entorno integrado tan deseado que permita la creación de interfaces de usuario sustituyendo todos los editores utilizados en UsiXML.

- Realizar una posible ampliación de la herramienta EgiuXML a nivel concreto, añadiendo algún objeto gráfico gráfico (widgets) específico que utilizaría algún editor de construcción y no se ha contemplado en este programa, así como poder manipular algunas propiedades más de los objetos utilizados. Sería muy interesante permitir a la herramienta EgiuXML que copiara y pegara elementos del portapapeles, ya añadidos gráficamente en diferentes modelos creados por esta aplicación, permitiendo de esa manera la reutilización de partes de diferentes interfaces para diseñar nuevas interfaces.

- La realización de otra posible ampliación de la herramienta EgiuXML, que consistiría en un programa que permita la modificación del código en UsiXML manipulando el propio código, su compilación y su posterior visualización con la paleta de elementos CIO que ofrece esta herramienta.

- La realización de una herramienta, capaz de recoger una especificación en el lenguaje UsiXML, aplicarle una serie de reglas de adaptación, y convertir la especificación en una presentación final en el lenguaje Java y visualizarlo. Este punto ya está siendo extendido por otro proyecto final de carrera.

- De la misma manera que se está elaborando estudios sobre el reconocimiento de imágenes de diferentes tipos (texto escrito manualmente o rasgos de la cara de una persona, etc.) sería interesante en el desarrollo de interfaces la realización de una herramienta que partiendo de una o varias imágenes de la interfaz de usuario de una aplicación, sea capaz de analizar la imagen y obtener la interfaz de usuario concreta asociada a la aplicación

- La realización de un programa que partiendo de una interfaz de usuario concreta, o en el mismo momento que se está diseñando, calcule el grado de usabilidad que tiene la interfaz que se está diseñando. El grado de usabilidad sería en función de la colocación, la cantidad y el tipo de elementos que componen la interfaz de usuario

## Bibliografía


(Abrams et al., 1999) Abrams, M., Phanouriou, C., Batongbacal, A. L., Williams, S. M., & Shuster, J. E. (1999). UIML: An appliance-independent XML user interface language. WWW8/ Computer Networks.

(Cachero, 2003) Cachero, C. (2003). OO-H: Una extensión a los métodos OO para el modelado y generación automática de interfaces hipermediales. Tesis doctoral. Universidad de Alicante.

(Calvary et al, 2003) Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., Vanderdonckt, J. (2003). A Unifying Reference Framework for Multi-Target User Interfaces. Interacting with Computers. Vol.15, No. 3, pages 289-308.

(ConcretUserInterface, 2007) Definición de interfaces de usuario concretas. (2007) http://www.awprofessional.com/articles/article.asp?p=167852&seqNum=7&r1=1 (7/10/07)

(Fincher, 2003) Fincher, S. (2003) Perspectives on HCI patterns: concepts and tools (introducing PLML). Interfaces, (56): pages 26-28.

(Gamma et al., 1994) Gamma, E., Johnson, R., Helm, R., Vlissides, J. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley. 1994.

(Ginda, 2004) Ginda R., (2004) JavaScript. Mozilla. Created at: September 9, 2004. http://www.mozilla.org/js/ (21/10/07)

(Griffiths et al, 1998) Griffiths, T., Barclay, P., McKirdy, J., Paton, N., Gray, P., Kennedy, J., Cooper, R., Goble, C., West A., and Smyth, M. (1998) Teallach: A Model-Based User Interface Development Environment for Object Databases. In Proceedings of UIDIS'99, pages 86-96, Edinburgh, UK, September 1999. IEEE Press.

(Gould et al., 1985) Gould, J. D., & Lewis, C. (1985) Designing for usability: Key principles and what designers think. Communications of the ACM, 28, 3, 300-311. Reprinted in Baecker, R. M., & Buxton, W. A. S. (Eds.), Readings in human-computer interaction: A multidisciplinary approach, San Mateo, CA: Morgan Kaufmann Publishers, pages 528-539.

(Jaquero, 2005) Jaquero, V.M.J.,(2005). Interfaces de Usuario Adaptativas Basadas en Modelos y Agentes Software. Tesis Doctoral. Universidad de Castilla-La Mancha.

(Knapp et al., 2004) Knapp, A., Koch, N., Zhang, G., Hassler, H-M. (2004). Modeling Business Processes in Web Applications with ArgoUWE. 7th International Conference on the Unified Modeling Language (UML2004), LNCS 3273. ©Springer Verlag.

(Koch, 2006) Koch, N. (2006) Transformations Techniques in the Model-Driven Development Process of UWE. *In Proc. of 2nd Model-Driven Web Engineering Workshop (MDWE 2006)*, ACM Vol. 155, Palo Alto, USA.

(Limbourg et al., 2004) Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., Jaquero, V.L. (2004). UsiXML: A Language Supporting Multi-Path Development of User Interfaces, Proc. of 9th IFIP Working Conference on Engineering for Human-Computer Interaction jointly with 11th Int. Workshop on Design, Specification, and Verification of Interactive Systems EHCI-DSVIS'2004 (Hamburg, July 11-13, 2004). LNCS, Vol. 3425, Springer-Verlag, Berlin, Germany.

(Lozano, 2001) Lozano, M.D., (2001). Entorno Metodológico Orientado a Objetos para la Especificación y Desarrollo de Interfaces de Usuario. Tesis Doctoral. Universidad de Valencia.

(Molina, 2003) Molina, P.J. (2003). Especificación de Interfaz de Usuario: De los Requisitos a la Generación Automática. Tesis doctoral. Universidad de Valencia.

(Montero, 2005). Montero, F., (2005). Integración de Calidad y Experiencia en el Desarrollo de Interfaces de Usuario Dirigido por Modelos. Tesis Doctoral. Universidad de Castilla-La Mancha.

(Mori et al., 2004) Mori G., Paternò F., Santoro C. (2004). Design and Development of Multidevice User Interfaces through Multiple Logical Descriptions. IEEE Transactions on Software Engineering, pages 507-520.

(Mozilla, 2003) Mozilla Project (2003). http://www.mozilla.org (15/10/07)

(Nunes, 2001) Nunes, N. J. (2001) Object Modeling for User-Centered Development and User Interface Design: The Wisdom Approach. Tesis Doctoral. Universidad de Madeira, Portugal.

(Paternò, 1999) Paterno, F. (1999). Model-based Design and Evaluation of Interactive Applications. Springer-Verlag. ISBN 1-85233-155-0.

(Paternò, 2000) Paternò, F. (2000). ConcurTaskTrees and UML: how to marry them? TUPIS.

(Paternò et al., 2006) Paternó, F., Santos, I. (2006). Designing and Developing Multi-User, Multi-Device Web Interfaces, Chapter 9, in Proc. of 6th Int. Conf. on Computer-Aided Design of User Interfaces CADUI'2006 Bucharest, Springer-Verlag, pages 111-122

(Penichet, 2007) Penichet, V.R. (2007). Modelo del Proceso para el Desarrollo de Interfaces en Entornos CSCW Centrado en los Usuarios y Dirigido por Tareas. Tesis Doctoral. Universidad de Castilla-La Mancha.

(Puerta, 1997) Puerta, A.R. (1997). A model based interface development environment, IEEE Software 14(4) pages.41-47.

(Puerta et al., 1999) Puerta, A.R., Eisenstein, J. (1999). Towards a General Computational Framework for Model-Based Interface Development Systems. IUI99: International Conference on Intelligent User Interfaces. Los Ángeles. http://citeseer.ist.psu.edu/puerta99towards.html (28/10/07)

(Puerta, 2001) Puerta, A., Eisenstein, J., Vanderdonckt, J. (2001). Applying Model-Based Techniques to the Development of UIs for Mobile Computers. In IUI01: International Conference on Intelligent User Interfaces. Santa Fe, NW, USA

(Puerta et al., 2001) Puerta, A.R. Eisenstein, J., Vanderdonckt J. (2001). Applying model-based techniques to the development of UIs for mobile computers. Intelligent User Interfaces 2001, pages 69-76.

(Rico, 2004) Rico, M. (2004). Interacción Persona-Agente en los Servicios Web Semánticos. Trabajo de Introducción a la Investigación. Universidad Autónoma de Madrid.

(Roberts et al., 1998) Roberts, D., Berry, D., Isensee, S. y Mullaly, J. (1998) Designing for the User with OVID: Bridging User Interface Design and Software Engineering. New Riders Publishing.

(Romero, 2007). Romero, J.L.M. (2007). Validación y Verificación de Interfaces de Usuario en él Ámbito del Desarrollo Basado en Modelos. XV Jornadas de Ingeniería del Software y Bases de Datos JISBD 2007 José Riquelme - Pere Botella (Eds) © CIMNE, Barcelona.

(Schlungnaum, 1996) Schlungbaum, E. (1996): Individual User Interfaces and Model-Based User Interface Software Tools. Research Report, Georgia Institute of Technology, Graphics, Visualization & Usability Center, Noviembre 1996, GIT-

GVU-96-28.

(Silva, 2000) Silva, P.P. (2000).User Interface Declarative Models and Development Environments: A Survey. In Interactive Systems: Design, Specification, and Verification (7th International Workshop DSV-IS, Limerick, Ireland, June, 2000), Ph. Palanque and F. Paternò (Ed.). LNCS Springer-Verlag, Vol. 1946, pages 207-226.

(Silva et al., 2003) Silva, P.P., Paton., N. W. (2003). User Interface Modeling in UMLi. IEEE Software, Vol.20 No. 4, pages 62-69.

(Szekely, 1996) Szekely, P. (1996) Retrospective and Challenges for Model Based Interface Development, in Vanderdonckt, J., (Editor), CADUI'96, Presses Universitaires de Namur, Namur.

(UML, 2007) Unified Modeling Language (2007). http://www.uml.org/#Links-Tutorials (21/10/07)

(UsiXML, 2007). User Interface eXtensible Markup Language (2007). http://www.usixml.org. (6/10/07)

(Vanderdonckt, 2005) Vanderdonckt, J. (2005). A MDA-Compliant Environment for Developing User Interfaces of Information Systems, pages 16-31, In: Proceedings of the 16th Conference on Advanced Information Systems Engineering, Oscar Pastor, Jo{a}o Falcao e Cunha (Ed.), Lecture Notes in Computer Science, Springer-Verlag, Porto, Portugal, Lecture Notes in Computer Science, Vol. 3520, June 2005, ISBN 3-540-26095-1