UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA
EN INFORMÁTICA

PROYECTO FIN DE CARRERA

"ACAUI: Abstracción de interfaces de usuario a partir de especificaciones concretas"

Francisco Javier Muñoz Márquez

Septiembre, 2007

UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA POLITÉCNICA SUPERIOR

DEPARTAMENTO DE SISTEMAS INFORMÁTICOS

PROYECTO FIN DE CARRERA

"ACAUI: Abstracción de interfaces de usuario a partir de especificaciones concretas"

Autor: Francisco Javier Muñoz Márquez

Director: Francisco Montero Simarro

Septiembre, 2007

Resumen

El proyecto consiste en el análisis y diseño de una aplicación software que permita **especificar una interfaz de usuario a nivel concreto**, y a partir de ella se pueda **abstraer su especificación a un nivel abstracto**. En este sentido, la realización de la **abstracción de interfaces de usuario** desde **especificaciones concretas de interfaces de usuario** se realiza mediante la conversión de ficheros que siguen el estándar impuesto por **UsiXML**. La **interfaz de usuario abstracta** resultante nos mostrará los contenedores y componentes abstractos obtenidos, y estos últimos con sus correspondientes facetas asociadas.

Normalmente, en **UsiXML** se avanza un paso en la jerarquía desde **interfaces de usuario abstractas** para obtener **interfaces de usuario concretas**, pero en este proyecto el paso se realiza justo en el sentido contrario, es decir, que partiendo de la **interfaz de usuario concreta** se obtiene la **interfaces de usuario abstracta**.

Por otro lado, en los últimos años se han estado investigando métodos que permitan incluir el diseño de la **interfaz de usuario** dentro de un proceso de **desarrollo basado en modelos**, para obtener beneficios 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.

En cambio, con el avance tecnológico de los últimos años se han creado nuevos dispositivos como: los teléfonos móviles, las PDAs y los ordenadores, que suponen un cambio importante en la forma en que el usuario interactúa con los sistemas. Debido a esta variedad de dispositivos, ha surgido la necesidad de crear soluciones para diseñar **interfaces de usuario** que funcionen sobre plataformas distintas.

Por este motivo para realizar la abstracción de la **interfaz de usuario** se sigue el **desarrollo basado en modelos** impuesto por **UsiXML**, buscando una tendencia a la estandarización, y un lenguaje de representación común de los datos interactivos. De esta manera, tomando como origen el mismo fichero de código que especifica una **interfaz de usuario** se puede utilizar independientemente del contexto de uso y de las restricciones impuestas por un dispositivo o plataforma específicos, y así se puede interpretar correctamente por distintos tipos de dispositivos como los citados anteriormente.

Agradecimientos

No puedo dejar pasar esta oportunidad para agradecer a todos aquellos que han hecho posible la elaboración de este proyecto fin de carrera.

Agradezco a mi familia el apoyo recibido durante estos años de carrera, ya que en todo momento me han animado a seguir estudiando. Si no hubiera sido por ellos, no podría haber realizado este proyecto fin de carrera.

También agradezco a mis amigos, que con su compañía han hecho este tiempo más ameno.

Por último, agradezco a mi tutor Francisco Montero Simarro, por asignarme este proyecto, y por estar siempre disponible para todas las consultas que le he realizado durante la realización del mismo.

A todos ellos, muchas gracias.

Se lo dedico a mi gente, aquellos con quien comparto agradables momentos, y en especial a los que siempre han confiado en mí

Índice General

Índice de figuras

Índice de tablas

Capítulo 1: Introducción

El proyecto consiste en el desarrollo de una herramienta nueva capaz de realizar una **abstracción de Interfaces de Usuario** (UI) a partir de **especificaciones concretas**, es decir, en el sentido inverso al utilizado normalmente en **UsiXML** (**lenguaje de especificación de interfaces de usuario**).

1.1 MOTIVACIÓN Y JUSTIFICACIÓN

En los últimos años se ha avanzado tecnológicamente a "velocidad de vértigo", por lo que cada vez se hace más necesario la **especificación de interfaces de usuario** independientes del dispositivo, de la plataforma y del contexto de uso, y esta situación provoca que cada vez se utilice más el **desarrollo de interfaces de usuario basado en modelos**, ya que se busca una tendencia a la estandarización, de un lenguaje de representación común para los datos interactivos y al mismo tiempo una evolución y continuo desarrollo de propuestas.

En este sentido se ha creado un **lenguaje de especificación de interfaces de usuario** denominado **UsiXML** que obedece los estándares impuestos por **XML**. La idea es utilizar el lenguaje **UsiXML** para que la representación de las **interfaces de usuario** sea la misma, independientemente de cual sea la plataforma y el dispositivo en el que se visualice dicha **interfaz de usuario**.

En la siguiente figura se muestra cómo partiendo del mismo fichero de código que especifica una **interfaz de usuario**, ésta se puede interpretar correctamente por distintos tipos de dispositivos como: teléfonos móviles, PDAs y ordenadores.

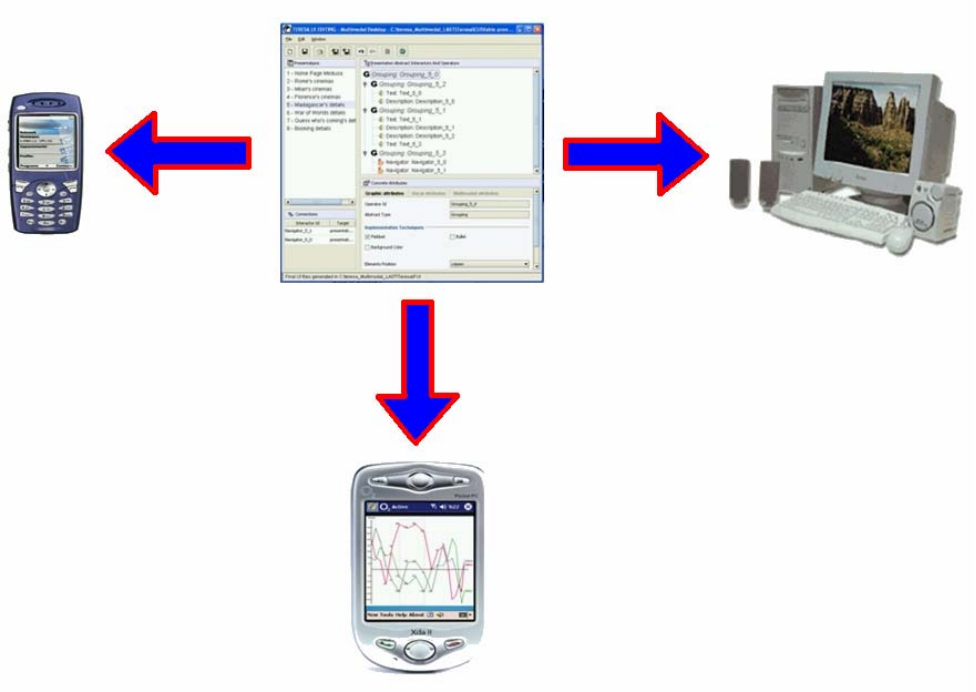


Figura 1.1 Desarrollo de interfaces basado en modelos

Apoyando a lo anterior, es de vital importancia que tras **desarrollar una interfaz de usuario** siguiendo un proceso de Ingeniería del software se consiga que ésta sea de calidad, entendiendo dicha calidad como el cumplimiento de los requisitos funcionales y no funcionales, la mantenibilidad de dicha **interfaz** durante y después de implementación, y la usabilidad a la hora de que el usuario final interactúe con ella.

1.2 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 para el cumplimiento de sus estándares.

- Seguir el **desarrollo basado en modelos** propuesto en **UsiXML**, avanzando un paso en la su jerarquía de modelos.

- Familiarizarnos con el lenguaje de programación Orientado a Objetos Java, y en concreto con el entorno JBuilder, mediante el que se creará la aplicación ACAUI (aplicación desarrollada en este proyecto).

- Conseguir desarrollar un 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 mantenible, para que posteriormente pueda ser modificado sin dificultad.

- Identificar la importancia de la relación existente entre la Ingeniería del Software y la Interacción Persona-Ordenador (IPO).

- Realizar una **abstracción de interfaces de usuario** desde **especificaciones concretas de interfaces de usuario**.

- Comprender la importancia de la **abstracción de interfaces de usuario concretas** para obtener **interfaces de usuario abstractas**, y así conocer con mayor detalle las necesidades que puedan tener los usuarios cuando interactúen con la aplicación.

1.3 ESTRUCTURA DEL DOCUMENTO

En este primer capítulo se puede encontrar una introducción que justifica la realización del proyecto, 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: **interfaces de usuario,** lenguajes de descripción de **interfaces de usuario basados en XML, desarrollo de interfaces de usuario basado en modelos** y **UsiXML**.

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: adquisición de requisitos, análisis de requisitos, diseño e implementación.

En el cuarto capítulo se realiza una explicación profunda del funcionamiento del producto software desarrollado (la aplicación ACAUI), indicando la utilidad de cada elemento que compone la interfaz de usuario. Además, hay varios ejemplos que muestran la conversión de **interfaces de usuario concretas** a **interfaces de usuario abstractas**, utilizando para dicha transformación ficheros que cumplen los estándares impuestos por **UsiXML**.

En el quinto capítulo se recogen las conclusiones obtenidas tras la realización del proyecto, trabajos relacionados con este proyecto y además se proponen una serie de trabajos futuros.

1.4 CONSIDERACIONES PREVIAS

Antes de empezar con el desarrollo de los capítulos sugeridos, es conveniente la definición aunque sea somera, de una serie de conceptos, que por su relación con el tema que se está tratando aparecerán con asiduidad a lo largo de todo el proyecto, 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.

Desde hace más de una década, y aunque prácticamente en exclusiva en el ámbito académico y 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.

Hay que comprender la importancia de la **abstracción de interfaces de usuario concretas a interfaces de usuario abstractas**, para conocer con mayor detalle las necesidades que tiene el usuario a la hora de interactuar con una aplicación. De esta manera, se tendrá un mayor conocimiento de las preferencias que tienen los usuarios sobre cómo debe ser una **interfaz de usuario** usable.

Actualmente, el **lenguaje de especificación de interfaces de usuario** denominado **UsiXML**, permite representar las **interfaces de usuario** independientemente de cual sea la plataforma y el dispositivo en el que se visualice

dicha **interfaz de usuario**. Esto se consigue mediante el seguimiento de los estándares impuestos por dicho lenguaje. **UsiXML** tiene un sitio Web oficial: http://www.usixml.org, donde se puede obtener información sobre este lenguaje.

Capítulo 2: Estudio del estado del arte

2.1 INTERFAZ DE USUARIO

La creación de las **interfaces de usuario** ha sido un área del desarrollo de software que ha evolucionado dramáticamente a partir de la década de los setenta. La **interfaz de usuario** es el vínculo entre el usuario y el programa de computadora. Las **interfaces de usuario** pueden adoptar muchas formas, que van desde la simple línea de comandos hasta las interfaces gráficas que proporcionan las aplicaciones más modernas.

La elaboración de una **interfaz de usuario** bien diseñada, exige una gran dedicación pues generalmente las **interfaces** son grandes, complejas y difíciles de implementar, depurar y modificar. Hoy en día las interfaces de manipulación directa (también llamadas **interfaces gráficas de usuario**, GUI por sus siglas en inglés) son prácticamente universales. Las interfaces que utilizan ventanas, íconos y menús se han convertido en estándar en los materiales computacionales (véase el Apéndice A).

La **interfaz** es el punto de encuentro entre el usuario y la computadora. En esta interacción el usuario juzga la utilidad de la interfaz, y el hardware y el software se convierten en simples herramientas sobre los cuales fue construida la interfaz. La definición de **interfaz** en sí es un tanto arbitraria, aunque esto dependería de la naturaleza de la tarea que se tiene enfrente.

Existen muchos tipos de software para la creación de **interfaces de usuario**. El sistema de ventanas permite la división de la pantalla en diferentes regiones rectangulares, llamadas “ventanas”. El sistema de ventanas XWindows para Unix divide la funcionalidad de la ventana en dos capas: el sistema de ventanas, el cual es la interfaz funcional, y el administrador de ventanas. 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. El administrador de ventanas le permite al usuario final el mover las ventanas por la pantalla, y es el responsable de desplegar las líneas de título bordes e íconos alrededor de las ventanas.

La parte central de un sistema de ventanas es el conjunto de herramientas (toolkit), el cual contiene los objetos gráficos (widgets[1]) más empleados tales como menús, botones, 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. (InterfazDeUsuario, 2007)

La **interfaz de usuario** es la parte del programa que permite a éste interactuar con el usuario. La **interfaz de usuario** es el aspecto más importante de cualquier aplicación. Una aplicación sin una **interfaz** fácil, impide que los usuarios saquen el máximo rendimiento del programa. Un programa muy poderoso con una interfaz pobremente elaborada tiene poco valor para un usuario no experto.

2.2 LENGUAJES DE DESCRIPCIÓN DE INTERFACES BASADOS EN XML

La definición de **interfaces de usuario** utilizando lenguajes declarativos 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.

Los **lenguajes basados en XML** se están perfilando como serios candidatos a soportar las especificaciones gracias a la versatilidad de mantenimiento, extensión y capacidades de refinamiento que proporcionan los documentos **XML**.

La tecnología **XML** (véase el apéndice B) 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. (LenguajesDescriptivosDeUI, 2007)

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

2.2.1 Notación para la descripción de los modelos

Para la especificación de los modelos, los entornos de desarrollo de Interfaces de Usuario basados en modelos (MB-UIDE) incorporan determinadas notaciones descriptivas que van desde especificaciones algebraicas, representaciones gráficas, hasta la utilización de lenguajes de especificación propios de cada entorno.

Una de las mayores desventajas del desarrollo de Interfaces basadas en Modelos es la complejidad que presentan algunas de estas notaciones que a menudo resultan difíciles de aprender y utilizar.

Para evitar esto, los MB-UIDE proporcionan herramientas de modelado que sirven a los desarrolladores para construir dichos modelos. 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. Se ha desarrollado una amplia gama de herramientas de modelado, a menudo especializadas en diferentes niveles del modelo.

Podemos observar que no existe una notación estándar para la descripción de los diferentes modelos dentro de los MB-UIDE desarrollados. Cada MB-UIDE utiliza 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 que se compone el modelo de interfaz.

2.2.2 Aproximaciones de lenguajes de descripción de interfaces basados en XML

2.2.2.1 AAIML (Alternate Abstract Interface Markup Language)

AAIML[2] es un **lenguaje basado en XML** en vías de desarrollo que se enmarca dentro de un proyecto más amplio para el desarrollo de un protocolo estándar para el acceso a **interfaces** alternativas. El objetivo de este proyecto es ampliar el mercado, haciendo más accesibles las **interfaces** y los dispositivos, por ejemplo a personas con discapacidades.

[2] http://xml.coverpages.org/userInterfaceXML.html#aaiml

El hecho de que los dispositivos electrónicos estén fabricados por diferentes compañías, requiere la definición de un estándar que permita utilizar una consola "alternativa" para controlar los dispositivos específicos. La especificación del concepto de Consola Remota Universal (URC) es una parte esencial del desarrollo de la definición de dicho estándar.

El concepto de URC permite a las personas con o sin discapacidades controlar remotamente cualquier dispositivo electrónico desde su dispositivo personal de control remoto que puede encontrarse situado en cualquier lugar.

El lenguaje AAIML debe servir para especificar la definición de una **interfaz de usuario abstracta** para un determinado servicio o dispositivo. Dicha **interfaz** sería transmitida desde el dispositivo a la URC. Este lenguaje debería ser lo suficientemente abstracto, para permitir que un dispositivo URC particular pueda traducir la **interfaz** proporcionada a los mecanismos de E/S concretos del dispositivo adaptado a las necesidades del usuario. (LenguajesDescriptivosDeUI, 2007)

2.2.2.2 AUIML (Abstract User Interface Markup Language)

El lenguaje AUIML[3] es un **lenguaje basado en XML** desarrollado por IBM y diseñado para permitir la definición de la semántica de la interacción con el usuario. Está centrado, por tanto, en la descripción de aspectos de interacción más que en aspectos de presentación.

Toda la información de la interacción se codifica una sola vez y se traduce utilizando una traducción dependiente del dispositivo final. Está diseñado para ser independiente de la plataforma, del lenguaje de programación final y de la tecnología de implementación.

Consta de dos principales conjuntos de elementos, los que se representan a través del modelo de datos, que definen la estructura de la información necesaria para soportar la interacción con el usuario, y los que se representan a través del modelo de presentación que definen el estilo de la presentación.

El lenguaje AUIML está todavía en desarrollo y no existen herramientas que soporten de forma completa su especificación, siendo ésta un documento de trabajo

[3] http://www.alphaworks.ibm.com/tech/auiml

interno de IBM. AUIML estaría englobado dentro de los niveles 2 y 3 del proceso de modelado. (LenguajesDescriptivosDeUI, 2007)

2.2.2.3 UIML (User Interface Markup Language)

El lenguaje UIML[4] 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.

Podemos decir que UIML se podría utilizar en los niveles 2 y 3 del esquema de modelado presentado y parcialmente en el nivel 1. La definición de la **interfaz** se enmarcaría en los niveles 1 y 2 y la definición del estilo en el nivel 3. Estas tareas incluso podrían ser llevadas a cabo por equipos de desarrollo diferentes.

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.

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. (LenguajesDescriptivosDeUI, 2007)

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

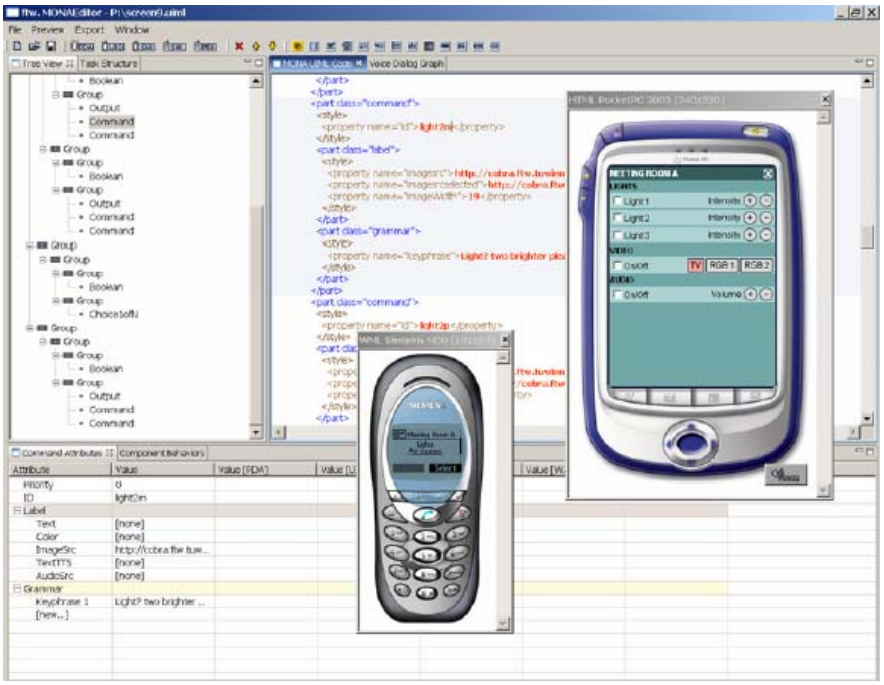


Figura 2.1 Entorno en el que se utiliza el lenguaje UIML

2.2.2.4 XIML (eXtensible Interface Markup Language)

El lenguaje XIML[5] es un lenguaje de especificación **basado en XML**. Se propone como lenguaje de especificación común e infraestructura de desarrollo para profesionales de la **interfaz de usuario** en todos los ámbitos, diseñadores, ingenieros de software o expertos en usabilidad.

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.

XIML es una colección organizada de elementos de interfaz categorizados en uno o más componentes principales. En su primera versión los componentes son tareas, dominio, usuario, diálogo y presentación, extraídos del estudio de los modelos declarativos del enfoque **basado en modelos**. Los tres primeros componentes se podrían considerar abstractos y los dos últimos específicos.

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

Por tanto, podemos decir que XIML podría ser utilizado en los niveles 1, 2 y 3 del proceso de modelado de **interfaces de usuario** presentado. La independencia de la plataforma de uso se hace posible gracias a la estricta separación entre la definición de la **interfaz** y la traducción de la misma. (LenguajesDescriptivosDeUI, 2007)

2.2.2.5 XUL (XML-based User-Interface Language)

El lenguaje XUL[6] es un lenguaje de descripción de **interfaces basado en XML**, específicamente diseñado para aplicaciones en red como navegadores, programas de correo. Está integrado dentro de la arquitectura de Mozilla para el desarrollo de Interfaces Web multiplataforma, dentro de la cuál se hace uso de tecnologías W3C ya existentes.

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. 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.

Podríamos utilizar XUL para realizar la definición de la **especificación de la interfaz concreta**, nivel 3. Contempla parcialmente el nivel 2 a través del resto de las tecnologías de las que hace uso. (LenguajesDescriptivosDeUI, 2007)

2.2.2.6 XFORMS (the neXt generation of web FORMS)

XForms[7] 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.

[6] http://xulplanet.com
[7] http://www.aulambra.com/markup/xforms.asp

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 3 de abstracción (parte del 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. (LenguajesDescriptivosDeUI, 2007)

2.3 DESARROLLO DE INTERFACES DE USUARIO BASADO EN MODELOS

El principal objetivo del desarrollo de **interfaces de usuario basado en modelos** es la construcción de **interfaces** mediante descripciones de alto nivel de los distintos aspectos de la **interfaz**: estructura y comportamiento, de modo que a partir de dichos modelos declarativos se pueda generar automáticamente la **interfaz de usuario** final.

Basándose en esta idea, se han construido diversos entornos de desarrollo y herramientas que evitan la necesidad de utilizar un lenguaje de programación concreto. En su lugar, el diseñador utiliza notaciones de mayor nivel de abstracción para especificar estos modelos o descripciones declarativas de la **interfaz**.

Estos lenguajes de modelado, describen la **interfaz de usuario** en diferentes niveles de abstracción según la fase del proceso de modelado en la cuál pueden ser utilizados. Algunos lenguajes pueden servir para especificar todos los modelos declarativos que componen el modelado de la **interfaz**, es decir, pueden ser usados durante el proceso de desarrollo completo. Sin embargo, otros sólo pueden ser utilizados en determinados niveles.

Al mismo tiempo, este **enfoque basado en modelos** declarativos proporciona un marco idóneo para la definición de **interfaces de usuario** independientes de contexto de uso, ya que permite utilizar descripciones de alto nivel de abstracción que pueden modelar aspectos independientes del contexto, o de las restricciones impuestas por un dispositivo o plataforma específicos.

2.3.1 Descripción basada en modelos de Interfaces de Usuario con 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 dispositivo, de plataforma y de modalidad.

Actualmente, el **desarrollo de Interfaces de Usuario** de aplicaciones interactivas es muy difícil a causa de la complejidad y la diversidad de los medios de desarrollo existentes y una alta cantidad de habilidades de programación requeridas por el desarrollador para alcanzar una **interfaz de usuario** usable: los lenguajes de marcado (por ejemplo HTML), lenguajes de programación (por ejemplo C++ o Java), desarrollos hábiles para la comunicación, habilidades para la ingeniería usable.

Estas dificultades son exacerbadas cuando el mismo **interfaz de usuario** debe ser desarrollado para múltiples contextos de uso tales como múltiples categorías de usuarios (por ejemplo diferentes preferencias, lenguajes nativos de habla diferente, potencialmente sufridos por incapacidad), diferentes plataformas de computación (por ejemplo un teléfono móvil, un Pocket PC, un quiosco interactivo, un portátil, pantallas planas), y varios medios de trabajo (por ejemplo fijo, móvil).

Aunque los diseñadores y los programadores están envueltos en estos tipos de proyectos, las herramientas disponibles son principalmente el objetivo del desarrollador. Por lo tanto, es algo difícil para los diseñadores diseñar una **interfaz de usuario** para múltiples contextos de uso mientras evitan reproducir múltiples **interfaces de usuario** para múltiples contextos de uso. Este trabajo propone un camino de separación de responsabilidades en este tipo de proyectos.

UsiXML (el cuál se presenta como candidato para el lenguaje de marcado extensible de interfaz de usuario) es un lenguaje XML de marcado adaptado que

describe la **interfaz de usuario** para múltiples contextos de uso tales como **Interfaces de Usuario** de Caracteres (CUIs), **Interfaces de Usuario** gráficas (GUIs), **Interfaces de Usuario** de Auditoria, **Interfaces de Usuario** Multimodales. En otras palabras, las aplicaciones interactivas con diferentes tipos de técnicas de interacción, modalidades de uso, y plataformas de computación pueden ser descritas de la forma que preserva el diseño independientemente de las características peculiares de la plataforma de computación física.

A continuación se definen las principales características de **UsiXML**:

- **UsiXML** es entendido por no desarrolladores, como analistas, 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 de la **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 descrita 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**. A continuación se definen las características que **UsiXML** no soporta:

- **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)

Por otro lado, **UsiXML** está basado en los principios de modelado MDA (Model-Driven Architecture), así como en el marco de referencia Camaleon, un proceso MBUI (desarrollo de interfaces de usuario basadas en modelos) que define los pasos y tipos de modelos en el **desarrollo de interfaces de usuario** para aplicaciones interactivas multicontexto. En particular, se proponen cuatro tipos de modelos, cuya jerarquía se muestra en la figura siguiente.

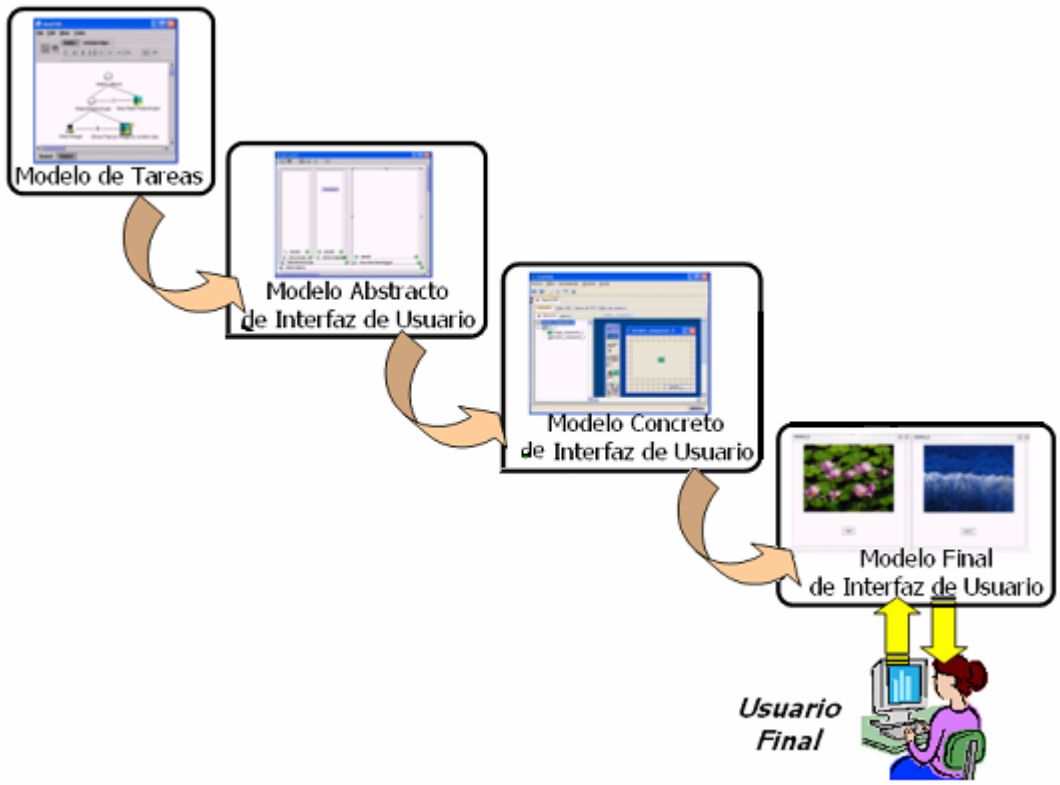


Figura 2.2 Jerarquía de modelos de UsiXML y procesos en su construcción

El **modelo de tareas y dominio** sirve para definir las tareas que se quieren realizar mediante la **interfaz** (similar a los casos de uso), así como conceptos orientados al dominio requeridos por las tareas.

El modelo de **interfaz de usuario abstracta (AUI, Abstract User Interface)** se utiliza para definir contenedores y componentes abstractos agrupando en estos la ejecución de las tareas antes definidas.

El modelo de **interfaz de usuario concreta (CUI, Concrete User Interface)** materializa la definición abstracta anterior en cada contexto de uso, mediante widgets y navegación entre interfaces.

Finalmente la **interfaz de usuario final (FUI, Final User Interface)** contiene la **interfaz de usuario** operacional, generada a partir de las capas anteriores. (UsiXMLRomero, 2007)

2.3.1.1 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.

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

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.

El modelo de tareas 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.

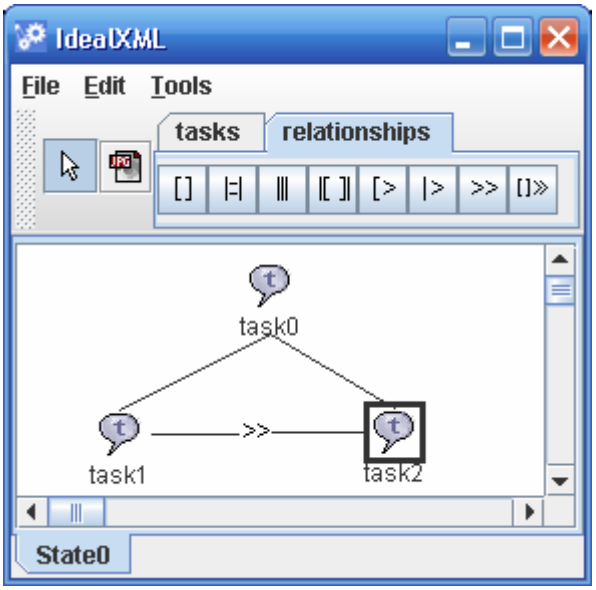


Figura 2.3 Ejemplo de un modelo de tareas

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.

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)

2.3.1.2 Interfaz de Usuario Abstracta (AUI)

Desde el punto de vista de la especificación de la **interfaz de usuario abstracta**, ésta se define de forma independiente de la plataforma y de la implementación. (Montero, 2005)

La **interfaz de usuario abstracta** enfoca la interacción de objetos soportados por el modelo de tareas. Es definida de términos de presentación, como conjunto de elementos **de interfaz de usuario**. Es una **interfaz de usuario** genérica independiente de cualquier contexto de uso. (TesisAachen, 2005)

El modelo de **interfaz de usuario abstracta** representa una expresión canónica de suministros y manipulación de conceptos del dominio y funciones que son tan independientes como sea posible de modalidades y especificaciones de plataformas de computación.

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. (Montero et al, 2005)

En el siguiente subapartado se dará una explicación de detalle de las facetas y sus tipos.

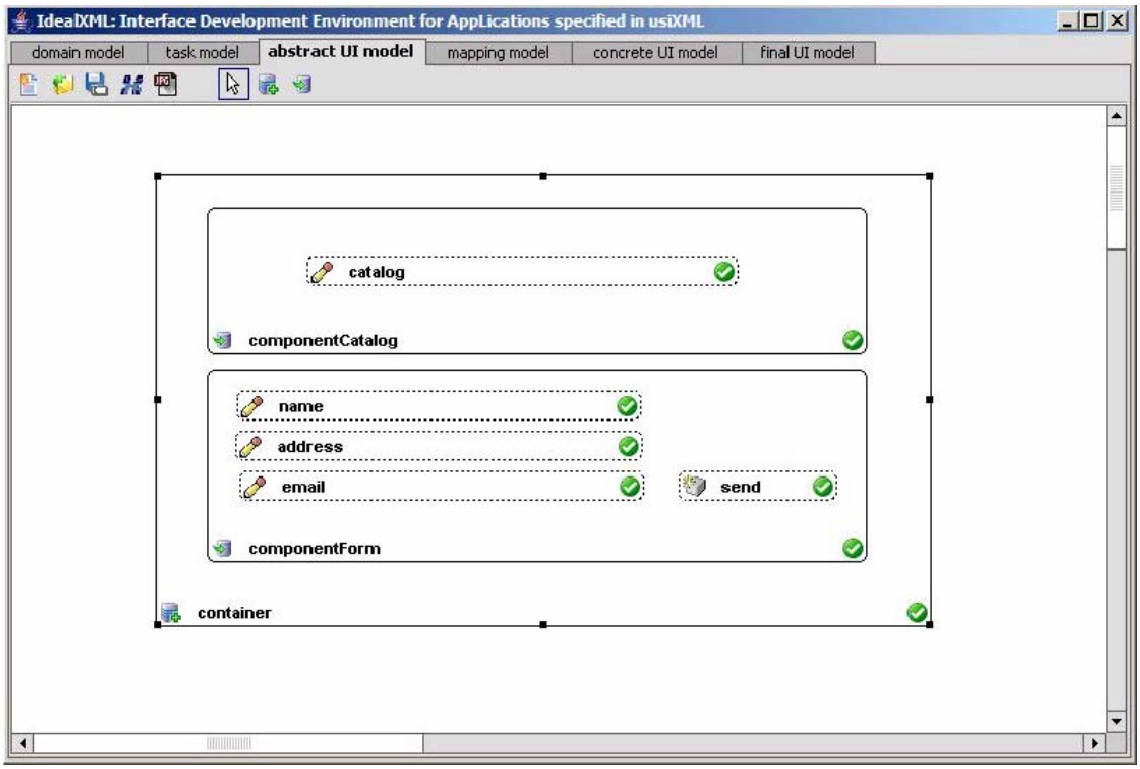


Figura 2.4 Ejemplo de una Interfaz de Usuario Abstracta

La **interfaz de usuario abstracta** es un árbol DOM que describe los objetos de la **interfaz de usuario** del programa dado. Una copia del árbol AUI es guardada por la interfaz y el sistema de activación. La sincronización del árbol AUI es automáticamente hecha por el sistema de activación usando el protocolo de **interfaz**. Los programas pueden manipular el árbol AUI usando la construcción de clases y las utilidades de XML.

2.3.1.2.1 Facetas AUI

Actualmente, una serie de **modelos** son usados para describir **las interfaces de usuario**. Estos **modelos** necesitan ser guardados en un repositorio para que puedan ser manipulados por diferentes herramientas usadas durante las etapas de generación de la **interfaz de usuario**. En la mayoría de los casos estos **modelos** son guardados usando un formato basado en **UsiXML**. Las facetas se utilizan para describir las capacidades de interacción de los objetos abstractos.

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 son tan independientes como sea posible de modalidades y plataforma de computación específica.

Se usa **especificación de interfaces de usuario abstractas en UsiXML** porque proporciona un reducido conjunto de elementos que permiten la descripción de **interfaces de usuario abstractas** en plataformas y modalidades diferentes. (Montero-López, 2005)

A continuación se muestran los objetos abstractos existentes:

Tabla 2.1 Objetos abstractos utilizados en la elaboración de modelos (abstractos)

Objeto abstracto	Icono
Contenedor	
Componente	

A continuación se muestran las facetas existentes:

Tabla 2.2 Facetas utilizadas en la elaboración de modelos abstractos

Faceta	Icono
Entrada	
Salida	
Control	
Navegación	

Un componente abstracto puede tener varias facetas asociadas, pero los contenedores abstractos no pueden tener facetas asociadas.

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 (por ejemplo, métodos del modelo dominado cuando existen). Un contenedor abstracto (AC) es una

entidad que permite agrupar de manera lógica otros contenedores abstractos o componentes abstractos. Los contenedores abstractos soportan la ejecución de un conjunto lógico/semántico de tareas conectadas. Los componentes de interacción abstractos y los contenedores abstractos 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.

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 et al, 2005)

2.3.1.3 Interfaz de Usuario Concreta (CUI)

La **interfaz de usuario concreta** es una descripción o especificación detallada de la apariencia e interactividad de las **interfaces de usuario** de 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)

La **interfaz de usuario concreta** es un modelo específico de **interfaz de usuario**, el cuál es supuesto de modelo de **interfaz de usuario** dependiendo de las restricciones de determinados contextos de uso, considerando así la parte sensible al contexto del modelo sensible al contexto de la tarea. (TesisAachen, 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 CIO es definido como una entidad que los usuarios pueden percibir y/o manipular. 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 et al, 2005)

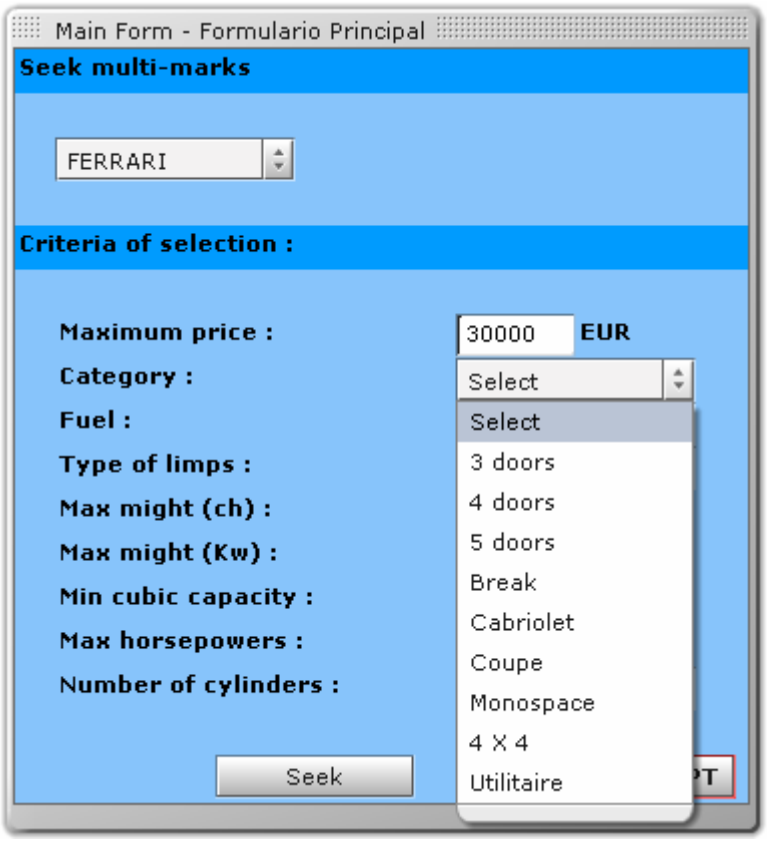


Figura 2.5 Ejemplo de una Interfaz de Usuario Concreta

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 retroalimentación dinámica.
- Menús, paletas, acciones del sistema en los elementos de menú y opciones de botón, accesos directos, y facilidades para deshacer.

Las salidas también incluyen:

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

Estas salidas pueden ser representadas usando

- Texto y diagramas
- Etiquetas y paneles
- Secuencias de interacción
- Layout generados por herramientas
- Prototipos

Un aspecto de las salidas es mostrar cómo las interacciones del usuario afectan al estado de las **interfaces**. (ConcreteUserInterface, 2007)

2.3.1.4 Interfaz de Usuario Final (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)

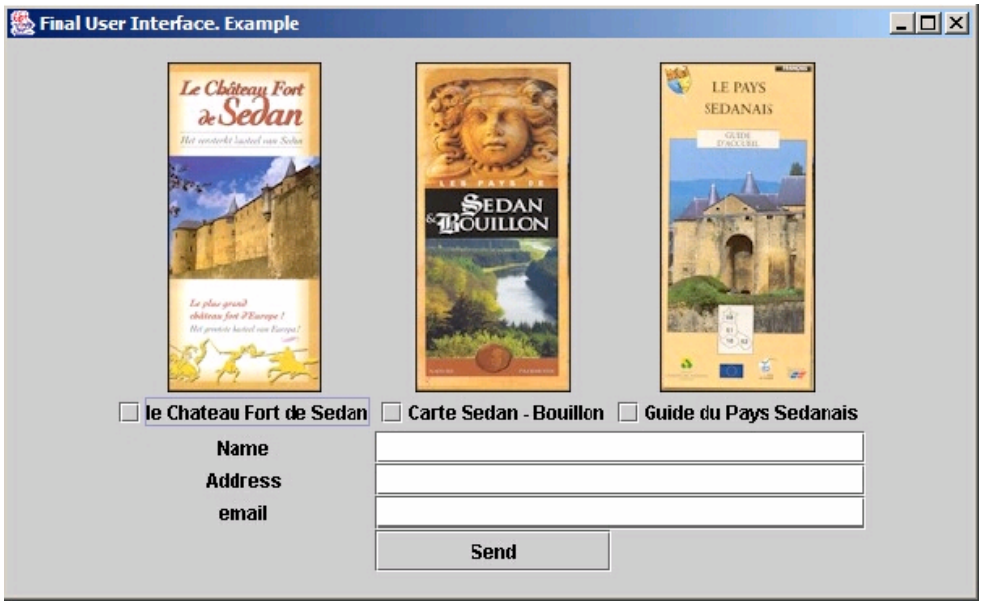


Figura 2.6 Ejemplo interfaz de usuario final

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**. (UsiXMLRomero, 2007)

2.3.2 Aproximaciones basadas en modelos para el diseño de interfaces de usuario

A lo largo de la última década, distintas aproximaciones han sido presentadas para el diseño de **interfaces de usuario basado en modelos**, donde se soporta en mayor o menor medida la generación automática de la **interfaz de usuario**. La mayoría de estas herramientas proceden del ámbito académico y en general sólo presentan prototipos de las herramientas CASE para soportar los métodos de diseño y especificación propuestos, aunque ya podemos encontrar algunos ejemplo de herramientas comerciales **basadas en modelos** como WebRatio[8], VisualWADE[9] u Olivanova[10].

A continuación se describen algunos de los métodos basados en modelos más destacados para la generación de **interfaces de usuario**, entre los que se encuentran: OVID, TERESA, Just-UI, OO-H, UML*i*, UWE e IDEAS. (Jaquero, 2005)

2.3.2.1 OVID (Object, View and Interaction Design)

La metodología OVID[11] 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.

El proceso básico de esta metodología es el siguiente:

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

[8] http://www.webratio.com
[9] http://www.visualwade.com
[10] http://www.care-t.com
[11] http://acm.org/sigchi/chi97/proceedings/tutorial/djr.htm

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 siguiente figura 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)

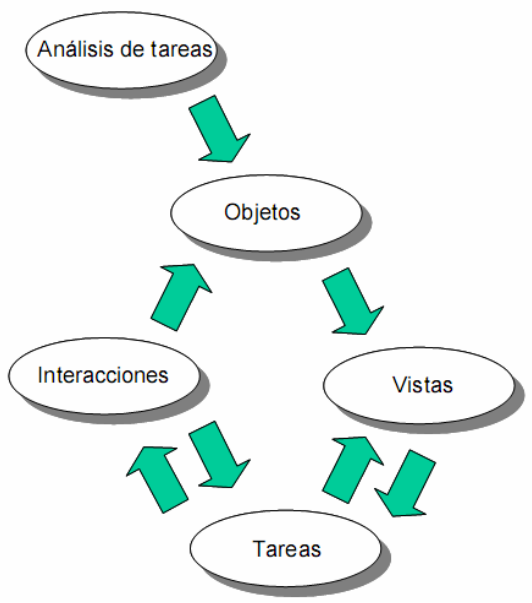


Figura 2.7 Ciclo de desarrollo en OVID

2.3.2.2 TERESA (Transformation Environment for inteRactivE Systems representAtions)

TERESA[12] es una herramienta diseñada para la generación de **interfaces de usuario** para distintas plataformas a partir de modelos de tareas creados utilizando la notación de *ConcurTaskTrees*. *ConcurTaskTrees* (*CTT*) es una notación basada en el lenguaje formal LOTOS para el análisis y generación de **interfaces de usuario** a partir

[12] http://giove.isti.cnr.it/teresa.html

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.

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 cual 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*.

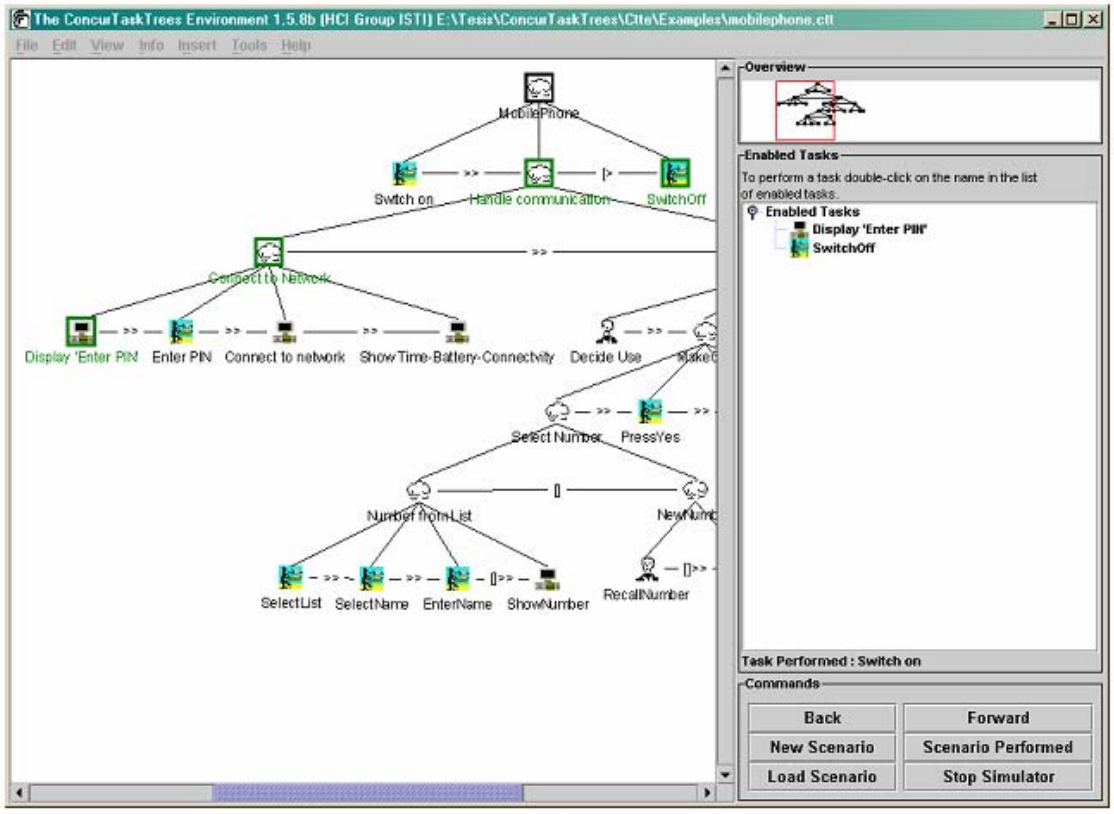


Figura 2.8 Especificación de la interacción en un teléfono móvil con CTT

La herramienta también permite convertir la notación gráfica al lenguaje LOTOS 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* (*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)

2.3.2.3 Just-UI

El propósito de Just-UI 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.

Para modelar la navegación 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. (Jaquero, 2005)

2.3.2.4 OO-H (Object Oriented Hypermedia Method)

OO-H[13] 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.

[13] http://gplsi.dlsi.ua.es/iwad/ooh_project

- 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**.

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. (Jaquero, 2005)

2.3.2.5 UWE

UWE 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 metamodelo 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. (Jaquero, 2005)

2.3.2.6 UML*i* (The Unified Modeling Language for Interactive Applications)

En UML*i*[14] Pinheiro da Silva 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

[14] http://trust.utep.edu/umli

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).

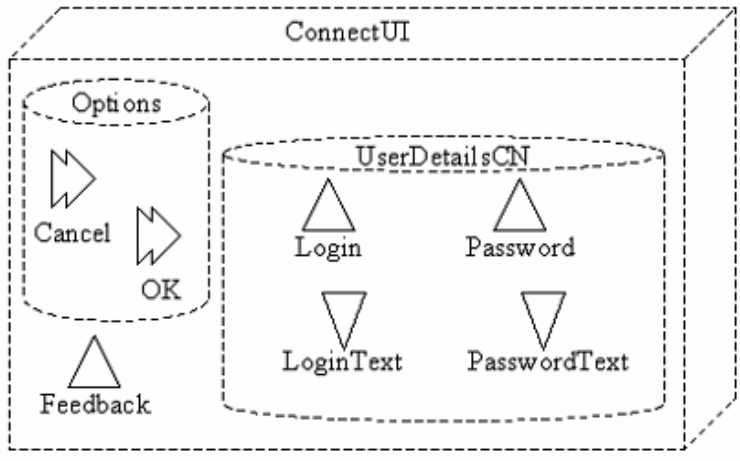


Figura 2.9 Especificación abstracta de una interfaz de usuario en UMLi

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*[15], una extensión del entorno ArgoUML de código abierto. (Jaquero, 2005)

2.3.2.7 IDEAS (Interface Development Environment within OASIS)

IDEAS 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 siguiente figura muestra los diagramas usados en cada una de las fases del desarrollo de una interfaz de usuario en IDEAS.

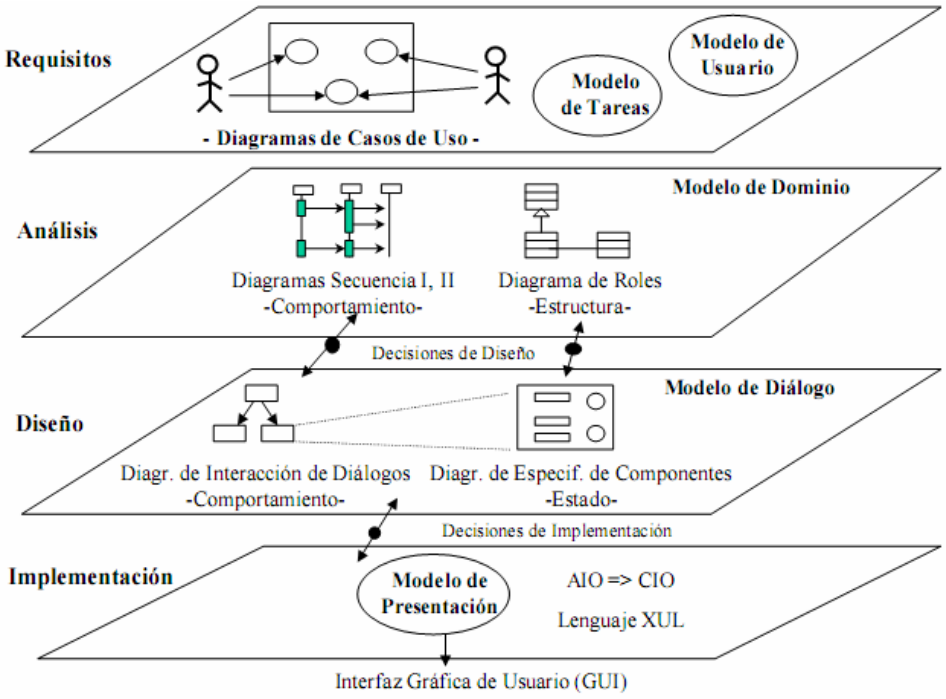


Figura 2.10 Etapas de diseño en IDEAS

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

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 (diagrama de estados internos).

En IDEAS también se plantea una **especificación abstracta de la interfaz de usuario** (diagrama de especificación de componentes). Esta especificación abstracta es traducida al lenguaje XUL para su visualización. (Lozano, 2001)

2.3.3 Abstracción de CUI a AUI

La estructura normal que se suele seguir a la hora de trabajar con **interfaces de usuario basadas en modelos** es la siguiente:

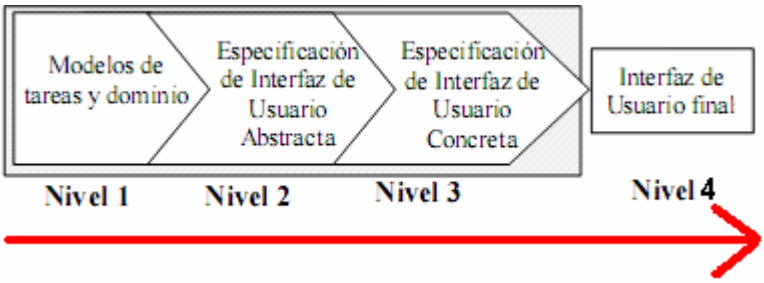


Figura 2.11 Modelado de interfaz normal

Es decir, que el nivel más bajo en la jerarquía de modelos lo forman los Modelos de tareas y dominio, y el nivel más alto lo forma la Interfaz de Usuario Final. Pero en el caso de este proyecto, no es así, sino que la transformación se realiza partiendo de la **Especificación de la Interfaz de Usuario Concreta** para obtener una **Especificación de la Interfaz de Usuario Abstracta**. Luego la transformación realizada se realiza en sentido contrario al utilizado habitualmente.

A continuación se muestra la secuencia de transformación seguida en el proyecto:

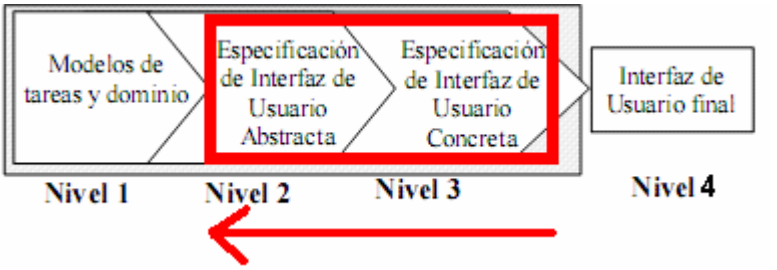


Figura 2.12 Modelado de interfaz en sentido inverso

Con lo que puede apreciarse un cierto grado de ingeniería inversa o reingeniería. La ingeniería inversa o reingeniería avanza en dirección opuesta a las tareas habituales de ingeniería, que consisten en utilizar datos técnicos para elaborar un producto determinado.

La ingeniería inversa es un método de resolución. Aplicar ingeniería inversa a algo supone profundizar en el estudio de su funcionamiento, hasta el punto de que podemos llegar a entender, modificar, y mejorar dicho modo de funcionamiento. (Wikipedia, 2007)

La reingeniería es una alternativa útil para la industria del software, ya que se trata de una actividad que permite incrementar la facilidad de mantenimiento, la reutilización y la evolución de sistemas software.

Capítulo 3: Detalles de implementación: ACAUI

ACAUI (**A**bstraction from **C**oncrete User Interface to **A**bstract **U**ser **I**nterface) es una herramienta nueva desarrollada en este proyecto, que permite realizar una **abstracción** partiendo de **interfaces de usuario concretas** para obtener **interfaces de usuario abstractas**.

En esta herramienta el usuario puede crear una **interfaz de usuario concreta** a su medida, con los distintos contenedores y componentes concretos (que se explican en el siguiente capítulo) disponibles. Una vez que el usuario finalice la creación de su **interfaz de usuario concreta**, y tras guardar el diagrama en un fichero con extensión ".cui" (el cual sigue el estándar impuesto por **UsiXML**), el usuario puede convertir dicha **interfaz de usuario concreta** en la **interfaz de usuario abstracta** asociada con sólo pulsar un botón. Cómo resultado de esta conversión, se obtiene un fichero con extensión ".usi" con la **especificación de la interfaz de usuario abstracta** asociada a la **interfaz de usuario concreta** que anteriormente se creó. Dicho fichero de **especificación abstracta** también sigue el estándar impuesto por **UsiXML**. Finalmente el módulo de la **interfaz de usuario abstracta** carga automáticamente dicho fichero y muestra al usuario los contenedores y componentes abstractos con sus facetas correspondientes.

En este caso, el paradigma de diseño **basado en modelos** intenta atajar las dificultades que afloran durante el diseño de una **interfaz de usuario**, y a su vez aporta nuevos beneficios: automatización o semiautomatización de la creación de la **interfaz de usuario**, reutilización de la experiencia, etc.

La siguiente figura muestra las etapas propuestas para desarrollar la aplicación ACAUI. El proceso 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 se estudian los requisitos descritos anteriormente y realizan diagramas de paquetes y de clases. La etapa de diseño crea una descripción abstracta de la futura interfaz de usuario. Finalmente, la etapa de implementación genera el código de la aplicación que será ejecutado dentro de la arquitectura.

A continuación se muestra un esquema con las fases seguidas en el desarrollo de la aplicación ACAUI:

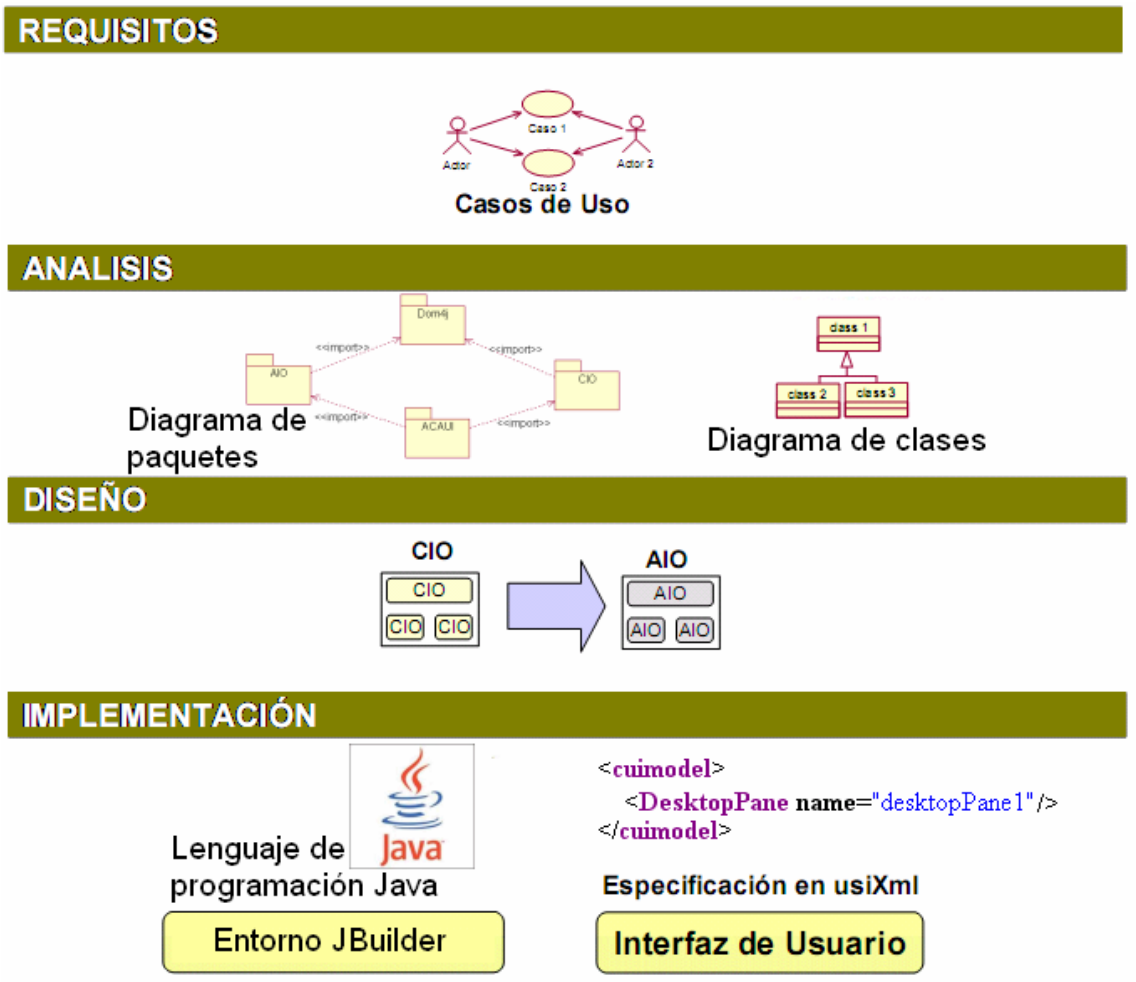


Figura 3.1 Fases de desarrollo para crear ACAUI

Antes de comenzar con las fases de desarrollo antes citadas, hay que dejar claro cuales son los principales objetivos de esta aplicación. El primer objetivo es: "**Gestionar las Interfaces de Usuario Concreta y Abstracta**", y el segundo objetivo es: "**Gestionar la abstracción de Interfaces de Usuario a partir de Interfaces de Usuario Concretas**".

A continuación se muestran unas tablas con la información más relevante de los objetivos citados anteriormente:

Tabla 3.1 Objetivo 1 del sistema

OBJ–01	Gestionar las Interfaces de Usuario Concreta y Abstracta
Versión	1.0
Autor	Francisco Javier Muñoz Márquez
Descripción	El sistema deberá gestionar la creación de las interfaces de usuario Concreta y Abstracta. La interfaz de usuario concreta estará compuesta por contenedores y componentes concretos acorde a las necesidades del usuario. La interfaz de usuario abstracta estará compuesta por contenedores abstractos, componentes abstractos y facetas asociadas a dichos componentes abstractos. Además deberá ser capaz de realizar las operaciones solicitadas por el usuario.
Importancia	Vital
Urgencia	Inmediata
Estado	Construido
Estabilidad	Media
Comentarios	Ninguno

Tabla 3.2 Objetivo 2 del sistema

OBJ–02	Gestionar la abstracción de Interfaces de Usuario a partir de Interfaces de Usuario Concretas
Versión	1.0
Autor	Francisco Javier Muñoz Márquez
Descripción	El sistema deberá realizar una abstracción, partiendo de la interfaz de usuario concreta creada por el usuario, para obtener la interfaz abstracta correspondiente.
Importancia	Vital
Urgencia	Inmediata
Estado	Construido
Estabilidad	Media
Comentarios	Ninguno

En los siguientes subapartados se describe de forma detallada cada una de las etapas del ciclo de desarrollo propuesto.

3.1 FASE DE ADQUISICIÓN DE REQUISITOS

Esta fase de adquisición de requisitos es 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 adquisición 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.

3.1.1 Identificación de los requisitos de información

Se han definido dos requisitos de información: uno relacionado con el objetivo 1, y otro relacionado con el objetivo 2. Será muy importante dar una definición correcta de dichos requisitos, ya que los casos de uso obtenidos para el sistema (se definen en la siguiente fase) están relacionados con estos requisitos de información. Una mala definición de dichos requisitos puede causar problemas a corto y a largo plazo en el desarrollo de la aplicación. En las dos tablas siguientes se definen dichos requisitos de información.

A continuación se muestra la información más relevante del requisito de información 1:

Tabla 3.3 Requisito de Información 1

IRQ–01	Información sobre las Interfaces de Usuario Concreta y Abstracta
Versión	1.0
Autor	Francisco Javier Muñoz Márquez
Objetivos asociados	OBJ-01. Gestionar las Interfaces de Usuario Concreta y Abstracta
Requisitos asociados	---
Descripción	El sistema deberá poder crear una interfaz de usuario concreta, según las peticiones solicitadas por el usuario, referentes a la personalización de dicha interfaz. Del mismo modo, el sistema también podrá crear una interfaz de usuario abstracta acorde a las indicaciones del usuario.
Importancia	Vital
Urgencia	Inmediata
Estado	Construido
Estabilidad	Alta
Comentarios	Ninguno

A continuación se muestra la información más relevante del requisito de información 2:

Tabla 3.4 Requisito de Información 2

IRQ–02	Información sobre la abstracción de Interfaces de Usuario a partir de Interfaces de Usuario Concretas
Versión	1.0
Autor	Francisco Javier Muñoz Márquez
Objetivos asociados	OBJ.-02. Gestionar la abstracción de Interfaces de Usuario a partir de Interfaces de Usuario Concretas
Requisitos asociados	---
Descripción	El sistema deberá poder transformar una interfaz de usuario concreta, obteniendo la interfaz de usuario abstracta correspondiente.
Importancia	Vital
Urgencia	Inmediata
Estado	Construido
Estabilidad	Alta
Comentarios	Ninguno

3.1.2 Identificación de requisitos funcionales: 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. (UML, 2007)

Relacionado con el requisito de información 1: "Información sobre las **Interfaces de Usuario Concreta y Abstracta**" se han creado dos casos de uso: "gestionar diagramas concretos" y "gestionar diagramas abstractos". En cuanto al requisito de información 2: "Información sobre la **abstracción de Interfaces de Usuario a partir de Interfaces de Usuario Concretas**" se ha creado un caso de uso denominado "convertir diagramas concretos a abstractos". Estos tres casos de uso se han representado en el mismo diagrama. Además, hay un actor denominado "usuario", que representa al usuario que interactúa con la aplicación.

A continuación se muestra el diagrama de casos de uso planteado para el sistema:

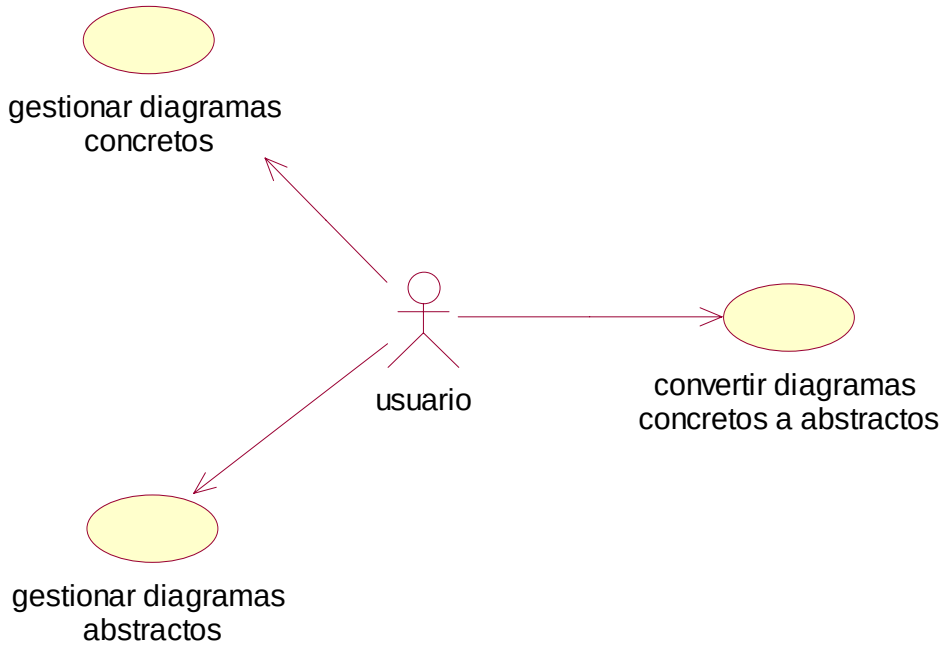


Figura 3.2 Diagrama de casos de uso planteado para el sistema

Como se puede observar, el diagrama anterior se compone de un actor denominado “usuario”, y por tres casos de uso denominados: “gestionar diagramas concretos”, “gestionar diagramas abstractos” y “convertir diagramas concretos a abstractos”. En las siguientes tablas se describe el actor y los casos de uso de la figura anterior.

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

Tabla 3.5 Actor 1: Usuario

ACT–01	Usuario
Versión	1.0
Autor	Francisco Javier Muñoz Márquez
Descripción	Este actor representa a cualquier usuario que interactúe con la aplicación creada en este proyecto.
Comentarios	Ninguno

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

Tabla 3.6 Caso de uso 1: Gestionar diagramas concretos

UC–01	Gestionar diagramas concretos	
Versión	1.0	
Autor	Francisco Javier Muñoz Márquez	
Objetivos asociados	OBJ.-01. Gestionar las Interfaces de Usuario Concreta y Abstracta	
Requisitos asociados	IRQ-01. Información sobre las Interfaces de Usuario Concreta y Abstracta	
Descripción	El usuario puede realizar una serie de operaciones con el diagrama concreto, como: crear un diagrama nuevo, guardar un diagrama, cargar un diagrama, editar un diagrama,… Editar un diagrama consiste en añadir contenedores o componentes concretos al diagrama, eliminarlos, cambiarlos de posición, modificar sus atributos,...	
Precondición	Ninguna	
Secuencia normal	**Paso**	**Acción**
	1	El usuario solicita al sistema la realización de una operación sobre el diagrama concreto.
	2	El sistema realiza la operación solicitada.
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 3.7 Caso de uso 2: Gestionar diagramas abstractos

UC–02	Gestionar diagramas abstractos	
Versión	1.0	
Autor	Francisco Javier Muñoz Márquez	
Objetivos asociados	OBJ.-01. Gestionar las Interfaces de Usuario Concreta y Abstracta	
Requisitos asociados	IRQ-01. Información sobre las Interfaces de Usuario Concreta y Abstracta	
Descripción	El usuario puede realizar una serie de operaciones con el diagrama abstracto, como: crear un diagrama nuevo, guardar un diagrama, cargar un diagrama, editar un diagrama, añadir contenedores y componentes abstractos al diagrama, eliminarlos, cambiarlos de posición, modificar sus atributos,... Además puede añadir una serie de facetas a los componentes abstractos.	
Precondición	Ninguna	
Secuencia normal	**Paso**	**Acción**
	1	El usuario solicita al sistema la realización de una operación en el diagrama abstracto.
	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 3.8 Caso de uso 3: Convertir diagramas concretos a abstractos

UC–03	Convertir diagramas concretos a abstractos	
Versión	1.0	
Autor	Francisco Javier Muñoz Márquez	
Objetivos asociados	OBJ.-02. Gestionar la abstracción de Interfaces de Usuario a partir de Interfaces de Usuario Concretas	
Requisitos asociados	IRQ-02. Información sobre la abstracción de Interfaces de Usuario a partir de Interfaces de Usuario Concretas	
Descripción	Se realiza la conversión de la Interfaz de Usuario Concreta que el usuario ha creado, y se obtiene la Interfaz de Usuario Abstracta equivalente.	
Precondición	El diagrama concreto debe estar previamente guardado.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario solicita al sistema la abstracción del diagrama concreto para obtener el diagrama abstracto.
	2	El sistema transforma el diagrama concreto a abstracto y lo muestra por pantalla.
Poscondición	Ninguna	
Excepciones	**Paso**	**Acción**
	2	Si el sistema detecta algún parámetro incorrecto, avisa al usuario de que no puede llevarse a cabo la transformación.
Rendimiento	**Paso**	**Cota de tiempo**
	2	1 segundo
Frecuencia	Alta	
Estabilidad	Alta	
Comentarios	Ninguno	

3.1.3 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. Describen no lo que el software hará, sino "cómo" lo hará.

Se han identificado dos requisitos no funcionales: *usabilidad* y *rendimiento*. Estos requisitos no funcionales suelen tener vital importancia en las **interfaces de usuario** utilizadas normalmente, ya que una aplicación cuya **interfaz** no sea usable será muy difícil de utilizar por los usuarios. También es importante el rendimiento de la aplicación, ya que si tarda mucho en procesar las órdenes indicadas por los usuarios, la aplicación será pronto reemplazada por otra más eficiente. Las siguientes dos tablas describen los requisitos no funcionales citados anteriormente.

Tabla 3.9 Requisito No Funcional 1: Usabilidad e interfaz amigable

RNF–01	Usabilidad e interfaz amigable
Versión	1.0
Autor	Francisco Javier Muñoz Márquez
Objetivos asociados	OBJ.-01. Gestionar las Interfaces de Usuario Concreta y Abstracta OBJ.-02. Gestionar la abstracción de Interfaces de Usuario a partir de Interfaces de Usuario Concretas
Requisitos asociados	IRQ-01. Información sobre las Interfaces de Usuario Concreta y Abstracta IRQ-02. Información sobre la abstracción de Interfaces de Usuario a partir de Interfaces de Usuario Concretas
Descripción	Para que el usuario consiga manejar con sencillez la aplicación que se está desarrollando, la interfaz de la aplicación debe ser amigable, y de este modo el usuario podrá intuir su funcionamiento.
Importancia	Vital
Urgencia	Inmediata
Estado	Construido
Estabilidad	Alta
Comentarios	Ninguno

Tabla 3.10 Requisito No Funcional 2: Rendimiento de las operaciones

RNF–02	Rendimiento de las operaciones
Versión	1.0
Autor	Francisco Javier Muñoz Márquez
Objetivos asociados	OBJ.-01. Gestionar las Interfaces de Usuario Concreta y Abstracta OBJ.-02. Gestionar la abstracción de Interfaces de Usuario a partir de Interfaces de Usuario Concretas
Requisitos asociados	IRQ-01. Información sobre las Interfaces de Usuario Concreta y Abstracta IRQ-02. Información sobre la abstracción de Interfaces de Usuario a partir de Interfaces de Usuario Concretas
Descripción	Se debe obtener un buen rendimiento a la hora de ejecutar la aplicación, sobre todo a la hora de realizar la **abstracción de la interfaz de usuario abstracta** a partir de la **interfaz de usuario concreta**. Si no fuera así, el usuario no estaría satisfecho con el rendimiento de la aplicación, y por muy buena que fuera en cuanto al cumplimiento de todos los requisitos funcionales, ésta perdería muchos adeptos.
Importancia	Vital
Urgencia	Inmediata
Estado	Construido
Estabilidad	Alta
Comentarios	Ninguno

3.2 FASE DE ANÁLISIS DE REQUISITOS

En esta 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 concreto, esta sección se divide en tres subapartados. El primero trata las correspondencias establecidas entre los contenedores y componentes concretos que el usuario dispondrá para crear la **interfaz de usuario** a su medida. En el segundo subapartado se muestra la estructura del diagrama de paquetes creado de acuerdo con los requisitos, de tal manera que permita simplificar las posteriores fases de diseño e implementación de la aplicación. Finalmente, en el tercer subapartado, se dará una descripción del diagrama de clases realizado para el paquete de objetos concretos **(CIO)**.

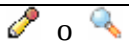
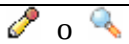
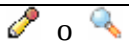
3.2.1 Asociación de facetas a contenedores y componentes concretos

Se han seleccionado los contenedores y componentes concretos más importantes disponibles en los principales entornos de programación. Se han analizado dichos componentes y contenedores concretos, y de ese análisis se han establecido una serie de facetas asociadas a cada componente concreto (los contenedores no tienen facetas asociadas), para establecer un criterio de asociación estándar a lo largo de todo el proyecto. En el apartado 2.3.1.2.1 de este documento puede encontrarse una definición de las facetas AUI.

Los contenedores concretos son: Panel y TabbedPane. Los contenedores concretos, son un tipo determinado de objetos que pueden tener dentro otros contenedores o componentes concretos. Los contenedores concretos no tienen ninguna faceta asociada, ya que las facetas sólo se asocian a los componentes concretos. Se han considerado los siguientes componentes concretos: Button, RadioButton, ToggleButton, CheckBox, Label, TextField, TextArea, TextPane, EditorPane, PasswordField, ComboBox, List y el ScrollBar. Cada componente tiene una o más facetas asociadas, dependiendo de las facetas que dicho componente utilice.

A continuación se muestran las facetas asociadas a cada uno de los siguientes objetos de interacción concretos:

Tabla 3.11 Asociación de facetas a contenedores y componentes concretos

Tipo	Icono	Contenedor/ componente	Facetas AUI
Panel			---
TabbedPane			---
Button			
RadioButton			
ToggleButton			
CheckBox			
Label			
TextField			
TextArea			
TextPane			
EditorPane			
PasswordField			
ComboBox			
List			
ScrollBar			

Como puede observarse en la tabla anterior, los contenedores concretos no tienen ninguna faceta asociada, ya que las facetas sólo se asocian a los componentes.

3.2.2 Diagrama de paquetes

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

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

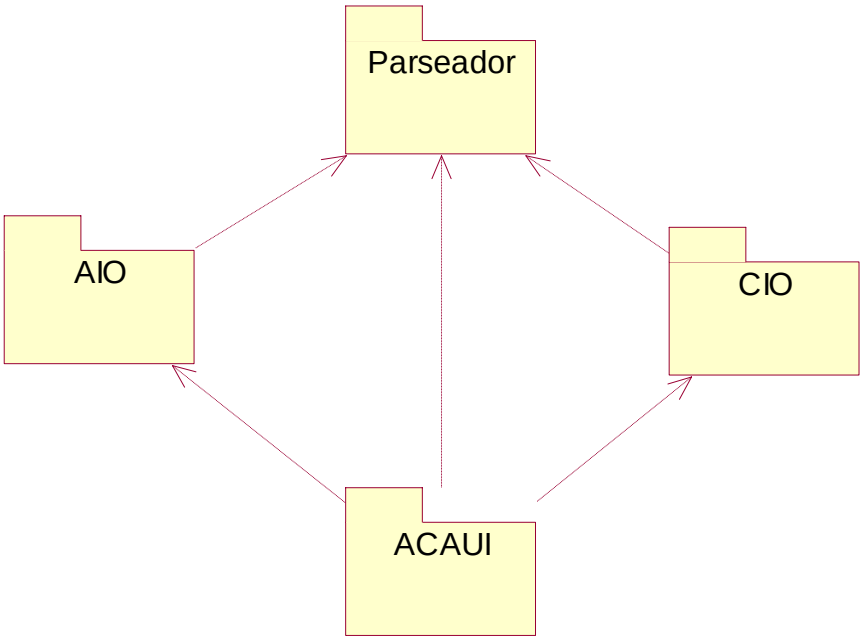


Figura 3.3 Diagrama de paquetes del sistema desarrollado

A continuación se definen brevemente los paquetes del diagrama:

- **ACAUI**: este paquete es el principal, ya que contiene la aplicación que lanzará a ejecución el programa. Contiene las clases necesarias para el manejo adecuado del árbol, para que se actualice automáticamente, y se pueda realizar operaciones a través de él.

  También contiene las clases necesarias para el funcionamiento de los menús principales y los menús contextuales, para permitir realizar ciertas operaciones mediante su uso. Además contiene las clases necesarias para la

implementación de un panel de propiedades de los objetos (contenedores y componentes) para facilitar el manejo al usuario final.

Está relacionado con los paquetes “AIO” y “CIO”, ya que estos paquetes implementan las **interfaces de usuario abstracta y concreta** respectivamente, y dichas interfaces son invocadas por este módulo. Y está relacionado con el paquete “Parseador” para realizar la transformación de interfaces, partiendo de la **interfaz de usuario concreta** para obtener la **interfaz de usuario abstracta**.

- **CIO**: este paquete contiene la parte correspondiente a la **interfaz de usuario concreta**. Contiene una serie de clases (mostradas en la figura 3.4) para permitir al usuario crear **interfaces concretas** con una serie de contenedores y componentes concretos.

- **AIO**: este paquete contiene la parte correspondiente a la **interfaz de usuario abstracta**. Permite al usuario crear **interfaces abstractas** con una serie de contenedores y componentes abstractos.

- **Parseador**: este paquete proporciona soporte para la lectura, procesamiento y creación de ficheros que siguen el estándar **UsiXML**, lo que nos facilitará la tarea para crear ficheros para las **interfaces de usuario concretas** y para las **interfaces de usuario abstractas**.

3.2.3 Diagrama 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)

Dentro del paquete “ACAUI” (definido en el subapartado anterior), cabe destacar el siguiente diagrama de clases, ya que debido a la herencia de las clases que extienden de JComponent, ha resultado mucho más sencilla la creación y modificación de elementos en la **interfaz de usuario concreta**.

El diagrama de clases que se ha utilizado para el paquete CIO es el siguiente:

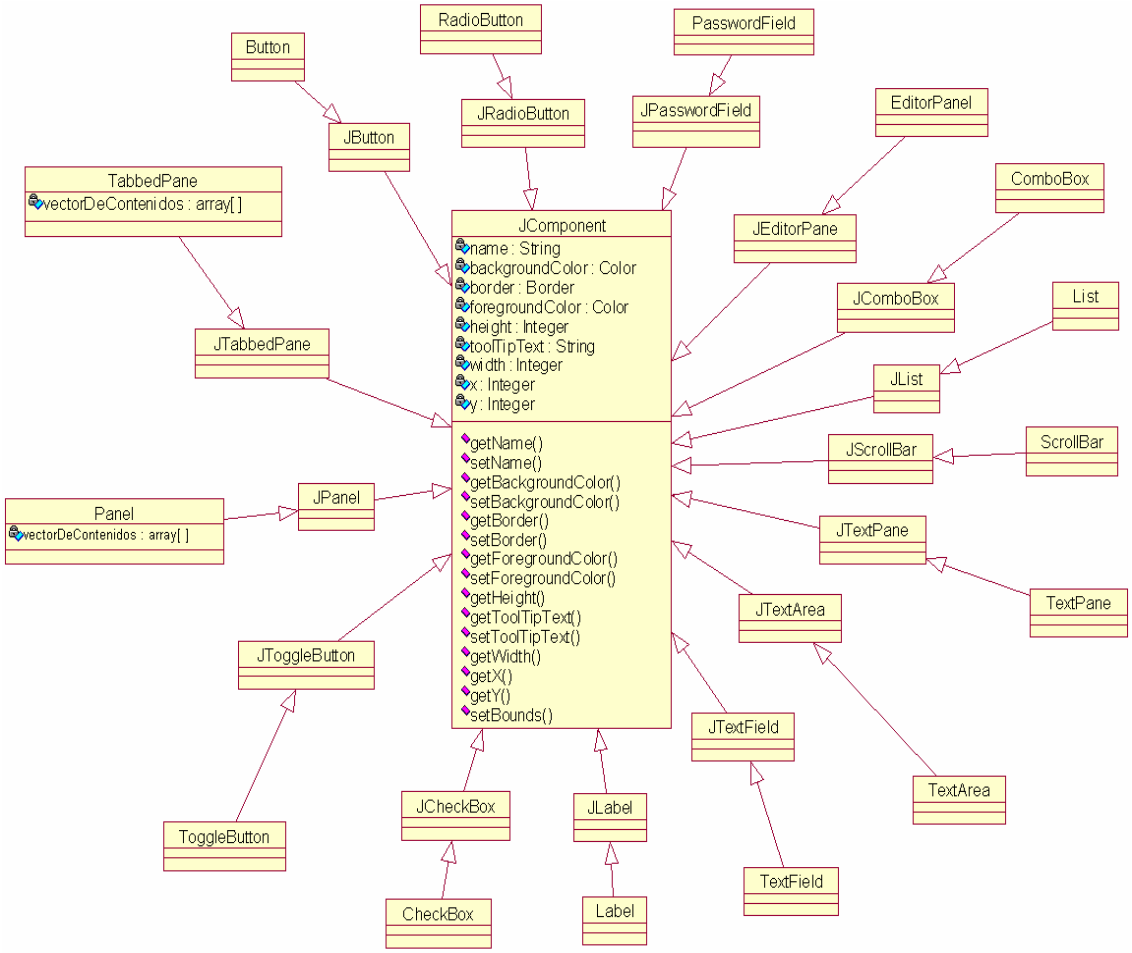


Figura 3.4 Diagrama de clases del paquete CIO

Como se puede observar para facilitar la implementación de los distintos componentes y contenedores usados en este módulo, se han reutilizado los proporciona por el lenguaje de programación 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.

Los dos primeros niveles los proporciona Java, y el tercero se ha añadido para modificar las clases del segundo nivel. Por ejemplo, la clase "Button" hereda las propiedades y métodos de la clase "JButton", y de esta manera se ahorra mucho tiempo y trabajo, ya que se aprovecha la implementación de dichas clases. Lo mismo ocurre con las demás clases del nivel inferior.

En las clases del tercer nivel se ha añadido funcionalidad como: posibilidad de mover objetos, posibilidad de redimensionar objetos, eliminar objetos, añadir objetos

dentro de contenedores (Panel y TabbedPane), cambio automático del puntero del ratón si el objeto puede redimensionarse o moverse,…

Las clases “Panel” y “TabbedPane” son contenedores concretos, es decir, dentro de ellas se pueden añadir componentes concretos, e 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, en los contenedores de tipo “Panel” o en los de tipo “TabbedPane”. Estos dos contenedores puede observarse que tienen un atributo denominado “vectorDeContenidos”, cuya utilidad es almacenar en dicho vector los objetos que se agreguen a los contenedores del tipo “Panel” y “TabbedPane”, y así facilitar la gestión del diagrama. De esta manera resultará mucho más fácil la modificación de objetos del diagrama, y el posterior almacenamiento del diagrama en un fichero.

3.3 FASE DE DISEÑO

En esta fase de diseño se crea una especificación de la **interfaz de usuario** suficientemente detallada para poder llegar a su implementación en la siguiente fase del proceso de desarrollo. La **interfaz de usuario** creada en esta fase de diseño permite describir la futura **interfaz de usuario** utilizando elementos de alto nivel.

Para realizar el diseño de las dos **interfaces de usuario** necesarias (una para realizar el diagrama de la **interfaz de usuario concreta**, y otra para mostrar el diagrama de la **interfaz de usuario abstracta** resultante) se ha realizado un diagrama de objetos de interacción abstracta para cada una de **las interfaces de usuario** necesarias. Mediante estos diagramas se ha realizado un diseño de manera abstracta de las dos **interfaces de usuario** con contenedores y componentes abstractos, y cada componente con sus facetas asociadas.

Es importante diferenciar el diseño de las dos **interfaces de usuario** necesarias. La primera interfaz se utiliza para que el usuario realice el diagrama de la **interfaz de usuario concreta**, y la segunda interfaz es para mostrarle al usuario el diagrama de la **interfaz de usuario abstracta** asociada.

A continuación se muestra el diseño de la interfaz de usuario que permite crear los diagramas de **especificación concreta**:

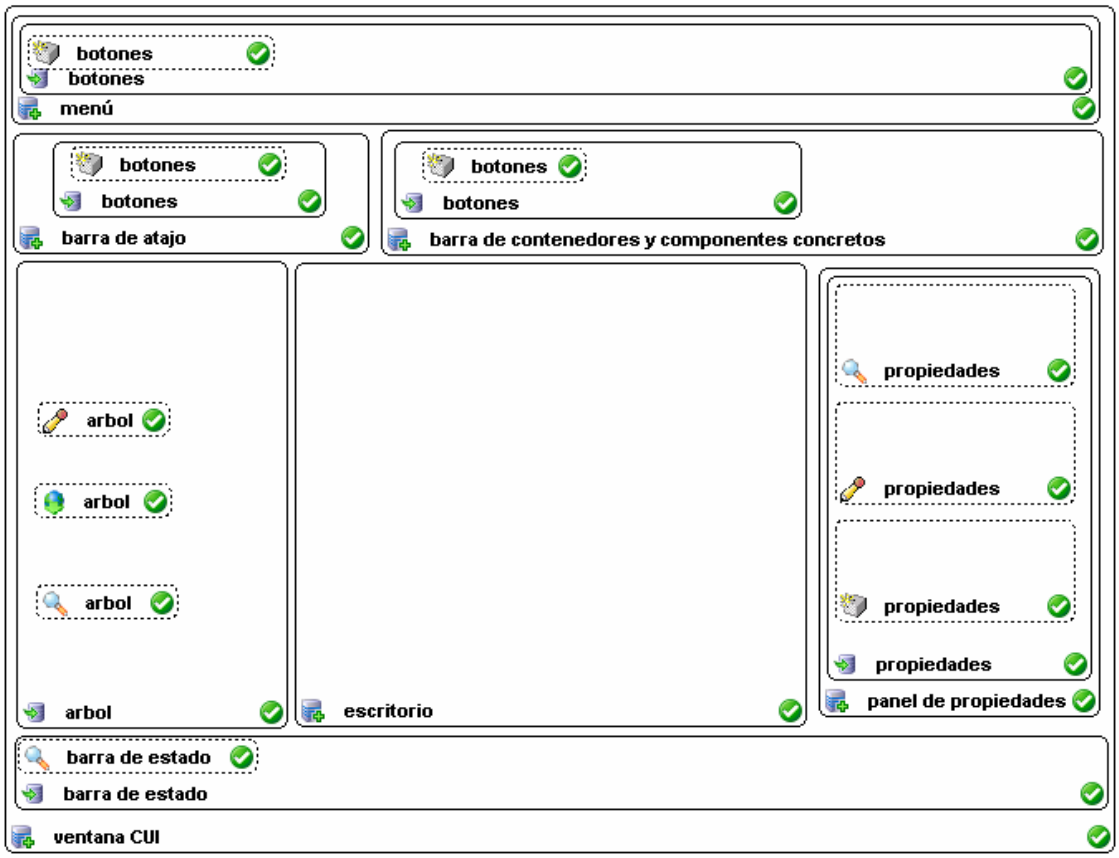


Figura 3.5 Diseño de la Interfaz de la aplicación (referente a la parte concreta)

Se observa en la figura anterior que la interfaz se divide en siete secciones claramente diferenciadas. Una primera sección para mostrar un menú al usuario, donde éste pueda elegir entre distintas opciones. Una segunda sección para una barra de botones de atajo que englobe algunas de las opciones más utilizadas del menú anterior. Una tercera sección que contenga unos botones correspondientes a los contenedores y componentes concretos que el usuario pueda insertar en la sección denominada escritorio. Una cuarta sección contendrá un árbol, para facilitar al usuario la navegación por la interfaz, también para mostrarle una estructura jerárquica de la interfaz que esté realizando, y dónde se espera que el usuario pueda realizar algunas operaciones. La quinta sección es el escritorio, que será donde el usuario cree su **interfaz de usuario concreta**, insertando contenedores y componentes concretos. La sexta sección es el panel de propiedades, donde se podrán realizar algunos cambios sobre los contenedores y componentes concretos que el usuario haya añadido al escritorio. Por último, se ha considerado útil una barra de estado para mostrar al usuario la última operación realizada.

Por otro lado, se ha realizado el diseño de la otra interfaz correspondiente a la **especificación abstracta,** es decir, la interfaz que muestra el resultado tras la abstracción de la **interfaz de usuario concreta** creada por el usuario.

A continuación se muestra el diseño de la interfaz de usuario correspondiente a la **especificación abstracta**:

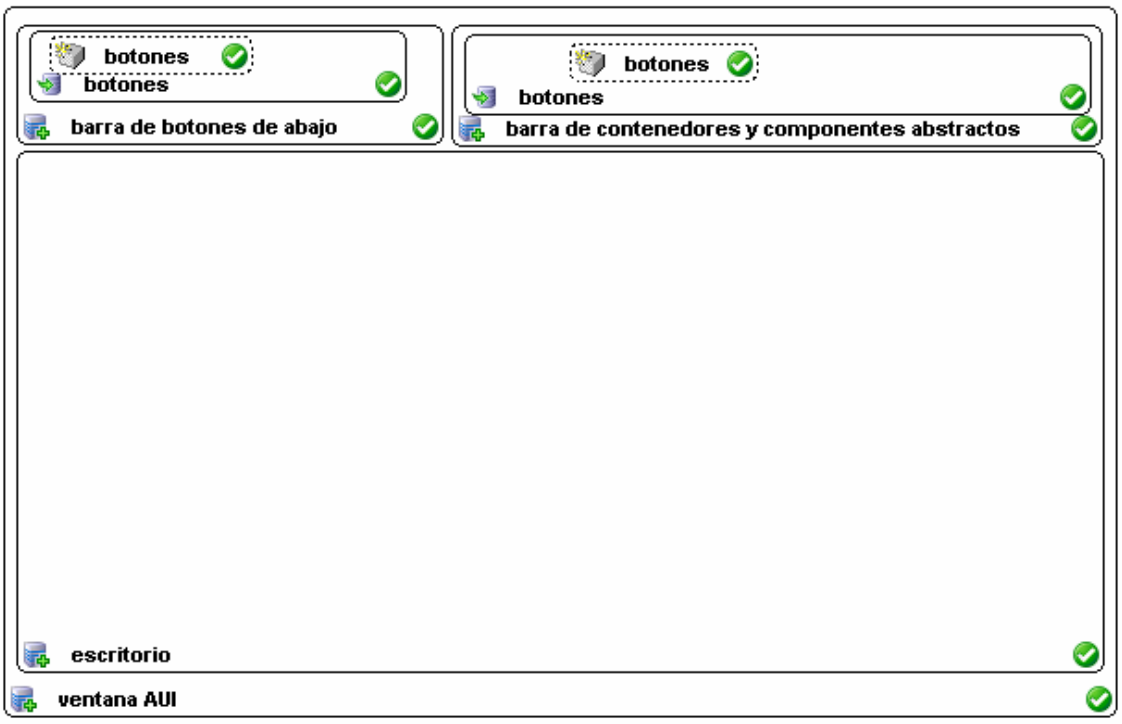


Figura 3.6 Diseño de la Interfaz de la aplicación (referente a la parte abstracta)

Se observa en la figura anterior que la interfaz se divide en tres secciones. La primera sección mostrará al usuario una serie de botones de atajo, con botones con opciones para guardar o cargar diagramas. La segunda sección mostrará una barra de botones correspondientes a los contenedores y componentes abstractos que el usuario podrá insertar en el escritorio. La última sección es el escritorio, que será donde se cargue el diagrama correspondiente a la **interfaz de usuario abstracta**, y además el usuario podrá realizar las modificaciones que considere necesarias sobre el diagrama creado.

3.4 FASE DE IMPLEMENTACIÓN

Una vez realizado el diseño de la aplicación en desarrollo, pasamos a la fase de implementación. Esta aplicación se ha implementado en Java, y en concreto con la herramienta Borland JBuilder. La elección de Borland JBuilder se debe a que a lo largo de la carrera 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.

Para implementar la aplicación ACAUI, se han seguido las especificaciones denotadas en las fases anteriores a la esta fase. Las interfaces de usuario concretas construidas (son necesarias dos: una principal y otra para los resultados) han sido obtenidas concretando detalles a partir de la especificación abstracta realizada en la fase de diseño.

Partiendo del diseño de la interfaz de usuario principal realizado en la figura 3.5, se ha obtenido la interfaz de usuario mostrada en la siguiente figura:

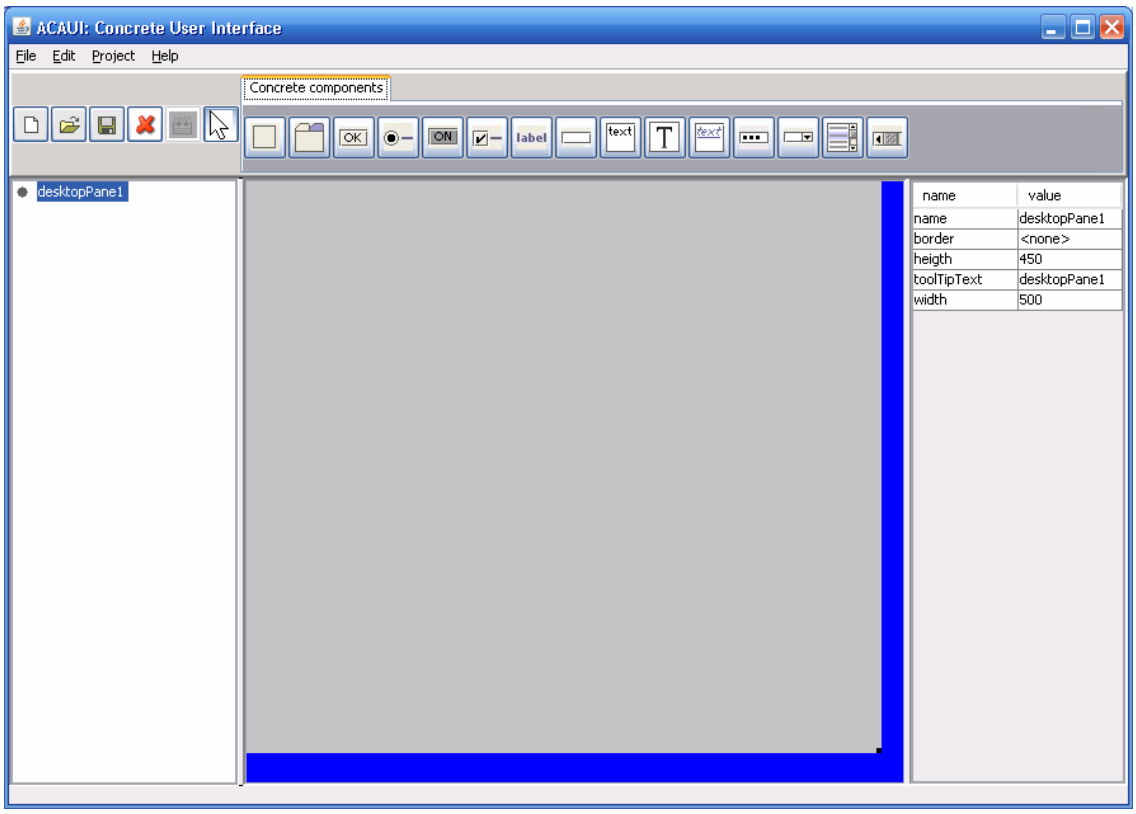


Figura 3.7 Interfaz Final de la aplicación (interfaz principal)

Como puede observarse en la figura anterior, se ha seguido al pié de la letra el diseño de esta interfaz realizado en la fase de diseño. Como se indicó en el diseño, la

interfaz se compone de siete secciones claramente diferenciadas. La primera sección consiste en un menú para que el usuario pueda realizar varias acciones como: abrir, guardar, guardar como, borrar, borrar todo, generar AUI, ayuda,… La segunda sección es una barra de botones de atajo que contiene algunas de las opciones más utilizadas del menú anterior como son: abrir, guardar, eliminar, generar AUI, y además tiene el cursor del ratón. La tercera sección contiene unos botones correspondientes a los contenedores y componentes concretos que el usuario puede insertar en el escritorio o en otros contenedores: Panel, TabbedPane, Button, RadioButton,… La cuarta sección contiene un árbol, para facilitar al usuario la navegación a lo largo de la interfaz, también para mostrarle una estructura jerárquica de la interfaz que está realizando, y dónde el usuario puede realizar algunas operaciones. La quinta sección es el escritorio, y será donde el usuario cree su **interfaz de usuario concreta**, insertando contenedores y componentes concretos. La sexta sección se es un panel de propiedades en el que se pueden realizar algunos cambios sobre los contenedores y componentes que el usuario haya añadido al escritorio; algunas de las propiedades que aparecen son: nombre, tipo de borde, altura, anchura,… La séptima y última sección es una barra de estado para informar al usuario de la última operación realizada.

Y la otra interfaz de usuario, es decir, la que muestra la interfaz de usuario resultante de la abstracción realizada, se ha obtenido partiendo de la interfaz de usuario abstracta de la figura 3.6:

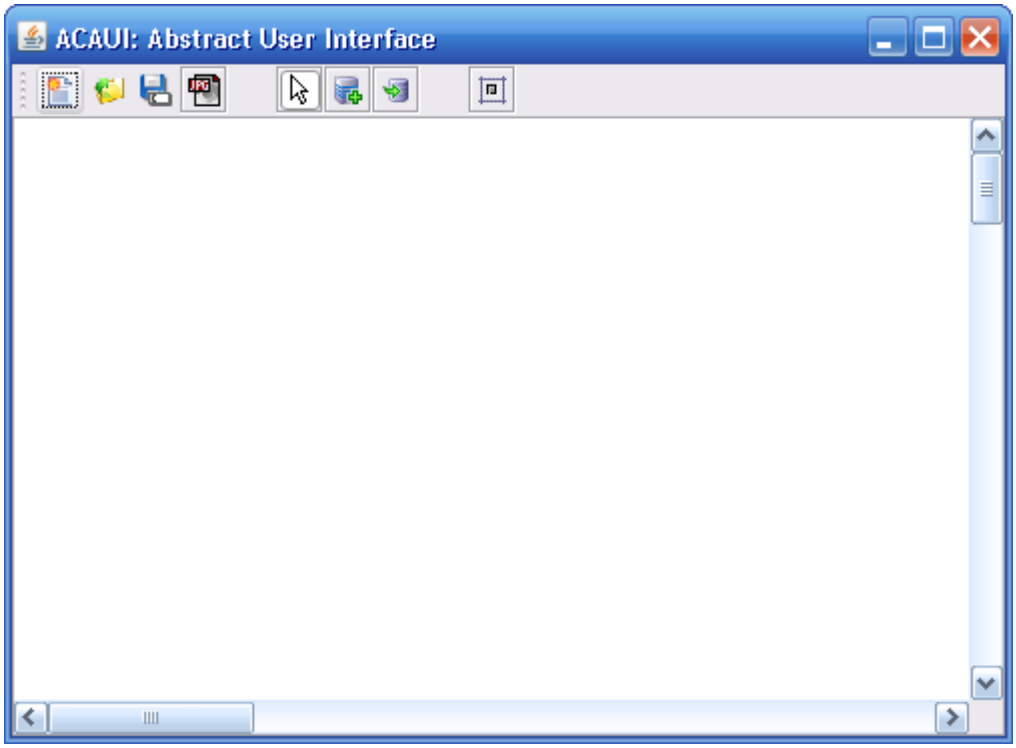


Figura 3.8 Interfaz Final de la aplicación (interfaz de resultados)

Se observa en la figura anterior, que se ha seguido correctamente el diseño de esta interfaz (dado en la fase de diseño). Como se indicó en el diseño, la interfaz se compone de tres secciones claramente diferenciadas. La primera sección muestra al usuario una serie de botones de atajo, con botones con las siguientes opciones: crear un nuevo diagrama, cargar un nuevo diagrama, guardar un diagrama o guardar el diagrama como imagen. La segunda sección muestra una barra de botones que contiene los siguientes botones: un botón con el cursor, un botón para añadir contenedores abstractos, otro botón para añadir componentes abstractos y otro botón para organizar el escritorio. La última sección es el escritorio, que es donde se carga la **interfaz de usuario abstracta**, y además el usuario puede realizar las modificaciones que considere necesarias sobre el diagrama creado.

Por otro lado, se ha utilizado la librería de código abierto "Dom4j" (**M**odelo de **O**bjetos del **D**ocumento para **J**ava) para facilitar el trabajo con los ficheros XML que se tienen que generar y manejar en esta aplicación. "Dom4j" permite trabajar con ficheros XML en la plataforma Java.

"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. Su página oficial es: www.dom4j.org, y en ella se puede encontrar mucha información sobre esta librería.

Por otro lado, se ha reutilizado el módulo AIO (**A**bstract **I**nteraction **O**bjects), ya que su funcionalidad era totalmente compatible con la necesaria para la interfaz de usuario abstracta.

Este módulo implementa toda la funcionalidad requerida por la interfaz de usuario a nivel abstracto. Permite crear diagramas de especificación abstracta, donde el usuario puede añadir contenedores y componentes abstractos, y a dichos componentes les podrá añadir facetas (entrada, salida, control y navegación).

Dicho módulo permite guardar los diagramas creados. Estos diagramas serán almacenados en ficheros con extensión ".usi", los cuales siguen el estándar UsiXML.

También permite cargar los diagramas previamente guardados con extensión ".usi". En estos ficheros se almacena toda la información relevante del diagrama correspondiente a la **interfaz de usuario abstracta**. Así, al pasarle un fichero de especificación abstracta con extensión ".usi", genera la **interfaz de usuario abstracta** correspondiente.

Capítulo 4: Caso de estudio

4.1 EXPLICACIÓN DETALLADA DE LA APLICACIÓN

A la hora de interactuar con la aplicación, el usuario debe seguir una secuencia como la mostrada a continuación:

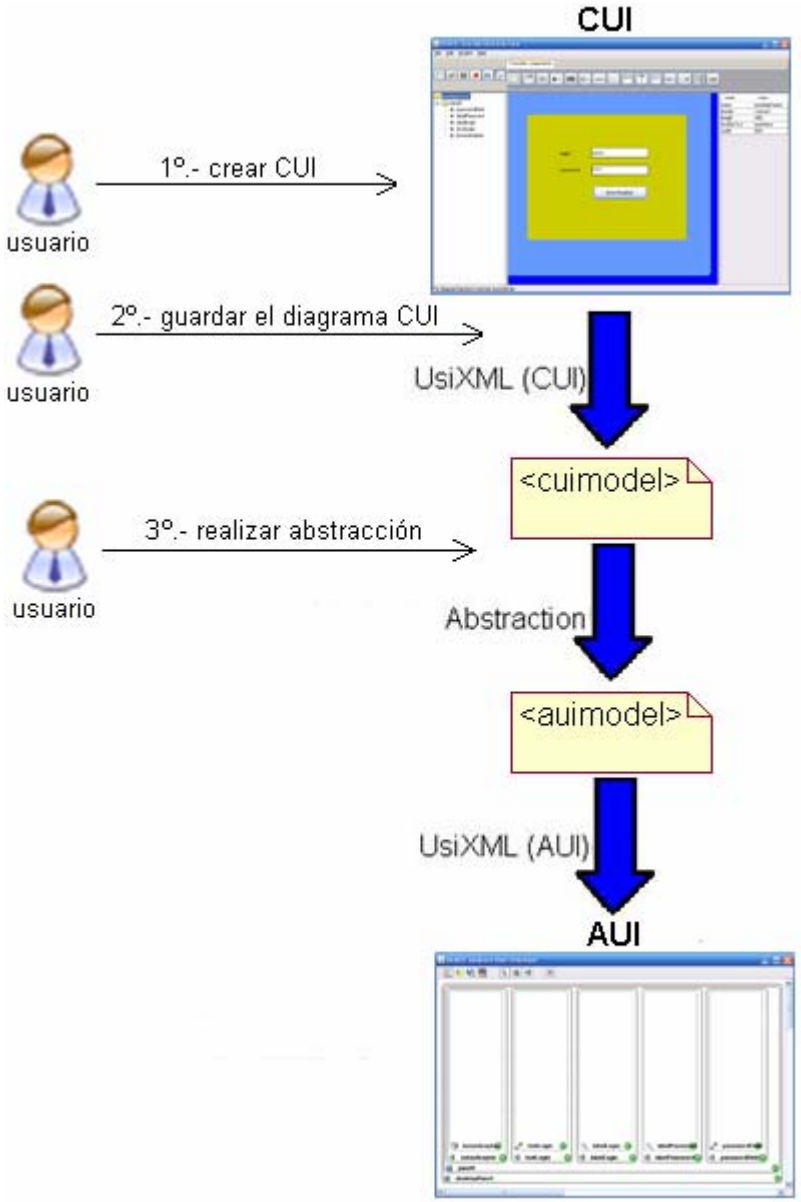


Figura 4.1 Traza de la secuencia seguida por el usuario

En primer lugar, el usuario debe crear la **interfaz de usuario concreta** mediante los componentes y contenedores concretos proporcionados por la aplicación.

Una vez creada la **interfaz de usuario concreta**, el siguiente paso es guardarla en un fichero con extensión ".cui", para lo cuál hay que pulsar el botón  mostrado en la barra de botones de atajo.

El último paso es realizar la **abstracción de la interfaz de usuario** creada anteriormente. Para realizar dicha abstracción, el usuario debe pulsar el botón  , y automáticamente el sistema realizará la abstracción, obteniéndose un fichero temporal de extensión ".usi", con la información de la **interfaz de usuario abstracta** asociada a la **interfaz de usuario concreta** que creó. Dicho fichero temporal se crea de manera oculta al usuario, por lo que el usuario no se cerciorará de la creación de este fichero. Después de la creación del fichero de **especificación abstracta**, Automáticamente se carga dicho fichero para mostrar la **interfaz de usuario abstracta**.

El entorno de la herramienta ACAUI es un entorno típico con un menú, una barra de botones de atajo, una barra de contenedores y componentes concretos, un árbol, un panel principal y un panel de propiedades:

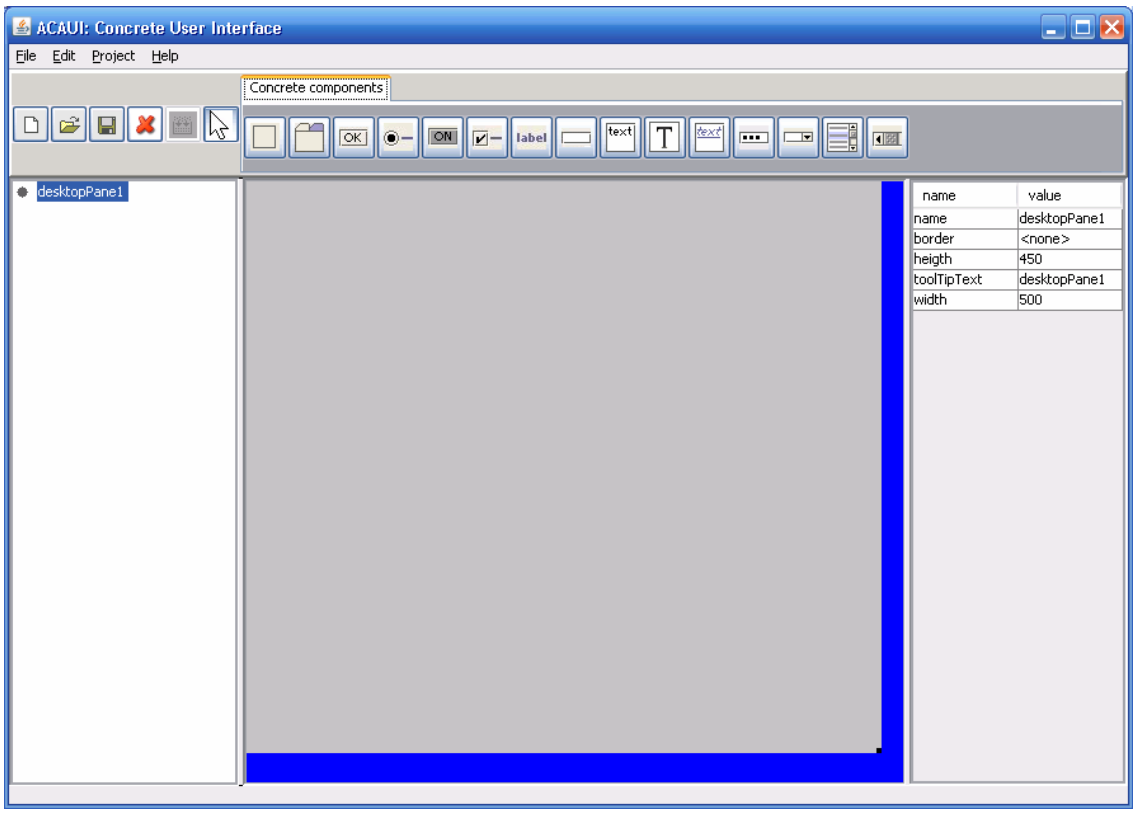


Figura 4.2 Entorno ACAUI en la interfaz de usuario concreta

El usuario dispone de una serie de menús y botones que le facilitarán la creación de la **interfaz de usuario concreta**. El usuario puede agregar:

- contenedores concretos como: Panel y TabbedPane.
- componentes concretos como: Button, RadioButton, ToggleButton, CheckBox, Label, TextField, TextArea, TextPane, EditorPane, PasswordField, ComboBox, List y ScrollBar.

El usuario puede añadir tantos componentes o contenedores como él considere, e incluso puede añadir contenedores y componentes dentro de otros contenedores.

A continuación se muestra la barra de contenedores y componentes que el usuario dispone para crear su **interfaz de usuario concreta**:

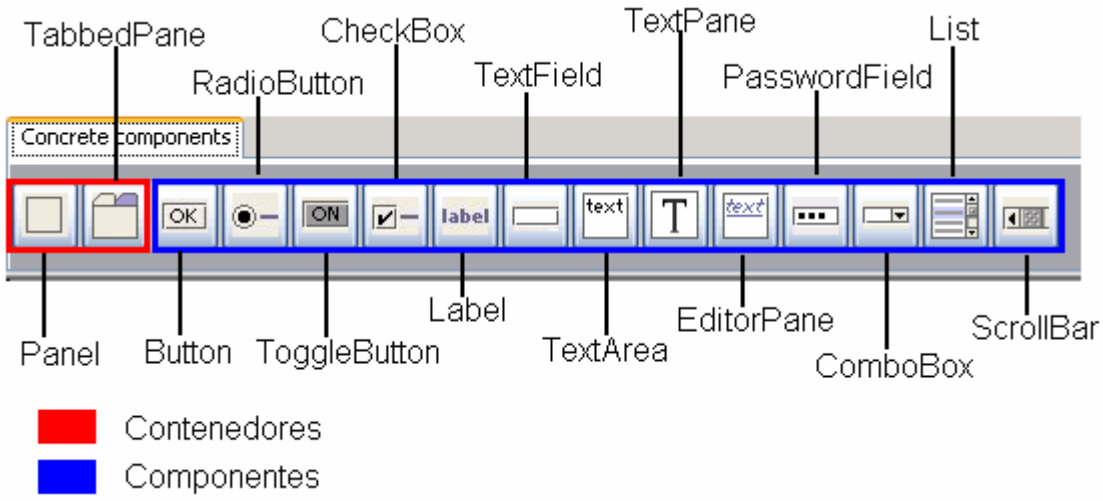


Figura 4.3 Barra de contenedores y componentes concretos

Al seleccionar uno de los componentes o contenedores de la figura anterior, el usuario puede agregar una instancia suya al panel inicial o al contenedor que desee.

En la parte derecha de la figura anterior, 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 pequeña guía para observar la estructura de la interfaz creada.

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

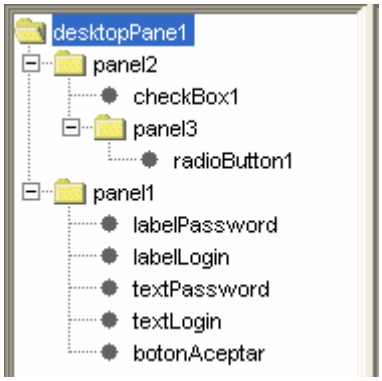


Figura 4.4 Árbol de componentes mostrado en la herramienta desarrollada

También puede observarse una barra de estado en la parte inferior izquierda de la figura 4.2. Esta barra de estado informa al usuario del último cambio realizado. El idioma de la barra de estado es el inglés, ya que es el idioma más utilizado a nivel mundial.

Por otro lado, en la parte derecha de la figura 4.2 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. A continuación se muestra un ejemplo del estado del panel de propiedades una vez seleccionado un componente de tipo Button:

name	value
name	boton1
border	<none>
heigth	33
text	Aceptar
toolTipText	OK
width	108

Figura 4.5 Panel de propiedades mostrado en la herramienta desarrollada

También cabe señalar la presencia en cada contenedor y componente concreto de un menú contextual con el que se puede realizar tres operaciones:

- set foreground color: para establecer el color del primer plano, es decir de las letras. Para facilitar el trabajo se despliega una paleta de colores como la de la figura 4.7.

- set background color: para establecer el color del fondo. Para facilitar el trabajo se despliega una paleta de colores como la de la figura 4.7.
- delete: permite eliminar el contenedor o componente concreto seleccionado.

A continuación se muestra dicho menú contextual:

Figura 4.6 Menú contextual mostrado en la herramienta desarrollada

A continuación se muestra la paleta de colores disponible:

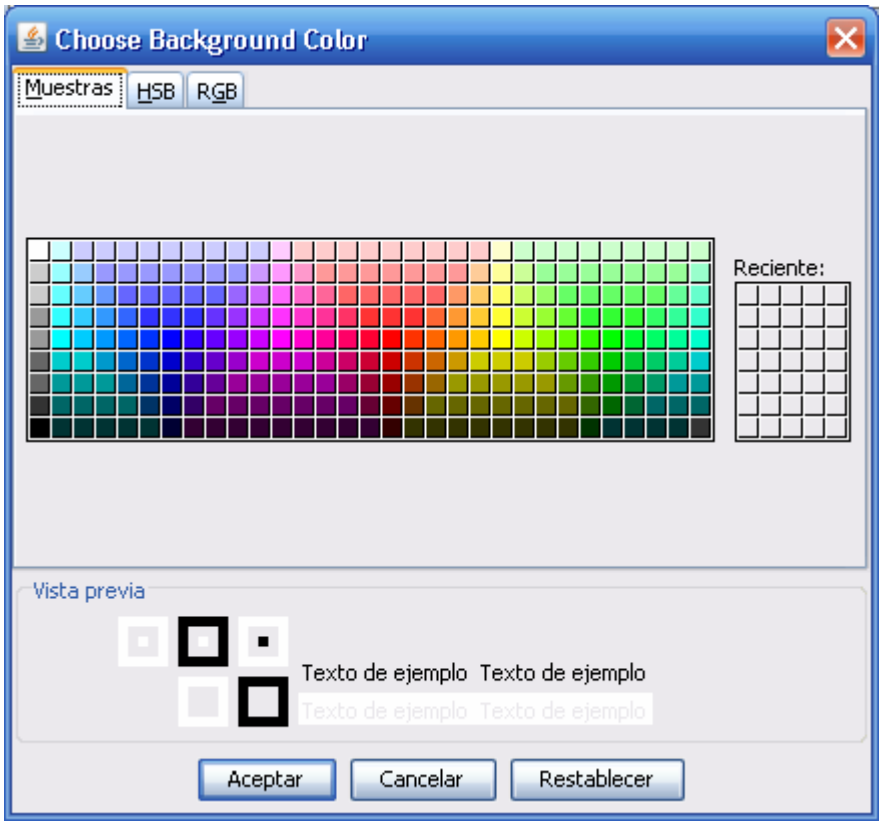


Figura 4.7 Paleta de colores de la herramienta desarrollada

Además se dispone de una serie de botones de atajo que le permiten realizar diversas operaciones útiles como: crear nuevo diagrama, abrir un diagrama, guardar un diagrama, eliminar un contenedor o componente del diagrama, convertir de CUI a AUI y seleccionar.

A continuación se muestra la barra de botones de atajo:

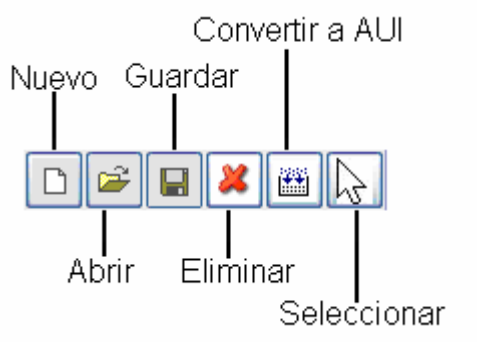


Figura 4.8 Barra botones de atajo de ACAUI en la interfaz de usuario concreta

Cabe destacar la importancia del botón “Convertir a AUI”, ya que se utiliza para la conversión de la interfaz de usuario concreta a la interfaz de usuario abstracta. Este botón aparecerá como desactivado hasta que el usuario guarde por primera vez el diagrama, ya que la transformación toma como entrada el fichero con extensión “.cui” generado al guardar el diagrama. En cuanto se guarde el diagrama, el botón aparecerá como activado, y el usuario ya podrá realizar la transformación con sólo pulsar el botón “Convertir a AUI”.

Cuando se realiza la abstracción de CUI a AUI, realmente lo que se hace es convertir el fichero “.cui” almacenado previamente, en un fichero temporal con extensión “.usi”.

Una vez se tiene el fichero “.usi”, éste es cargado por un entorno que se superpone al anterior, en el que nos muestra la interfaz de usuario abstracta obtenida.

A continuación se nos muestra el entorno ACAUI en la interfaz de usuario abstracta:

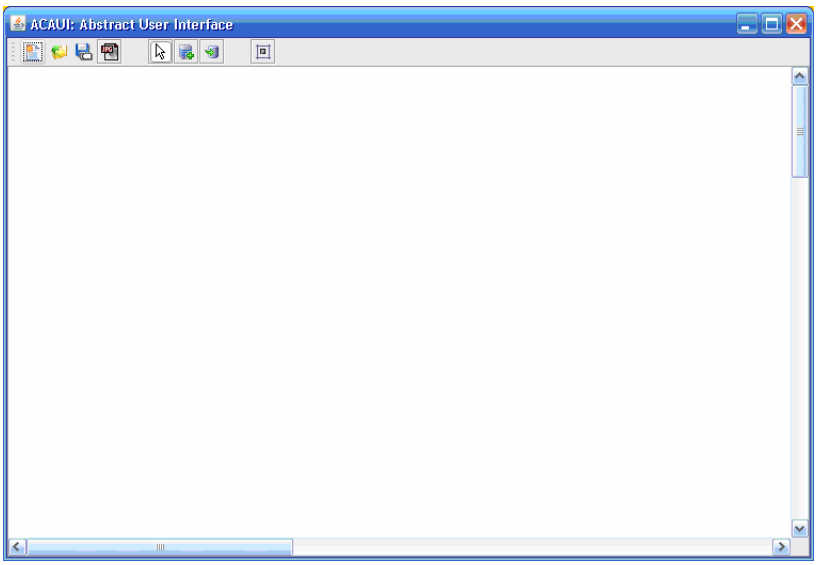


Figura 4.9 Entorno ACAUI en la interfaz de usuario abstracta

En la figura anterior se muestra la interfaz de usuario abstracta obtenida tras convertir la interfaz de usuario concreta anterior. Además, el usuario dispone de una barra de botones que vamos a dividir en dos partes.

Por un lado contiene cuatro botones: seleccionar, contenedor, componente y organizar. El botón *seleccionar* permite seleccionar un elemento de la **interfaz de usuario abstracta**. El botón *contenedor* permite crear un contenedor en el panel principal, pero no es otros contenedores. El botón *componente* permite crear un componente en el panel principal, pero no en otros contenedores. El botón *organizar* permite organizar todos los contenedores, componentes y facetes que haya en la **interfaz de usuario abstracta**.

A continuación se muestra una figura con los botones anteriormente explicados:

Figura 4.10 Barra de contenedores y componentes abstractos

Por otro lado, en la siguiente figura se pueden observar cuatro botones. Estos botones son los utilizados para realizar las operaciones típicas de: nuevo, abrir, guardar y guardar como gráfico (guarda el diagrama en un archivo con extensión ".jpg").

Figura 4.11 Barra botones de atajo de ACAUI en la interfaz de usuario abstracta

Cada contenedor abstracto dispone de un menú de propiedades, que podemos ver al hacer clic derecho sobre el contenedor deseado. En él podemos ver las siguientes opciones:

- add container: permite añadir un contenedor al contenedor abstracto seleccionado.
- add component: permite añadir un componente al contenedor abstracto seleccionado.
- edit: permite editar las propiedades del contenedor abstracto. En la figura 4.13 puede observarse un ejemplo de edit.
- set color: permite establecer el color del contenedor abstracto seleccionado. Para facilitar la tarea, se nos proporciona una paleta de colores que podemos ver en la figura 4.7.
- arrange: permite organizar todos los elementos contenidos en el contenedor abstracto seleccionado.
- delete: permite eliminar el contenedor abstracto seleccionado.

A continuación se muestra una figura con el menú de propiedades de los contenedores abstractos anteriormente explicados:

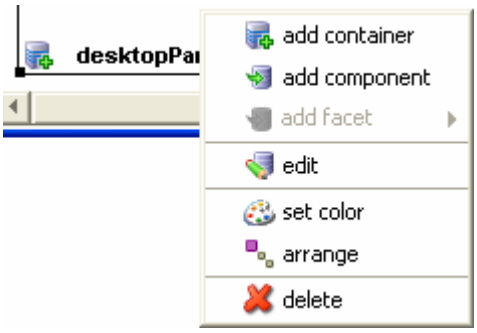


Figura 4.12 Menú contextual de los contenedores abstractos

A continuación se muestra una figura con las propiedades del contenedor:

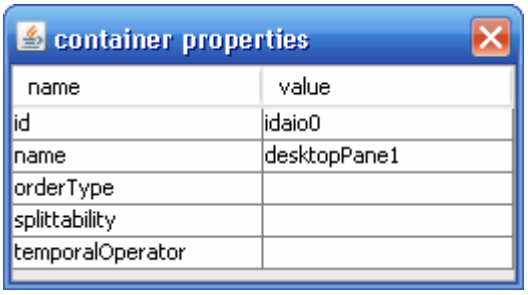


Figura 4.13 Propiedades del contenedor abstracto

En el caso de los componentes abstractos, el menú de propiedades es parecido al de los contenedores abstractos, pero teniendo en cuenta que en los componentes

abstractos no se pueden insertar ni contenedores ni componentes abstractos. En cambio, los componentes abstractos suelen llevar unas facetas asociadas. Dichas facetas pueden ser:

- input: faceta de entrada.
- output: faceta de salida.
- navigation: faceta de navegación.
- control: faceta de control.

En la sección 2.3.1.2.1 de este documento se explican con detalle las facetas AUI.

A continuación se muestra una figura con el menú de propiedades de los componentes abstractos:

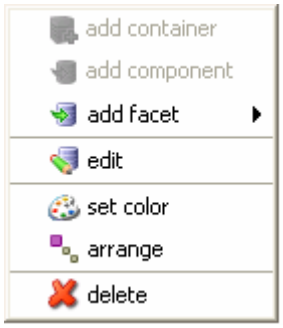


Figura 4.14 Menú contextual de los componentes abstractos

En cuanto a las facetas, tienen el siguiente menú contextual asociado:

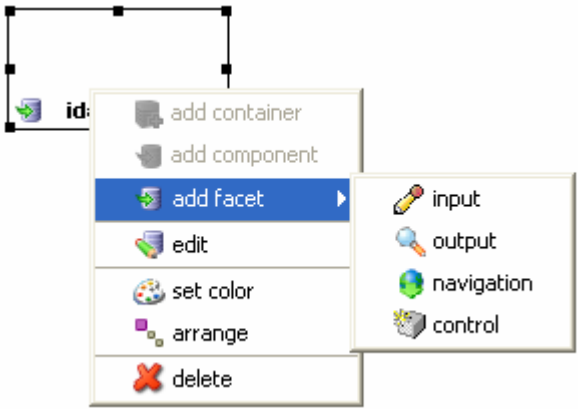


Figura 4.15 Menú contextual de las facetas

Donde los ítems que aparecen tienen el mismo comportamiento que los explicados anteriormente para contenedores y componentes abstractos.

4.2 MANUAL DE 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 y también ha sido traducido a inglés. 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, así como las opciones de los menús,…

Además, se han realizado varios ejemplos prácticos, de tal manera que el usuario pueda observar paso a paso cómo se utiliza la herramienta.

La estructura seguida por el manual de ayuda es la siguiente:

1.- ¿Qué es ACAUI?
2.- ¿Cómo utilizar la herramienta ACAUI?
3.- Ejemplos
    3.1.-Ejemplo1: Pantalla de acceso
    3.2.-Ejemplo2: Traductor
    3.3.-Ejemplo3: Días vividos
    3.4.-Ejemplo4: PuTTY Terminal
4.- Acerca de

El manual se puede consultar en la sección “Help” del menú, y luego en “Help topics (English)” o “Help topics (Spanish)” según se quiera consultar la ayuda en inglés o en castellano.

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

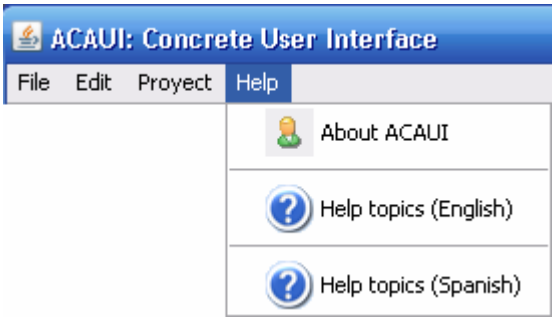


Figura 4.16 Menú para abrir la ayuda

A continuación se muestra una instantánea del manual de ayuda:

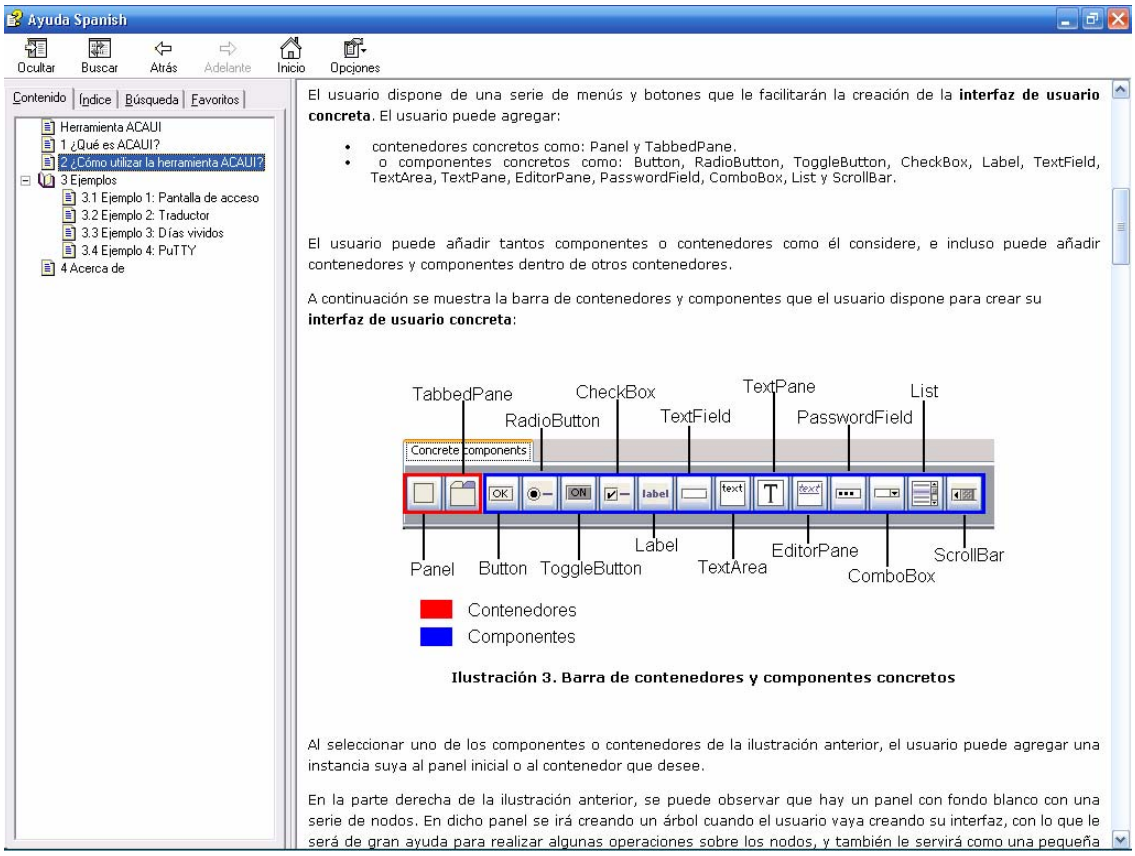


Figura 4.17 Manual de ayuda

4.3 EJEMPLOS

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

4.3.1 Ejemplo 1: Pantalla de acceso

Este primer ejemplo consiste en la creación de una pantalla de acceso típica, que se suele utilizan frecuentemente para que sólo puedan entrar los usuarios autorizados, y cada uno puede tener permisos diferentes.

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

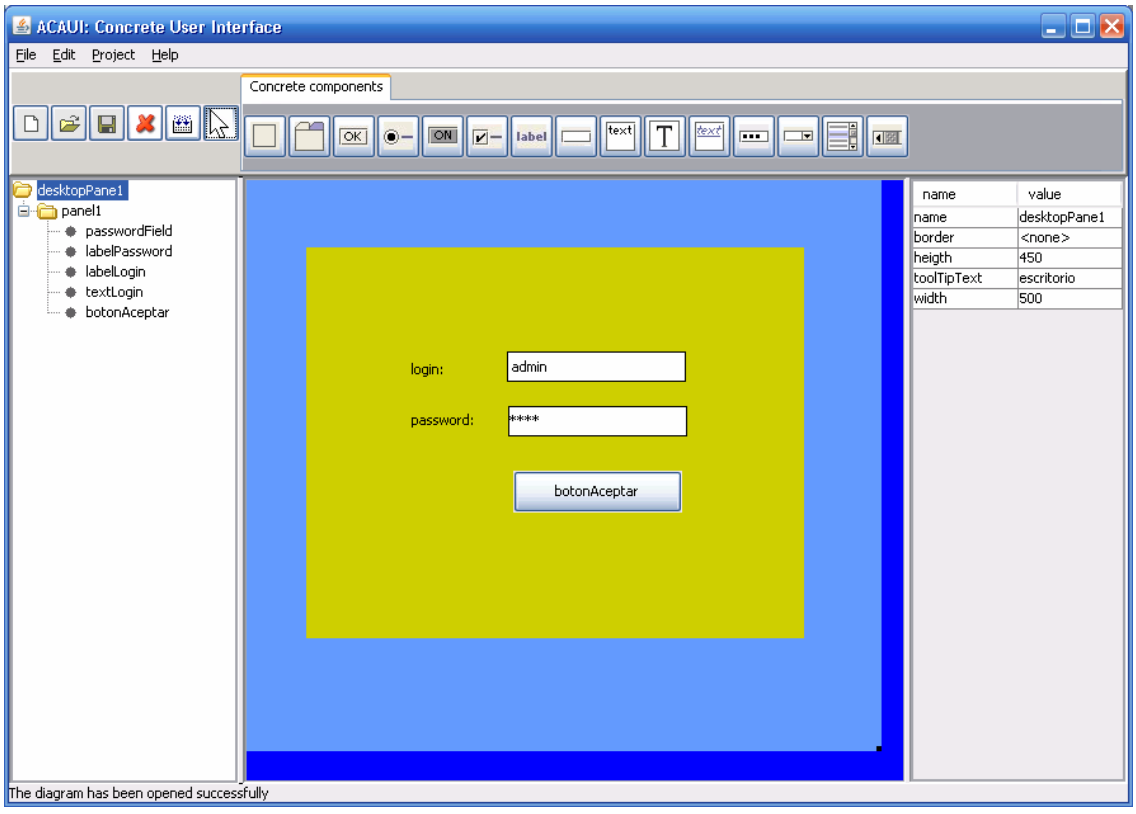


Figura 4.18 Interfaz de usuario concreta de Pantalla de acceso

Se compone de un contenedor principal de tipo *DesktopPane* denominado "desktopPane1", que dentro de tiene otro contenedor de tipo *Panel* denominado

"panel1"; éste último dentro tiene cinco componentes, de los cuales uno es de tipo *PasswordField* denominado "passwordField", dos son de tipo *Label* denominados "labelPassword" y "labelLogin", uno de tipo *TextField* denominado "textPassword", y otro componente de tipo *Button* denominado "botonAceptar".

Como se puede observar, el usuario ha personalizado la interfaz como ha querido, cambiando los colores que vienen por defecto por otros más llamativos, y ajustando el tamaño de cada contenedor o componente concreto al que considere más apropiado.

Durante la creación del mismo, en todo momento se ha indicado lo que el usuario estaba haciendo en una barra de estado situada en la parte inferior izquierda de la herramienta de la figura anterior.

Al guardar la interfaz de usuario concreta creada anteriormente, la información del diagrama se almacena en un fichero denominado "LoginPassword.cui", siguiendo el estándar de UsiXML, en concreto la parte referente a **Concrete User Interfaces (CUI)**. Como puede observarse en la siguiente figura, el fichero se compone de una serie de nodos, y cada nodo con sus atributos correspondientes.

A continuación se muestra el fichero "LoginPassword.cui" asociado a la interfaz de usuario concreta de la figura anterior:

```
<cuimodel>
- <DesktopPane name="desktopPane1" backgroundRed="102" backgroundGreen="153" backgroundBlue="255" border="<none>" foregroundRed="0"
  foregroundGreen="0" foregroundBlue="0" height="450" toolTipText="escritorio" width="500" x="-1" y="0">
  - <Panel name="panel1" backgroundRed="204" backgroundGreen="204" backgroundBlue="0" border="<none>" foregroundRed="0"
    foregroundGreen="0" foregroundBlue="0" height="308" toolTipText="panel1" width="392" x="47" y="53">
      <Button name="botonAceptar" backgroundRed="226" backgroundGreen="226" backgroundBlue="226" border="<none>" foregroundRed="0"
      foregroundGreen="0" foregroundBlue="0" height="33" text="botonAceptar" toolTipText="botonAceptar" width="108" x="174" y="176"/>
      <TextField name="textLogin" backgroundRed="255" backgroundGreen="255" backgroundBlue="255" border="Line" foregroundRed="0"
      foregroundGreen="0" foregroundBlue="0" height="24" text=" admin" toolTipText="textField1" width="141" x="158" y="82"/>
      <Label name="labelLogin" backgroundRed="226" backgroundGreen="226" backgroundBlue="226" border="<none>" foregroundRed="0"
      foregroundGreen="0" foregroundBlue="0" height="24" text="login:" toolTipText="label1" width="46" x="97" y="84"/>
      <Label name="labelPassword" backgroundRed="226" backgroundGreen="226" backgroundBlue="226" border="<none>" foregroundRed="0"
      foregroundGreen="0" foregroundBlue="0" height="36" text="password:" toolTipText="label2" width="58" x="98" y="120"/>
      <PasswordField name="passwordField" backgroundRed="255" backgroundGreen="255" backgroundBlue="255" border="Line" echoChar="*"
      foregroundRed="0" foregroundGreen="0" foregroundBlue="0" height="24" text="1234" toolTipText="passwordField" width="141" x="159"
      y="125"/>
    </Panel>
  </DesktopPane>
</cuimodel>
```

Figura 4.19 Código de especificación de Pantalla de acceso a nivel concreto

Como puede observarse en la figura anterior, el primer nodo es el nodo con la etiqueta *cuimodel*, que abarca todo el fichero. Dentro de dicho nodo hay otro nodo de

tipo *DesktopPane* denominado “desktopPane1”, que tiene atributos sobre su: nombre, color de fondo (descompuesto en sus tres colores primarios: rojo, verde y azul), color del primer plano (también descompuesto en sus tres colores primarios: rojo, verde y azul), altura, texto de consejo, anchura, posiciones “x” e “y” en la que se sitúa.

El nodo *Panel* denominado “panel1” tiene los mismos atributos (aunque con valores distintos en este caso) que el nodo de tipo *DesktopPane* descrito anteriormente. El nodo “panel1” contiene cinco nodos dentro de él: uno de tipo *Button*, uno de tipo *TextField*, otro de tipo *PasswordField* y otros dos de tipo *Label*. El *DesktopPane* tiene como atributos: *name*, *border*, *height*, *toolTipText* y *width*. El nodo de tipo *Panel* tiene los mismo atributos que el nodo *DesktopPane*. Los nodos de tipo *Label* y *TextField* además tienen el atributo *text*, que sirve para escribir un texto en el componente. Por último, el componente de tipo *PasswordField* es similar a los anteriores, pero además tiene el atributo *echoChar*, que sirve para especificar el carácter que se verá en vez del texto escrito por el usuario.

Una vez guardado el fichero anterior, el usuario ya puede realizar la conversión pulsando el botón  , para así obtener el fichero de **especificación de la interfaz a nivel abstracto**.

El fichero de **especificación a nivel abstracto** obtenido es el mostrado el la figura siguiente:

```
<auimodel>
 - <abstractContainer id="idaio0" name="desktopPane1">
   - <abstractContainer id="idaio1" name="panel1">
     - <abstractIndividualComponent id="idaio2" name="botonAceptar">
         <control id="idaio3" name="botonAceptar"/>
       </abstractIndividualComponent>
     - <abstractIndividualComponent id="idaio4" name="textLogin">
         <input id="idaio5" name="textLogin"/>
       </abstractIndividualComponent>
     - <abstractIndividualComponent id="idaio6" name="labelLogin">
         <output id="idaio7" name="labelLogin"/>
       </abstractIndividualComponent>
     - <abstractIndividualComponent id="idaio8" name="labelPassword">
         <output id="idaio9" name="labelPassword"/>
       </abstractIndividualComponent>
     - <abstractIndividualComponent id="idaio10" name="passwordField">
         <input id="idaio11" name="passwordField"/>
       </abstractIndividualComponent>
     </abstractContainer>
   </abstractContainer>
</auimodel>
```

Figura 4.20 Código de especificación de Pantalla de acceso a nivel abstracto

Este fichero ha sido creado siguiendo el estándar UsiXML, en concreto la parte referente a **Abstract User Interfaces (AUI).**

Como puede observarse, hay un nodo de tipo *auimodel* que engloba a todo el fichero. Dentro de él se puede observar un primer nodo de tipo *abstractContainer* con el identificador “idaio0” y el nombre “desktopPane1”. Este nodo es un contenedor de tipo abstracto que se corresponde con el contenedor concreto de tipo *DesktopPane* de la figura 4.19.

Dentro del nodo de identificador “idaio0” tenemos otro nodo también de tipo *abstractContainer* con el identificador “idaio1”, que se corresponde con el contenedor concreto de tipo *Panel* de la figura 4.19.

Dentro del nodo de identificador “idaio1” hay cinco *abstractIndividualComponent*, que se corresponden con los componentes concretos denominados “botonAceptar”, “textLogin”, “labelLogin”, “labelPassword” y

“passwordField” de la figura 4.19. Cada uno de ellos tiene unas facetas asociadas (en el apartado 3.2.1 de este documento puede verse la asociación realizada entre cada uno de los componentes concretos y las facetas AUI).

El componente abstracto de identificador “idaio2” representa al componente concreto de tipo *Button* de la figura 4.19. Tiene una faceta de control asociada, que indica que el componente puede enlazar con las funciones del sistema, como por ejemplo que acceda a la base de datos a comprobar si el login y el password son correctos.

El componente abstracto con identificador “idaio4”, se corresponde con el *TextField* del fichero de especificación concreta. Tiene una faceta de entrada, lo que quiere decir que el componente soporta una acción de entrada, como por ejemplo que el usuario escriba su “login”.

El componente abstracto con identificador “idaio6”, se corresponde con el primer *Label* del fichero de especificación concreta. Tiene una faceta de salida, lo que quiere decir que el componente muestra algo por pantalla al usuario, como por ejemplo el texto “login:” para hacerle una referencia de donde tiene que poner su login.

El siguiente componente, es decir, el componente abstracto con identificador “idaio8” se corresponde con el segundo *Label* del fichero de especificación concreta. Al igual que el componente “idaio6” tiene una faceta de salida, pero en este caso es para indicarle al usuario donde tiene que poner su contraseña.

El último componente abstracto tiene identificador “idaio10”, y se corresponde con el *PasswordField* del fichero de especificación concreta. Tiene una faceta de entrada, que sirve para que el usuario escriba su contraseña.

Como resultado de interpretar el fichero anterior, se obtiene la siguiente interfaz abstracta, donde se refleja todo lo explicado correspondiente al fichero de **especificación de usuario a nivel abstracto** asociado:

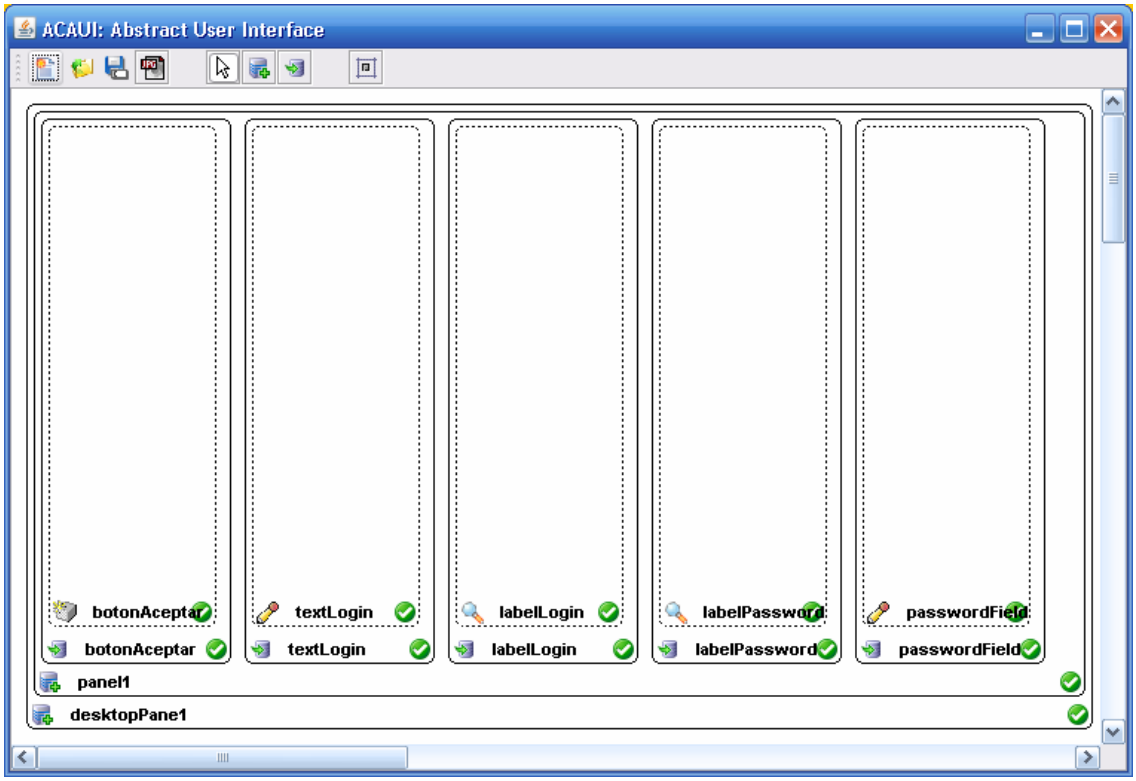


Figura 4.21 Interfaz de usuario abstracta de Pantalla de acceso

4.3.2 Ejemplo 2: Traductor

Este segundo ejemplo consiste en la creación de la interfaz de usuario de un traductor español-inglés e inglés-español. Este modelo de interfaz es frecuentemente utilizada en los traductores por la facilidad de manejarla.

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

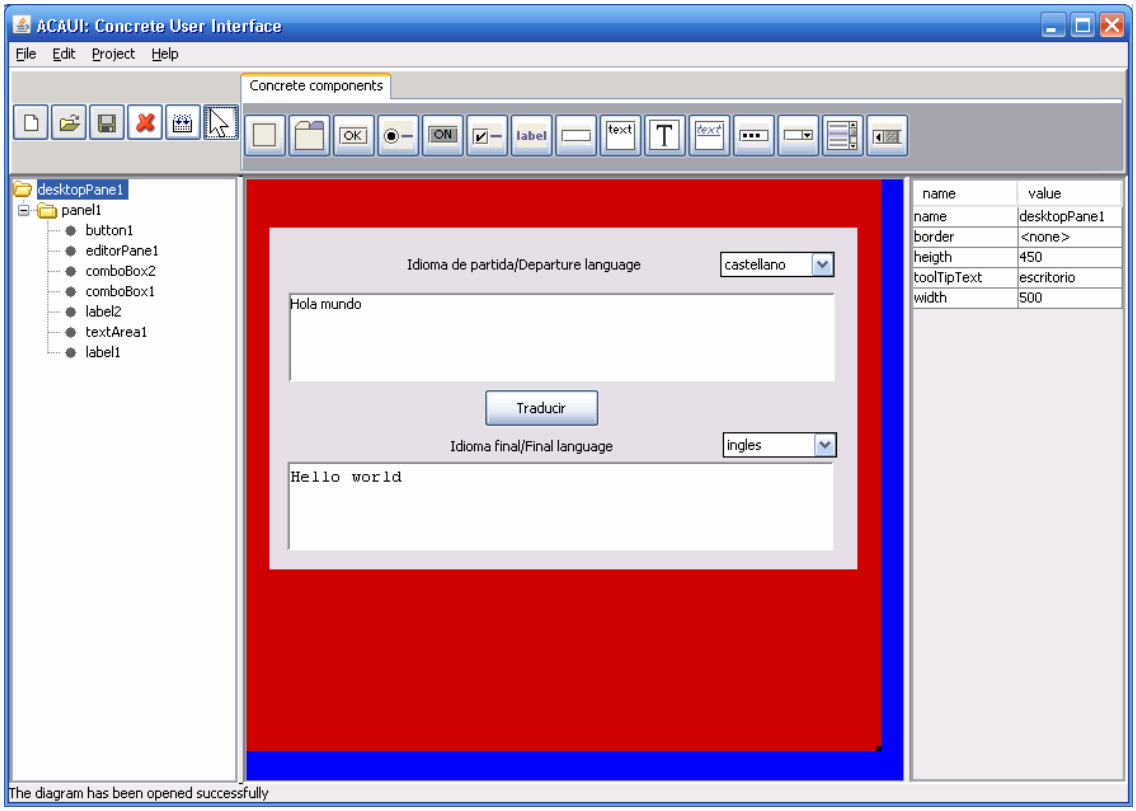


Figura 4.22 Interfaz de usuario concreta del traductor

Dicha interfaz se compone de un contenedor principal de tipo *DesktopPane* denominado “desktopPane1”, que contiene a un contenedor de tipo *Panel* denominado “panel1”; éste último contiene siete componentes, de los cuales uno es del tipo *Button* denominado “button1”, otro es de tipo *EditorPane* denominado “editorPane1”, dos son de tipo *ComboBox* denominados “comboBox1” y comboBox2”, dos son de tipo *Label* denominados “label1” y “label2” y uno de tipo *TextArea* denominado “textArea1”.

Para facilitar la elección del idioma de partida y del idioma final, la interfaz proporciona dos ComboBox, dando a elegir dos opciones: español o inglés. En el

EditorPane (donde pone “Hola mundo”) es donde hay que escribir la frase que se desea traducir, y en el *TextArea* (donde pone Hello world) es donde debería aparecer la frase traducida. El boton “Traducir” traduciría la frase del *EditorPane* de arriba, y la pondría en el *TextArea* de abajo. Las *Label* se utilizan para mostrar al usuario información sobre la interfaz.

La información del diagrama se ha guardado en un fichero denominado “Traductor.cui”, siguiendo el estándar de UsiXML, en concreto la parte referente a **Concrete User Interfaces (CUI)**. Como puede observarse en la siguiente figura, el fichero se compone de una serie de nodos, y cada nodo con sus atributos correspondientes, según han sido configurados en la interfaz anterior.

A continuación se muestra el fichero “Traductor.cui” asociado a la interfaz de usuario concreta de la figura anterior:

```
<cuimodel>
- <DesktopPane name="desktopPane1" backgroundRed="204" backgroundGreen="0" backgroundBlue="0" border="<none>" foregroundRed="0"
  foregroundGreen="0" foregroundBlue="0" height="450" toolTipText="escritorio" width="500" x="-1" y="0">
  - <Panel name="panel1" backgroundRed="224" backgroundGreen="223" backgroundBlue="227" border="<none>" foregroundRed="0"
    foregroundGreen="0" foregroundBlue="0" height="269" toolTipText="panel1" width="463" x="18" y="38">
      <Label name="label1" backgroundRed="226" backgroundGreen="226" backgroundBlue="226" border="<none>" foregroundRed="0"
      foregroundGreen="0" foregroundBlue="0" height="16" text="Idioma de partida/Departure language" toolTipText="label1" width="194" x="130"
      y="21"/>
      <TextArea name="textArea1" backgroundRed="255" backgroundGreen="255" backgroundBlue="255" border="LoweredBevel" editable="false"
      foregroundRed="0" foregroundGreen="0" foregroundBlue="0" height="70" text="Hello world" toolTipText="textArea1" width="430" x="14"
      y="184"/>
      <Label name="label2" backgroundRed="226" backgroundGreen="226" backgroundBlue="226" border="<none>" foregroundRed="0"
      foregroundGreen="0" foregroundBlue="0" height="19" text="Idioma final/Final language" toolTipText="label2" width="134" x="152" y="163"/>
      <ComboBox name="comboBox1" backgroundRed="255" backgroundGreen="255" backgroundBlue="255" border="Line" data="español,inglés"
      foregroundRed="0" foregroundGreen="0" foregroundBlue="0" height="20" toolTipText="comboBox1" width="90" x="355" y="19"/>
      <ComboBox name="comboBox2" backgroundRed="255" backgroundGreen="255" backgroundBlue="255" border="Line" data="inglés,español"
      foregroundRed="0" foregroundGreen="0" foregroundBlue="0" height="20" toolTipText="comboBox2" width="90" x="357" y="161"/>
      <EditorPane name="editorPane1" backgroundRed="255" backgroundGreen="255" backgroundBlue="255" border="LoweredBevel" editable="true"
      foregroundRed="0" foregroundGreen="0" foregroundBlue="0" height="70" text="Hola mundo" toolTipText="editorPane1" width="430" x="15"
      y="51"/>
      <Button name="button1" backgroundRed="236" backgroundGreen="233" backgroundBlue="216" border="<none>" foregroundRed="0"
      foregroundGreen="0" foregroundBlue="0" height="30" text="Traducir" toolTipText="button1" width="80" x="169" y="127"/>
    </Panel>
  </DesktopPane>
</cuimodel>
```

Figura 4.23 Código de especificación del traductor a nivel concreto

Se observa en la figura anterior, que el primer nodo es etiquetado con la etiqueta *cuimodel*, y abarca todo el fichero. Dentro de dicho nodo hay otro nodo que es un contenedor concreto de tipo *DesktopPane* denominado “desktopPane1”. Dentro de dicho contenedor hay otro contenedor concreto de tipo *Panel* denominado “Panel1”, con una serie de atributos. Este contenedor contiene siete componentes concretos dentro de él: dos de tipo *Label*, uno de tipo *TextArea*, dos de tipo *ComboBox*, uno de tipo

EditorPane, y otro de tipo *Button*. Cada uno de estos componentes concretos tiene una serie de atributos asociados.

Hay que indicar que el *EditorPane* permite escribir en él al usuario porque el campo “editable es true”, pero en el *TextArea* no le permite escribir, sino que sólo le permite leer, esto es debido a que el campo “editable es false”. El atributo editable puede ser modificado por el usuario.

Una vez guardado el fichero anterior, el usuario ya puede realizar la conversión pulsando el botón  , para así obtener el fichero de **especificación de la interfaz a nivel abstracto**.

El fichero de **especificación a nivel abstracto** obtenido es el mostrado el la figura siguiente:

```
<auimodel>
- <abstractContainer id="idaio0" name="desktopPane1">
  - <abstractContainer id="idaio1" name="panel1">
    - <abstractIndividualComponent id="idaio2" name="label1">
        <output id="idaio3" name="label1"/>
      </abstractIndividualComponent>
    - <abstractIndividualComponent id="idaio4" name="textArea1">
        <output id="idaio5" name="textArea1"/>
      </abstractIndividualComponent>
    - <abstractIndividualComponent id="idaio6" name="label2">
        <output id="idaio7" name="label2"/>
      </abstractIndividualComponent>
    - <abstractIndividualComponent id="idaio8" name="comboBox1">
        <control id="idaio9" name="comboBox1"/>
        <input id="idaio10" name="comboBox1"/>
        <output id="idaio11" name="comboBox1"/>
      </abstractIndividualComponent>
    - <abstractIndividualComponent id="idaio12" name="comboBox2">
        <control id="idaio13" name="comboBox2"/>
        <input id="idaio14" name="comboBox2"/>
        <output id="idaio15" name="comboBox2"/>
      </abstractIndividualComponent>
    - <abstractIndividualComponent id="idaio16" name="editorPane1">
        <input id="idaio17" name="editorPane1"/>
      </abstractIndividualComponent>
    - <abstractIndividualComponent id="idaio18" name="button1">
        <control id="idaio19" name="button1"/>
      </abstractIndividualComponent>
    </abstractContainer>
  </abstractContainer>
</auimodel>
```

Figura 4.24 Código de especificación del traductor a nivel abstracto

Este fichero ha sido creado siguiendo el estándar UsiXML, en concreto la parte referente a **Abstract User Interfaces (AUI).**

Como puede observarse, hay un nodo de tipo *auimodel* que engloba a todo el fichero. Dentro de él hay un primer nodo de tipo *abstractContainer* con el identificador "idaio0" y el nombre "desktopPane1". Este nodo es un contenedor de tipo abstracto que se corresponde con el contenedor concreto de tipo *DesktopPane* de la figura 4.23.

Dentro del contenedor abstracto de identificador "idaio0" tenemos otro contenedor abstracto también de tipo *abstractContainer* con el identificador "idaio1", que se corresponde con el contenedor concreto de tipo *Panel* de la figura 4.23.

Dentro del contenedor abstracto de identificador "idaio1" hay siete componentes abstractos, todos de tipo *abstractIndividualComponent*, que se corresponden con los siete componentes concretos denominados "label1", "textArea1", "label2", "comboBox1", "comboBox2", "editorPane1" y "button1" respectivamente, de la figura 4.23. Cada uno de ellos tiene unas facetas asociadas (en el apartado 3.2.1 de este documento puede verse la asociación realizada entre cada uno de los componentes concretos y las facetas AUI).

El componente abstracto de identificador "idaio2" tiene una faceta de salida asociada. Esta faceta de salida indica que el componente muestra algo por pantalla al usuario, en este caso lo que le muestra es: "Idioma de partida/Departure language".

El componente abstracto con identificador "idaio4", se corresponde con el *TextArea* del fichero de especificación concreta. Tiene una faceta de salida, para mostrar al usuario el resultado de la traducción.

El siguiente componente es el componente abstracto con identificador "idaio6" se corresponde con el segundo *Label* del fichero de especificación concreta. Al igual que el componente "idaio2" tiene una faceta de salida, y en este caso muestra: "Idioma final/Final language".

El siguiente componente es el componente abstracto con identificador "idaio8", que se corresponde con el primer *ComboBox* del fichero de especificación concreta. Este componente tiene tres facetas asociadas: una de control, otra de entrada y otra de salida. En este caso la faceta de control representa la posibilidad de elección del idioma de partida dentro del *ComboBox*. La faceta de entrada indica la posibilidad de que el usuario introduzca algún dato en el *ComboBox*. La faceta de salida indica que los ítems de dicho combo son mostrados por pantalla al usuario.

El siguiente componente tiene asociado el identificador “idaio12”, y su descripción es similar a la del componente anterior. La única diferencia, es que éste se utiliza para seleccionar el idioma final.

El siguiente componente tiene el identificador “idaio16”, y representa al componente concreto de tipo *EditorPane* de la figura 4.23. Éste componente abstracto tiene una faceta de entrada asociada, que se utiliza para que el usuario introduzca la frase a traducir.

Por último, el componente abstracto de identificador “idaio18” representa al componente concreto de tipo *Button* de la figura 4.23. Tiene una faceta de control asociada, que indica que el componente puede enlazar con las funciones del sistema, por ejemplo, en este caso su funcionalidad sería la de traducir una frase.

Como resultado de interpretar el fichero anterior, se obtiene la siguiente interfaz abstracta, donde se refleja todo lo explicado correspondiente al fichero de **especificación de usuario a nivel abstracto** asociado:

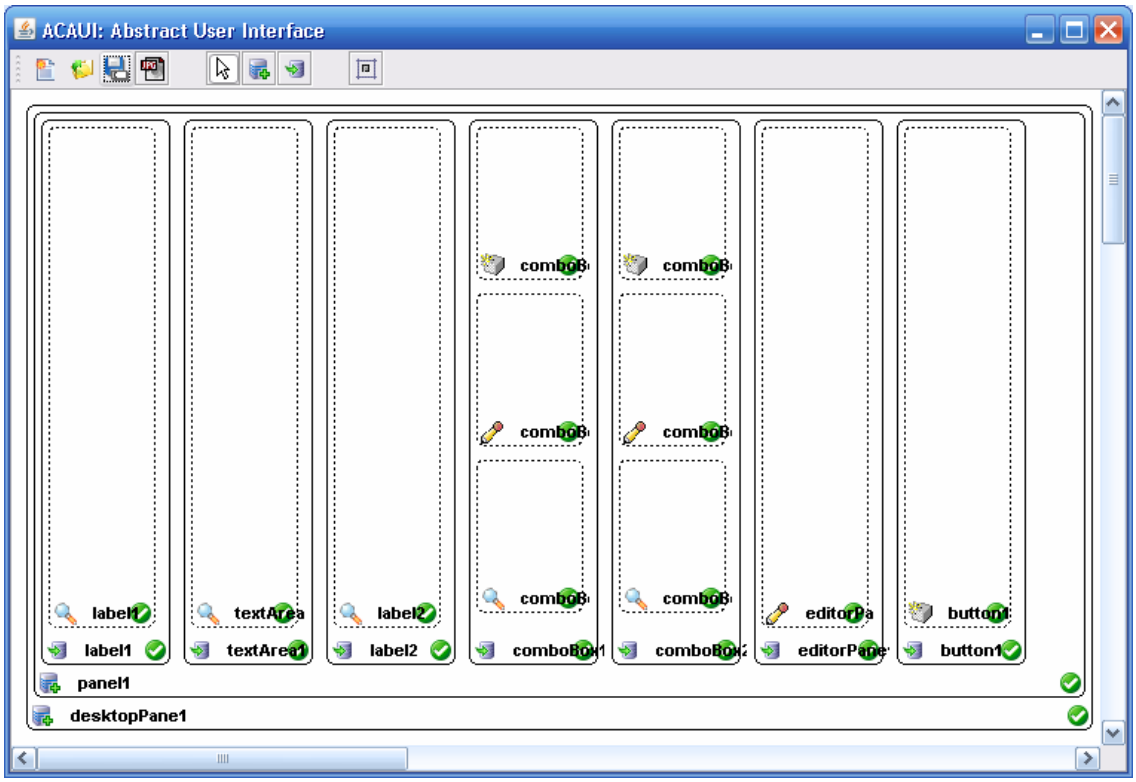


Figura 4.25 Interfaz de usuario abstracta del traductor

4.3.3 Ejemplo 3: Días vividos

Este tercer ejemplo consiste en la creación de la interfaz de usuario de un programa para calcular los días vividos de una persona en función de la fecha actual y del día de nacimiento.

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

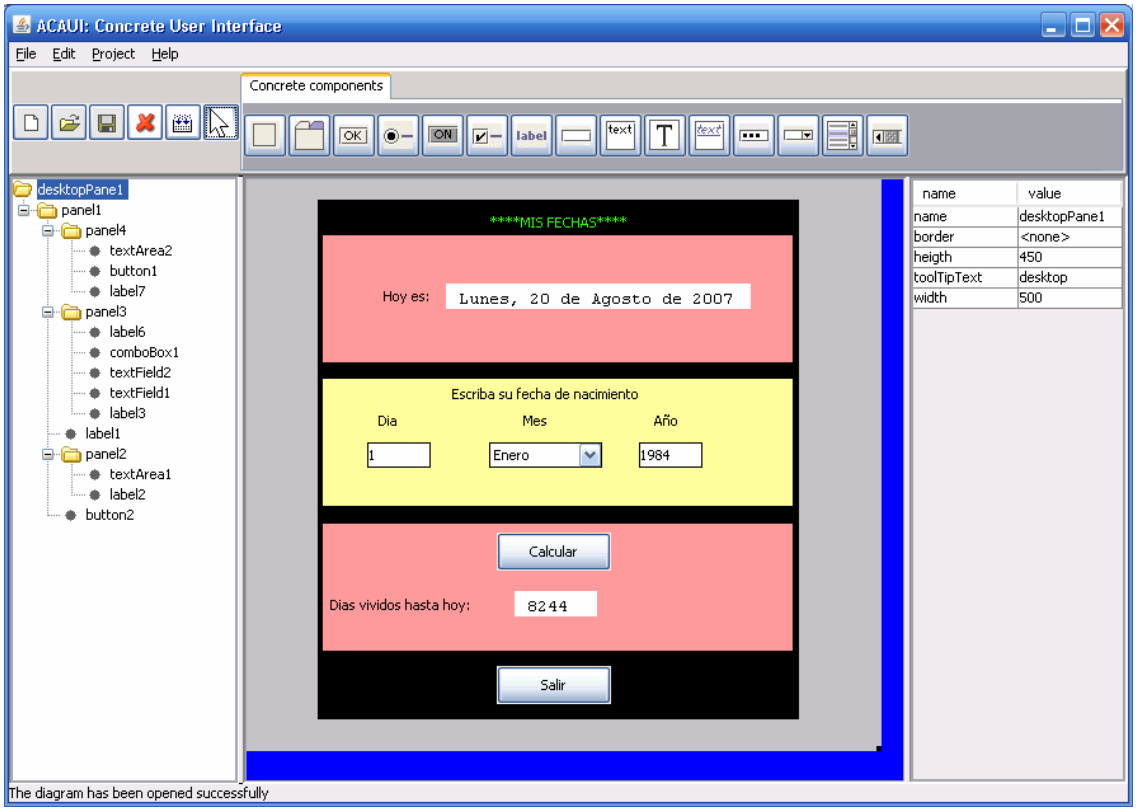


Figura 4.26 Interfaz de usuario concreta de Días vividos

Dicha interfaz se compone de un contenedor principal de tipo *DesktopPane* denominado “desktopPane1”, que contiene a un contenedor de tipo *Panel* denominado “panel1”; éste último contiene a su vez a dos componentes, uno de tipo *Label* denominado “label1”, y otro de tipo *Button* denominado “Button2”, y también contiene a tres contenedores, todos del tipo *Panel* denominados “panel2”, “panel3” y “panel4”.

Empezando de arriba abajo en la figura anterior, el panel denominado “panel2” contiene a dos componentes concretos: uno de tipo *label* denominado “label2”, y otro de tipo *textArea* denominado “textArea1”.

El siguiente panel es el denominado “panel3”, que contiene cinco componentes concretos: dos de tipo *label* denominados “label6” y “label3”, dos de tipo *textField* denominados “textField2” y “textField1”, y otro de tipo *ComboBox* denominado “comboBox1”.

El último panel se denomina “panel4” que dentro tiene a tres componentes: uno de tipo *Button* denominado “button1”, uno de tipo *textArea* denominado “textArea2” y otro de tipo *label* denominado “label7”.

Hay que resaltar tanto “textArea1” como “textArea2” tienen el campo “editable a false”, ya que se utilizan para mostrar datos al usuario, y no para que este los introduzca.

La idea es que el sistema ponga la fecha actual en el componente “textArea1” situado en el panel de arriba. Los componentes del segundo panel son para que el usuario ponga su fecha de nacimiento, y así cuando pulse el botón con el texto “Calcular”, el sistema calcule los días que dicha persona lleva vividos desde el día que nació, y los muestre en el componente “textArea2”.

La información del diagrama se ha guardado en un fichero denominado “DiasVividos.cui”, siguiendo el estándar de UsiXML, en concreto la parte referente a **Concrete User Interfaces (CUI)**. Como puede observarse en la siguiente figura, el fichero se compone de una serie de nodos, y cada nodo con sus atributos correspondientes, según han sido configurados en la interfaz anterior.

A continuación se muestra el fichero “DiasVividos.cui” asociado a la interfaz de usuario concreta de la figura anterior:

```
<cuimodel>
- <DesktopPane name="desktopPane1" backgroundRed="192" backgroundGreen="192" backgroundBlue="192" border="<none>" foregroundRed="0"
  foregroundGreen="0" foregroundBlue="0" height="450" toolTipText="desktop" width="500" x="-1" y="0">
  - <Panel name="panel1" backgroundRed="0" backgroundGreen="0" backgroundBlue="0" border="<none>" foregroundRed="0"
    foregroundGreen="0" foregroundBlue="0" height="409" toolTipText="panel1" width="379" x="56" y="16">
      <Button name="button2" backgroundRed="236" backgroundGreen="233" backgroundBlue="216" border="<none>" foregroundRed="0"
      foregroundGreen="0" foregroundBlue="0" height="30" text="Salir" toolTipText="button2" width="90" x="141" y="367"/>
    - <Panel name="panel2" backgroundRed="255" backgroundGreen="153" backgroundBlue="153" border="<none>" foregroundRed="0"
      foregroundGreen="0" foregroundBlue="0" height="100" toolTipText="panel2" width="370" x="4" y="28">
        <Label name="label2" backgroundRed="236" backgroundGreen="233" backgroundBlue="216" border="<none>" foregroundRed="0"
        foregroundGreen="0" foregroundBlue="0" height="25" text="Hoy es:" toolTipText="label2" width="56" x="48" y="36"/>
        <TextArea name="textArea1" backgroundRed="255" backgroundGreen="255" backgroundBlue="255" border="<none>" editable="false"
        foregroundRed="0" foregroundGreen="0" foregroundBlue="0" height="20" text=" Lunes, 20 de Agosto de 2007" toolTipText="textArea1"
        width="240" x="97" y="38"/>
      </Panel>
      <Label name="label1" backgroundRed="236" backgroundGreen="233" backgroundBlue="216" border="<none>" foregroundRed="51"
      foregroundGreen="255" foregroundBlue="0" height="24" text="*****MIS FECHAS*****" toolTipText="label1" width="111" x="136" y="6"/>
    - <Panel name="panel3" backgroundRed="255" backgroundGreen="255" backgroundBlue="153" border="<none>" foregroundRed="0"
      foregroundGreen="0" foregroundBlue="0" height="100" toolTipText="panel3" width="370" x="4" y="141">
        <Label name="label3" backgroundRed="236" backgroundGreen="233" backgroundBlue="216" border="<none>" foregroundRed="0"
        foregroundGreen="0" foregroundBlue="0" height="15" text="Dia Mes Año" toolTipText="label3" width="268" x="44" y="26"/>
        <TextField name="textField1" backgroundRed="255" backgroundGreen="255" backgroundBlue="255" border="Line" foregroundRed="0"
        foregroundGreen="0" foregroundBlue="0" height="20" text="1" toolTipText="textField1" width="50" x="35" y="50"/>
        <TextField name="textField2" backgroundRed="255" backgroundGreen="255" backgroundBlue="255" border="Line" foregroundRed="0"
        foregroundGreen="0" foregroundBlue="0" height="20" text="1984" toolTipText="textField2" width="50" x="249" y="50"/>
        <ComboBox name="comboBox1" backgroundRed="255" backgroundGreen="255" backgroundBlue="255" border="Line"
        data="Enero,Febrero,Marzo,Abril,Mayo,Junio,Julio,Agosto,Septiembre,Octubre,Noviembre,Diciembre" foregroundRed="0" foregroundGreen="0"
        foregroundBlue="0" height="20" toolTipText="comboBox1" width="65" x="141" y="50"/>
        <Label name="label6" backgroundRed="236" backgroundGreen="233" backgroundBlue="216" border="<none>" foregroundRed="0"
        foregroundGreen="0" foregroundBlue="0" height="18" text="Escriba su fecha de nacimiento" toolTipText="label6" width="169" x="102" y="3"/>
      </Panel>
    - <Panel name="panel4" backgroundRed="255" backgroundGreen="153" backgroundBlue="153" border="<none>" foregroundRed="0"
      foregroundGreen="0" foregroundBlue="0" height="100" toolTipText="panel4" width="370" x="4" y="255">
        <Label name="label7" backgroundRed="236" backgroundGreen="233" backgroundBlue="216" border="<none>" foregroundRed="0"
        foregroundGreen="0" foregroundBlue="0" height="18" text="Dias vividos hasta hoy:" toolTipText="label7" width="125" x="11" y="55"/>
        <Button name="button1" backgroundRed="236" backgroundGreen="233" backgroundBlue="216" border="<none>" foregroundRed="0"
        foregroundGreen="0" foregroundBlue="0" height="30" text="Calcular" toolTipText="button1" width="90" x="137" y="7"/>
        <TextArea name="textArea2" backgroundRed="255" backgroundGreen="255" backgroundBlue="255" border="<none>" editable="false"
        foregroundRed="0" foregroundGreen="0" foregroundBlue="0" height="20" text=" 8244" toolTipText="textArea2" width="65" x="137" y="53"/>
      </Panel>
    </Panel>
  </DesktopPane>
</cuimodel>
</cuimodel>
```

Figura 4.27 Código de especificación de Días vividos a nivel concreto

Se observa en la figura anterior, que el primer nodo es etiquetado con la etiqueta *cuimodel*, y abarca todo el fichero. Dentro de dicho nodo hay otro nodo que es un contenedor concreto de tipo *DesktopPane* denominado “desktopPane1”. Dentro de dicho contenedor hay otro contenedor concreto de tipo *Panel* denominado “panel1. Dicho contenedor denominado “panel1”contiene a su vez dos componentes concretos: uno de tipo *Label* y otro de tipo *Button*, con una serie de atributos asociados; además contiene a tres contenedores concretos de tipo *Panel* denominados “panel2”, “panel3” y “panel4” respectivamente.

El nodo denominado “panel2” contiene dos nodos asociados a componentes concretos: uno de tipo *Label* y otro de tipo *TextArea*. Cada uno de estos componentes concretos tiene una serie de atributos asociados, en función de cómo los haya configurado el usuario.

El nodo denominado “panel3” contiene a cinco nodos asociados a componentes concretos: dos de tipo *Label*, otros dos de tipo *TextField*, y otro de tipo *ComboBox*. Cada uno de estos componentes concretos tiene una serie de atributos asociados.

El último nodo denominado “panel4” contiene a tres componentes concretos: uno de tipo *Label*, uno de tipo *TextArea* y otro de tipo *Button*. Cada uno de estos componentes concretos tiene una serie de atributos asociados, procedentes de la configuración realizada.

Una vez guardado el fichero anterior, el usuario ya puede realizar la conversión pulsando el botón  , para así obtener el fichero de **especificación de la interfaz a nivel abstracto**.

El fichero de **especificación a nivel abstracto** obtenido es el mostrado el la siguiente figura:

```
<auimodel>
- <abstractContainer id="idaio0" name="desktopPane1">
  - <abstractContainer id="idaio1" name="panel1">
    - <abstractIndividualComponent id="idaio2" name="button2">
        <control id="idaio3" name="button2"/>
      </abstractIndividualComponent>
    - <abstractContainer id="idaio4" name="panel2">
      - <abstractIndividualComponent id="idaio5" name="label2">
          <output id="idaio6" name="label2"/>
        </abstractIndividualComponent>
      - <abstractIndividualComponent id="idaio7" name="textArea1">
          <output id="idaio8" name="textArea1"/>
        </abstractIndividualComponent>
      </abstractContainer>
    - <abstractIndividualComponent id="idaio9" name="label1">
        <output id="idaio10" name="label1"/>
      </abstractIndividualComponent>
    - <abstractContainer id="idaio11" name="panel3">
      - <abstractIndividualComponent id="idaio12" name="label3">
          <output id="idaio13" name="label3"/>
        </abstractIndividualComponent>
      - <abstractIndividualComponent id="idaio14" name="textField1">
          <input id="idaio15" name="textField1"/>
        </abstractIndividualComponent>
      - <abstractIndividualComponent id="idaio16" name="textField2">
          <input id="idaio17" name="textField2"/>
        </abstractIndividualComponent>
      - <abstractIndividualComponent id="idaio18" name="comboBox1">
          <control id="idaio19" name="comboBox1"/>
          <input id="idaio20" name="comboBox1"/>
          <output id="idaio21" name="comboBox1"/>
        </abstractIndividualComponent>
      - <abstractIndividualComponent id="idaio22" name="label6">
          <output id="idaio23" name="label6"/>
        </abstractIndividualComponent>
      </abstractContainer>
    - <abstractContainer id="idaio24" name="panel4">
      - <abstractIndividualComponent id="idaio25" name="label7">
          <output id="idaio26" name="label7"/>
        </abstractIndividualComponent>
      - <abstractIndividualComponent id="idaio27" name="button1">
          <control id="idaio28" name="button1"/>
        </abstractIndividualComponent>
      - <abstractIndividualComponent id="idaio29" name="textArea2">
          <output id="idaio30" name="textArea2"/>
        </abstractIndividualComponent>
      </abstractContainer>
    </abstractContainer>
  </abstractContainer>
</auimodel>
```

Figura 4.28 Código de especificación de Días vividos a nivel abstracto

Este fichero ha sido creado siguiendo el estándar UsiXML, en concreto la parte referente a **Abstract User Interfaces (AUI).**

Como puede observarse, hay un nodo de tipo *auimodel* que engloba a todo el fichero. Dentro de él hay un primer nodo de tipo *abstractContainer* con el identificador "idaio0" y el nombre "desktopPane1". Este nodo es un contenedor de tipo abstracto que se corresponde con el contenedor concreto de tipo *DesktopPane* de la figura anterior.

Dentro del contenedor abstracto de identificador "idaio0" tenemos otro contenedor abstracto también de tipo *abstractContainer* con el identificador "idaio1", que se corresponde con el contenedor concreto de tipo *Panel* denominado "panel1" de la figura anterior.

Dentro del contenedor abstracto de identificador "idaio1" hay dos componentes y tres contenedores abstractos. Dichos componentes abstractos son de tipo *abstractIndividualComponent*, que se corresponden con los componentes concretos denominados "label1" y "button2" de la figura anterior.

El componente abstracto de identificador "idaio2" representa al componente concreto de tipo *Button* denominado "button2" de la figura anterior. Tiene una faceta de control asociada que indica que el componente puede enlazar con las funciones del sistema, por ejemplo, en este caso su funcionalidad sería la de finalizar la ejecución del programa.

El componente abstracto de identificador "idaio9" tiene una faceta de salida asociada. Esta faceta de salida indica que el componente muestra algo por pantalla al usuario, ya que en la interfaz de usuario concreta muestra: "MIS FECHAS".

Y los contenedores abstractos son identificados con los identificadores: "idaio4", "idaio11" y "idaio24" respectivamente.

El contenedor abstracto de identificador "idaio4" contiene dos componentes abstractos de identificadores "idaio5" y "idaio7", que se corresponden con un *Label* y un *TextArea* respectivamente. El primero tiene una faceta de salida, ya que en la interfaz concreta mostraba por pantalla el mensaje "Hoy es:". El segundo componente representa al componente concreto de tipo *TextArea* de la figura anterior. Éste componente abstracto tiene una faceta salida asociada. Dicha faceta se utiliza para mostrarle datos al usuario, en concreto la fecha actual. Cada componente concreto tiene unas facetas asociadas (en el apartado 3.2.1 de este documento puede verse la asociación realizada entre cada uno de los componentes concretos y las facetas AUI).

El contenedor abstracto de identificador “idaio11” contiene cinco componentes abstractos. Los componentes abstractos “idaio12” y “idaio22” se corresponden con los dos de tipo *Label* que indican “Escriba su fecha de nacimiento” y “Día Mes Año” respectivamente. Ambos utilizan la faceta de salida para mostrar datos por pantalla al usuario. Los componentes abstractos “idaio14” y “idaio16” se corresponden con los dos de tipo *TextField* correspondientes con los componentes concretos “textField1” y “textField2” de la figura 4.27. Ambos utilizan la faceta de entrada, ya que aceptan datos introducidos por el usuario. El otro componente abstracto tiene el identificador “idaio18” y se corresponde con el *ComboBox* del fichero de especificación concreta. Este componente tiene tres facetas asociadas: una de control, otra de entrada y otra de salida. En este caso la faceta de control representa la elección del mes. La faceta de entrada permite al usuario introducir datos en el *ComboBox*. La faceta de salida indica que los ítems de dicho combo son mostrados por pantalla al usuario.

Por último, el contenedor de identificador “idaio24” contiene tres componentes abstractos. El primero tiene identificador “idaio25” y se corresponde con el componente concreto de tipo *Label* denominado “label7”. Tiene una faceta de salida asociada. Esta faceta de salida indica que el componente muestra algo por pantalla al usuario, en este caso muestra: “Días vividos hasta hoy:”. El siguiente componente abstracto tiene identificador “idaio29” y se corresponde con el componente concreto “textArea2”. Éste componente abstracto tiene una faceta salida asociada, ya que muestra se utiliza para mostrarle datos al usuario, en concreto los días que lleva vividos. Por último el componente de identificador “idaio27” se corresponde con el componente concreto “button1”. Tiene una faceta de control asociadas que indica que el componente puede enlazar con las funciones del sistema, en este caso su funcionalidad sería la de calcular los días vividos.

Como resultado de interpretar el fichero anterior, se obtiene la siguiente interfaz abstracta, donde se refleja todo lo explicado correspondiente al fichero de **especificación de usuario a nivel abstracto** asociado:

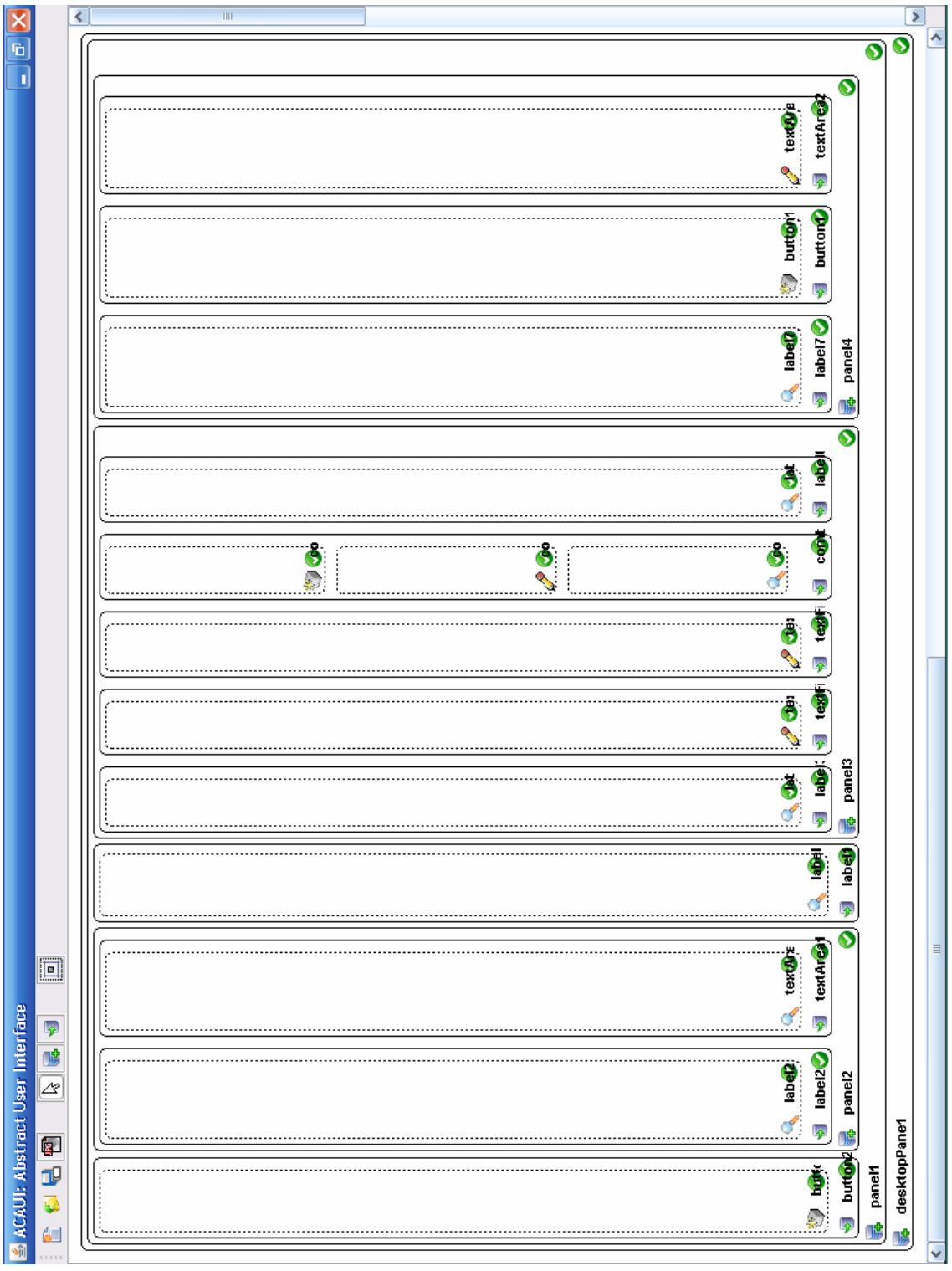


Figura 4.29 Interfaz de usuario abstracta de Días vividos

4.3.4 Ejemplo 4: PuTTY

Este cuarto y último ejemplo consiste en la creación de la interfaz de usuario semejante a la del conocido programa de uso profesional denominado “PuTTY”. Éste programa se utiliza para realizar conexiones con distintos host, mediante la correcta configuración de los distintos parámetros.

Para evitar la saturación de elementos en esta interfaz y así conseguir la correcta comprensión del ejemplo, las pestañas Keyboard, Selection, Terminal y Telnet no han sido completadas.

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

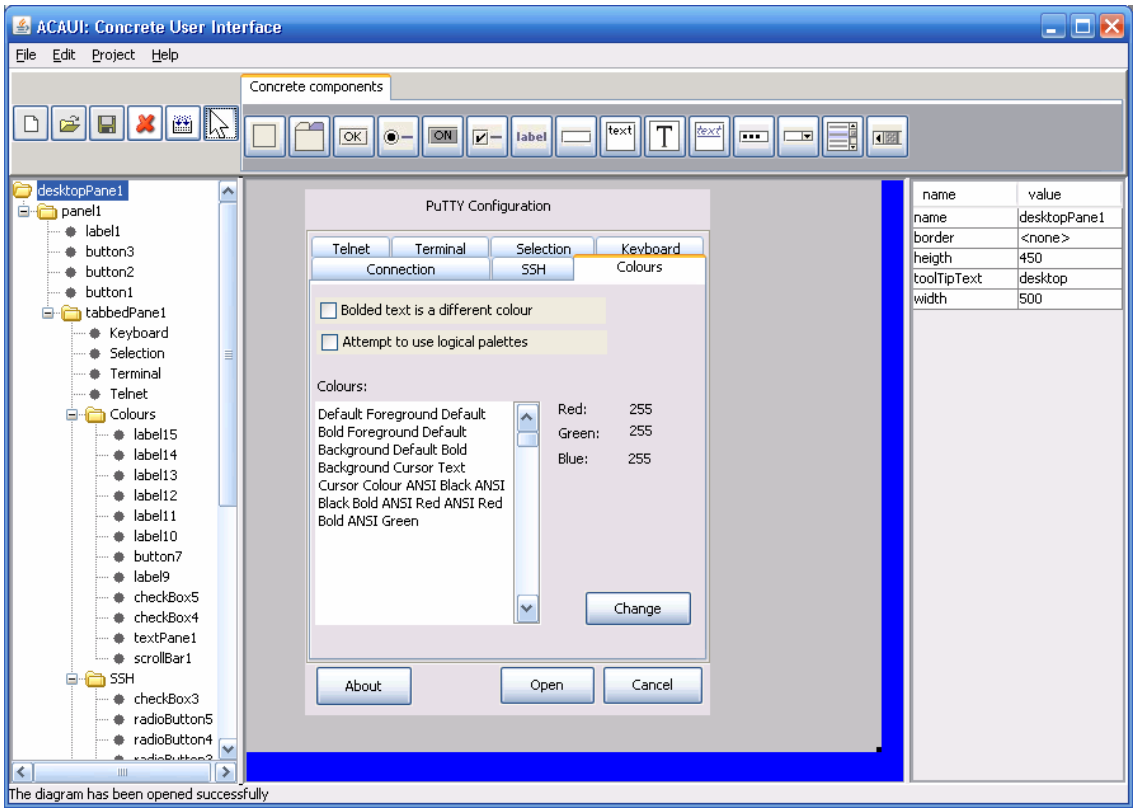


Figura 4.30 Interfaz de usuario concreta de PuTTY

Dicha interfaz se compone de un contenedor principal de tipo *DesktopPane* denominado “desktopPane1”, que contiene a un contenedor de tipo *Panel* denominado “panel1”; éste último contiene cuatro componentes: tres de tipo *Button* con el texto “About”, “Open” y “Cancel” respectivamente, y uno de tipo *label* denominado “label1”

con el texto “PuTTY Configuration”; y también contiene un contenedor de tipo *TabbedPane* denominado “tabbedPane1”.

Dentro del contenedor de tipo *TabbedPane*, hay siete contenedores de tipo *Pane* denominados: “Connection”, “SSH”, “Colours”, “Telnet”, “Terminal”, “Selection” y “Keyboard” respectivamente.

Dentro del contenedor “Connection” hay: cuatro componentes de tipo *Label* denominados “label2”, “label3”, “label4” y “label5”; tres componentes de tipo *textField* denominados “textField1”, “textField2” y “textField3”; dos componentes de tipo *RadioButton* denominados “radioButton1” y “radioButton2”; uno de tipo *TextArea* denominado “textArea1”; uno de tipo *CheckBox* denominado “checkBox1”; y tres de tipo *Button* denominados “button4”, “button5” y “button6”.

Dentro del contenedor “SSH” hay: tres componentes de tipo *Label* denominados “label6”, “label7” y “label8”; dos de tipo *TextField* denominados “textField4” y “textField5”; tres de tipo *RadioButton* denominados “radioButton3”, “radioButton4” y “radioButton5”; y dos de tipo *CheckBox* denominados “checkBox2” y “checkBox3”.

Dentro del contenedor “Colours” hay: un componente de tipo *ScrollBar* denominado “scrollBar1”; uno de tipo *TextPane* denominado “textPane1”; dos de tipo *CheckBox* denominados “checkBox4” y “checkBox5”; siete de tipo *Label* denominados “label9”, “label10”, “label11”, “label12”, “label13”, “label14” y “label15”; y uno de tipo *Button* denominado “button7”.

Los otros cuatro contenedores (“Telnet”, “Terminal”, “Selection” y “Keyboard”) no han sido completados para no saturar mucho el ejemplo.

Hay que destacar que tanto el “textArea1” como el “textPane1” tienen el campo “editable a false”, ya que ambos se utilizan para mostrar datos al usuario, y en ellos el usuario no puede escribir.

La idea es que el usuario pueda introducir los parámetros de configuración de la conexión mediante los campos de texto (*TextField*) adecuados, los *RadioButton* y los *CheckBox*. Una vez configurado todo correctamente, se establecería una conexión con el host destino.

La información del diagrama se ha guardado en un fichero denominado “PuttyTerminal.cui”, siguiendo el estándar de UsiXML, en concreto la parte referente a **Concrete User Interfaces (CUI)**. Como puede observarse en la siguiente figura, el

fichero se compone de una serie de nodos, y cada nodo con sus atributos correspondientes, según han sido configurados en la interfaz anterior.

A continuación se muestra el fichero “PuttyTerminal.cui” asociado a la interfaz de usuario concreta de la figura anterior:

```
<cuimodel>
 - <DesktopPane name="desktopPane1" backgroundRed="192" backgroundGreen="192" backgroundBlue="192" border="<none>" foregroundRed="0"
   foregroundGreen="0" foregroundBlue="0" height="450" toolTipText="desktop" width="500" x="-1" y="0">
    - <Panel name="panel1" backgroundRed="224" backgroundGreen="223" backgroundBlue="227" border="<none>" foregroundRed="0"
      foregroundGreen="0" foregroundBlue="0" height="414" toolTipText="panel1" width="319" x="46" y="7">
       - <TabbedPane name="tabbedPane1" backgroundRed="236" backgroundGreen="233" backgroundBlue="216" border="Etched"
         foregroundRed="0" foregroundGreen="0" foregroundBlue="0" height="342" toolTipText="tabbedPane1" width="319" x="0" y="32">
          - <Panel name="Connection" backgroundRed="224" backgroundGreen="223" backgroundBlue="227" border="<none>" foregroundRed="0"
            foregroundGreen="0" foregroundBlue="0" height="296" toolTipText="panel2" width="310" x="4" y="41">
               <Label name="label2" backgroundRed="236" backgroundGreen="233" backgroundBlue="216" border="<none>" foregroundRed="0"
               foregroundGreen="0" foregroundBlue="0" height="23" text="Host Name" toolTipText="label2" width="95" x="12" y="24"/>
               <TextField name="textField1" backgroundRed="255" backgroundGreen="255" backgroundBlue="255" border="<none>"
               foregroundRed="0" foregroundGreen="0" foregroundBlue="0" height="20" text="" toolTipText="textField1" width="212" x="10" y="44"/>
               <TextField name="textField2" backgroundRed="255" backgroundGreen="255" backgroundBlue="255" border="<none>"
               foregroundRed="0" foregroundGreen="0" foregroundBlue="0" height="21" text="" toolTipText="textField2" width="64" x="233" y="44"/>
               <Label name="label3" backgroundRed="236" backgroundGreen="233" backgroundBlue="216" border="<none>" foregroundRed="0"
               foregroundGreen="0" foregroundBlue="0" height="17" text="Port" toolTipText="label3" width="44" x="236" y="27"/>
               <Label name="label4" backgroundRed="236" backgroundGreen="233" backgroundBlue="216" border="<none>" foregroundRed="0"
               foregroundGreen="0" foregroundBlue="0" height="21" text="Protocol:" toolTipText="label4" width="64" x="107" y="77"/>
               <RadioButton name="radioButton1" backgroundRed="236" backgroundGreen="233" backgroundBlue="216" border="<none>"
               foregroundRed="0" foregroundGreen="0" foregroundBlue="0" height="29" text="Telnet" toolTipText="radioButton1" width="79" x="159"
               y="72"/>
               <RadioButton name="radioButton2" backgroundRed="236" backgroundGreen="233" backgroundBlue="216" border="<none>"
               foregroundRed="0" foregroundGreen="0" foregroundBlue="0" height="30" text="SSH" toolTipText="radioButton2" width="64" x="241"
               y="72"/>
               <TextField name="textField3" backgroundRed="255" backgroundGreen="255" backgroundBlue="255" border="<none>"
               foregroundRed="0" foregroundGreen="0" foregroundBlue="0" height="22" text="" toolTipText="textField3" width="213" x="6" y="125"/>
               <Label name="label5" backgroundRed="236" backgroundGreen="233" backgroundBlue="216" border="<none>" foregroundRed="0"
               foregroundGreen="0" foregroundBlue="0" height="16" text="Stored Sessions" toolTipText="label5" width="165" x="7" y="107"/>
               <TextArea name="textArea1" backgroundRed="255" backgroundGreen="255" backgroundBlue="255" border="<none>" editable="false"
               foregroundRed="0" foregroundGreen="0" foregroundBlue="0" height="115" text="Default Settings" toolTipText="textArea1" width="213"
               x="6" y="152"/>
               <CheckBox name="checkBox1" backgroundRed="236" backgroundGreen="233" backgroundBlue="216" border="<none>"
               foregroundRed="0" foregroundGreen="0" foregroundBlue="0" height="28" text="Close Window on Exit" toolTipText="checkBox1"
               width="178" x="5" y="269"/>
               <Button name="button4" backgroundRed="236" backgroundGreen="233" backgroundBlue="216" border="<none>" foregroundRed="0"
               foregroundGreen="0" foregroundBlue="0" height="28" text="Load" toolTipText="button4" width="65" x="231" y="156"/>
               <Button name="button5" backgroundRed="236" backgroundGreen="233" backgroundBlue="216" border="<none>" foregroundRed="0"
               foregroundGreen="0" foregroundBlue="0" height="28" text="Save" toolTipText="button5" width="65" x="231" y="190"/>
               <Button name="button6" backgroundRed="236" backgroundGreen="233" backgroundBlue="216" border="<none>" foregroundRed="0"
               foregroundGreen="0" foregroundBlue="0" height="28" text="Delete" toolTipText="button6" width="65" x="231" y="224"/>
            </Panel>
          - <Panel name="SSH" backgroundRed="224" backgroundGreen="223" backgroundBlue="227" border="<none>" foregroundRed="0"
            foregroundGreen="0" foregroundBlue="0" height="296" toolTipText="panel3" width="310" x="4" y="41">
               <Label name="label6" backgroundRed="236" backgroundGreen="233" backgroundBlue="216" border="<none>" foregroundRed="0"
               foregroundGreen="0" foregroundBlue="0" height="26" text="Terminal-type string" toolTipText="label6" width="104" x="7" y="14"/>
               <CheckBox name="checkBox2" backgroundRed="236" backgroundGreen="233" backgroundBlue="216" border="<none>"
               foregroundRed="0" foregroundGreen="0" foregroundBlue="0" height="19" text="Don't allocate a pseudo-terminal" toolTipText="checkBox2"
               width="182" x="7" y="62"/>
               <Label name="label7" backgroundRed="236" backgroundGreen="233" backgroundBlue="216" border="<none>" foregroundRed="0"
               foregroundGreen="0" foregroundBlue="0" height="20" text="Auto-login username" toolTipText="label7" width="108" x="8" y="110"/>
               <TextField name="textField4" backgroundRed="255" backgroundGreen="255" backgroundBlue="255" border="<none>"
               foregroundRed="0" foregroundGreen="0" foregroundBlue="0" height="22" text="xterm" toolTipText="textField4" width="155" x="145"
               y="17"/>
               <TextField name="textField5" backgroundRed="255" backgroundGreen="255" backgroundBlue="255" border="<none>"
               foregroundRed="0" foregroundGreen="0" foregroundBlue="0" height="22" text="" toolTipText="textField5" width="155" x="146" y="106"/>
               <Label name="label8" backgroundRed="236" backgroundGreen="233" backgroundBlue="216" border="<none>" foregroundRed="0"
               foregroundGreen="0" foregroundBlue="0" height="20" text="Cipher:" toolTipText="label8" width="72" x="8" y="147"/>
               <RadioButton name="radioButton3" backgroundRed="236" backgroundGreen="233" backgroundBlue="216" border="<none>"
               foregroundRed="0" foregroundGreen="0" foregroundBlue="0" height="26" text="3DES" toolTipText="radioButton3" width="67" x="78"
               y="143"/>
               <RadioButton name="radioButton4" backgroundRed="236" backgroundGreen="233" backgroundBlue="216" border="<none>"
               foregroundRed="0" foregroundGreen="0" foregroundBlue="0" height="28" text="Blowfish" toolTipText="radioButton4" width="74" x="152"
               y="142"/>
               <RadioButton name="radioButton5" backgroundRed="236" backgroundGreen="233" backgroundBlue="216" border="<none>"
               foregroundRed="0" foregroundGreen="0" foregroundBlue="0" height="27" text="DES" toolTipText="radioButton5" width="64" x="235"
               y="142"/>
               <CheckBox name="checkBox3" backgroundRed="236" backgroundGreen="233" backgroundBlue="216" border="<none>"
               foregroundRed="0" foregroundGreen="0" foregroundBlue="0" height="21" text="Attempt TIS authentication" toolTipText="checkBox3"
               width="164" x="6" y="185"/>
            </Panel>
```

```
- <Panel name="Colours" backgroundRed="224" backgroundGreen="223" backgroundBlue="227" border="<none>" foregroundRed="0"
  foregroundGreen="0" foregroundBlue="0" height="296" toolTipText="panel4" width="310" x="4" y="41">
    <ScrollBar name="scrollBar1" backgroundRed="212" backgroundGreen="208" backgroundBlue="200" border="Etched"
    foregroundRed="236" foregroundGreen="233" foregroundBlue="216" height="176" orientation="HORIZONTAL_SPLIT"
    toolTipText="scrollBar1" width="22" x="159" y="92"/>
    <TextPane name="textPane1" backgroundRed="255" backgroundGreen="255" backgroundBlue="255" border="<none>" editable="false"
    foregroundRed="0" foregroundGreen="0" foregroundBlue="0" height="176" text="Default Foreground Default Bold Foreground Default
    Background Default Bold Background Cursor Text Cursor Colour ANSI Black ANSI Black Bold ANSI Red ANSI Red Bold ANSI Green"
    toolTipText="textPane1" width="156" x="3" y="92"/>
    <CheckBox name="checkBox4" backgroundRed="236" backgroundGreen="233" backgroundBlue="216" border="<none>"
    foregroundRed="0" foregroundGreen="0" foregroundBlue="0" height="21" text="Bolded text is a different colour" toolTipText="checkBox4"
    width="203" x="3" y="9"/>
    <CheckBox name="checkBox5" backgroundRed="236" backgroundGreen="233" backgroundBlue="216" border="<none>"
    foregroundRed="0" foregroundGreen="0" foregroundBlue="0" height="19" text="Attempt to use logical palettes" toolTipText="checkBox5"
    width="203" x="3" y="36"/>
    <Label name="label9" backgroundRed="236" backgroundGreen="233" backgroundBlue="216" border="<none>" foregroundRed="0"
    foregroundGreen="0" foregroundBlue="0" height="18" text="Colours:" toolTipText="label9" width="71" x="5" y="71"/>
    <Button name="button7" backgroundRed="236" backgroundGreen="233" backgroundBlue="216" border="<none>" foregroundRed="0"
    foregroundGreen="0" foregroundBlue="0" height="28" text="Change" toolTipText="button7" width="77" x="215" y="241"/>
    <Label name="label10" backgroundRed="236" backgroundGreen="233" backgroundBlue="216" border="<none>" foregroundRed="0"
    foregroundGreen="0" foregroundBlue="0" height="19" text="Red:" toolTipText="label10" width="40" x="195" y="89"/>
    <Label name="label11" backgroundRed="236" backgroundGreen="233" backgroundBlue="216" border="<none>" foregroundRed="0"
    foregroundGreen="0" foregroundBlue="0" height="21" text="Green:" toolTipText="label11" width="49" x="195" y="107"/>
    <Label name="label12" backgroundRed="236" backgroundGreen="233" backgroundBlue="216" border="<none>" foregroundRed="0"
    foregroundGreen="0" foregroundBlue="0" height="26" text="Blue:" toolTipText="label12" width="50" x="195" y="124"/>
    <Label name="label13" backgroundRed="236" backgroundGreen="233" backgroundBlue="216" border="<none>" foregroundRed="0"
    foregroundGreen="0" foregroundBlue="0" height="21" text="255" toolTipText="label13" width="50" x="251" y="88"/>
    <Label name="label14" backgroundRed="236" backgroundGreen="233" backgroundBlue="216" border="<none>" foregroundRed="0"
    foregroundGreen="0" foregroundBlue="0" height="20" text="255" toolTipText="label14" width="45" x="251" y="105"/>
    <Label name="label15" backgroundRed="236" backgroundGreen="233" backgroundBlue="216" border="<none>" foregroundRed="0"
    foregroundGreen="0" foregroundBlue="0" height="20" text="255" toolTipText="label15" width="42" x="250" y="127"/>
  </Panel>
  <Panel name="Telnet" backgroundRed="224" backgroundGreen="223" backgroundBlue="227" border="<none>" foregroundRed="0"
  foregroundGreen="0" foregroundBlue="0" height="296" toolTipText="panel5" width="310" x="4" y="41"/>
  <Panel name="Terminal" backgroundRed="224" backgroundGreen="223" backgroundBlue="227" border="<none>" foregroundRed="0"
  foregroundGreen="0" foregroundBlue="0" height="296" toolTipText="panel6" width="310" x="4" y="41"/>
  <Panel name="Selection" backgroundRed="224" backgroundGreen="223" backgroundBlue="227" border="<none>" foregroundRed="0"
  foregroundGreen="0" foregroundBlue="0" height="296" toolTipText="panel7" width="310" x="4" y="41"/>
  <Panel name="Keyboard" backgroundRed="224" backgroundGreen="223" backgroundBlue="227" border="<none>" foregroundRed="0"
  foregroundGreen="0" foregroundBlue="0" height="296" toolTipText="panel8" width="310" x="4" y="41"/>
 </TabbedPane>
 <Button name="button1" backgroundRed="236" backgroundGreen="233" backgroundBlue="216" border="<none>" foregroundRed="0"
 foregroundGreen="0" foregroundBlue="0" height="32" text="About" toolTipText="button1" width="77" x="8" y="375"/>
 <Button name="button2" backgroundRed="236" backgroundGreen="233" backgroundBlue="216" border="<none>" foregroundRed="0"
 foregroundGreen="0" foregroundBlue="0" height="31" text="Open" toolTipText="button2" width="76" x="153" y="375"/>
 <Button name="button3" backgroundRed="236" backgroundGreen="233" backgroundBlue="216" border="<none>" foregroundRed="0"
 foregroundGreen="0" foregroundBlue="0" height="31" text="Cancel" toolTipText="button3" width="80" x="234" y="375"/>
 <Label name="label1" backgroundRed="236" backgroundGreen="233" backgroundBlue="216" border="<none>" foregroundRed="0"
 foregroundGreen="0" foregroundBlue="0" height="24" text="PuTTY Configuration" toolTipText="label1" width="134" x="96" y="1"/>
 </Panel>
 </DesktopPane>
</cuimodel>
```

Figura 4.31 Código de especificación de PuTTY a nivel concreto

Como puede observarse en la figura anterior se refleja todo lo descrito anteriormente referente a la figura 4.20, aunque en este caso mediante un fichero en el que cada uno de los distintos componentes o contenedores se representan con un nodo y con un conjunto de atributos asociados.

Una vez guardado el fichero anterior, el usuario ya puede realizar la conversión pulsando el botón  , para así obtener el fichero de **especificación de la interfaz a nivel abstracto**.

El fichero de **especificación a nivel abstracto** obtenido es el mostrado el la siguiente figura:

```
<auimodel>
 − <abstractContainer id="idaio0" name="desktopPane1">
   − <abstractContainer id="idaio1" name="panel1">
     − <abstractContainer id="idaio2" name="tabbedPane1">
       − <abstractContainer id="idaio3" name="Connection">
         − <abstractIndividualComponent id="idaio4" name="label2">
             <output id="idaio5" name="label2"/>
           </abstractIndividualComponent>
         − <abstractIndividualComponent id="idaio6" name="textField1">
             <input id="idaio7" name="textField1"/>
           </abstractIndividualComponent>
         − <abstractIndividualComponent id="idaio8" name="textField2">
             <input id="idaio9" name="textField2"/>
           </abstractIndividualComponent>
         − <abstractIndividualComponent id="idaio10" name="label3">
             <output id="idaio11" name="label3"/>
           </abstractIndividualComponent>
         − <abstractIndividualComponent id="idaio12" name="label4">
             <output id="idaio13" name="label4"/>
           </abstractIndividualComponent>
         − <abstractIndividualComponent id="idaio14" name="radioButton1">
             <input id="idaio15" name="radioButton1"/>
           </abstractIndividualComponent>
         − <abstractIndividualComponent id="idaio16" name="radioButton2">
             <input id="idaio17" name="radioButton2"/>
           </abstractIndividualComponent>
         − <abstractIndividualComponent id="idaio18" name="textField3">
             <input id="idaio19" name="textField3"/>
           </abstractIndividualComponent>
         − <abstractIndividualComponent id="idaio20" name="label5">
             <output id="idaio21" name="label5"/>
           </abstractIndividualComponent>
         − <abstractIndividualComponent id="idaio22" name="textArea1">
             <output id="idaio23" name="textArea1"/>
           </abstractIndividualComponent>
         − <abstractIndividualComponent id="idaio24" name="checkBox1">
             <input id="idaio25" name="checkBox1"/>
           </abstractIndividualComponent>
         − <abstractIndividualComponent id="idaio26" name="button4">
             <control id="idaio27" name="button4"/>
           </abstractIndividualComponent>
         − <abstractIndividualComponent id="idaio28" name="button5">
             <control id="idaio29" name="button5"/>
           </abstractIndividualComponent>
         − <abstractIndividualComponent id="idaio30" name="button6">
             <control id="idaio31" name="button6"/>
           </abstractIndividualComponent>
         </abstractContainer>
       − <abstractContainer id="idaio32" name="SSH">
         − <abstractIndividualComponent id="idaio33" name="label6">
             <output id="idaio34" name="label6"/>
           </abstractIndividualComponent>
         − <abstractIndividualComponent id="idaio35" name="checkBox2">
             <input id="idaio36" name="checkBox2"/>
           </abstractIndividualComponent>
         − <abstractIndividualComponent id="idaio37" name="label7">
             <output id="idaio38" name="label7"/>
           </abstractIndividualComponent>
         − <abstractIndividualComponent id="idaio39" name="textField4">
             <input id="idaio40" name="textField4"/>
           </abstractIndividualComponent>
         − <abstractIndividualComponent id="idaio41" name="textField5">
             <input id="idaio42" name="textField5"/>
           </abstractIndividualComponent>
         − <abstractIndividualComponent id="idaio43" name="label8">
             <output id="idaio44" name="label8"/>
           </abstractIndividualComponent>
         − <abstractIndividualComponent id="idaio45" name="radioButton3">
             <input id="idaio46" name="radioButton3"/>
           </abstractIndividualComponent>
         − <abstractIndividualComponent id="idaio47" name="radioButton4">
             <input id="idaio48" name="radioButton4"/>
           </abstractIndividualComponent>
         − <abstractIndividualComponent id="idaio49" name="radioButton5">
             <input id="idaio50" name="radioButton5"/>
           </abstractIndividualComponent>
         − <abstractIndividualComponent id="idaio51" name="checkBox3">
             <input id="idaio52" name="checkBox3"/>
           </abstractIndividualComponent>
         </abstractContainer>
```

```
- <abstractContainer id="idaio53" name="Colours">
  - <abstractIndividualComponent id="idaio54" name="scrollBar1">
      <navigation id="idaio55" name="scrollBar1"/>
    </abstractIndividualComponent>
  - <abstractIndividualComponent id="idaio56" name="textPane1">
      <output id="idaio57" name="textPane1"/>
    </abstractIndividualComponent>
  - <abstractIndividualComponent id="idaio58" name="checkBox4">
      <input id="idaio59" name="checkBox4"/>
    </abstractIndividualComponent>
  - <abstractIndividualComponent id="idaio60" name="checkBox5">
      <input id="idaio61" name="checkBox5"/>
    </abstractIndividualComponent>
  - <abstractIndividualComponent id="idaio62" name="label9">
      <output id="idaio63" name="label9"/>
    </abstractIndividualComponent>
  - <abstractIndividualComponent id="idaio64" name="button7">
      <control id="idaio65" name="button7"/>
    </abstractIndividualComponent>
  - <abstractIndividualComponent id="idaio66" name="label10">
      <output id="idaio67" name="label10"/>
    </abstractIndividualComponent>
  - <abstractIndividualComponent id="idaio68" name="label11">
      <output id="idaio69" name="label11"/>
    </abstractIndividualComponent>
  - <abstractIndividualComponent id="idaio70" name="label12">
      <output id="idaio71" name="label12"/>
    </abstractIndividualComponent>
  - <abstractIndividualComponent id="idaio72" name="label13">
      <output id="idaio73" name="label13"/>
    </abstractIndividualComponent>
  - <abstractIndividualComponent id="idaio74" name="label14">
      <output id="idaio75" name="label14"/>
    </abstractIndividualComponent>
  - <abstractIndividualComponent id="idaio76" name="label15">
      <output id="idaio77" name="label15"/>
    </abstractIndividualComponent>
  </abstractContainer>
  <abstractContainer id="idaio78" name="Telnet"> </abstractContainer>
  <abstractContainer id="idaio79" name="Terminal"> </abstractContainer>
  <abstractContainer id="idaio80" name="Selection"> </abstractContainer>
  <abstractContainer id="idaio81" name="Keyboard"> </abstractContainer>
</abstractContainer>
- <abstractIndividualComponent id="idaio82" name="button1">
    <control id="idaio83" name="button1"/>
  </abstractIndividualComponent>
- <abstractIndividualComponent id="idaio84" name="button2">
    <control id="idaio85" name="button2"/>
  </abstractIndividualComponent>
- <abstractIndividualComponent id="idaio86" name="button3">
    <control id="idaio87" name="button3"/>
  </abstractIndividualComponent>
- <abstractIndividualComponent id="idaio88" name="label1">
    <output id="idaio89" name="label1"/>
  </abstractIndividualComponent>
</abstractContainer>
</abstractContainer>
</auimodel>
```

Figura 4.32 Código de especificación de PuTTY a nivel abstracto

Este fichero ha sido creado siguiendo el estándar UsiXML, en concreto la parte referente a **Abstract User Interfaces (AUI).**

Como puede observarse, hay un nodo de tipo *auimodel* que engloba a todo el fichero. Dentro de él hay un primer nodo de tipo *abstractContainer* con el identificador

"idaio0" y el nombre "desktopPane1". Este nodo es un contenedor de tipo abstracto que se corresponde con el contenedor concreto de tipo *DesktopPane* de la figura 4.31.

Dentro del contenedor abstracto de identificador "idaio0" tenemos otro contenedor abstracto también de tipo *abstractContainer* con el identificador "idaio1", que se corresponde con el contenedor concreto de tipo *Panel* denominado "panel1" de la figura 4.31.

Dentro del contenedor abstracto de identificador "idaio1" hay cuatro componentes abstractos de tipo *abstractIndividualComponent*, denominados "idaio82", "idaio84", "idaio86", "idaio88", que se corresponden con los cuatro componentes concretos de la figura 4.31 denominados "button1", "button2", "button3" y "label1" respectivamente. El componente abstracto de identificador "idaio82", tiene una faceta de control asociada que indica que el componente puede enlazar con las funciones del sistema, por ejemplo, en este caso su funcionalidad sería la de mostrar información acerca del entorno. El componente abstracto de identificador "idaio84" sería semejante al anterior, pero su faceta de control consistiría en establecer la conexión. El componente abstracto de identificador "idaio86" sería semejante a los anteriores, pero su faceta de control consistiría en cancelar la conexión. Por último, el componente abstracto de identificador "idaio88" utiliza la faceta de salida, ya que muestra por pantalla el mensaje "PuTTY Configuration".

Además el contenedor "idaio1" contiene a otro contenedor abstracto (*abstractContainer*) de identificador "idaio2", que se corresponde con el componente concreto de tipo *TabbedPane* de la figura 4.31.

El contenedor abstracto de identificador "idaio2" se denomina "tabbedPane1" y contiene siete contenedores abstractos con identificadores: "idaio3", "idaio32", "idaio53", "idaio78", "idaio79", "idaio80" e "idaio81" respectivamente.

El contenedor de identificador "idaio3" se denomina "Connection", y contiene a catorce componentes abstractos de identificadores: "idaio4", "idaio6", "idaio8", "idaio10", "idaio12", "idaio14", "idaio16", "idaio18", "idaio20", "idaio22", "idaio24", "idaio26", "idaio28" e "idaio30" respectivamente.

El contenedor de identificador "idaio32" se denomina "SSH", y contiene a diez componentes abstractos de identificadores: "idaio33", "idaio35", "idaio37", "idaio39", "idaio41", "idaio43", "idaio45", "idaio47", "idaio49" e "idaio51" respectivamente.

El contenedor de identificador "idaio53" se denomina "Colours", y contiene a doce componentes abstractos de identificadores: "idaio54", "idaio56", "idaio58",

"idaio60", "idaio62", "idaio64", "idaio66", "idaio68", "idaio70", "idaio72", "idaio74" e "idaio76" respectivamente.

Cada uno de los componentes abstractos citados anteriormente, tienen unas facetas asociadas (en el apartado 3.2.1 de este documento puede verse la asociación realizada entre cada uno de los componentes concretos y las facetas AUI).

Como resultado de interpretar el fichero anterior, se obtiene la siguiente interfaz abstracta, donde se refleja todo lo explicado correspondiente al fichero de **especificación de usuario a nivel abstracto** asociado:

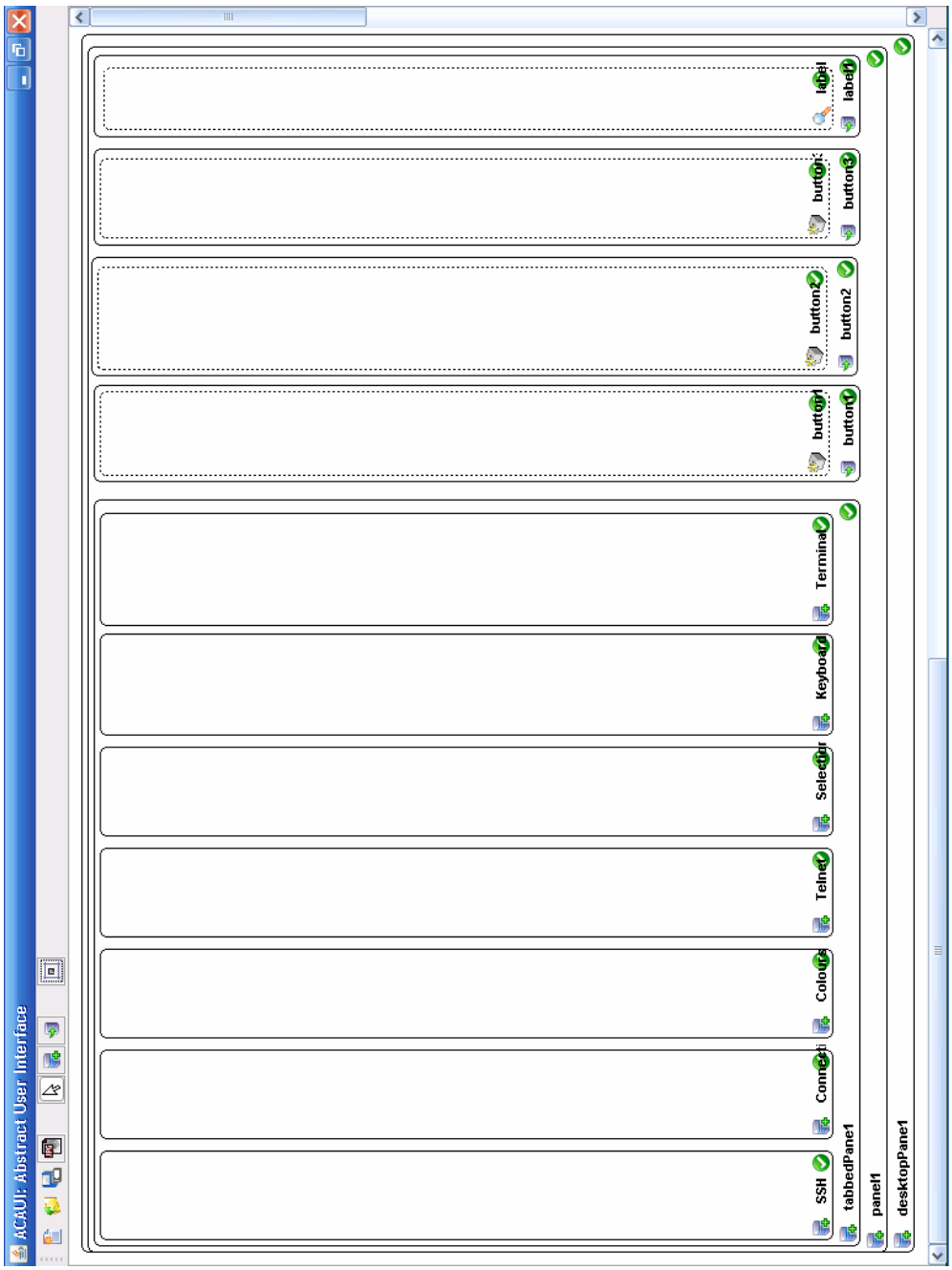


Figura 4.33 Interfaz de usuario abstracta de PuTTY

Capítulo 5: Conclusiones y relaciones con otros proyectos

Una vez terminada la presentación del proyecto realizado, llega el momento de hacer un compendio de las conclusiones obtenidas del trabajo realizado. También se hablará de la relación que este proyecto tiene con otros proyectos ya desarrollados y con otros proyectos que se han desarrollado en paralelo por otros estudiantes. En el último apartado del capítulo se hablará de posibles trabajos futuros relacionados con este trabajo.

5.1 CONCLUSIONES

Este proyecto ha centrado el interés desde el principio en tres conceptos: realización de una **abstracción de interfaces de usuario** desde **especificaciones concretas de interfaces de usuario**, el seguimiento de un **desarrollo basado en modelos** (**UsiXML**) y la comprensión de la importancia de la **abstracción de interfaces de usuario**.

La realización de una **abstracción de interfaces de usuario** desde **especificaciones concretas de interfaces de usuario** se ha realizado mediante la conversión de ficheros que siguen el estándar impuesto por **UsiXML**. Se crean dos ficheros, uno contiene la información de la **interfaz de usuario concreta** realizada por el usuario, y el otro fichero el cuál se obtiene como resultado de la abstracción realizada a partir del fichero anterior, contiene la información de la **interfaz de usuario abstracta**.

Esta abstracción de **interfaces de usuario** se realiza en el sentido inverso al normal seguido en **UsiXML**, ya que se realiza partiendo de especificaciones de **interfaces de usuario concretas**, y a partir de ellas se realiza una abstracción para obtener **interfaces de usuario abstractas**. Como se ha dicho anteriormente, el sentido normal de transformación utilizado en **UsiXML** es justo al contrario, es decir, partiendo de la **interfaz de usuario abstracta** se obtiene la **interfaz de usuario concreta**.

Como se ha ido mencionando a lo largo del documento, se ha seguido un **desarrollo basado en modelos** (**UsiXML**), buscando una tendencia a la estandarización, y un lenguaje de representación común de los datos interactivos. Esto sirve por ejemplo para que partiendo de la misma **interfaz de usuario**, ésta se pueda utilizar independientemente del contexto de uso y de las restricciones impuestas por un dispositivo o plataforma específicos, y así poderse interpretar correctamente por distintos tipos de dispositivos como: teléfonos móviles, PDAs y ordenadores.

Por último, hay que comprender la importancia de la **abstracción de interfaces de usuario concretas a interfaces de usuario abstractas**, para conocer con mayor detalle las necesidades que tiene el usuario a la hora de interactuar con una aplicación. Una vez que se ha obtenida la interfaz de usuario abstracta, pueden verse las facetas (de entrada, de salida, de control y de navegación) que tiene la **interfaz de usuario**, y de esta manera considerar si es correcta la cantidad de facetas de cada tipo, o si es necesario quitar o añadir facetas de algún determinado tipo a la interfaz, para facilitar el manejo de la **interfaz al usuario**. Esta abstracción también es útil para restaurar **interfaces de usuario** de aplicaciones antiguas cuyo código fuente se ha extraviado.

5.2 RELACIÓN CON OTROS PROYECTOS

Hay diversos proyectos relacionados con el actual, tanto proyectos ya finalizados como en desarrollo. Entre los proyectos ya finalizados relacionados están: IdealXML, TranformiXML, FlashiXML, VisualiXML, KnowiXML, GrafiXML, VisiXML, SketchiXML, FormiXML y ReversiXML. Estos proyectos, al igual que ocurre con el actual, siguen el **desarrollo de interfaces de usuario basadas en modelos** propuesto en **UsiXML**. En la página www.usixml.org se puede obtener información de todas estas aplicaciones.

Por ejemplo, la aplicación IdealXML es una herramienta orientada a patrones, que parte del modelo de tareas para obtener un diagrama **abstracto de interfaces de usuario**. Por otro lado, GrafiXML es una herramienta gráfica que **dibuja interfaces de usuario** para múltiples plataformas de computación. Se puede guardar una **interfaz de usuario** en varios formatos como Java o XHTML

En la siguiente figura puede observarse donde se situaría cada una de las aplicaciones anteriores en la jerarquía seguida en **UsiXML**:

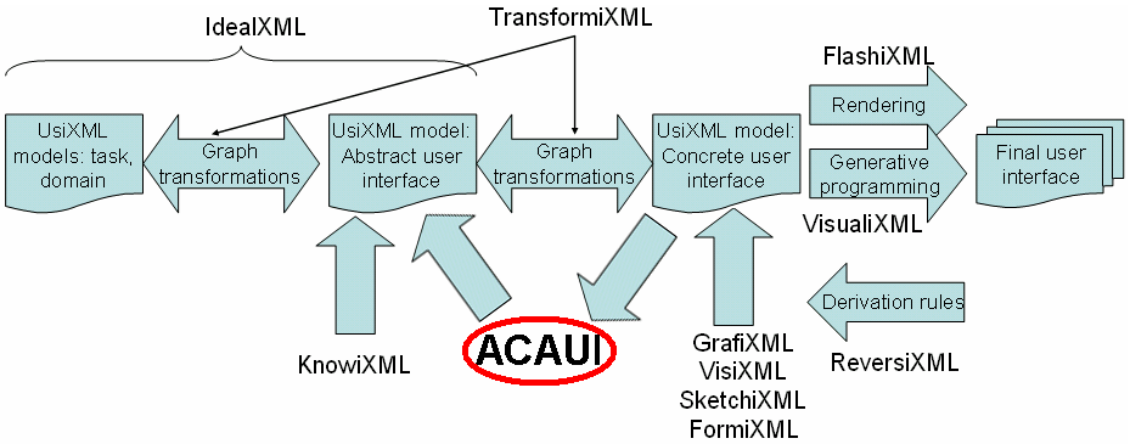


Figura 5.1 Aplicaciones realizadas siguiendo el lenguaje UsiXML

Como puede observarse, la aplicación **ACAUI** (desarrollada durante este proyecto) se sitúa en el medio de la jerarquía de **UsiXML**. Se aprecia que dicha aplicación parte de "**UsiXML model: Concrete user interface**" y tras realizar la abstracción se obtiene "**UsiXML model: Abstract user interface**".

Por otro lado, durante este curso se han desarrollado otros proyectos fin de carrera relacionados con el actual. Estos proyectos son los siguientes:

Tabla 5.1 Proyectos fin de carrera relacionados

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

Cabe destacar que el proyecto “Especificación de modelos de tareas a partir de especificaciones de interfaces de usuario” empieza justo en el mismo sitio donde termina proyecto actual, ya que siguiendo la jerarquía de **interfaces de usuario basadas en modelos**, el proyecto actual comienza en **interfaces de usuario concretas** y termina en **interfaces de usuario abstractas**, y el proyecto citado anteriormente comienza en **interfaces de usuario abstractas** para terminar en el modelo de tareas.

También cabe mencionar el proyecto “Transformación de especificaciones concretas a finales de interfaces de usuario”, ya que tanto en dicho proyecto como el actual, comienzan desde **interfaces de usuario concretas**, pero el proyecto citado anteriormente da un paso hacia delante en la jerarquía de **interfaces de usuario basadas en modelos**, ya que la convierte en una **interfaz de usuario final**. Y por otro lado el proyecto actual da un paso hacia atrás en dicha jerarquía, ya que tras la transformación lo que se obtiene es una **interfaz de usuario abstracta**.

5.3 TRABAJO FUTURO

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 un programa que partiendo de una **interfaz de usuario concreta**, calcule el grado de usabilidad que tiene dicha **interfaz de usuario**. La usabilidad de la interfaz creada se calcularía en función de la situación, la cantidad y el tipo de los elementos que componen la **interfaz de usuario** creada.

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

- Sería importante la realización de una herramienta que partiendo de una imagen (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 creación de un entorno integrado que permita la creación de **interfaces de usuario** desde una herramienta centralizada, es decir, consistiría en la integración de todos los editores utilizados en **UsiXML** en una única herramienta.

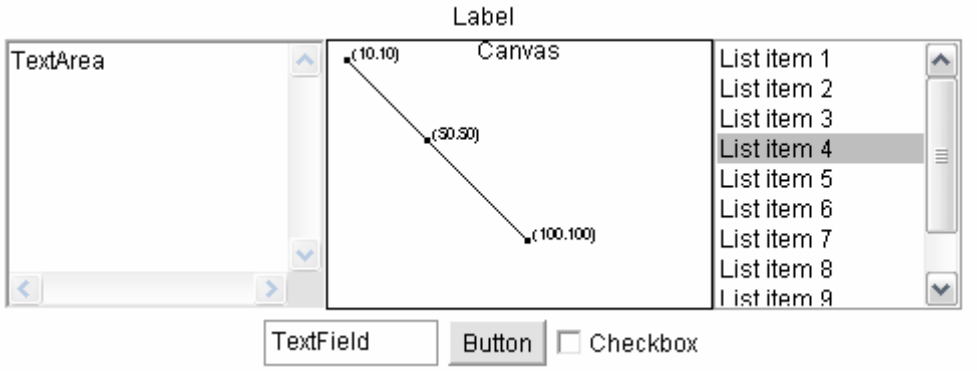
Apéndice A: El AWT como Kit Gráfico

El AWT (Abstract Windows Toolkit) es la parte de Java que se ocupa de construir interfaces gráficas del usuario. AWT permite hacer interfaces gráficas mediante artefactos de interacción con el usuario, como botones, menús, texto, botones para selección, barras de deslizamiento, ventanas de diálogo, selectores de archivos,… Y por supuesto despliegue gráfico general.

Partes que componen una interfaz gráfica:

1. Un container, que es la ventana o parte de la ventana donde se situarán los componentes (botones, barras de desplazamiento, etc).
2. Los componentes: menús, botones, barras de desplazamiento, cajas, áreas de texto, botones de opción y selección, etc.
3. El modelo de eventos. El usuario controla la aplicación actuando sobre los componentes.

La siguiente ilustración muestra algunos de estos artefactos de interacción:



Estos artefactos de interacción se denominan widgets. En la ventana los artefactos se organizan en una jerarquía de componentes gráficas:

- En la parte superior hay una etiqueta que dice “Label”. La aplicación usa las etiquetas para mostrar texto.

- A la izquierda hay un área para texto que contiene:

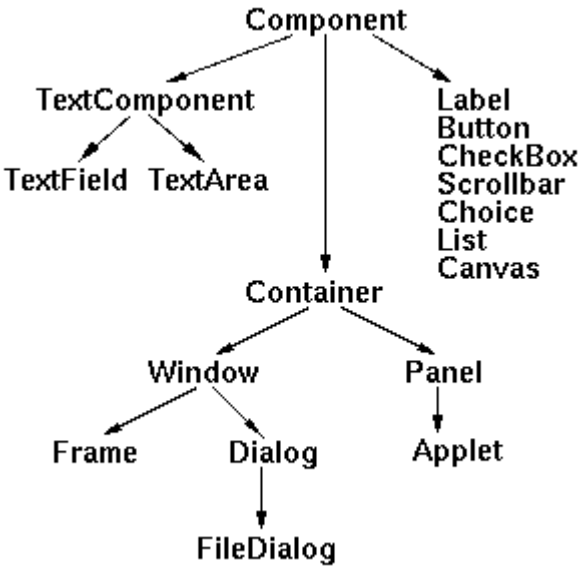
    - Una superficie para ingresar texto
    - Una barra de deslizamiento vertical
    - Una barra de deslizamiento horizontal

- A la derecha se observa una lista de ítems que contiene:
    - Una superficie para mostrar texto
    - Una barra de deslizamiento (sólo aparece si es necesario)
- Al centro hay un canvas, donde la aplicación dibuja figuras geométricas y/o texto.
- Abajo hay un panel de componentes con:
    - Un campo para ingreso de texto (una sola línea)
    - Un botón
    - Un botón de activado/desactivado que dice **“**Checkbox”

Es el programador de la interfaz gráfica el que diseña esta jerarquía de componentes.

I. Jerarquía de clases de AWT

Cada una de las componentes de una ventana en AWT se representa mediante uno o más objetos de la aplicación. Estos objetos pertenecen a las clases que se observan en la siguiente jerarquía de clases para AWT:



La clase de los contenedores sirve para crear áreas en la ventana cuyo único fin es colocar otras componentes en su interior. Hay dos tipos de contenedores:

- Panel: sirve para colocar botones, etiquetas, etc. En particular un applet es un panel.

- Window: sirve para crear nuevas ventanas independientes del browser Web. Es decir ventanas que serán manejadas por el administrador de ventanas de la plataforma (Motif, Windows, etc.). Una ventana independiente puede ser:

    - Frame es un tipo de ventana en donde se pueden colocar menús.

    - Dialog es un tipo de ventana para dialogar con el usuario. Se usan para colocar botones, etiquetes, etc. Es decir cumple la misma función que un panel, pero en una ventana independiente. En particular FileDialog es un artefacto para que el usuario escoja un archivo.

II. Construcción de una jerarquía de artefactos de interacción

Un artefacto de interacción se construye creando el objeto correspondiente de la jerarquía de clases. Por ejemplo el botón de la ventana de la figura se creó con:

```
Button boton= new Button("Button")
```

El argumento que recibe el constructor es la etiqueta que se colocará sobre el botón. El aspecto y comportamiento de este botón es dependiente de la plataforma. La aplicación se basa sólo en que será informada cuando el botón sea presionado por el usuario para ejecutar alguna acción.

Los otros artefactos se crearon con:

- Etiquetas: Label etiq= new Label("Label", Label.CENTER);

Label.CENTER es una constante de la clase Label y puesta como argumento en el constructor indica que el nombre de la etiqueta estará centrado en el rectángulo que se le asigne dentro de la ventana.

- Areas de texto: TextArea textArea= new TextArea("TextArea", 5, 20);

Usualmente un artefacto posee varios constructores con distintos parámetros. En este caso se especificó que el contenido inicial es "TextArea" y se necesitan 5 filas y 20 columnas. Sin embargo el contenedor puede asignar más espacio para distribuir mejor los artefactos en la ventana.

- La lista de ítems:

```
List lista = new List(3, false);
for (int i = 1; i <= 10; i++) {
  lista.addItem("List item " + i);
}
```

Las componentes de AWT se manipulan mediante métodos. En éste la lista se llena usando **addItem**.

- Etc.

III. Artefactos complejos

Los artefactos que hemos visto hasta el momento son simples porque la plataforma sabe como dibujarlos y qué tamaño deben ocupar. Es decir la aplicación no necesita especificar tamaño ni forma.

Canvas es un artefacto complicado porque AWT no conoce en absoluto el aspecto que debe tener en la ventana. Por lo tanto es la aplicación la que se debe preocupar de especificar qué tamaño debe tener y cómo se dibuja. Lo anterior se logra creando una nueva clase derivada a partir de Canvas. En ella se redefinen algunos métodos que realizan las funciones que un canvas normal no sabe hacer:

```
class MyCanvas extends Canvas{
   public void paint(Graphics g) {
      int w = size().width;
      int h = size().height;
      g.drawLine(10,10, 100,100);
   }

   public Dimension minimumSize() {
      return new Dimension(150,130);
   }

   public Dimension preferredSize() {
      return minimumSize();
   }
}
```

El principio es que todas las componentes tienen un método paint que AWT invoca cada vez que la componente se descubre y por lo tanto hay que redibujarla. La mayoría de las componentes (como botones, listas, etc.) saben dibujarse porque en la

clase respectiva se redefinió el método paint con el código apropiado para mostrar su aspecto.

Sin embargo el aspecto de un canvas es absolutamente dependiente de la aplicación. Por lo tanto es la aplicación la que redefine paint con sus propias instrucciones para dibujar el artefacto. Si este método no se redefine, el canvas no será dibujado cuando la ventana sea descubierta. El método paint recibe un argumento "g" que acepta primitivas gráficas como drawLine.

Del mismo modo, AWT invoca los métodos minimunSize y preferredSize para conocer el tamaño mínimo y preferido de un componente. Botones, listas, barras de deslizamiento,… han redefinido estos métodos.

Entonces en el nuevo canvas también se redefinen estos métodos para indicar el tamaño apropiado. Si estos no se redefinen el canvas podría quedar reducido a un tamaño 0. El nuevo canvas puede determinar su tamaño real en la ventana consultando su propio método *size*.

IV. Organizando los artefactos en un contenedor

Los contenedores usan organizadores (*layout manager*) para distribuir los artefactos en la superficie de despliegue de la mejor forma. Cada tipo de contenedor tiene su propio organizador. Por ejemplo los paneles usan FlowLayout que coloca los artefactos de izquierda a derecha.

Para el contenedor de la raíz (GUI) definimos explícitamente un organizador por medio de:

```
public class Gui extends Frame {
  GUI() {
    setLayout(new BorderLayout());
    ...
  }
```

Observe que como no se especifica objeto en la invocación, el método se aplica al objeto principal, que en este caso es la raíz (un frame).

Para crear el panel y colocar en su interior los artefactos se usa:

```
// Crear panel
Panel panel = new Panel();
```

```
// Crear artefactos
texto= new TextField("TextField");
boton= new Button("Button");
check= new Checkbox("Checkbox");

// Agregarlos al panel
panel.add(texto);
panel.add(boton);
panel.add(check);
```

Para colocar los artefactos en el frame (es decir en un objeto de la clase GUI) se usa el método *add*, pero con un argumento adicional que dice si cada artefacto se agrega a la izquierda, arriba, etc. Esto es específico del organizador BorderLayout.

```
// Define el organizador
setLayout(new BorderLayout());

// Crear artefactos
...

// Agregar artefactos
add("Center", canvas);
add("North", etiqueta);
add("West", areaTexto);
add("East", lista);

// Agregar el panel
add("South", bottomPanel);
```

Cuando se invoca *pack* en el frame se invoca el organizador que asigna un rectángulo a cada artefacto. A estas alturas todavía no se muestra en la pantalla la ventana. Para realizar esto se invoca el método *show* del frame. (AWT, 2007)

V. Relación entre las clases Swing y AWT

Javasoft, en vista de la precariedad de que hace gala el AWT, y para asegurarse que los elementos que desarrolla para generar interfaces gráficas sean fácilmente transportables entre plataformas, se ha unido con Netscape, IBM y Lighthouse Design para crear un conjunto de clases que proporcionen una sensación visual agradable y sean más fáciles de utilizar por el programador. Esta colección de clases son las *Java Foundation Classes* (JFC), que están constituidas por cinco grupos de clases, al menos en este momento: AWT, Java 2D, Accesibilidad, Arrastrar y Soltar y Swing.

Swing, es la parte más importante y la que más desarrollada se encuentra. Ha sido creada en conjunción con Netscape y proporciona una serie de componentes muy bien descritos y especificados de forma que su presentación visual es independiente de la plataforma en que se ejecute el applet o la aplicación que utilice estas clases. **Swing** simplemente extiende el AWT añadiendo un conjunto de componentes, *JComponents*, y sus clases de soporte. Hay un conjunto de componentes de Swing que son análogos a los de AWT, y algunos de ellos participan de la arquitectura MVC (*Modelo-Vista-Controlador*), aunque **Swing** también proporciona otros widgets nuevos como árboles, pestañas (JTabbedPane), JTextField, JToggleButton, etc.

La estructura básica del AWT se basa en *Componentes y Contenedores*. Estos últimos contienen Componentes posicionados a su respecto y son Componentes a su vez, de forma que los eventos pueden tratarse tanto en Contenedores como en Componentes, corriendo por cuenta del programador (todavía no hay herramientas de composición visual) el encaje de todas las piezas, así como la seguridad de tratamiento de los eventos adecuados. Con Swing se va un paso más allá, ya que todos los *JComponentes* son subclases de **Container**, lo que hace posible que widgets Swing puedan contener otros componentes, tanto de AWT como de Swing, lo que hace prever interesantes posibilidades.

Cuando se empieza a utilizar *Swing*, se observa que JavaSoft ha dado un gran paso adelante respecto al AWT. Ahora los Componentes del interfaz gráfico son *Beans* y utilizan el nuevo modelo de Delegación de Eventos de Java. Swing proporciona un conjunto completo de Componentes, todos ellos *lightweight*, es decir, ya no se usan componentes "*peer*" dependientes del sistema operativo, y además, *Swing* está totalmente escrito en Java. Todo ello redunda en una mayor funcionalidad en manos del programador, y en la posibilidad de mejorar en gran medida la cosmética de las interfaces gráficas de usuario.

Son muchas las ventajas que ofrece el uso de *Swing*. Por ejemplo, la navegación con el teclado es automática, cualquier aplicación Swing se puede utilizar sin ratón, sin tener que escribir ni una línea de código adicional. Las etiquetas de información, o "*tool tips*", se pueden crear con una sola línea de código. Además, Swing aprovecha la circunstancia de que sus Componentes no están renderizados sobre la pantalla por el sistema operativo para soportar lo que llaman "*pluggable look and feel*", es decir, que la apariencia de la aplicación se adapta dinámicamente al sistema operativo y plataforma en que esté corriendo.

Los Componentes Swing no soportan el modelo de Eventos de Propagación, sino solamente el modelo de Delegación incluido desde el JDK 1.1; por lo tanto, si se

van a utilizar componentes Swing, se debe programar exclusivamente en el nuevo modelo.

El paso de AWT a Swing es muy sencillo y no hay que descartar nada de lo que se haya hecho con el AWT. Afortunadamente, los programadores de Swing han tenido compasión y, en la mayoría de los casos es suficiente con añadir una "J" al componente AWT para que se convierta en un componente Swing.

Es muy importante entender y asimilar el hecho de que Swing es una extensión del AWT, y no un sustituto encaminado a reemplazarlo. Aunque esto sea verdad en algunos casos en que los componentes de Swing se corresponden a componentes del AWT; por ejemplo, el **JButton** de Swing puede considerarse como un sustituto del **Button** del AWT, y una vez que se usen los botones de Swing se puede tomar la decisión de no volver a utilizar jamás un botón de AWT, pero, la funcionalidad básica de Swing descansa sobre el AWT. Todo esto es para evitar que el lector salte directamente a Swing, ya que sería conveniente que primero entendiese el AWT y cómo Swing mejora al AWT. (TutorialJava, 2007)

Apéndice B: XML

XML, sigla en inglés de e**X**tensible **M**arkup **L**anguage ("lenguaje de marcas extensible"), es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C). Es una simplificación y adaptación del SGML (Standard Generalized Markup Language) y permite definir la gramática de lenguajes específicos (de la misma manera que HTML es a su vez un lenguaje definido por SGML). Por lo tanto XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades. Algunos de estos lenguajes que usan XML para su definición son XHTML, SVG, MathML.

XML no ha nacido sólo para su aplicación en Internet, sino que se propone como un estándar para el intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos, editores de texto, hojas de cálculo y casi cualquier cosa imaginable.

XML es una tecnología sencilla que tiene a su alrededor otras que la complementan y la hacen mucho más grande y con unas posibilidades mucho mayores. Tiene un papel muy importante en la actualidad ya que permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil. (Wikipedia, 2007)

I. Estructura de un documento XML

La tecnología XML busca dar solución al problema de expresar información estructurada de la manera más abstracta y reutilizable posible. Que la información sea estructurada quiere decir que se compone de partes bien definidas, y que esas partes se componen a su vez de otras partes. Entonces se tiene un árbol de pedazos de información. Ejemplos son un tema musical, que se compone de compases, que están formados a su vez con notas. Estas partes se llaman elementos, y se las señala mediante etiquetas.

Una etiqueta consiste en una marca hecha en el documento, que señala una porción de este como un elemento, un pedazo de información con un sentido claro y definido. Las etiquetas tienen la forma <nombre>, donde nombre es el nombre del elemento que se está señalando.

A continuación se muestra un ejemplo para entender la estructura de un documento XML:

```
<?xml version=”1.0”?>
<!DOCTYPE MENSAJE SYSTEM “mensaje.dtd”>
<mensaje>
    <remitente>
        <nombre>Alfredo Reino</nombre>
        <mail>alf@ibium.com</mail>
    </remitente>
    <destinatario>
        <nombre>Bill Clinton</nombre>
        <mail>president@WhiteHouse.gov</mail>
    </destinatario>
    <asunto>Hola Bill</asunto>
    <texto>
        <parrafo>¿Hola que tal? Hace <enfasis>mucho</enfasis> que
        no escribes. A ver si llamas y quedamos para tomar algo. </parrafo>
    </texto>
</mensaje>
```

II. Partes de un documento XML

Un documento XML está formado por el prólogo y por el cuerpo del documento.

Prólogo

Aunque no es obligatorio, los documentos XML pueden empezar con unas líneas que describen la versión XML, el tipo de documento y otras cosas.

El prólogo contiene:

- una declaración XML. Es la sentencia que declara al documento como un documento XML.
- una declaración de tipo de documento. Enlaza el documento con su DTD (Document Type Definition), o el DTD puede estar incluido en la propia declaración o ambas cosas al mismo tiempo.
- uno o más comentarios e instrucciones de procesamiento.

Cuerpo

A diferencia del prólogo, el cuerpo no es opcional en un documento XML, el cuerpo debe contener al menos un elemento raíz, característica indispensable también para que el documento esté bien formado.

III. Elementos

Los elementos XML pueden tener contenido (más elementos, caracteres o ambos), o bien ser elementos vacíos.

IV. Atributos

Los elementos pueden tener atributos, que son una manera de incorporar características o propiedades a los elementos de un documento.

V. Entidades predefinidas

Entidades para representar caracteres especiales para que no sean interpretados como marcado en el procesador XML.

VI. Secciones CDATA

Es una construcción en XML para especificar datos utilizando cualquier carácter sin que se interprete como marcado XML.

VII. Comentarios

Comentarios a modo informativo para el programador que han de ser ignorados por el procesador.

Los comentarios en XML tienen el siguiente formato:

```
<!-- Esto es un comentario -->
<!-- Otro comentario -->
```

Bibliografía

Ref. (Aachen, 2005): TassaneeEr-Jongmanee “XML-Driven Device Independent User Interface Build Rich Client Applications Using XML”. Tesis Aachen 2005.
Ruta del enlace:
www-i5.informatik.rwth-aachen.de/lehrstuhl/staff/chatti/theses/tassanee/thesis.pdf
Fecha de publicación: abril de 2005

Ref. (AWT, 2007): El kit gráfico AWT.
Ruta del enlace: http://www.dcc.uchile.cl/~lmateu/Java/Apuntes/awt.htm
Fecha de visita: 18 de abril de 2007

Ref. (CAMELEON, 2007): Página web oficial del Proyecto Cameleon.
Ruta del enlace: http://giove.cnuce.cnr.it/cameleon.html
Fecha de visita: 5 de julio de 2007

Ref. (ConcreteUserInterface, 2007): Definición de interfaces de usuario concretas.
Ruta del enlace:
http://www.awprofessional.com/articles/article.asp?p=167852&seqNum=7&rl=1
Fecha de visita: 16 de marzo de 2007

Ref. (InterfazDeUsuario, 2007): Definición de interfaz de usuario.
Ruta del enlace: http://www.fismat.umich.mx/~crivera/tesis/node6.html
Fecha de visita: 24 de marzo de 2007

Ref. (Jaquero, 2005):
Ruta del enlace: Tesis doctoral de Víctor López Jaquero.
http://www.dsi.uclm.es/personal/VictorManuelLopez/mipagina/archivos/thesis.pdf
Fecha de publicación: julio de 2005

Ref. (LenguajesDescriptivosDeUI, 2007):
Ruta del enlace: griho.udl.es/i2004/BajarPonencia/69.pdf
Fecha de visita: 4 de julio de 2007

Ref. (Lozano, 2001): Entorno metodológico orientado a objetos para la especificación y desarrollo de interfaces de usuario.
Fecha de publicación: 2001

Ref. (MDA Vanderdonckt, 2005): Medios para desarrollar interfaces de usuario de los sistemas de información.
Ruta del enlace:
http://www.isys.ucl.ac.be/bchi/publications/2005/Vanderdonckt-CAiSE2005-Web.ppt
Fecha de visita: 9 de agosto de 2007

Ref. (Montero et al, 2005): Francisco Montero, Víctor López Jaquero, Jean Vanderdonckt, Pascual González, María Lozano, Quentin Limbourg, "Solving the Mapping Problem in User Interface Design by Seamless Integration in IDEALXML". Ruta del enlace: http://www.dsi.uclm.es/personal/FranciscoMSimarro/idealxml_archivos/Montero-DSV-IS2005.pdf
Fecha de visita: 24 de abril de 2007

Ref. (Montero, 2005): Montero, F. "Integración de calidad y experiencia en el desarrollo de interfaces de usuario dirigido por modelos". Tesis UCLM 2005. Ruta del enlace: www.isys.ucl.ac.be/bchi/publications/Ph.D.Theses/Montero-PhD2005.pdf
Fecha de publicación: julio de 2005

Ref. (TERESA, 2007): Página web oficial de TERESA.
Ruta del enlace: http://giove.cnuce.cnr.it/teresa.html
Fecha de visita: 5 de julio de 2007

Ref. (TutorialJava, 2007): Tutorial de Java.
Ruta del enlace: http://www.itapizaco.edu.mx/paginas/JavaTut/froufe
Fecha de visita: 18 de abril de 2007

Ref. (UsiXML, 2007): Página web oficial de UsiXML.
Ruta del enlace: http://www.usixml.org
Fecha de visita: 12 de abril de 2007

Ref. (UsiXMLRomero, 2007): Descripción basada en modelos de interfaces de usuario con UsiXML.
Ruta del enlace: www.dsic.upv.es/workshops/dsdm06/files/dsdm06-09-Romero.pdf
Fecha de visita: 26 de julio de 2007

Ref. (UML, 2007): Información del lenguaje unificado de modelado (UML).
Ruta del enlace: ftp://jano.unicauca.edu.co/cursos/EnfasisIV/uml/UML.pdf
Fecha de visita: 14 de abril de 2007

Ref. (UMLCreangel, 2007): Información de UML.
Ruta del enlace: http://www.creangel.com/uml
Fecha de visita: 14 de abril de 2007

Ref. (Wikipedia, 2007): Enciclopedia online.
Ruta del enlace: http://es.wikipedia.org/wiki/Portada
Fecha de visita: 10 de abril de 2007