UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA

EN INFORMÁTICA

PROYECTO FIN DE CARRERA

TicXML: Generando diferentes interfaces de usuario finales a partir de una única especificación declarativa

Juan Carlos Peña

Septiembre, 2007

UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA POLITÉCNICA SUPERIOR

Departamento de Sistemas Informáticos

PROYECTO FIN DE CARRERA

TicXML: Generando diferentes interfaces de usuario finales a partir de una única especificación declarativa

Presentado por: Juan Carlos Peña

Dirigido por: Francisco Montero Simarro

Víctor Manuel López Jaquero

Septiembre, 2007

Índice de contenidos

Índice de figuras

Índice de tablas

Agradecimientos

Quiero aprovechar estas líneas para intentar expresar el profundo agradecimiento que debo a las personas que me han ayudado a la realización de este Proyecto Fin de Carrera. No sólo quisiera mostrar mi agradecimiento a las personas que han sido parte activa del proyecto, también quisiera destacar el papel de las personas que me han acompañado durante todos estos años.

El director del proyecto, Paco, ha hecho que el camino, en ocasiones incierto, de la investigación haya sido un interesante periplo, guiándome de forma magistral. Gracias por su trato profesional y sobre todo, por el trato personal que he recibido.

Verónica, que ha supuesto una ayuda activa, siempre intentando ayudar de cualquier manera y que sin duda, ha supuesto un apoyo moral muy importante. Gracias.

Mi familia, que se ha preocupado por mí de manera positiva, apoyándome y esforzándose por comprender ciertas cosas que no les son muy familiares. Aunque suene a tópico, he de decir que les debo mucho, y que sin su ayuda no habría sido posible llegar hasta aquí.

Mis amigos, que siempre han estado ahí desde hace muchos años y que en este sentido me han reforzado y ayudado. Gracias por contribuir a pasar tan buenos momentos y hacer que me sienta orgulloso de poder contar con vosotros.

Los compañeros de carrera, con los que he compartido muchos momentos durante estos años, haciendo que en el punto final uno ya sienta cierta nostalgia.

Los compañeros de trabajo, que en la recta final del proyecto han compartido conversaciones y me han ayudado aportando su experiencia.

Quisiera finalizar esta sección señalando que el esfuerzo que he tenido que realizar para la consecución del presente trabajo me ha resultado menos pesado gracias a las personas que he citado. Siento que en ocasiones no haya podido responder de la manera que hubiese querido por no contar con todo el tiempo que hubiera deseado, espero que no hayáis desesperado, haré lo posible por que este agradecimiento no quede sólo en unas pocas líneas. Sin más, muchas gracias por vuestra ayuda, vuestra forma de ser y por estar ahí.

Resumen

Para lograr que los humanos interactuaran con las máquinas se han venido desarrollando Interfaces de Usuario como punto de contacto y diálogo entre ambos. Con el desarrollo de la tecnología y la implantación de sistemas en múltiples escenarios, surge una gran diversidad de Interfaces de Usuarios diferentes.

En el contexto en el que se sitúa este proyecto final de carrera, el de la informática, cuenta con multitud de interfaces ya que cualquier interacción realizada entre usuario y ordenador se ha de realizar mediante una Interfaz de Usuario. En ocasiones el usuario se encuentra con gran cantidad de interfaces diferentes que intentan expresar lo mismo, pero cada una con un estilo determinado. Esta circunstancia unida a la faraónica tarea de tener que desarrollar un mismo sistema cada vez que se quiere implantar en una nueva plataforma, ha llevado a la investigación de una solución que resuelva la problemática presentada.

La solución pasa por utilizar modelos que puedan recoger los aspectos más relevantes de las Interfaces de Usuario. En este sentido, la arquitectura dirigida por modelos (MDA, *Model Driven Architecture)* resulta ser un respaldo a la idea presentada, donde se aboga por que los modelos dejen de ocupar el papel de meras representaciones con pretensiones de documentación para pasar a ser elementos activos en el proceso de desarrollo del software. Estos modelos servirán como base para realizar transformaciones sobre ellos con el objetivo de obtener implementaciones o ejecuciones de forma automática.

Para obtener un modelo de la Interfaz de Usuario se estudian las propuestas existentes en este sentido, optando por escoger los modelos presentados por la propuesta de UsiXML. El proyecto final de carrera tiene como punto de partida el modelo de interfaz de usuario a nivel concreto de la propuesta anteriormente citada.

Comúnmente las Interfaces de Usuario son implementadas sobre alguno de los lenguajes de programación, se opta por escoger dos que se consideran representativos; en primer lugar se tratará con el lenguaje orientado a objetos Java, que pasa por ser uno de los más populares en la época en la que se trata el proyecto; en segundo lugar se tratará con el lenguaje etiquetado HTML, que puede considerarse como el más utilizado para soportar páginas web en Internet.

Para transformar el modelo de la interfaz en alguno de los lenguajes de programación es necesario definir una serie de ideas y técnicas que permitan

transformar el modelo en código. Para su puesta en práctica se usa el lenguaje de transformación XSLT que sirve como soporte para realizar las transformaciones mediante plantillas.

El proyecto final de carrera lleva asociado la construcción de la herramienta *TicXML*, que se encarga de dar soporte a la metodología presentada, ofreciendo al usuario la posibilidad de realizar las transformaciones de una forma sencilla y transparente. Esta herramienta contribuye al desarrollo de UsiXML integrándose en el organigrama de las herramientas disponibles para la citada propuesta.

Las principales conclusiones que se obtienen de la realización del Proyecto Fin de Carrera pasan por señalar la necesidad de utilizar el modelado, considerándolo especialmente interesante como parte activa del proceso de ingeniería del software. El seguimiento de la filosofía presentada, así como la implantación del sistema desarrollado (o alguno similar a él, adaptado y/o ampliado a cada necesidad), facilitaría enormemente la construcción en multitud de plataformas, así como la migración entre ellas, consiguiendo algo muy importante en cualquier industria, ahorro en tiempo y costes.

Capítulo 1 : INTRODUCCIÓN

1.1 Contextualización de este proyecto

La interfaz de usuario (en lo sucesivo, UI, *User Interface*) es la forma en la que los usuarios pueden comunicarse con un sistema, ésta comprende todos los puntos de contacto entre el usuario y el sistema.

Existen dos tipos de UI: interfaces alfanuméricas (o interpretes de comandos) y las interfaces graficas de usuario (en lo sucesivo, GUI, Graphics User Interfaces), que son sobre las que se centrará este proyecto final de carrera; este tipo de interfaz se basa en el uso y la representación del lenguaje visual para conseguir una interacción "amigable"1 con un sistema informático, esto se consigue gracias a un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz, habitualmente las acciones se realizan mediante manipulación directa2 para facilitar la interacción usuario-sistema.

Otra clasificación posible para las UIs es si son hardware o software, en el primer caso nos estaríamos refiriendo al conjunto de dispositivos que hacen posible la interacción usuario-sistema, ejemplos posibles son: un teclado o un ratón. En el caso de las UI software se refieren a los programas que permiten expresar las acciones al sistema, ejemplos posibles en sistemas operativos son: el entorno Windows (Fig. 1.1) y el entorno X-Window de Linux (Fig. 1.2).

Figura 1.1 Interfaz de usuario de Windows

[1] Con alto grado de usabilidad

[2] Schneiderman, Ben, Designing the User Interface. Reading, MA: Addison-Wesley: 1998.

Figura 1.2 Interfaz de usuario X-Window de Linux

En este marco, en el de las aplicaciones de escritorio, se encuentran multitud de plataformas y lenguajes que ofrecen diversos sistemas finales, se pueden citar algunos ejemplos como: páginas web o programas de escritorio (dentro de estos existen varias implementaciones). De esta manera, el usuario puede estar percibiendo la misma información de diversas formas, presentado en multitud de formatos, pudiendo producir el efecto de que un simple cambio en la forma o localización de los componentes pueda llevar al desconcierto. Un mismo mensaje se representa de una manera u otra dependiendo de la implementación final (Fig. 1.3), en este caso se muestra un sistema operativo diferente dependiendo de la plataforma final.

Figura 1.3 Aplicaciones de escritorio, PDA y teléfono móvil

Se evidencia (Fig. 1.3) que en diversas ocasiones el contenido, en esencia, es el mismo pero la presentación varía de una implementación a otra. De esta forma, se puede observar que sería interesante separar el contenido de la presentación, pudiendo así manipular ambos de forma separada.

En la actualidad, los procesos de desarrollo se vienen apoyando en modelos, que en algunas ocasiones son utilizados como representaciones previas

al sistema que se quiere desarrollar, y en otras ocasiones, como elementos activos para los siguientes pasos. En la segunda casuística citada, se observa un gran potencial en cuanto a portabilidad, reusabilidad y mantenimiento de las UIs; en este sentido se puede abordar la transformación de los modelos iniciales en modelos o representaciones que vayan un paso más allá en la cadena de producción.

En algunos casos, los modelos están limitados a ser una guía o representación, ya que no es posible su reutilización o transformación (con los medios técnicos de la actualidad), un ejemplo de esto puede ser que los planos de una casa no son suficientes como para que se construya de forma automática la misma (Fig. 1.4).

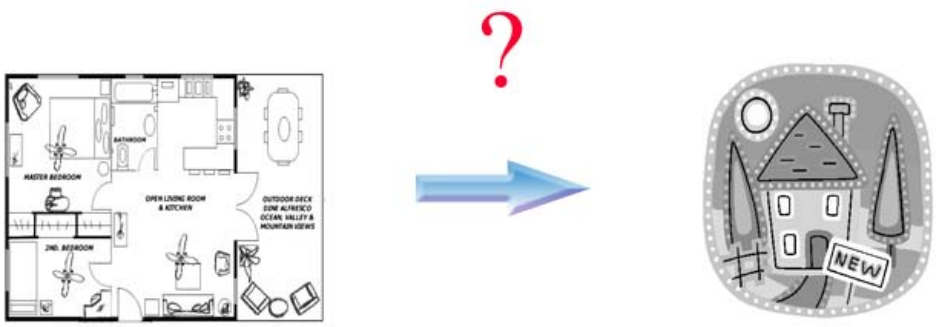


Figura 1.4 Metáfora de transformación de modelo, caso no posible

Por otro lado, es posible encontrar otros campos en los que estas ideas sí podrían ser llevadas a cabo, por ejemplo en el mundo de la ingeniería del software y, más concretamente en el desarrollo de UI, es posible realizar cierto aprovechamiento de los modelos; en este sentido irá dedicado gran parte del esfuerzo del presente proyecto final de carrera.

1.2 Motivación y objetivos del proyecto final de carrera

La principal motivación del presente proyecto final de carrera se centrará en el desarrollo de un sistema que proporcione los pasos necesarios para obtener resultados tácitos, en forma de código, a partir de modelos que especifiquen la configuración de una interfaz de usuario.

Para ello se fijan una serie de objetivos a alcanzar en el presente trabajo de investigación asociado al proyecto final de carrera:

- Estudiar la manera en la que se desarrollan las interfaces de usuario, centrándose en las partes comunes y replicadas.
- Identificar un marco que respalde la transformación de modelos.
- Examinar las propuestas existentes en torno a los modelos sobre las interfaces de usuario.
- Seleccionar una propuesta concreta que sirva como base para el desarrollo del sistema.
- Estudiar diversos lenguajes de implementación de interfaces de usuario.
- Definir las ideas y técnicas necesarias para transformar los modelos de interfaces de usuario en código.
- Familiarizarse con algún lenguaje de transformación que permita convertir los modelos en implementaciones.
- Implementar una herramienta que integre de una forma cómoda las transformaciones.
- Diseñar dos casos de estudio que prueben la herramienta y la metodología que se va a seguir; incrementando la complejidad del primer caso al segundo.

1.3 Estructura del presente documento

En el **capítulo primero**, el presente, se justifica la realización del Proyecto Fin de Carrera identificando la motivación asociada al mismo, dicha motivación se basa en el desarrollo de una herramienta que facilite el aprovechamiento de los modelos en las interfaces de usuario, para concluir en implantaciones reales, posibilitando así un marco para la portabilidad y reusabilidad, entre otras ventajas.

El **capítulo segundo** se presentan las ideas sobre las que se ha basado la realización del proyecto final de carrera, concretamente se centrará en la filosofía de la arquitectura dirigida por modelos. Seguidamente se presentan diversos lenguajes de especificación de modelos para este caso concreto, ofreciendo finalmente la conclusión obtenida del estudio de las propuestas presentadas y justificando la elección de un modelo concreto.

El **capítulo tercero** muestra la metodología llevada a cabo para la creación de una solución asociada a la investigación del presente trabajo. En este capítulo se parte de las ideas y propuestas presentadas en el capítulo anterior, la combinación de ambas junto con las investigaciones llevadas a cabo en el proyecto final de carrera cristalizará en una herramienta que soporte y automatice los pasos necesarios para las transformaciones deseadas.

El **capítulo cuarto** muestra dos casos de estudio que recorren toda la propuesta metodológica presentada en el capítulo anterior, para ello se hará uso de la herramienta desarrollada a tal efecto.

Finalmente, en el **capítulo quinto**, se expresan las conclusiones y los posibles trabajos futuros relacionados con este proyecto fin de carrera.

1.4 Consideraciones previas

Cabe destacar que uno de los pilares fundamentales del trabajo se trata de la propuesta de modelos de interfaces de usuario UsiXML (USer Interface eXtensible Markup Language) (UsiXML, 2007; Vanderdonckt, J., 2005).

UsiXML nace con el propósito de describir la interfaz de usuario a partir de diversos modelos de diseño, independientemente de las características concretas de cada lenguaje ó plataforma, es decir, un solo modelo y muchos dispositivos o plataformas finales. Por ello y por otras características que se detallaran en el capítulo segundo se ha optado por escoger como base esta propuesta.

UsiXML se trata de una propuesta de la universidad católica de Louvain (Bélgica), actualmente es seguida, documentada y ampliada por personas del todo el mundo. En la dirección http://www.usixml.org se puede encontrar toda la información referente a la propuesta, así como las herramientas asociadas a ella y multitud de documentación.

Capítulo 2 : LENGUAJES DE DESCRIPCIÓN Y ESPECIFICACIÓN DE INTERFACES DE USUARIO

2.1 Introducción

El desarrollo de interfaces de usuario se encuentra incluido en el mundo de la ingeniería del software. En este paradigma, las estructuras de datos, los objetos, las componentes, los aspectos, etc. Han sido y son elementos en los que apoyar las aplicaciones software, en estos momentos y desde hace un tiempo gracias al consenso alcanzado con el lenguaje de modelado UML y con la riqueza semántica y sintáctica que ofrece, se puede apostar por el desarrollo de productos software dirigido por modelos. En este sentido la propuesta de la arquitectura dirigida por modelos (*Model Driven Architecture* o MDA) provee de ideas y propuestas para sustentar que los modelos sean un elemento clave en el proceso de desarrollo de software, abandonando el papel clásico de meras representaciones de partes del software para tomar un papel decisivo, útil y básico para la generación automática de código.

Esta filosofía no es completamente nueva en el desarrollo específico de interfaces de usuario donde técnicas y desarrollos basados en modelos para el diseño de interfaces de usuario (Model-Based User Interface Development Enviroment, MB-UIDE) se han venido utilizando desde comienzos de los años 90. Estas técnicas vienen proponiendo el uso de modelos para especificar todos aquellos aspectos relacionados con la interfaz de usuario y que influyen en la misma.

El modelado con UML tiene algunas limitaciones, como la imposibilidad de reflejar la interacción y las limitaciones y deficiencias en lo que al tratamiento y consideración de las interfaces de usuario se refiere. Para poner solución a estas limitaciones, se están proponiendo diferentes lenguajes de especificación de interfaces de usuario, fundamentalmente de carácter declarativo. Entre ellos cabe destacar: UsiXML, propuesta que cubre todas las características de las interfaces de usuario; XIML, que cubre de una manera abstracta y genérica el problema; TERESA, define el desarrollo en tres fases y las sucesivas transformaciones entre ellas; Thinlet, como propuesta basada en código puro, con entradas y salidas determinadas; y Laszlo con características similares al anterior.

Este proyecto se centrará en UsiXML y concretamente se partirá de su especificación concreta para posibilitar la obtención de código a través de un

método declarativo por considerar éste el método que permite mayores facilidades de mantenimiento y modificación posterior.

2.2 La arquitectura dirigida por modelos (Model-Driven Architecture o MDA)

La arquitectura dirigida por modelos (en lo sucesivo, MDA, Model-Driven Architecture) (Brown, 2004) está siendo incluida en los desarrollos de diversas organizaciones por varias razones. MDA hace un uso eficiente del sistema de modelos en el proceso de desarrollo de software, y esto apoya la reutilización de las mejores prácticas creando las familias de sistemas.

Ha sido definida por el Grupo de Dirección de Objeto (en lo sucesivo, OMG, Object Management Group), MDA es un modo de organizar y manejar arquitecturas de la empresa apoyadas por instrumentos automatizados y servicios tanto para la definición de los modelos como para la facilitación de transformaciones entre tipos diferentes modelos.

En el desarrollo de software, las aplicaciones de hoy en día requieren un acercamiento a la arquitectura del software que ayude a los arquitectos a desarrollar sus soluciones de manera flexible. Este acercamiento debería permitir a la reutilización de esfuerzos existentes en el contexto de nuevas capacidades que ponen en práctica la funcionalidad de negocio en una manera oportuna, hasta como la infraestructura objetivo sí mismo se desarrolla. Este desafío se centra en dos ideas importantes:

- **Arquitecturas Orientadas a Servicios (SOA, Service-Oriented Architectures)**. Las soluciones empresariales pueden ser vistas como conjuntos de servicios conectados con contratos que definen las interfaces de servicio. Los diseños de sistemas resultantes se suelen llamar arquitecturas orientadas a servicios (SOAs). La flexibilidad puede ser realzada, en la arquitectura del sistema, organizando el sistema como una colección de servicios encapsulados que hacen llamadas y operaciones definidas en sus interfaces de servicios. Algunas organizaciones expresan sus soluciones en términos de servicios y sus interconexiones.

- **Cadenas de Producción de Software**. A menudo, hay una buena concordancia entre las organizaciones que desarrollan software y las que lo mantienen. Se observan accesos que se repiten en cada nivel de un proyecto de software empresarial, de tener modelos de dominio estándar que capturan procesos de negocio esenciales y conceptos del dominio, de manera que los

desarrolladores implementes soluciones específicas para hacer diseños en código.

Estas dos ideas tienen una influencia significante en las propuestas de OMG. Un consorcio de las organizaciones de software que desarrolla y soporta especificaciones para mejorar la práctica de desarrollo de software. El concepto MDA del OMG proporciona un acercamiento abierto, neutro del vendedor para la interoperabilidad del sistema vía las normas de modelado establecidas del OMG: UML, MOF, XML, XMI y CWM. Las descripciones de soluciones de la empresa pueden ser construidas usando estas normas de modelado y transformadas en una plataforma principal abierta o propietaria.

Antes de entrar en detalle en MDA, se relatarán los conceptos y beneficios fundamentales del modelado en el desarrollo de software.

2.2.1 La razón del modelado, transformaciones de modelos y MDA

Los modelos deben ser desarrollados como precursores a la implementación del sistema físico, o pueden ser sacados de un sistema existente o de un sistema en desarrollo como ayuda al entendimiento de su comportamiento.

Porque muchos aspectos de un sistema pueden ser interesantes, se pueden usar diversos conceptos de modelado y notación para resaltar una o varias perspectivas particulares. La transformación convierte modelos que ofrecen una perspectiva particular desde un nivel de abstracción a otro, normalmente desde una vista más abstracta hacia una menos abstracta, añadiendo mayor detalle proporcionado por las reglas de transformación.

Los modelos y el desarrollo de software conducido por modelos son el corazón del acercamiento de MDA.

En el mundo de la ingeniería del software, el modelado tiene una rica tradición, desde los primeros días de la programación. Las innovaciones más recientes se han centrado en notaciones y herramientas que permiten a los usuarios expresar las perspectivas del sistema de la valoración de los arquitectos de software y los desarrolladores de diversas maneras listas para situarse en código de algún lenguaje de programación que pueda ser compilado en un sistema operativo y plataforma particulares. En este momento se emplea mayoritariamente UML (Universal Modeling Language, o Lenguaje de Modelado Universal), UML permite avanzar a los equipos de desarrollo a capturar la variedad de características del sistema en sus correspondientes modelos.

Se hace mucho mas difícil de manejar la evolución de las soluciones así como incrementa la complejidad, o como el sistema evoluciona en el tiempo, o cuando los miembros originales del equipo del diseño no están directamente disponibles para el equipo de mantenimiento del sistema.

Una mejora sería la de proporcionar visualizaciones de código en alguna notación de modelado apropiada. Como los desarrolladores crean o analizan una aplicación, a menudo quieren visualizar el código por alguna notación gráfica que ayude su entendimiento de la estructura del código o el comportamiento. También puede ser posible manipular la notación gráfica como una alternativa a la corrección del código basado por texto, para que la interpretación visual se haga una representación directa del código.

En una visión de desarrollo centrado en el modelo, el modelado del sistema tiene el suficiente detalle para poder generar toda la implementación del sistema desde los modelos. El proceso de generación de código debe aplicar una serie de patrones para transformar los modelos en código, frecuentemente permitiendo al desarrollar elegir qué patrón elegir.

2.2.2 MDA: Un consenso en crecimiento

El modelado tiene un gran impacto en la ingeniería del software para resolver muchos problemas a escala empresarial. Sin embargo, hay gran diversidad en lo que los modelos representan y cómo son usados. Un pregunta interesante: ¿cuál de estas aproximaciones puede ser descrita como "conducido por modelos"? si se crea una visualización de una parte del sistema ¿significa que estoy haciendo MDA?, desafortunadamente, no hay ninguna respuesta definitiva. Más bien hay un acuerdo general en que MDA es más estrechamente asociado con accesos en los cuales el código es (semi-)automáticamente generado a partir de modelos más abstractos, y aquellas lenguas de especificación de estándar de empleo para describir aquellos modelos. Exploraremos este concepto en la siguiente sección.

2.2.3 MDA en teoría

Hay muchas opiniones sobre qué es MDA y qué no. La visión más autorizada la podemos recoger del OMG, un consorcio de industria de más de 800 empresas, organizaciones, e individuos. Como un estándar arquitectónico que surge, MDA se cae en una tradición larga de apoyo de OMG y codificación de numerosas normas calculadoras durante las dos décadas pasadas. OMG ha sido el responsable del desarrollo de algunos de los más conocidos e influenciables estándares de la industria para la especificación de sistemas y la

interoperabilidad, incluyendo entre ellos: CORBA, IDL, IIOP, UML, MOF, XML, XMI, CWM, OMA. Además, OMG ha realzado estos datos específicos para apoyar industrias específicas como la atención de salud, la fabricación, telecomunicaciones, y otros.

2.2.4 Los principios de MDA

Cuatro son los principios de la visión de OMG sobre MDA:

- Modelos expresados en una notación bien definida son una piedra angular a sistemas comprensivos para soluciones de escala de la empresa.
- La construcción de sistemas puede ser organizada en torno a una serie de modelos e imponiendo una serie de transformaciones entre modelos, organizados en un en un marco arquitectónico de capas y transformaciones.
- Un soporte formal para describir modelos en un conjunto de metamodelos facilita la integración significativa y la transformación entre modelos, y es la base para la automatización por herramientas.
- La aceptación de este acercamiento basado en el modelo requiere que los estándares de la industria proporcionen garantía a los consumidores, y la competencia entre vendedores.

Una herramienta de MDA debería soportar transformaciones de un modelo en varios pasos, desde el modelo inicial de análisis hasta el código ejecutable.

Los desarrolladores de MDA reconocen que las transformaciones pueden ser aplicadas para abstraer las descripciones de los aspectos de un sistema para añadir el detalle, hacer la descripción más concreta, o convertir entre representaciones. La distinción entre las clases diferentes de modelos nos permite pensar en el software y el desarrollo de sistema como una serie de refinamientos entre representaciones diferentes modelos. Estos modelos y sus refinamientos son una parte crítica de la metodología de desarrollo para las situaciones que incluyen refinamientos entre modelos que representan los diversos aspectos del sistema, la inclusión de detalles a un modelo, o la conversión entre las clases diferentes de modelos.

Las siguientes tres ideas son importantes para convertir la naturaleza abstracta de un modelo en la implementación detallada que lo representa:

- **Clasificación de modelos.** Podemos clasificar el software y los modelos de sistema en términos de como explícitamente representan los aspectos de las plataformas objetivo. En el software y el desarrollo del sistema son restricciones importantes por la opción de lenguas, hardware, conexión a una red con una determinada topología, protocolos de comunicaciones e infraestructura, etc. Cada uno de estos elementos puede ser considerado de una solución. Un acercamiento de MDA nos ayuda a enfocar que es esencial el diseño de los aspectos de negocio de una solución, separado de los detalles de la plataforma.

- **Independencia de la plataforma**. La noción de plataforma es compleja y muy dependiente del contexto. Por ejemplo, en ocasiones la plataforma será el sistema operativo y las herramientas asociadas, en otras ocasiones será una infraestructura tecnológica representada por modelos bien definidos de programación como J2EE o .Net, en otras ocasiones será una instancia de una topología de red. En cualquier caso, es más importante pensar en termines de qué modelos y niveles de abstracción se van a usar para los diversos propósitos, que invertir tiempo en definición de la plataforma.

- **Transformación de modelos y refinamiento**. Pensando que el desarrollo del software y el sistema es un conjunto de refinamientos entre modelos, las transformaciones entre modelos se convierten en los elementos clave del proceso de desarrollo. Para llevarlas a cabo será necesario el conocimiento especializado en el dominio del negocio, las tecnologías usadas para la implementación, o ambos. Se puede mejorar la eficiencia y calidad de los sistemas capturando estas transformaciones explícitamente y rehusándolas coherentemente a través de soluciones. Si los diferentes modelos abstractos están bien definidos, se pueden utilizar transformaciones estándar. Por ejemplo, con modelos diseñados en UML e implementaciones en J2EE, se podrá, en muchos casos, usar una transformación UML-a-J2EE con patrones que puedan ser aplicados, validados y automatizados.

Siendo la base estas representaciones los modelos, y apoyando las transformaciones, como una serie de metamodelos. La capacidad de analizar, para automatizar, y transformar modelos requiere un modo claro e inequívoco de describir la semántica de los modelos. Los modelos y las transformaciones entre ellos serán especificados usando estándares de carácter abierto.

Ejemplo

Ejemplo simplificado de un modelo independiente de la plataforma (PIM, platform independent model) y sus transformaciones en tres diferentes modelos dependientes de la plataforma (PSM, platform-specific models).

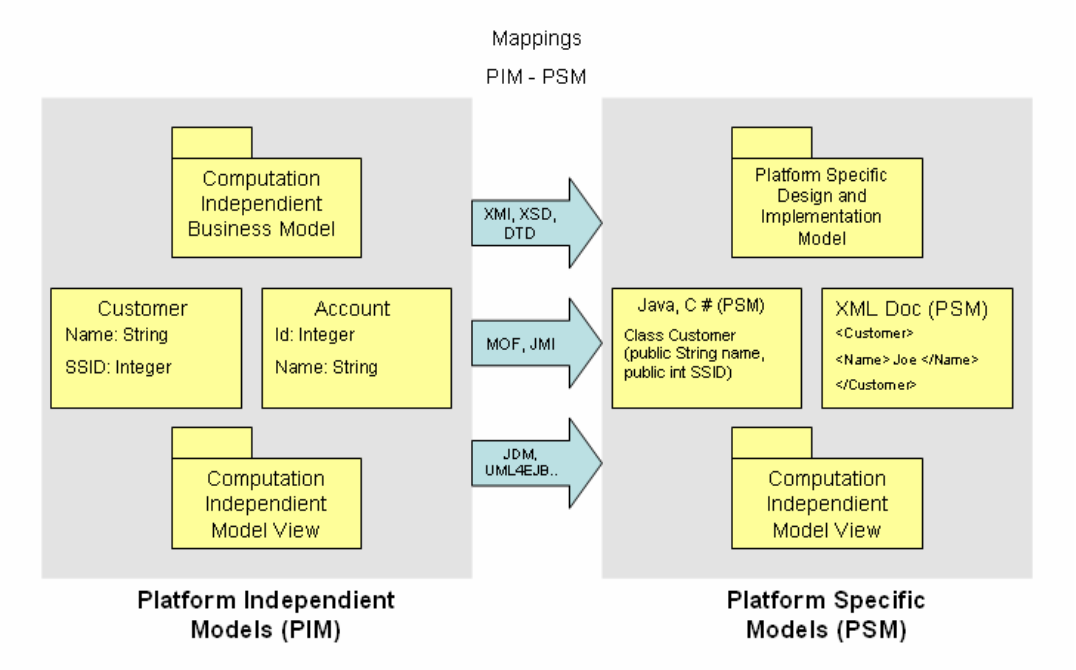


Figura 2.1 Ejemplo de transformación de PIM a PSM

Este proyecto realizará una correspondencia entre PIM y PSM, en la figura 2.2 se pueden observar ejemplos de estas correspondencias, encontrándose sombreada la zona que ocupa este proyecto final de carrera.

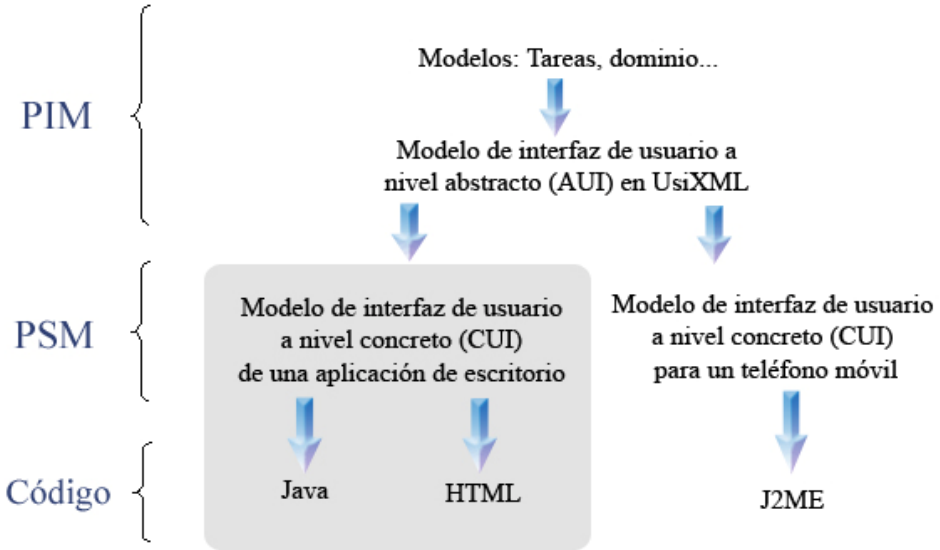


Figura 2.2 Transformaciones de PIM a PSM y código

2.3 Desarrollos basados en modelos para el diseño de Interfaces de Usuario

Se entiende por modelo a una idealización de la realidad utilizada para plantear un problema, normalmente de manera simplificada en términos relativos. En concreto, los modelos para el diseño de alguna parte de un sistema software son aquella representación declarativa de todos los aspectos relevantes, donde se incluyen y consideran cada uno de sus componentes, así como aspectos de diseño afines. En un modelo se ven recogidos los aspectos de algún nivel del sistema, la combinación de varios modelos puede especificar completamente el comportamiento de un sistema.

En el área del diseño de UI, son ampliamente utilizados los entornos de desarrollo basados en modelos, es lo que se conoce como Model-Based User Interface Development Enviroment (MB-UIDE) (Montero, 2005). El propósito de esta propuesta es que la especificación de todos aquellos aspectos relacionados con la interfaz de usuario y que influyen en la misma se hagan utilizando modelos.

Para la realización de un modelo o un conjunto de modelos existen varias propuestas y opciones. La evolución en el modelado ha ido desde la utilización de modelos propios por parte de cada organización, pasando por especificaciones algebraicas hacia la previsible futura estandarización. En este sentido, la propuesta de Lenguaje Unificado de Modelado (UML, Universal Modeling Language), se sitúa como el lenguaje de modelado de sistemas de software más utilizado en la actualidad, aún cuando todavía no es un estándar oficial, está apoyado en gran manera por el OMG (Object Management Group). Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema de software.

No obstante, los modelos utilizados no tienen porqué ser los mismos en organizaciones distintas, de hecho, aún no existe un consenso en cuanto a los modelos a seguir y las notaciones a utilizar. En cualquier caso, aparecen diversos modelos característicos, algunos de ellos se han introducido de forma constante y otros han ido apareciendo conforme las nuevas necesidades han ido apareciendo. Los modelos más utilizados en el desarrollo de UI son: modelos de dominio, de tareas, de usuario, de diálogo, de presentación, de plataforma, de contexto y de comportamiento. Seguidamente se comentarán algunos de ellos:

El modelo de dominio recoge información acerca de los objetos que manipulará el usuario, juntos con sus características y su comportamiento.

El modelo de tareas recoge una especificación de las tareas que el usuario llevará a cabo en el sistema con el fin de realizar sus objetivos. Las tareas

pueden ser de diferentes tipos y se descompondrán en acciones. Un ejemplo de notación utilizada en este tipo de modelo podría ser la notación CTT (ConcurTaskTrees) (Paternò, 1999).

El modelo de usuario recoge las características del usuario final. Estas características es conveniente que sean tenidas en cuenta en el desarrollo de la interfaz de usuario, así como la adaptación y/o adaptatividad y algún sistema de ayuda al usuario.

Con estos modelos se especifican los elementos característicos para la posterior fase de desarrollo de la aplicación y de su interfaz de usuario. No obstante, la verdadera potencia de los MB-UIDE son las relaciones que se establecen entre los diversos modelos.

2.4 Lenguajes de especificación de Interfaces de Usuario

Para especificar los aspectos de una interfaz de usuario se ha de recurrir a algún tipo de lenguaje de especificación. Las diferentes propuestas que se presentarán a continuación, utilizan el estándar XML para especificar los modelos correspondientes a las interfaces de usuario.

XML (eXtensible Markup Language) (XML, 2006) es un meta-lenguaje que permite definir otros lenguajes de marcación. Con XML el diseñador puede definir sus propios comandos. XML no ha nacido sólo para su aplicación en Internet, sino que se propone como lenguaje de aplicación para intercambio de información estructurada entre diferentes plataformas. En definitiva XML es una especificación genérica para formatear documentos según una sintaxis estándar e independiente de los lenguajes de programación y las plataformas.

Por las características anteriores, XML encaja perfectamente con la filosofía de las siguientes propuestas que a continuación se presentan como propuestas de lenguajes de especificación de interfaces de usuario.

2.4.1 UsiXML

UsiXML (USer Interface eXtensible Markup Language) (UsiXML, 2007; Vanderdonckt, J., 2005) es una propuesta de la universidad católica de Louvain (Bélgica), con el propósito de describir la interfaz de usuario a partir de diversos modelos de diseño, independientemente de las características concretas de cada lenguaje ó plataforma, es decir, un solo modelo y muchos dispositivos o plataformas finales.

En la actualidad el desarrollo de UI es muy costoso por la complejidad y diversidad de los entornos de desarrollo existentes, a esto hay que añadir cada vez una mayor cualificación para el desarrollo de las UI: lenguajes de marcas (ej: HTML), lenguajes de programación (ej: C++ o Java), conocimiento de sistemas de comunicación, uso de la ingeniería de la usabilidad. Estas dificultades se incrementan cuando la misma UI ha de ser desarrollada en múltiples contextos, múltiples categorías de usuarios (ej: diferentes idiomas, diferentes preferencias, discapacitados...), diversas plataformas (ej: teléfonos móviles, PDA, quioscos interactivos, ordenadores de sobremesa, pantallas táctiles...) y diversos entornos de trabajo (ej: móvil, estático).

Para los diseñadores y programadores que se ven envueltos en este tipo de proyectos, las herramientas son principalmente el problema con el que se encuentra el desarrollador. De este modo, es muy difícil diseñar una UI para múltiples contextos. Esta propuesta ofrece una vía para separar responsabilidades en este tipo de proyectos.

La propuesta hace uso del lenguaje xml para describir la UI desde múltiples contextos: la interfaz de usuario a nivel concreto (en lo sucesivo, CUI, Concrete User Interface), a nivel gráfico, a nivel de auditoria y a nivel multimodal.

Algunas de las características a destacar de la propuesta son:

- **No son necesarios conocimientos técnicos especiales**, con una simple visión de lo que se quiere poner en la interfaz es suficiente para especificarla en este lenguaje. UsiXML está pensado para los no desarrolladores (ej: analistas, diseñadores, jefes de proyecto...) sin requerirles conocimientos concretos en lenguajes de marcas ó lenguajes de programación, por supuesto, la propuesta también va dirigida a los desarrolladores.

- El lenguaje de UsiXML es un **lenguaje declarativo** que captura la esencia de la UI, de forma similar a como piensan los humanos.

- UsiXML describe la UI a un **alto nivel de abstracción** los elementos que la constituyen: componentes, controles, técnicas de interacción... Usado en combinación con compiladores y entornos de desarrollo permite el desarrollo completo de la UI.

- Es **independiente del dispositivo**: una UI puede ser descrita independientemente de los dispositivos como un ratón, teclado, etc.

- Es **independiente de la plataforma**: una UI puede ser descrita independientemente de la plataforma en la que se ejecutará como pueda ser un teléfono móvil, una PDA, un ordenador, etc.

- Es **independiente del modo de interacción**: una UI puede ser descrita independientemente del modo de interacción como pueda ser la interacción gráfica, la interacción vocal, interacción tridimensional, etc.

- Soporta la **independencia de modo**: una UI puede ser descrita independientemente del modo de interacción, ya sea interacción gráfica, vocal, 3D...

- Permite la **reutilización de elementos** previamente descritos en anteriores UI para crear nuevas UI en nuevas aplicaciones.

La cobertura de UsiXML es amplia, no obstante, la propuesta no pretende acaparar todas las actividades del desarrollo de UI:

- UsiXML no determina el lenguaje de la implementación de la UI. Propone la integración con alguno de estos formatos: cHTML, WML, HTML, XHTML, VoiceXML, VRML, Java, C++,.... Pero es tarea a realizar la transformación de UsiXML a alguno de estos formatos.

- No describe los detalles a bajo nivel de los elementos, como pueda ser los atributos del sistema operativo, eventos y primitivas.

- El código UsiXML no puede ser compilado o ejecutado por sí mismo: se necesita una transformación ó renderización que forme parte del proceso de desarrollo.

- No pretende dar soporte a todos los atributos, eventos y primitivas de todos los componentes existentes en todos los entornos. Contempla un subconjunto de los más comunes.

Este trabajo se centrará en el apartado concreto de la especificación de la CUI, en este nivel se encontrarán elementos típicos de la interfaz gráfica como puedan ser botones, etiquetas, campos de introducción de texto, etc. Estos elementos tienen una serie de características, reflejadas en el código xml en forma de atributos, que servirán de punto de partida de la transformación hacia una interfaz de usuario a nivel final (en lo sucesivo, FUI, Final User Interface).

En la figura 2.3 se puede observar el esquema básico de UsiXML para el modelo de CUI.

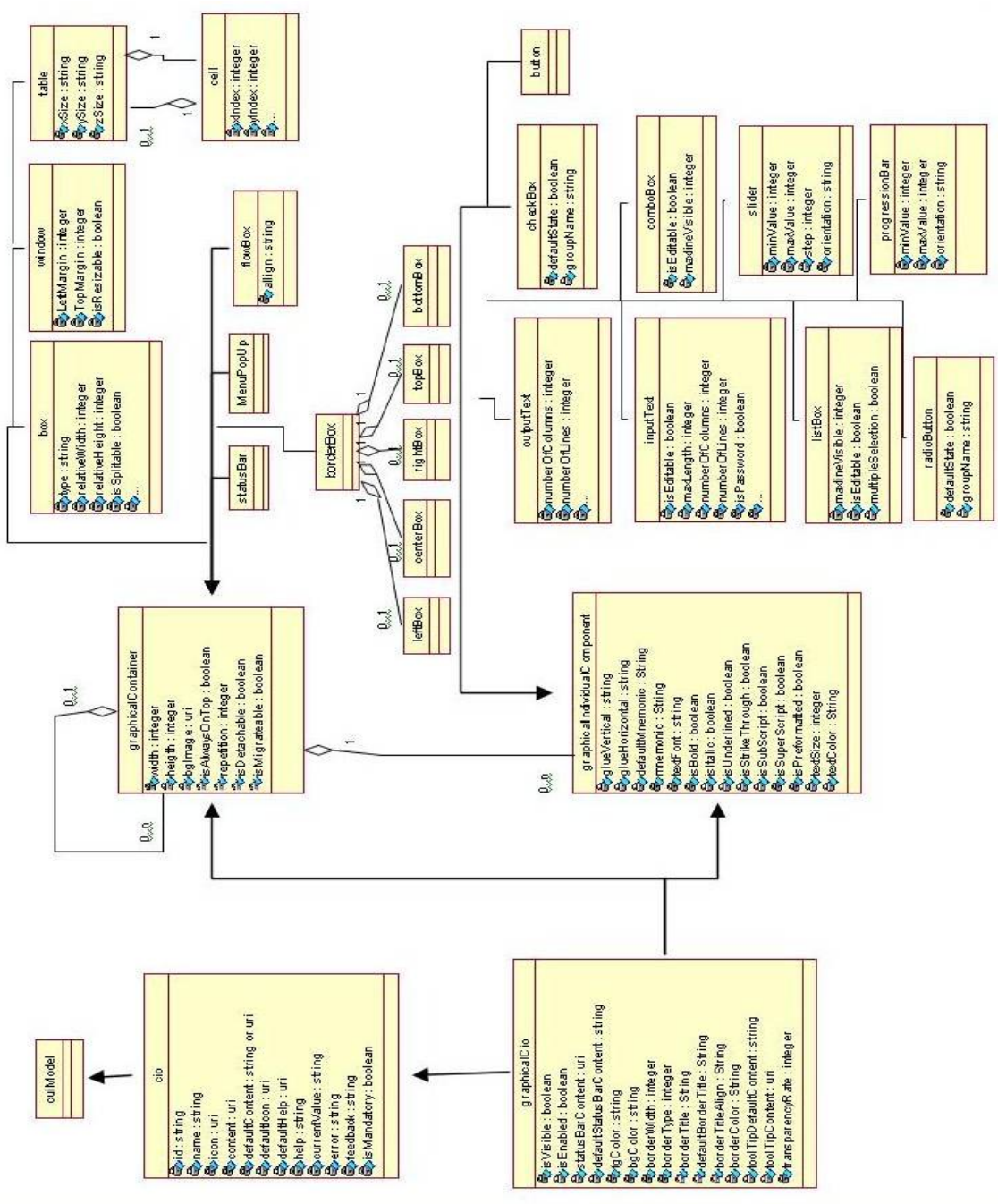


Figura 2.3 Diagrama UML del modelo CUI

A continuación se muestran diversos componentes individuales que son los que dan forma a la especificación de la interfaz de usuario en UsiXML (tabla 2.1), corresponden al último nivel del anterior diagrama de clases (Fig. 2.1), y servirán de punto de partida para el presente trabajo.

Componente	Código UsiXML	Descripción
window	<window id="w0" name="w_0" width="400" height="350"> *"contenido"* </window>	Típicamente este componente contendrá otros contenedores como 'box', 'tables', etc
box	<box id="b1" name="b_1" type="vertical"> ... </box>	Hace la de función de contenedor de componentes gráficos.
button	<button id="b1" name="b_1" isVisible="true" isEnabled="true"/>	Los botones tendrán, además de las propiedades propias de un elemento gráfico, propiedades como si están activos o no.
radioButton	<radioButton id="r1" name="r_1" isVisible="true" isEnabled="true"/>	Botón de radio, características similares al anterior.
comboBox	<comboBox id="c1" name="c1" isVisible="true" isEnabled="true"/>	Múltiples opciones desplegables, características similares al anterior.
checkBox	<checkBox id="c3" name="c_3" isVisible="true" isEnabled="true"/>	Casilla de marcado, características similares al anterior.
listBox	<listBox id="l1" name="l_1" isVisible="true" isEnabled="true" multiple_selection="false"/>	Lista con diversas opciones a seleccionar, con la propiedad 'multiple_selection' se permite seleccionar varias opciones.
inputText	<inputText id="i1" name="i_1" isVisible="true" isEnabled="true" isPassword="false" maxLength="50" numberOfColumns="15" numberOfLines="2" isEditable="true"/>	Componente de introducción de texto, propiedades interesantes son la de la máxima longitud para introducir los datos, número de filas y columnas, si es posible editarlo y si es campo contraseña.
table	<table id="t3" name="t_3" isVisible="true" isEnabled="true" xSize="3" ySize="5"/>	Componente que representa a las tablas.
outputText	<outputText id="o1" name="o_1" isVisible="true" isEnabled="true" isBold="true"/>	Representa a la etiqueta de texto que comúnmente acompaña a otros componentes gráficos.

Tabla 2.1 Componentes CUI de UsiXML

2.4.1.1 *UsiXML y sus herramientas disponibles*

La propuesta de UsiXML cuenta con un buen número de herramientas que individualmente realizan algún(os) paso(s) de la cadena de acciones abarcada por la propuesta.

En la figura 2.4 se pueden ver algunas de las herramientas existentes y el papel que desempeñan en la propuesta.

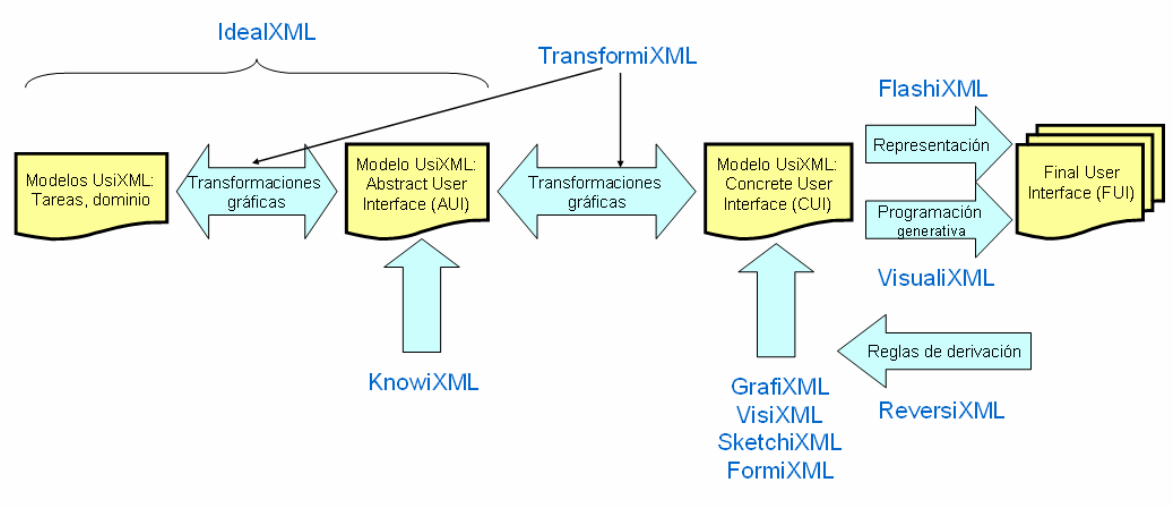


Figura 2.4 Herramientas disponibles en UsiXML

A continuación se detallarán algunas de las herramientas consideradas como más importantes, representativas de alguno de los pasos esenciales.

IdealXML

IdealXML es una herramienta orientada a patrones (IdealXML, 2007). Es posible editar a través de texto las características asociadas a los patrones, como: nombre, alias, problema, contexto, solución, sinopsis, etc. También es posible editar diagramas usando las notaciones más comunes (diagramas de clases y CTT) de la ingeniería del software y la interacción persona-ordenador. Los diagramas utilizan las notación UsiXML y los patrones son guardares usando el lenguaje PLML (Pattern Language Markup Language).

La herramienta no da soporte a especificaciones a nivel concreto, pero si a nivel abstracto, de tareas, del dominio y mapping o correspondencia. Lo que puede resultar muy interesante de cara a completar la descripción de la UI.

A continuación se muestra el aspecto general de la herramienta (Fig. 2.5), con los apartados correspondientes al modelo del dominio, el modelo de tareas y el modelo AUI.

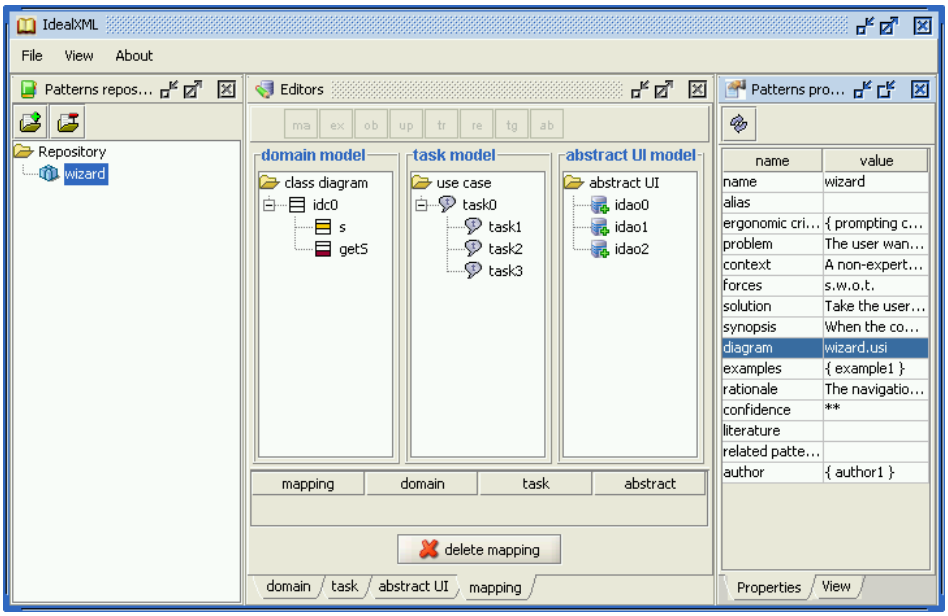


Figura 2.5 Aspecto general de la herramienta IdealXML

GrafiXML

Permite colocar los elementos típicos de una UI de manera gráfica e intuitiva, generando un fichero UsiXML que refleja la disposición propuesta por el usuario (GrafiXML, 2007). Esta herramienta es usada en el presente trabajo para obtener los archivos UsiXML necesarios para las transformaciones.

En la figura 2.6 se puede observar una visión de UI en la herramienta GrafiXML. Inicialmente, en el modo compositor, podemos arrastrar los componentes gráficos a una ventana, de forma similar a como lo hace cualquier herramienta de desarrollo visual.

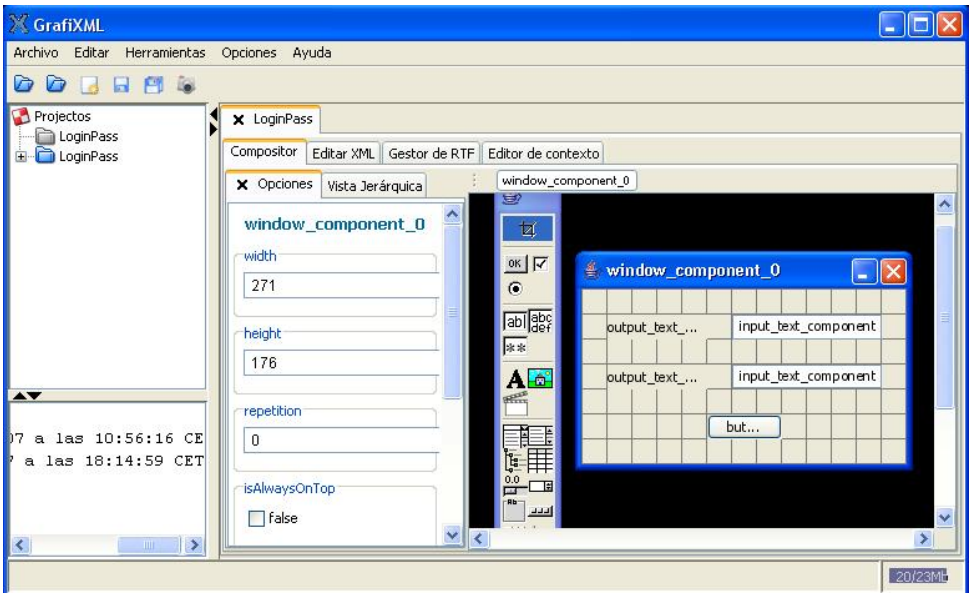


Figura 2.6 Aspecto general de la herramienta GrafiXML

A continuación se puede observar el código en UsiXML que genera la disposición dada (Fig. 2.7), para ello hay que cambiar de modo al indicado en la pestaña 'Editar XML'.

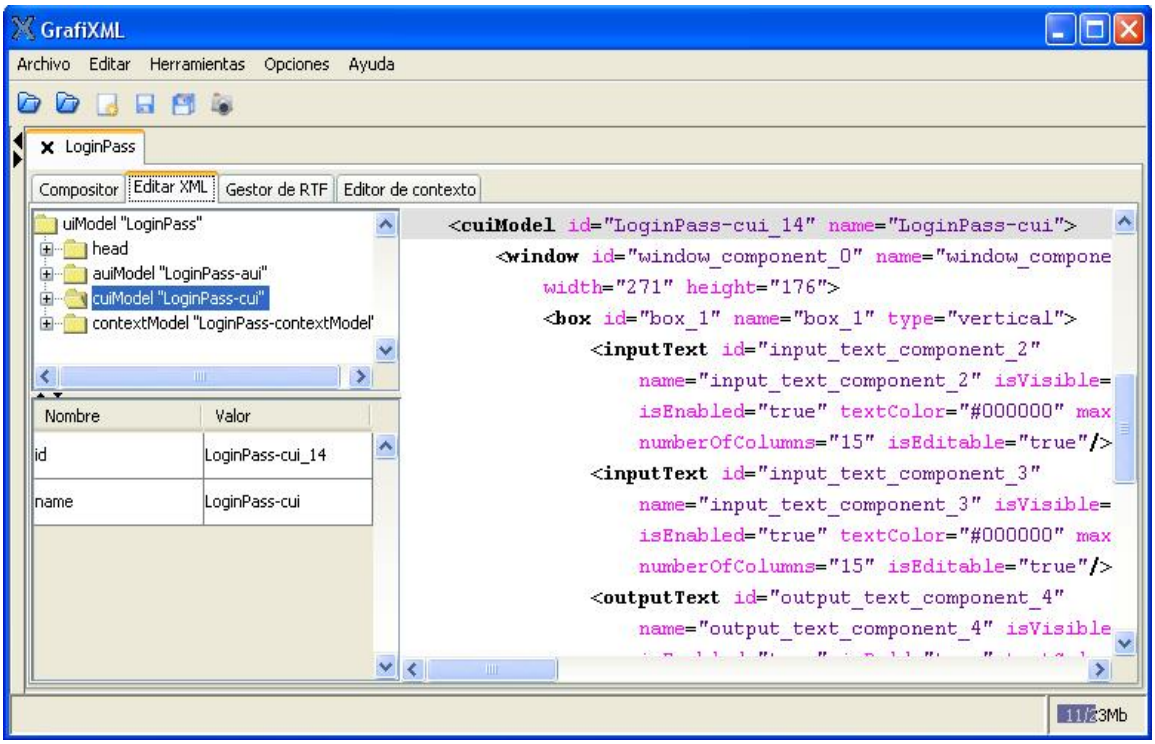


Figura 2.7 Código obtenido en GrafiXML

FlashiXML

Es una herramienta que a partir de especificaciones en UsiXML genera UI en flash, todo ello integrado en un programa flash (FlashiXML, 2007).

En las siguientes ilustraciones se puede observar un ejemplo del sencillo funcionamiento de la herramienta (figuras 2.8 y 2.9):

Figura 2.8 Inicio en FlashiXML

Figura 2.9 Resultado obtenido en FlashiXML

ReversiXML

Esta herramienta tiene un propósito completamente distinto a las demás, se centra en actos de ingeniería inversa, a partir de una implementación final como una página web en HTML, extrae el modelo UsiXML que tendría que tener asociado (ReversiXML, 2007). Abre amplias posibilidades para transformar, renovar y transportar aplicaciones ya desarrolladas.

Como resultados muestra el código en UsiXML junto con unas estadísticas sobre ambos documentos y el proceso de transformación.

2.4.2 XIML

Importantes esfuerzos académicos e industriales se han sucedido para estandartizar la representación de los diversos tipos de datos para facilitar la interoperabilidad de usos. No hay, sin embargo, ningún esfuerzo comparable apuntado a datos de interacción, los datos que se relacionan con interfaces de usuario.

Se introduce XIML (eXtensible Interface Markup Language) (XIML, 2004), una propuesta de representación común para datos de interacción. XIML tiene los siguientes principios: (1) apoya el diseño, la operación, la organización, y funciones de evaluación, (2) es capaz de relacionar los elementos de datos abstractos y concretos de un interfaz, (3) permite a sistemas basados en conocimiento explotar los datos capturados. En este papel, se introduce las características de XIML, su alcance y validación, y un camino propuesto para la adopción de la industria.

2.4.2.1 *Introducción*

La industria del software está haciendo esfuerzos para llegar a un acuerdo en el modelado, de forma que se utilice un estándar para que las aplicaciones interoperen e intercambien datos. Desde hace unos años, el mundo del industrial y académico han contribuido a la construcción de estándares para este nuevo modelo. Estos esfuerzos incluyen, entre otros, la diseminación y la adopción de un formato de representación de datos común (XML), la definición de protocolos estándar para la interoperabilidad de aplicación (SOAP).

Los beneficios de la interoperabilidad de las aplicaciones y la facilidad para el intercambio de datos entre aplicaciones, son evidentes. No solo por la integración, también por el soporte y la construcción en diversos marcos y procesos de negocio, que anteriormente no podían ser soportados.

XIML es un lenguaje basado en XML que permite la definición e interrelación entre los datos.

2.4.2.2 *Requisitos de XIML*

Para definir un mecanismo de representación para la interacción de datos, es necesario establecer los requisitos como una representación en términos de expresividad, alcance y tecnologías soporte. En la figura 2.8 se pueden observar los tipos de requerimientos que se consideran fundamentales para XIML.

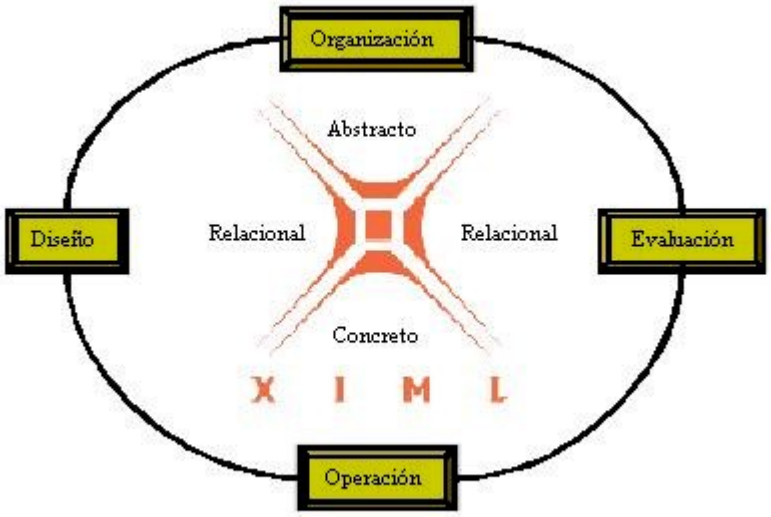


Figura 2.10 Requisitos fundamentales para XIML

- **Repositorio central de datos:** Estos repositorios de datos deben cubrir el alcance de una interfaz de usuario o de una colección de interfaces de usuario.

- **Apoyo comprensivo del ciclo de vida:** El lenguaje debe permitir el soporte durante el ciclo de vida de la interfaz de usuario. Esto incluye el diseño, la operación, y fases de evaluación.

- **Elementos abstractos y concretos:** XIML debe poder representar los aspectos abstractos de una interfaz de usuario, como pueda ser en qué contexto se interactuará, y los elementos concretos, como los componentes gráficos a mostrar.

- **Soporte relacional:** El lenguaje debe de poder relacionar los diversos elementos capturados en esta representación.

- **Tecnología soporte:** Se apoya en XML como base de XIML.

2.4.2.3 Estructura de XIML

El lenguaje XIML se apoya en dos estudios: uno es el que estudia las ontologías y sus representaciones, el otro es el trabajo en los modelos de interfaz. A continuación se puede observar las unidades de representación usadas en el lenguaje (Fig. 2.9).

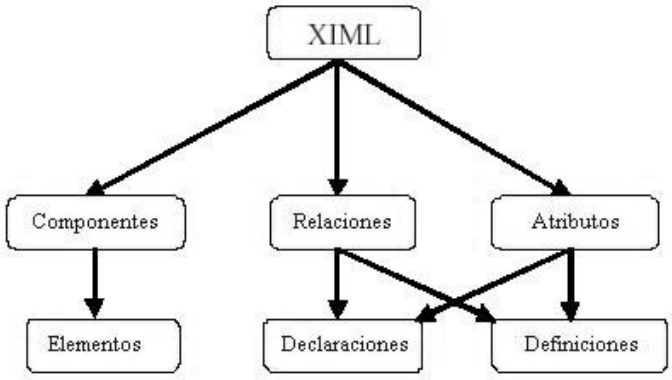


Figura 2.11 Unidades de representación usadas en XIML

Componentes: En su sentido más básico, XIML es una colección organizada de los elementos de interfaz que son clasificados en uno o varios componentes de interfaz principales. El lenguaje no limita el número y los tipos de los componentes que pueden ser definidos. En su primera versión XIML definido cinco tipos de componentes de interfaz:

- Tarea: Captura el proceso de negocio y/o las tareas de usuario que la interfaz soporta. El componente puede definir tareas y subtareas.

- Dominio: Es una colección organizada de objetos de datos y clases de los objetos que son estructurados en una jerarquía.

- Usuario: Define una jerarquía (un árbol) de usuarios. Un usuario en la jerarquía puede representar un grupo de usuarios o un usuario individual.

- Presentación: Define una jerarquía de los elementos de interacción que comprenden los objetos concretos que se comunican con usuarios en un interfaz. Ejemplos de esto podrían ser una ventana, un botón, un desplazador…

- Dialogo: Define una colección estructurada de elementos que determinan las acciones de interacción que están disponibles para los usuarios en una interfaz. Por ejemplo, un 'click'.

Relaciones: Los elementos de datos de interacción capturados por varios componentes XIML constituyen un cuerpo de conocimiento explícito sobre una interfaz de usuario. Hay, sin embargo, un cuerpo más extenso de conocimiento que está compuesto de las relaciones entre varios elementos en una especificación XIML. Una relación en XIML es una definición que une dos o más elementos XIML dentro de un componente o a través de componentes. Por ejemplo, "los datos de tipo A son mostrados con los elementos de presentación B ó los elementos de presentación C" es un enlace entre un elemento componente de dominio y un elemento componente de presentación. XIML soporta relaciones de tipo definición, que especifican la forma canónica de la relación, y relaciones de tipo declaración que especifican la instancia actual de la relación.

Atributos: Son características o propiedades de elementos a los que se les puede asignar un valor. El valor de un atributo puede ser un tipo básico de datos, o puede ser una instancia de algún elemento existente.

2.4.2.4 Validación de XIML

Para validar la expresividad y utilidad de XIML, hay que hacer una serie de pruebas. Estas actividades tienen el objetivo principal de comprobar la viabilidad de XIML para satisfacer los objetivos marcados. Las actividades de validación son: codificación a mano de la representación de interfaces, el desarrollo de interfaz multiplataforma, la dirección de interacción inteligente, la tarea del modelado y la ingeniería inversa.

2.4.3 TERESA

Para los autores de esta propuesta, el aumento de nuevos tipos de plataformas incrementa la complejidad en el diseño. Hay una necesidad de nuevos métodos y de herramientas para apoyar el desarrollo de los diversos usos que se pueden alcanzar con una variedad de dispositivos.

Esta propuesta presenta una solución (Mori et al., 2004; TERESA, 2007), basada en tres niveles de abstracción, que permite a los diseñadores centrarse en los aspectos lógicos relevantes, evitando que tengan que perder tiempo en ocuparse de los detalles.

Definen una serie de transformaciones capaces de obtener interfaces desde abstracciones, considerando las plataformas disponibles y su interacción mientras sea posible. Para apoyar las transformaciones se aporta la herramienta 'TERESA' que proporciona varios niveles de ayuda automática y varias posibilidades para adaptar tales transformaciones a las necesidades particulares.

El proceso está estructurado en tres fases principales:

En la primera fase se procede al modelado a alto nivel de las tareas de la aplicación multiplataforma. Los diseñadores desarrollan un solo modelo, que trata los contextos posibles del uso y varias plataformas implicadas, incluyendo un modelo del dominio para identificar todos los objetos que tienen que ser manipulados para realizar tareas y las relaciones entre tales objetos. El propósito de este modelo es proporcionar una descripción de las tareas apoyadas por el uso multiplataforma. Que cada tarea, indique qué plataformas pueden soportarlo y qué dependencias entre las tareas que se pueden realizar a través de diversas plataformas.

En la segunda fase, se procede con la transformación del modelo anterior para las diversas plataformas consideradas. Aquí, los diseñadores tienen que adaptar el modelo según la plataforma objetivo y, en caso de necesidad, refinar el modelo para adaptarlo a la especificidad de la plataforma elegida. Después de esto, los diseñadores transformarán el modelo original en un interfaz de usuario abstracto (en lo sucesivo, AUI, Abstract User Interface). Seguidamente, se realizará la transformación de una AUI a una CUI, siendo esta transformación totalmente dependiente de la plataforma destino.

En la tercera fase, se procede con la generación de código. La generación es iniciada partiendo de una CUI en el entorno de software elegido, este paso se puede automatizar completamente.

La herramienta TERESA está basada en transformaciones. Para representar los modelos tiene un lenguaje propio, usando el estándar XML, en el que se puede determinar los aspectos de los diferentes modelos a especificar. Utilizando las sucesivas transformaciones puede llegar a CUIs.

A continuación se puede observar el aspecto de la herramienta (Fig. 2.10), como ejemplo se muestra un modelo inicial, y en el menú se encuentran las diversas transformaciones:

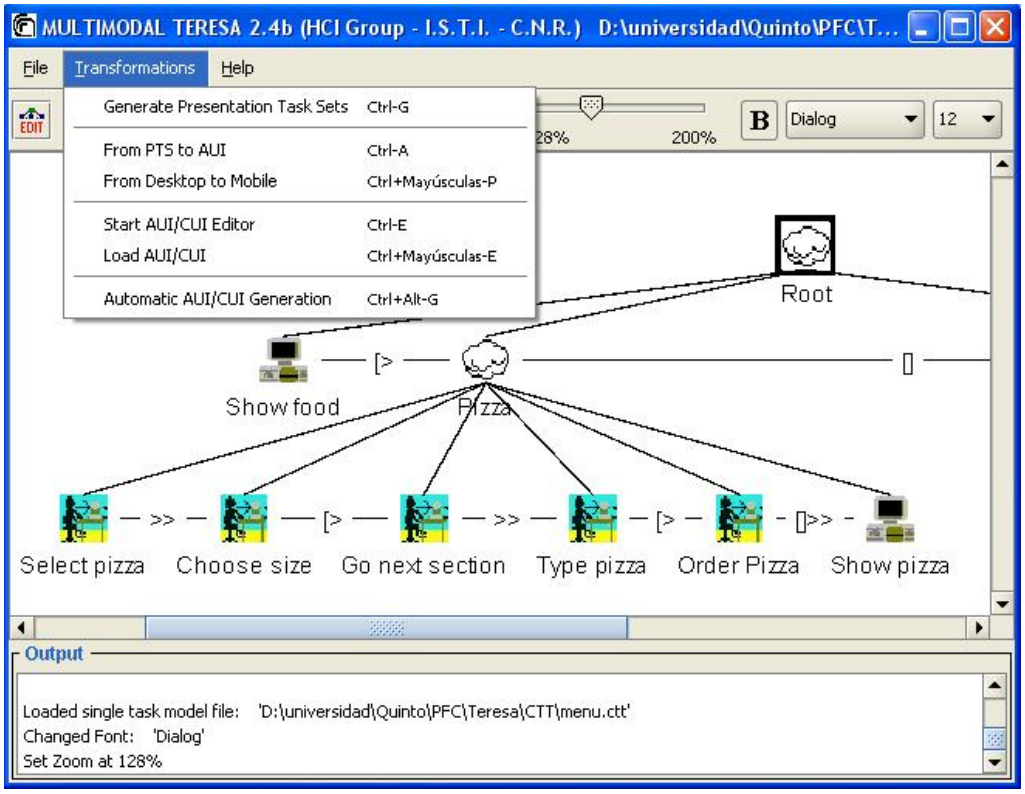


Figura 2.12 Aspecto general de la herramienta TERESA

2.4.4 Thinlet

Thinlet es una propuesta con una filosofía distinta (Thinlet, 2007). La diferencia radica en la forma de realizar las transformaciones, siendo en esta propuesta la transformación realizada por código en el sentido clásico, es decir, se utiliza un algoritmo secuencial en un lenguaje de programación imperativo, en el caso de Thinlet es el lenguaje Java el utilizado.

Separa la descripción de la interfaz de usuario y la lógica de negocio. Para especificar el modelo de la interfaz de usuario utiliza un lenguaje propio basado en XML. En la tabla 2.2 se encuentran recogidos algunos ejemplos de este lenguaje:

Componente	Código Thinlet
button	`<button text="Button" icon="image.gif" alignment="left" tooltip="ToolTip" />`
comboBox	`<combobox text="ComboBox"> <choice text="Choice" icon="image.gif" /> <choice text="Disabled" enabled="false" /> </combobox>`
checkBox	`<checkbox text="CheckBox" icon="image.gif" selected="true"/>`
textField	`<textfield text="TextField" columns="10" />`
label	`<label text="Label" icon="image.gif" alignment="center" />`

Tabla 2.2 Ejemplos de componentes en Thinlet

En la siguiente captura se puede observar una aplicación hecha con esta herramienta (Fig. 2.11), se trata de un buscador de libros consultando la base de datos del buscador amazon.

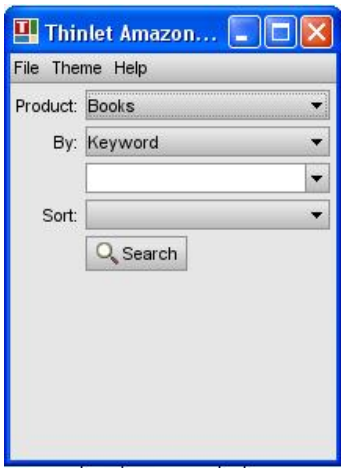


Figura 2.13 Ejemplo realizado con Thinlet

2.4.5 Laszlo

Esta propuesta (Laszlo, 2007) va en el mismo sentido que la anterior propuesta, más que una propuesta y estándar se trata de una herramienta con su lenguaje propio.

La herramienta que la sustenta, 'OpenLaszlo', es de libre distribución y sirve para crear aplicaciones web sin instalación, con las capas de un interfaz de usuario de escritorio. Los programas escritos para esta herramienta pueden ser escritos en XML o JavaScript y serán transformados de forma transparente a Flash o HTML Dinámico (DHTML), La herramienta provee de soporte para animación, contexto, datos, comunicaciones e interfaces de usuario declarativos.

El lema de los creadores de la propuesta es “escribe una vez, ejecuta donde quieras”. Una aplicación en este lenguaje puede ejecutarse en cualquier navegador web en todos los sistemas operativos comunes.

En la tabla 2.3 se muestra algún ejemplo de componentes en este lenguaje:

Componente	Código Laszlo

button	`<button height="22"enabled="false">`
	`  texto`
	`</button>`

comboBox	`<combobox width="100">`
	`  <textlistitem text="azul" selected="true"/>`
	`  <textlistitem text="rojo"/>`
	`</combobox>`

checkBox	`<checkbox text="soltero"/>`
textField	`<edittext width="200" text="text entry here"/>`
label	`<text> texto </text>`

Tabla 2.3 Ejemplos de componentes en Laszlo

A continuación, se puede observar una aplicación hecha con esta herramienta (Fig. 2.12), se trata de catálogo de música ofertado por Amazon.

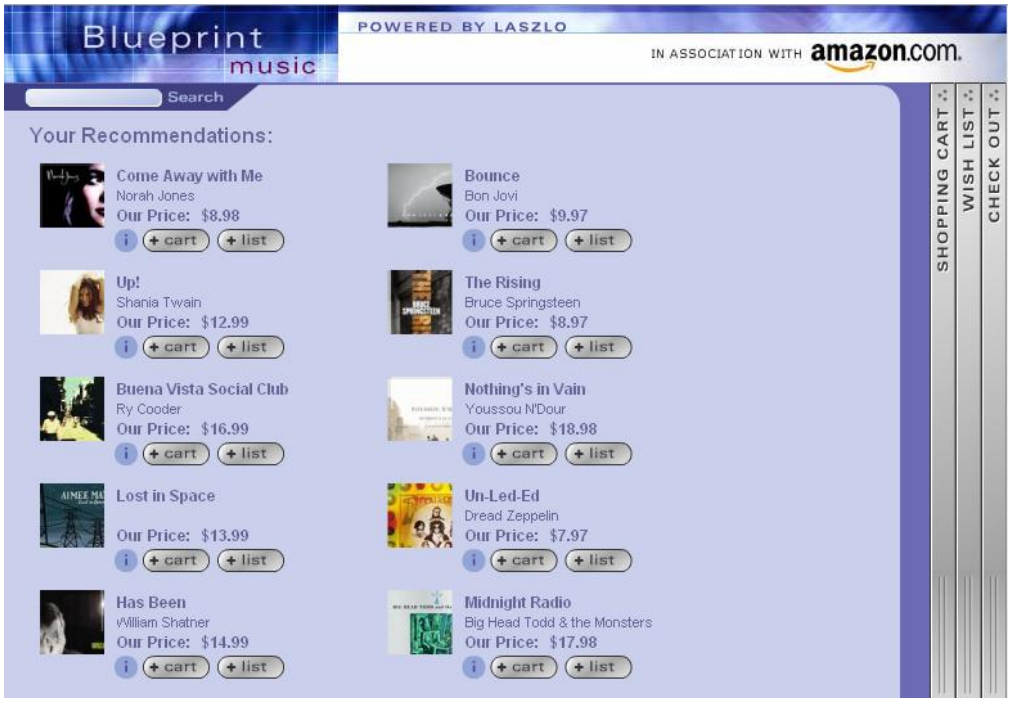


Figura 2.14 Ejemplo realizado con Laszlo

2.5 Conclusiones

La tendencia actual en el desarrollo de software, y las interfaces de usuario no son una excepción, pasa por los desarrollos dirigidos por modelos. Atendiendo a las definiciones de modelos y los desarrollos basados en modelos, cabe destacar la importancia de la recopilación de características en modelos, formando parte de un conjunto de ellos para así formar una estructura general sobre cómo se quiere que sea la interfaz de usuario y qué funcionalidad incorporará.

La arquitectura dirigida por modelos (MDA), se sitúa como un trabajo en progreso; muchas de las definiciones de MDA están en curso. En sentido estricto, la idea se apoya sobre diferentes modelos abstractos de un sistema, y las transformaciones bien definidas entre ellos. En sentido más general, la idea se apoya en modelos que varían en los diferentes niveles de abstracción y que sirven como base para las arquitecturas de software que en última instancia son implementadas por diversas tecnologías. Hoy en día, MDA es interpretado muy ampliamente, y muchas organizaciones han dado su apoyo o conformidad a MDA en sus soluciones. En definitiva MDA propone una filosofía del uso de los modelos no como simples representaciones que recojan aspectos de los sistemas software para que los siguientes desarrolladores en la cadena los interpreten, sino en la utilización de cada modelo para que genere las características más importantes en la siguiente fase de forma automática o semiautomática. A modo de ejemplo se puede citar que un diagrama de clases, denotado por ejemplo en UML, pueda ser transformado de forma automática en instancias de clases, atributos, operaciones... en algún lenguaje final como pueda ser Java, C, etc. En este sentido versará gran parte del presente trabajo, cuyos detalles e ideas sobre estas transformaciones serán expuestos en capítulos siguientes.

Para la representación de los modelos se han presentado diversos lenguajes de especificación, todos ellos basados en XML. A continuación se muestran las características más importantes de cada una:

- La propuesta UsiXML está ampliamente desarrollada y documentada, lo cual facilita enormemente su expansión, comprensión y aceptación por parte de los desarrolladores interesados en hacer algo dirigido a las nuevas tendencias en el desarrollo de software, utilizando los modelos no sólo como simples representaciones. En torno a esta propuesta están disponibles múltiples herramientas cuyo propósito suele ser centrarse en alguna parte del mecanismo de transformación. En algunos casos servirán como generadores de códigos UsiXML a partir de implementaciones o indicaciones, y en otros casos convertirán el código UsiXML en alguna implementación o paso más elaborado que el

modelo inicial. Por todo ello, ha sido elegida como la especificación a seguir para el presente Proyecto Fin de Carrera.

- La propuesta XIML, es introducida como lenguaje de representación de interfaces para el soporte universal de la funcionalidad a través del ciclo de vida de una interfaz de usuario: diseño, desarrollo, operación, dirección, organización, y evaluación. Los desarrolladores expresan que hay bastantes pruebas para apoyar la continuación de este esfuerzo a su siguiente fase, que es la diseminación de la lengua dentro del ámbito académico y comunidades de investigación de la industria.

- La propuesta TERESA abarca muy ampliamente el proceso de desarrollo de interfaces de usuario, desde los primeros pasos hasta la generación de la implementación final. La herramienta que acompaña a la propuesta funciona en todas sus fases y hace que la propuesta esté respaldada quedando demostrada la aptitud práctica. Como desventaja cabe señalar el uso de un lenguaje propio para especificar los modelos, lo que obliga al usuario potencial a aprender el lenguaje, aunque ya conozca otros de índole similar como UsiXML.

- La propuesta Thinlet es una propuesta basada en código, tiene la ventaja que para los desarrolladores que podrán crear herramientas de este tipo como están acostumbrados a hacerlo, al utilizar los lenguajes de programación imperativos, no supondrá mayor esfuerzo que plantearse esta situación de transformación de códigos, de forma similar a como pueda hacerlo un compilador. Como puntos débiles de esta propuesta cabe señalar que tiene un lenguaje muy restringido y con pocas posibilidades. La conversión que realiza es de un único modelo, y no utiliza más de un modelo y representación a diferentes niveles como otras propuestas ya mencionadas, esto puede hacerle ser menos potente.

- La propuesta Laszlo resulta interesante para el caso particular de las páginas Web, ya que al estar especializado en ello puede resultar una solución óptima para los desarrolladores web. En contrapunto, cabe destacar que la falta de generalidad, puesto que en este tipo de propuestas se espera una flexibilidad en la implementación final lo mayor posible.

En los capítulos siguientes se mostrará cómo realizar alguna de las transformaciones propuestas por la filosofía MDA, en concreto se tratará el problema de la conversión de la especificación de una interfaz de usuario a nivel concreto (CUI), representada en el lenguaje UsiXML, hasta una interfaz de usuario final (FUI).

Capítulo 3 : EL PROBLEMA. DETALLES DE IMPLEMENTACIÓN

3.1 Introducción

En el capítulo anterior se ha descrito el problema y se han introducido las bases, de manera teórica, que serán utilizadas para abordar el problema que ha ocupado la gran parte del trabajo realizado en este proyecto final de carrera. Tomando como punto de partida las ideas y propuestas comentadas anteriormente, especialmente en el capítulo segundo, este otro capítulo tendrá como objetivo presentar la forma en que se han puesto en práctica. Concretamente y en esencia, el problema al que se ha hecho frente es al de transformar una especificación de una interfaz de usuario a nivel concreto (CUI) a una implementación (descripción utilizando un lenguaje de alto nivel) de una interfaz de usuario final (FUI).

El punto de partida, la interfaz de usuario a nivel concreto en UsiXML, se asumirá que está facilitada por la herramienta GrafiXML (GrafiXML, 2007), a partir de ella se ha desarrollado un marco de trabajo soportado por una herramienta, TicXML, que permitirá ver de forma clara el reflejo en código UsiXML a nivel final.

Con lo obtenido, se podrá obtener código fuente; por lo tanto, en este capítulo, se estudiarán las particularidades e ideas de los códigos finales. Particularmente en este proyecto final de carrera hemos considerados dos tipos de lenguajes: los de texto plano como Java y los de etiquetado como HTML. Con ello demostraremos la viabilidad de nuestra forma de proceder y demostraremos de forma empírica que es posible aplicarlas tanto para lenguajes procedurales como declarativos.

Para obtener código final, es necesario someter a una serie de transformaciones el código original. Para describir cómo han sido definidas estas transformaciones, en este capítulo se presentarán las ideas básicas y la metodología seguida. El método elegido para definir las transformaciones es mediante el uso de un lenguaje de transformación, XSLT, que permite modificar archivos escritos en XML mediante reglas y plantillas hasta trasformarlos en los archivos de salida deseados.

Estos pasos se ilustran en la figura 3.1, que corresponde a la hoja de ruta de la metodología a seguir, en el transcurso del capítulo se irá recuperando e indicando en qué paso se encuentra en cada momento.

Figura 3.1 Hoja de ruta de la metodología presentada

Toda la descripción recogida en este capítulo, finalmente cristalizará en una herramienta, TicXML (*Transform in a Click usiXML*), que integrará el motor de las transformaciones con las posibles salidas de una forma sencilla y transparente al usuario.

3.2 Punto de partida: Especificación CUI en UsiXML

Como ya se presentó en el capítulo anterior, UsiXML permite la descripción de una interfaz de usuario a diversos niveles de abstracción, este proyecto final de carrera se centrará en el nivel concreto de la especificación (Fig. 2.2), en dicho nivel se especifican los componentes típicos de la UI, entre ellos se pueden encontrar algunos ampliamente utilizados como botones, campos de introducción de texto, botones de radio, etiquetas, ventanas, etc.

La forma de proceder ha sido:

Primero. De los diversos componentes de la especificación CUI (veáse tabla 2.1 y Fig. 2.2), se escogerá un subconjunto de ellos a fin de ejemplificar de forma práctica las posibilidades existentes en este campo.

Segundo. Los componentes tienen una serie de propiedades de las cuales se escogerán un subconjunto representativo de ellas, cabe destacar que no todas las propiedades podrán ser extrapolables a todas las implementaciones finales, ya que las implementaciones finales tendrán una serie de restricciones propias.

Tercero. Aunque el presente proyecto final de carrera se centre en el nivel CUI, una especificación en UsiXML consta de más modelos. A continuación se muestra el esquema general de una especificación en UsiXML:

```
<uiModel …>
    <auiModel …>
     …
    </auiModel>

    <cuiModel …>
     "componentes"
    </cuiModel>

   <contextModel …>
    </contextModel>
</uiModel>
```

Se pueden observar los tres modelos principales que tiene una especificación UsiXML: AUI (Abstract User Interface, Interfaz de Usuario a nivel Abstracto), CUI (Concrete User Interface, Interfaz de Usuario a nivel Concreto) y el modelo de contexto. Como ya se ha dicho, el modelo CUI será el punto de partida y el que realmente será de interés para este proyecto.

Para generar las especificaciones concretas de las que partiremos en este proyecto se utilizará la herramienta GrafiXML (Fig. 3.1), presentada en el capítulo anterior, otra opción válida es usar ficheros UsiXML ya existentes o, aunque menos recomendable, podemos crear especificaciones propias creando un fichero de texto (Fig. 3.5) atendiendo a la sintaxis de UsiXML. Otra solución interesante es la utilización de ambas técnicas: obtener el modelo de GrafiXML y personalizarlo (nombres de variables, eliminación/adición de etiquetas...) en un editor de texto, esta última opción será la que se utilice en este proyecto. El reflejo del punto de partida en la hoja de ruta se puede observar resaltado en las figuras 3.2 y 3.4, donde se separa en dos pasos básicos este punto de partida:

Figura 3.2 Hoja de ruta: Definición del modelo

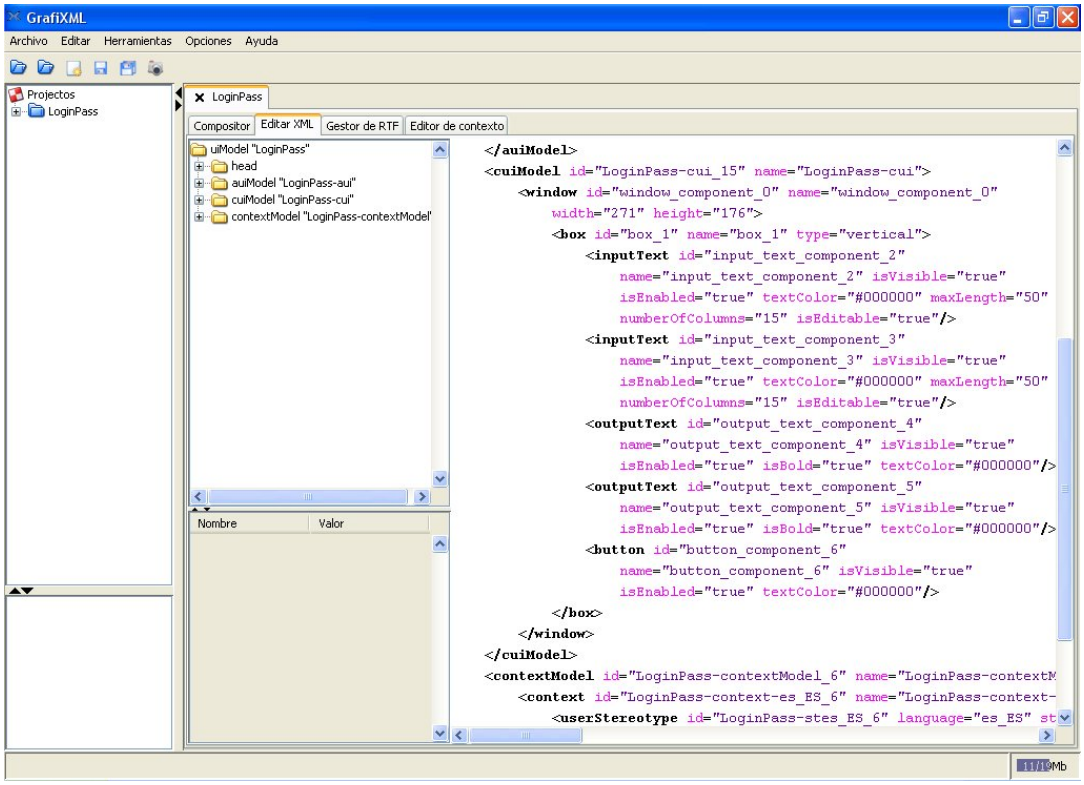


Figura 3.3 Fichero UsiXML obtenido en la herramienta GrafiXML

En este punto se obtendrá la especificación CUI de UsiXML en un fichero de texto, como ya se ha señalado, es posible modificarlo y personalizarlo mediante un editor de texto (Fig. 3.5).

Figura 3.4 Hoja de ruta: Especificación CUI-UsiXML

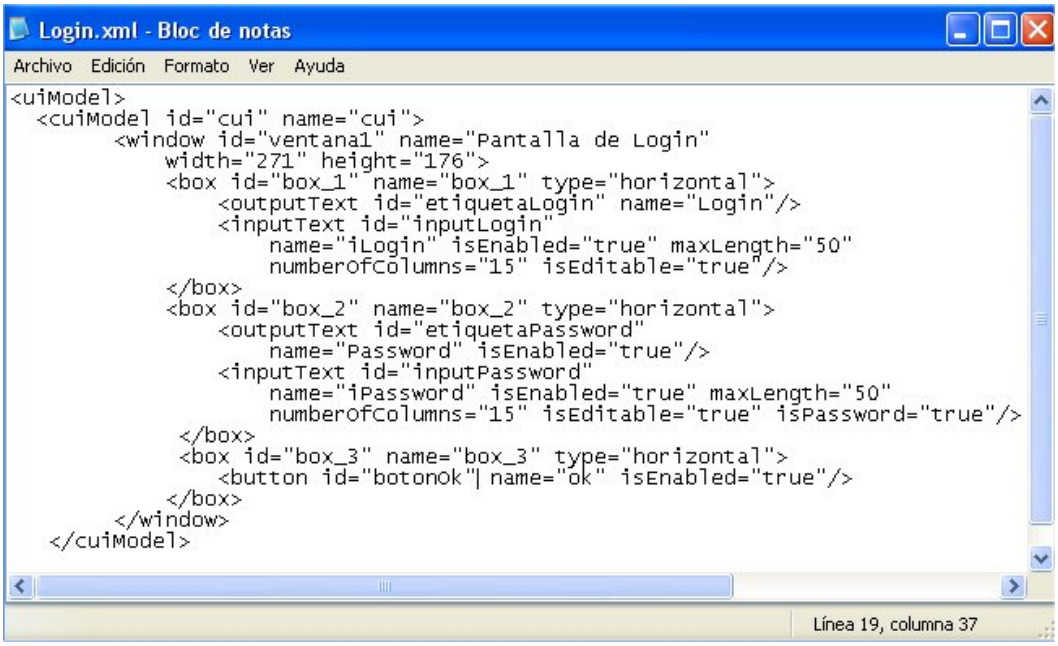


Figura 3.5 Fichero UsiXML escrito en un editor de texto

3.3 El objetivo de una interfaz de usuario final

El objetivo de la transformación será obtener una interfaz de usuario final, es decir, una descripción de la interfaz donde se utilice un lenguaje de alto nivel de abstracción y donde fruto de su compilación o interpretación se pueda lograr visualizar la interfaz de usuario asociada. Para ello y previamente es conveniente estudiar qué partes componen dicha interfaz.

De forma genérica, una interfaz de usuario gráfica está compuesta de diferentes objetos manipulables por el usuario. Una buena forma de visualizar qué componentes ofrece un lenguaje determinado es observar un IDE (Integrated Development Enviroment, Entorno de Desarrollo Integrado) asociado a dicho lenguaje de programación, tarea sencilla si estos entornos se encuentran disponibles, o en otro caso visualizar diversas implementaciones en las que se pueda observar los componentes que la forman.

Seguidamente se comentan los lenguajes de las implementación final, elegidos para describir una interfaz de usuario en este proyecto final de carrera, como ya se dijo anteriormente, se tratará de los lenguajes Java y HTML.

3.3.1 Interfaces de usuario en Java

En primer lugar se estudiará el lenguaje Java (Eckel 2002; Java, 2007; JavaAlmanac, 2007). Dicho lenguaje de programación es orientado a objetos y fue desarrollado por Sun Microsystems a principios de los 90. Es uno de los lenguajes más utilizados actualmente. La paleta de componentes (Fig. 3.6) disponible en el entorno de programación NetBeans (NetBeans, 2007), un IDE de libre distribución para Java.

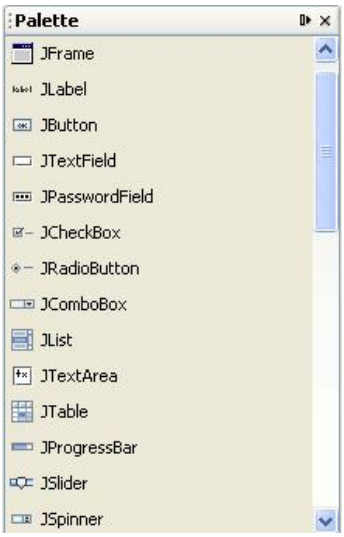


Figura 3.6 Componentes gráficos en Java

En Java, típicamente, se utiliza una función llamada constructor que permite crear instancias de objetos, con los métodos (funciones) se puede personalizar estas instancias, accediendo a los atributos asociados a cada clase de objetos. En ocasiones, en el propio constructor se pueden agregar parámetros que modifiquen la instancia del objeto creado. En la tabla 3.1 se recoge, con más detalle, el análisis llevado a cabo sobre cada uno de los componentes disponibles en Java Swing, así como su reflejo en código con el constructor y los métodos utilizados y una breve descripción de cada componente.

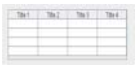
Componente	Métodos utilizados	Descripción	Imagen
JFrame	*Constructor:* `JFrame(String texto)` *Métodos:* `setSize(int x,int y)`	Ventana contenedora de otro tipo de componentes, también llamado caja de diálogo.	
JLabel	*Constructor:* `JLabel(String texto)`	Etiqueta de texto, suele describir otros componentes de la interfaz.	
JButton	*Constructor:* `JButton()` *Métodos:* `setText(String text)` `setEnabled(boolean op)`	Un botón que suele servir para desencadenar alguna acción.	
JTextField	*Constructor:* `JTextField()` *Métodos:* `setColumns(int c)` `setEnabled(boolean op)`	Campo de introducción de texto, suele utilizarse para introducir datos.	
JPasswordField	*Constructor:* `JPasswordField()` *Métodos:* `setColumns(int c)` `setEnabled(boolean op)`	Campo de introducción de texto, lo que introduce el usuario es ocultado.	
JCheckBox	*Constructor:* `JCheckbox()` *Métodos:* `setText(String txt)` `setEnabled(boolean op)`	Marcado de opciones, función marcar/desmarcar.	
JRadioButton	*Constructor:* `JRadiobutton()` *Métodos:* `setText(String txt)` `setEnabled(boolean op)`	Botón de radio que suele formar parte de una agrupación de varios en forma de opciones.	

Componente	Métodos utilizados	Descripción	Imagen

JComboBox	*Constructor:*
	`JComboBox(String *opciones)`
	Métodos:
	`setEnabled(boolean op)`
		Cuando se pulsa en este campo se despliegan una serie de opciones, una siempre estará seleccionada.	

JList	*Constructor:*
	`JList(String *opcs)`
	Métodos:
	`setEnabled(boolean op)`
	`setSelectionMode(Mode m)`
		Similar al anterior, salvo que se muestran varias opciones al a vez y es posible la selección múltiple.	

JTextArea	*Constructor:* `JTextArea()`
	Métodos:
	`setColumns(int c)`
	`setRows(int r)`
		Área de texto, comúnmente utilizada para recoger cantidades grandes de texto.	

JTable	*Constructor:* `JTable(int x, int y)` *Métodos:* `setEnabled(bolean op)`	Un componente tabla, útil para mostrar o introducir datos.	

JProgressBar	*Constructor:*
	`JProgressBar()`
	Métodos:
	`setMinimum(int min)`
	`setMaximum(int max)`
	`setOrientation(Orient o)`
	`setIndeterminate(boolean b)`
		Barra de progreso utilizada para informar al usuario del progreso de algunas operaciones como por ejemplo una carga.	

JSlider	*Constructor:*
	`JSlider(Orientation or ,int min, int max)`
	Métodos:
	`setEnabled(boolean b)`
		Deslizador utilizado para que el usuario manipule directamente datos, como por ejemplo porcentajes.	

JSpinner	*Constructor:* `JSpinner()` *Métodos:* `setEnabled(boolean b)`	Campo de introducción de texto acompañado de botones de modificación del valor numérico.	

Tabla 3.1 Componentes gráficos en Java

Instancias de estos componentes pueden ser alojados en una clase, formando una interfaz de usuario, la disposición del esqueleto de la clase no resulta muy compleja, basta con incluir las librerías necesarias y en la función de ejecución una llamada a una función que muestre los componentes. Podría ser algo así:

```
/* Importar librerías… */

public class Main
{
   public Main(){}

   public static void main(String[] args)
   {
       /* Aquí los componentes generados en la transformación */

       /* Hacer visible la ventana o popup generado en la especificación */
       variable_ventana_generada.setVisible(true);
   }
}
```

Otras características interesantes que se pueden agregar son: manejadores de eventos, ejemplos de esos manejadores son aquellos con los que es posible la terminación de la ejecución cuando se cierra una ventana mostrada, o aquellos que facilitan hacer llamadas a funciones para que no se permita redimensionar las ventanas, etc. . Aunque no son imprescindibles, se consideran como opciones de optimización muy interesantes, muchas de ellas han sido incluidas en el proyecto.

3.3.2 Interfaces de usuario en HTML

El lenguaje de marcación HTML (HTML, 2007), fue diseñado para estructurar textos y presentarlos en forma de hipertexto, que es el formato estándar de las páginas web. HTML tiene las características propias de un lenguaje de marcas, tiene unas marcas definidas, las cuales tienen una serie de atributos, las marcas pueden incluir a otras marcas formando jerarquías.

Los diversos componentes estarán formados por etiquetas con sus correspondientes atributos. Para ver los componentes existentes en HTML se ha consultado la referencia web de la bibliografía donde se detallan los componentes básicos, otra opción habría sido consultar algún editor como Dreamweaver o Frontpage. En la tabla 3.2 se muestra un resumen de los componentes disponibles en una interfaz de usuario cuando se utiliza HTML, en dicha tabla se ha omitido la descripción de los componentes ya que son similares a las descritas en la tabla anterior.

Componente	Etiqueta y atributos	Imagen
Etiqueta	"Nada", se sitúa el texto entre las etiquetas	Etiqueta
Botón	*Etiqueta:* `<button>"nombre"</button>` *Atributos:* `type, disabled`	ok
Introducción texto	*Etiqueta:* `<input>` *Atributos:* `readonly, disabled, name, size, maxlength`	
Introducción password	*Etiqueta:* `<input type="password">` *Atributos:* `readonly, disabled, name, size, maxlength`	
CheckBox	*Etiqueta:* `<input type="checkbox">` *Atributos:* `disabled, name`	Opcion
RadioButton	*Etiqueta:* `<input type="radio">` *Atributos:* `disabled, name`	Opcion
ComboBox	*Etiqueta:* `<select>` `<option>"op"</opction> </select>` *Atributos:* `disabled, name`	Opcion 1
ListBox	*Etiqueta:* `<select>` `<option>"op"</opction> </select>` *Atributos:* `disabled, name, size`	Opcion1 Opcion2 Opcion3
TextArea	*Etiqueta:* `<textarea>` *Atributos:* `name, rows, cols, readonly, disabled`	
Tabla	*Etiqueta:* `<table> … </table>` *Atributos:* `border, width`	

Tabla 3.2 Componentes gráficos en HTML

De la misma manera que en Java, en HTML existe un "esqueleto" de programa, que es el siguiente. La particularidad de este esqueleto es que la base para la elaboración de cualquier página web como en el caso del apartado anterior para la definición lo era para la descripción de una clase en Java:

```
<HTML>

<HEAD> <TITLE> Titulo, cogido de la especificación CUI </TITLE>
</HEAD>

<BODY>
  Aquí los componentes generados en la transformación
</BODY>

</HTML>
```

Las especificaciones y particularidades mostradas en estos apartados por los anteriores lenguajes, serán el objetivo de las transformaciones que sean necesarias realizar para obtener una configuración final de los componentes de cada lenguaje. En el siguiente apartado se enseñará qué procedimientos se seguirán para conseguir esta transformación.

En este punto, cabe retomar la hoja de ruta para observar el objetivo a donde se ha llegado (Fig. 3.7):

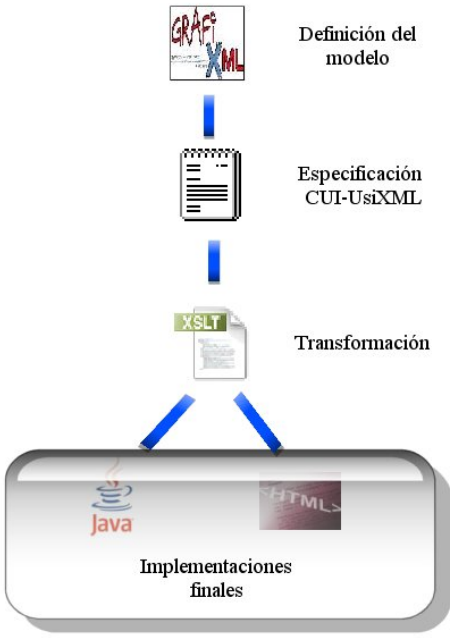


Figura 3.7 Hoja de ruta: Implementaciones finales

3.4 Transformaciones de CUI a FUI

La siguiente tarea a realizar será la de la transformación propiamente dicha. La hoja de ruta en este momento corresponde a la siguiente figura:

Figura 3. 8 Hoja de ruta: Transformación

Partiendo de una especificación en UsiXML habrá que tener en cuenta cómo está estructurada la plataforma destino para realizar una transformación óptima. En este punto cabe destacar que la mayor diferencia será cómo se estructura la implementación final, si se trata de un lenguaje basado en texto plano como Java, o si se trata de un lenguaje etiquetado como HTML.

Como primer paso se necesitará saber cuál es el prototipo de programa en el lenguaje final, es decir, el esqueleto básico que se mostró en los anteriores apartados. Este esqueleto será colocado de forma fija en la transformación final.

Una vez que se tenga el esqueleto de programa, se puede proceder a la introducción de componentes de interfaz de usuario. Los componentes se irán introduciendo en el orden que la especificación origen determine. En cuanto a cómo colocar los componentes se ha seguido la norma de incluir cualquier componente gráfico (botón, etiqueta, etc.) en un componente contenedor que indique la disposición de sus componentes gráficos, básicamente se han considerado dos disposiciones en los contenedores, horizontal y vertical. Con la combinación de ambas disposiciones se puede dar cobertura a la gran mayoría de interfaces de usuario existentes. En la figura 3.9 se muestra un ejemplo de disposiciones en una interfaz conocida como es la del buscador google. Si por alguna circunstancia los componentes gráficos no se encontrarán en ningún contenedor, se situarán de forma horizontal formando una serie. Las especificaciones completas acerca de estas circunstancias, y de las demás, vendrán dadas por el fichero origen UsiXML.

Figura 3.9 Disposiciones horizontal y vertical en el buscado google

El código correspondiente a los componentes vendrá condicionado por la elección de la forma más cómoda, en términos de ahorro de código, de adaptación con el código original y de simplicidad a la hora de visualizarlo. Se definen unos componentes genéricos, éstos se obtendrán en las implementaciones finales por medio de componentes directos que ya existan o, en su caso, se conseguirán por medio de uso de la combinación de otros componentes.

En la tabla 3.3 se observa en forma de código de los componentes en el origen y los objetivos. En esta tabla, por razones de simplicidad, se ha omitido algún componente común a todos los componentes, por ejemplo la propiedad Enabled o Accesible, en los lenguajes finales se podrá indicar esta característica mediante alguna función, por ejemplo en Java variable.setEnabled, o el atributo disabled en el caso de HTML.

Componente	Código UsiXML	Código Java	Código HTML
Ventana	<window id="w0" name="w_0" width="400" height="350"> *"contenido"* </window>	Jframe w0 = new JFrame(new String("w_0")); ventanaRegistro.setSize(40,30);	<HEAD> <TITLE>w_0</TITLE> </HEAD>
Box	<box id="b1" name="b_1" type="vertical"> "*contenido*" </box>	Box b1 = new Box(BoxLayout.Y_AXIS);	<tr align="center"> "contenido" </tr>
Etiqueta	<outputText id="o1" name="o_1" isEnabled="true"/>	JLabel o1 = new JLabel("o_1");	o_1
Botón	<button id="b1" name="b_1" isVisible="true" isEnabled="true"/>	JButton b1 = new JButton(); b1.setText("b_1");	<button type="button">b_1 </button>
Introducción texto	<inputText id="i1" name="i_1" isEnabled="true" numberOfColumns="15" maxLength="50" isEditable="true"/>	JTextField i1 = new JTextField(); i1.setColumns(15);	<input name="i1" size="15" maxlength="50"/>
Introducción password	<inputText id="i1" name="i_1" isEnabled="true" isPassword="true" numberOfColumns="15" maxLength="50" isEditable="true"/>	JPasswordField i1 = new JPasswordField(); i1.setColumns(15);	<input type="password" name="i1" size="15" maxlength="50"/>
CheckBox	<checkBox id="c3" name="c_3" isEnabled="true"/>	JcheckBox c3 = new JCheckBox(); c3.setText("c_3");	<input type="checkbox" name="c3">c_3 </input>
RadioButton	<radioButton id="r1" name="r_1" isEnabled="true"/>	JRadioButton r1 = new JRadioButton(); r1.setText("r_1");	<input type="radio" name="r1">r_1 </input>
ComboBox	<comboBox id="c1 name="c1" isEnabled="true"/>	String[] opciones {"o1"}; JComboBox c1 = new JComboBox(opciones);	<select name="c1"> <option>o1</opction> </select>

Tabla 3.3 (1/2) Componentes en UsiXML, Java y HTML

Componente	Código UsiXML	Código Java	Código HTML
ListBox	<listBox id="l1" name="l_1" isEnabled="true" multiple_selection= "false"/>	String[] opciones = {"o1"}; JList l1 = new JList(opciones); l1.setSelectionMode(0);	<select name="l1" size="3"> <option>o1</option> </select>
TextArea	<inputText id="i1" name="i_1" isEnabled="true" numberOfColumns="15" numberOfLines="2" isEditable="true"/>	JTextArea i1 = new JTextArea(); i1.setRows(2); i1.setColumns(15);	<textarea name="i1" rows="2" cols="15"/>
Tabla	<table id="t3" name="t_3" isEnabled="true" xSize="2" ySize="2"/>	JTable t3 = new JTable(2,2);	<table border="1" width="200"> <tr><td>.....</td> <td>..... </td></tr> <tr><td>.....</td> <td>..... </td></tr> </table>
Spin	<spin id="s0" name="s_0" isEnabled="true" orientation="horizo ntal"/>	JSpinner s0 = new JSpinner();	<input name="s0" size="1"> <button type="button">+ </button> <button type="button">- </button>
Barra de progreso	<progressionBar id="p3" name="p_3" isEnabled="true" orientation="horizo ntal" minValue="0" maxValue="100"/>	JProgressBar p3 = new JProgressBar(); p3.setMinimum(0); p3.setMaximum(100); p3.setOrientation(…);	-
Deslizador	<slider id="s1" name="s_1" isEnabled="true orientation="horizo ntal" minValue="0" maxValue="100">	JSlider s1 = new JSlider(JSlider.Horizont al,0,100);	-

Tabla 3.3 (2/2) Componentes en UsiXML, Java y HTML

En este contexto y con los objetivos descritos, se pueden estudiar las características comunes que se utilizan en las diversas implementaciones finales. Para visualizarlo de forma gráfica (Fig. 3.10 y Fig. 3.11) se ha escogido el ejemplo significativo de un botón.

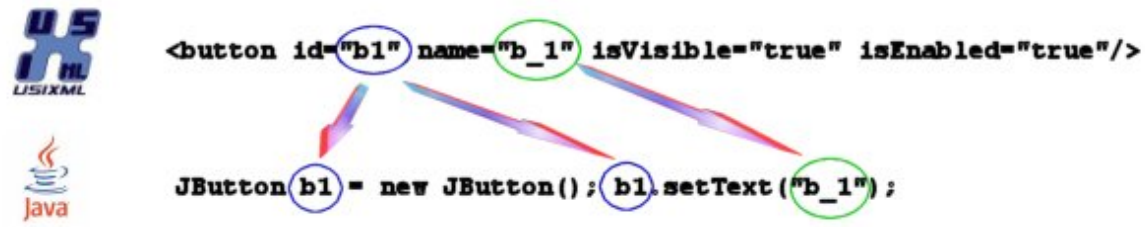


Figura 3.10 Transformación de UsiXML a Java

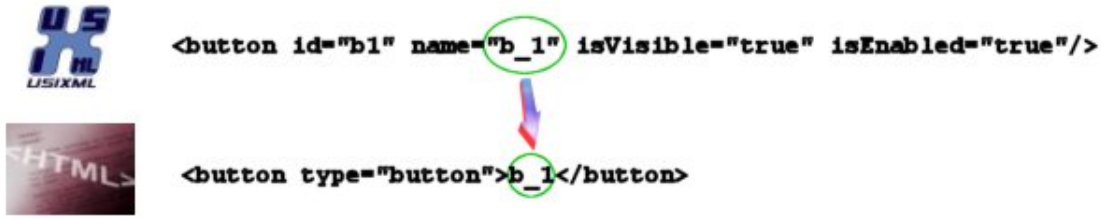


Figura 3. 11 Transformación de UsiXML a HTML

En este ejemplo se refleja que recolocando adecuadamente valores de entrada como el nombre o el identificador, se puede crear un objeto de ese tipo en otro lenguaje final. Dado que las características de los componentes de un lenguaje a otro son similares, la problemática es la de colocarlos adecuadamente. En consecuencia, la idea de estas transformaciones radica en partes de código fijas combinadas con atributos extraídos de la especificación original en UsiXML.

Algunas de las transformaciones resultarán algo más complicadas, en algunos casos la existencia de un atributo en un mismo componente puede dar como resultado una componente final distinta, por ejemplo, si un componente InputText de UsiXML tiene el atributo isPassword con valor 'verdadero' dará como consecuencia un componente final diferente. Por ello, se tienen que hacer diferenciaciones en las transformaciones y no será suficiente con la reescritura de los parámetros de entrada. En estos, casos será necesario tener en cuenta estos valores para dar lugar a diferentes componentes finales, esto se puede ver en las figuras 3.12 y 3.13, en el primer caso obtendremos un componente JTextField y en el segundo un componente de tipo JPasswordField.

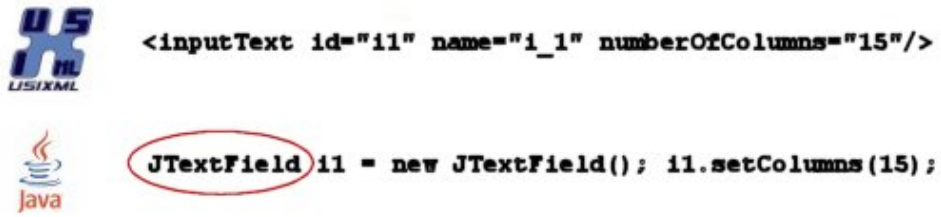


Figura 3.12 Transformaciones: Diversas implementaciones finales (1)

Figura 3. 13 Transformaciones: Diversas implementaciones finales (2)

En otros casos, la aparición de un atributo hará que haya que incluir algún código que refleje esa característica, aunque esto se tomará con carácter opcional. Por ejemplo, para cualquier componente gráfico, es posible que el valor del atributo enabled de la especificación original en UsiXML sea falso, en ese caso en las implementaciones finales habrá que incluir un trozo de código que refleje esto. Para mostrar esta característica se ha modificado los ejemplos mostrados en las figuras 3.10 y 3.11, el resultado de dichas modificaciones se puede observar en las figuras 3.14 y 3.15.

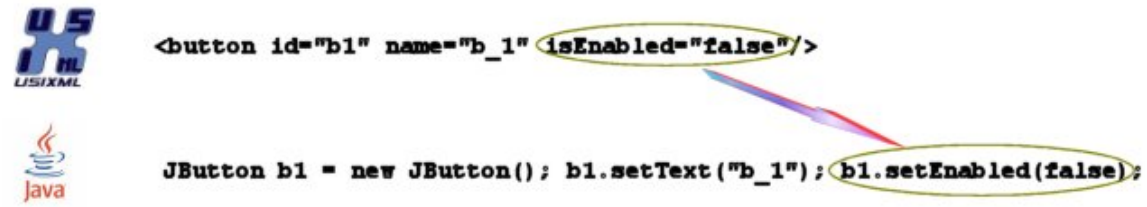


Figura 3.14 Transformación de UsiXML a Java

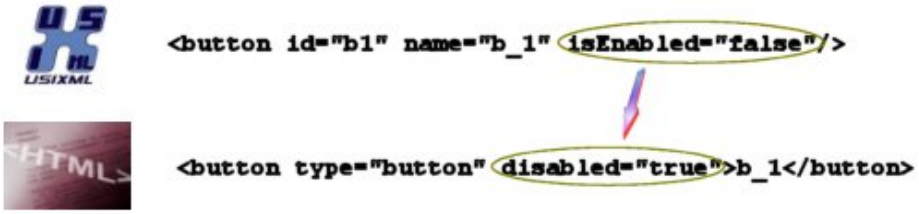


Figura 3.15 Transformación de UsiXML a HTML

En el caso de los contenedores gráficos, hay que tener en cuenta factores como su tamaño, orientación, componentes gráficos que contendrán, etc. Las características de forma y tamaño se podrán reflejar de una manera similar a la expuesta anteriormente, mediante atributos y/o funciones. Sobre los

componentes que contendrán se aprovecha la característica del lenguaje etiquetado, es decir, un componente de tipo contenedor contendrá uno o varios componentes gráficos como se muestra a continuación:

```
<Contenedor … … …>
      <Componente_gráfico1 …/>
      …
      <Componente_gráficoN …/>
</Contenedor>
```

A la hora de reflejar este comportamiento en las especificaciones finales hay que tener en cuenta el tipo de codificación, es decir, si se trata de un lenguaje de texto como Java o se trata de un lenguaje etiquetado como HTML. En el primer caso la variable que controle el contenedor se encargará de ir agregando los componentes que contenga de alguna manera, por ejemplo usando alguna función añadirComponenteGráfico o similar, en el segundo caso se podrá aprovechar la circunstancia de tener el mismo tipo de codificación del origen, esto es, estructurar de forma jerárquica de la misma manera (si es posible) que está hecho en el original. En las figuras 3.16 y 3.17 se muestra un ejemplo relacionado con lo expuesto.

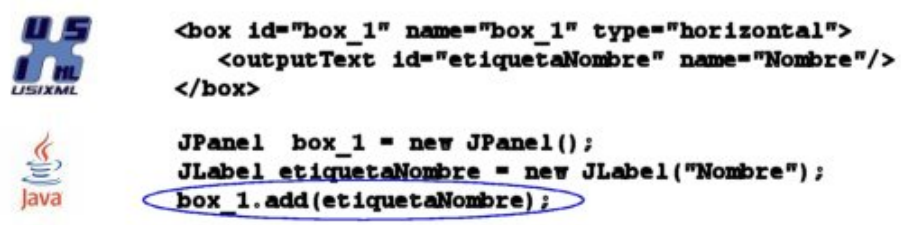


Figura 3.16 Adición de componentes a contenedores en Java

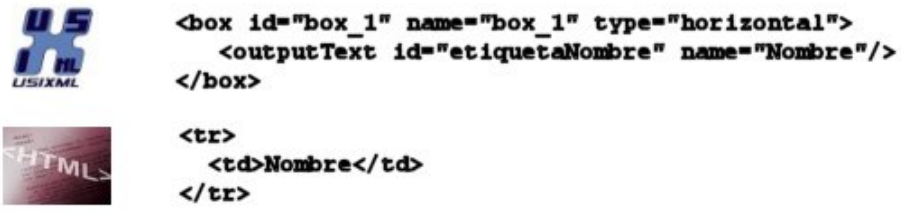


Figura 3.17 Adición de componentes a contenedores en HTML

Para implementar las transformaciones se ha utilizado el lenguaje XSLT, cuyas principales características se detallan en el siguiente apartado.

3.5 Transformaciones XSLT

XSLT o XSL Transformaciones (XSLT, 2007) es un estándar de la organización W3C que presenta una forma de transformar documentos XML en otros e incluso a formatos que no son XML. Las hojas de estilo XSLT realizan la transformación del documento utilizando una o varias reglas de plantilla: unidas al documento fuente a transformar, esas reglas de plantilla alimentan a un procesador de XSLT, el cual realiza las transformaciones deseadas colocando el resultado en un archivo de salida o, como en el caso de una página web, directamente en un dispositivo de presentación, como el monitor de un usuario. Actualmente, XSLT es muy utilizado ya que permite separar contenido y presentación, aumentando así la productividad.

El lenguaje XSLT es de naturaleza declarativa, el orden en el que se sitúen las plantillas, reglas, etc. no es determinante. La forma de proceder del procesador XSLT es la de ir buscando patrones o plantillas que encajen con las definidas en el archivo que determina la transformación.

El lenguaje XSLT tiene sus propias etiquetas, las hay que se utilizan para definir plantillas y hacer llamadas a las mismas, otras para emular estructuras condicionales, de selección, etc. también para funciones como aplicar atributos del fichero XML original. De esta manera, se puede combinar la utilización del archivo origen en UsiXML con un archivo de tipo XSL que haga las veces de plantilla de transformación, para obtener el deseado fichero de la implementación final. De las transformaciones, se pueden obtener dos tipos de ficheros: texto y xml (etiquetado). La forma de realizar transformaciones será totalmente diferente, ya que al estar XSLT basado en XML el texto ha de introducirse mediante etiquetas, es decir, habrá que indicar expresamente que se desea introducir texto. En caso de tener como resultado un archivo etiquetado, se podrán incluir las etiquetas de forma normal.

En el presente trabajo se han abordado las dos casuísticas, con Java y HTML respectivamente.

A continuación se puede observar la estructura de un documento XSLT, donde se puede observar la plantilla raíz, que es la primera que se ejecutará y que a su vez puede ir aplicando las plantillas que se deseen o aplicando cualquier regla.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0">

  <xsl:template match='/'>
     …
  </xsl:template>
</xsl:stylesheet>
```

Algunas de las etiquetas más importantes utilizadas en XSLT se encuentran recogidas en la tabla 3.4, junto con una breve descripción de su uso.

Etiqueta	Ejemplo	Descripción
Text	`<xsl:text>import </xsl:text>`	Introduce texto, necesario cuando el archivo de salida sea en modo texto.
Template	`<xsl:template match="/"> .... </xsl:template>`	Las plantillas hacen las veces de funciones.
Call-Template	`<xsl:call-template name="button"/>`	Para hacer llamadas a plantillas.
Value-Of	`<xsl:value-of select="@n"/>`	Inserta el valor de un atributo del origen.
If	`<xsl:if test = "@n ='1'"> <xsl:text>si</xsl:text> </xsl:if>`	Sentencia condicional, se realiza lo que contiene si la condición es cierta.
For-each	`<xsl:for-each select="box"> ... </xsl:for-each>`	Hace la acción contenida para cada atributo que cumpla la condición. Especialmente útil en archivos XML ya que podrá existir una o múltiples ocurrencias de una misma etiqueta.
Param	`<xsl:param name="n" />`	Para recoger parámetros en las plantillas.
With-Param	`<xsl:with-param name="n">5</xsl:with-param>`	Para pasar parámetros a las plantillas.

Tabla 3.4 Etiquetas importantes en XSLT

Por tanto, habrá que definir tantos ficheros de transformación como implementaciones finales se deseen. Para llevar a cabo las transformaciones es necesario un procesador de XSLT, para el presente trabajo se ha utilizado el programa 'Stylus Studio 2006 XML Enterprise Edition' (Fig. 3.18). Este tipo de programas ayudan al trabajo con XML y XSLT, ofreciendo la posibilidad de visualizar los documentos en forma de árbol, esquema y texto. Además es posible definir un escenario para una transformación XSLT, en el que se especificará la

ruta del archivo XML origen, junto con otros parámetros opcionales como ficheros de salida, una vez realizado el escenario el programa irá guiando la transformación mostrando el nombre de los atributos a aplicar, o las etiquetas, etc. de forma similar a como lo hace un IDE moderno de cualquier lenguaje de programación. Una vez esté definido todo esto, es posible ejecutar y ver el resultado de diversas formas, en texto, html o árbol.

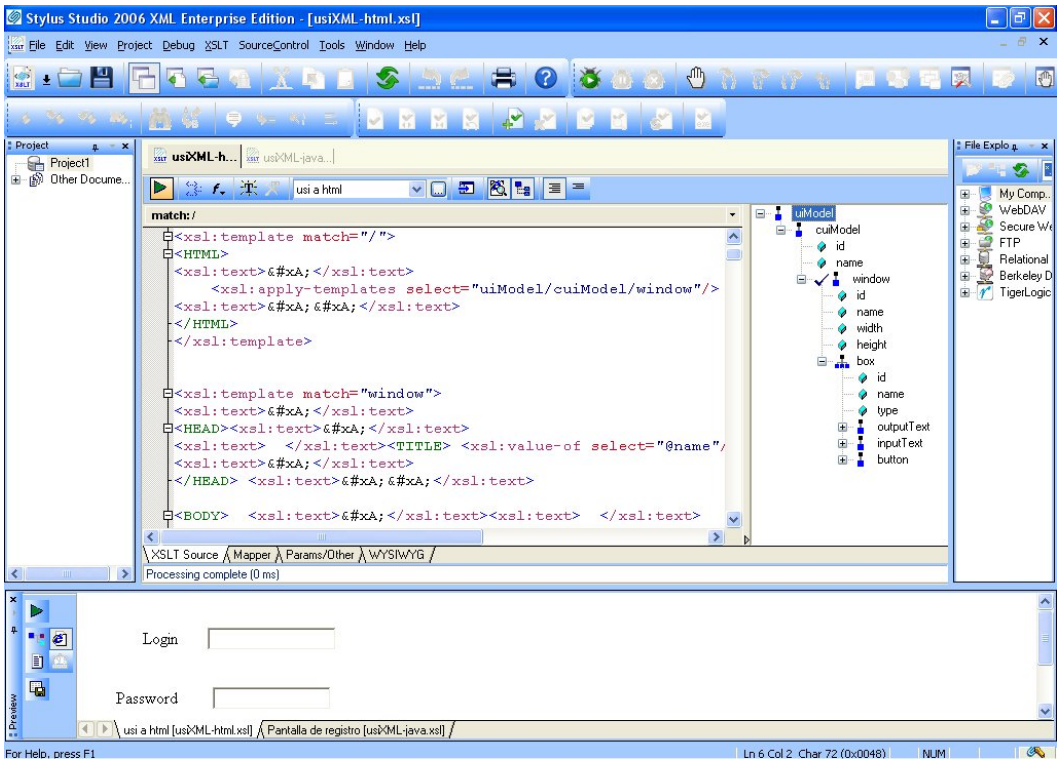


Figura 3.18 Herramienta Stylus Studio 2006 XML Enterprise Edition

Una vez se han definido y probado las diversas transformaciones, se procede a la construcción de una herramienta que permita seleccionar y visualizar los ficheros, transformarlos en implementaciones finales y ejecutarlos de forma transparente al usuario. La herramienta se llama TicXML y se describirá en detalle en el siguiente apartado.

3.6 La herramienta TicXML

La herramienta TicXML (Transform In a Click usiXML) se ha construido utilizando el lenguaje de programación Java, para ello se ha utilizado el IDE de libre distribución NetBeans (NetBeans, 2007). La idea preliminar a la hora de desarrollar esta esta herramienta es facilitar una aplicación visual e intuitiva en forma de asistente o wizard.

En este punto conviene retomar el contexto de las herramientas en UsiXML y el punto que la nueva herramienta ocupa en este organigrama. En la figura 3.19 se puede ver la herramienta TicXML de forma destacada.

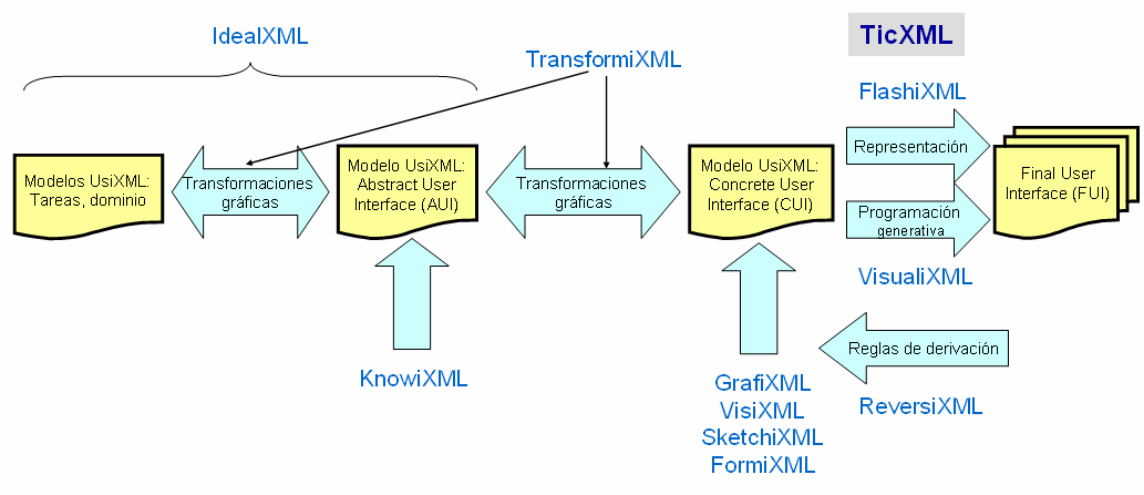


Figura 3.19 Herramientas en UsiXML, incorporación de TicXML

TicXML (Fig. 3.20) permite la selección del archivo con la especificación UsiXML mediante un asistente con en el que se podrá ir navegando por el sistema de ficheros del usuario hasta la selección del archivo, de forma análoga se procede con el directorio destino, donde se alojará el fichero resultante, por defecto utilizará el mismo directorio que en el que se encuentre la especificación origen.

Una vez se haya definido el archivo origen, se podrá visualizar el contenido en forma de árbol, por considerarse una forma cómoda para visualizar este tipo de archivos (Fig. 3.21). También se ofrece la posibilidad de cambiar la implementación destino, por defecto estará marcada la opción de Java. Finalmente se podrá realizar la transformación pulsando sobre el botón 'Siguiente'. El aspecto inicial de TicXML se puede observar en la siguiente figura:

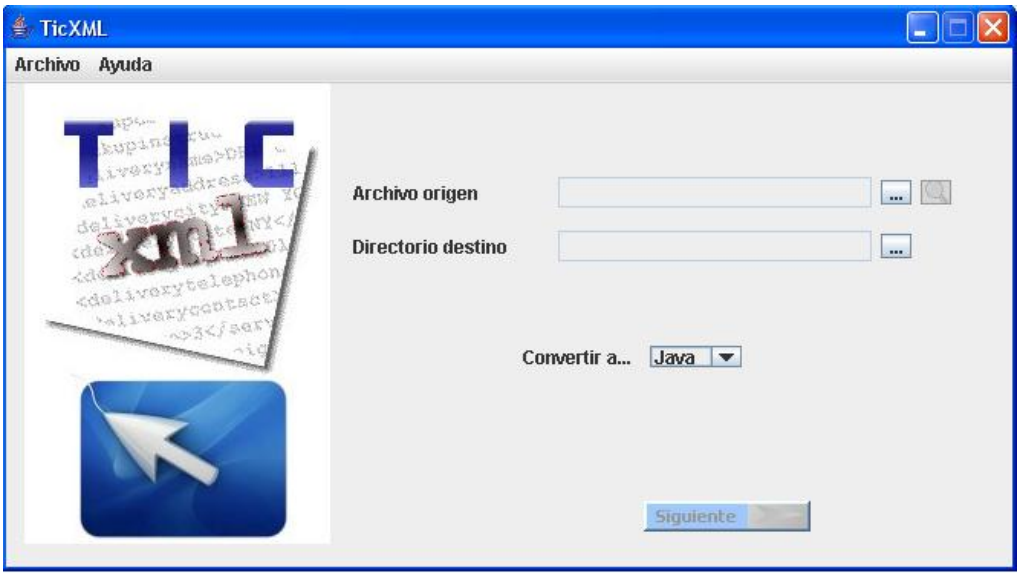


Figura 3.20 Aspecto inicial de la herramienta TicXML

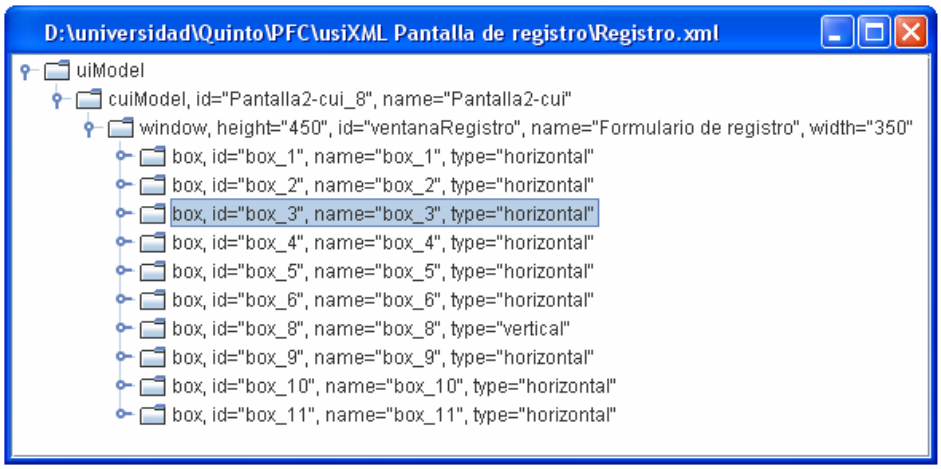


Figura 3.21 Muestra del archivo XML con la herramienta TicXML

Al pulsar sobre el botón 'Siguiente', el programa realiza la transformación y pasa a una segunda pantalla (Fig. 3.22) donde se muestra el código obtenido en texto, este código es fácilmente transportable a cualquier aplicación mediante los controles de copiar y pegar, aunque no será necesario ya que se habrá creado un archivo con la extensión del formato fijado, por ejemplo un .java si se trata de una transformación a ese lenguaje. En este punto se ofrece la posibilidad de ejecutar el programa obtenido, al pulsar sobre el botón 'Ejecutar', la herramienta lanzará la ejecución externa del programa recién obtenido mediante la transformación.

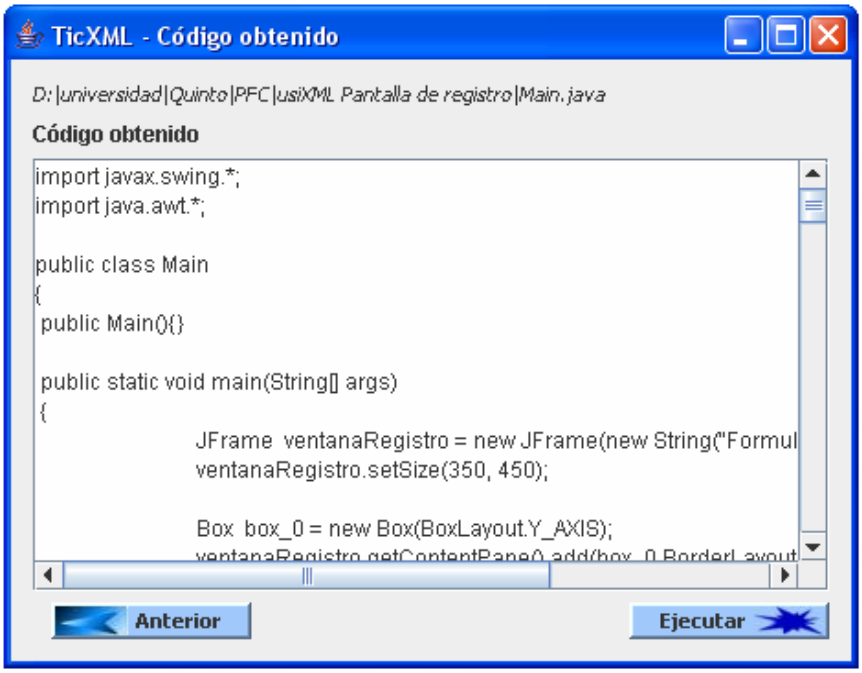


Figura 3.22 Segunda pantalla de la herramienta TicXML. Código obtenido

La herramienta provee de menús de ayuda que puedan ser útiles para clarificar el sencillo funcionamiento de la misma, así como una corta información acerca del programa y el autor.

Para el manejo de las transformaciones XSLT se han utilizado las librerías Dom4j (Dom4j, 2007) creadas a tal efecto, con las que se da soporte para cargar y transformar los archivos XML, obteniendo así el archivo resultante.

La mejor forma de probar las ideas y herramientas presentadas en este capítulo se estima mediante la realización de ejemplos de prueba, por ello, en el siguiente capítulo se detallarán dos casos de estudio que aborden la metodología presentada en su completitud.

Capítulo 4 : CASOS DE ESTUDIO

En el capítulo anterior se detallaron los pasos necesarios para realizar transformaciones de CUI a FUI y la integración de todos ellos en la herramienta TicXML. En este capítulo se abordarán dos ejemplos, presentados en dos casos de estudio diferentes, donde se mostrarán los pasos a seguir para lograr las interfaces de usuario objetivo. El primer caso de estudio será un ejemplo muy sencillo que ilustre de una forma clara el procedimiento a llevar a cabo sin entrar en un alto nivel de detalle, y como segundo ejemplo se mostrará un ejemplo más complicado y elaborado que permita centrarse en algunas particularidades interesantes.

Los casos de estudio girarán principalmente entorno a la herramienta TicXML, con la que se posibilita de una forma rápida y sencilla las transformaciones que ocupan el proyecto final de carrera. Posibilitando también al usuario no familiarizado en la problemática presentada en capítulos anteriores obtener salidas que le puedan resultar interesantes y útiles.

4.1 Ejemplo 1: Petición de usuario y contraseña

En este caso práctico, se realizará un recorrido por la metodología propuesta con el ejemplo de una sencilla interfaz de petición de usuario y contraseña. En la figura 4.1 se muestra la interfaz de ingreso de usuario y contraseña para el servicio de correo electrónico de Google (http://www.gmail.com), que muestra un ejemplo para este tipo de interfaces.

Figura 4.1 Ejemplo de interfaz de petición de usuario y contraseña

El primer paso que hay que dar es el de conseguir una especificación de la interfaz de usuario a nivel concreto que refleje estos aspectos en UsiXML. Básicamente esta interfaz constará de dos etiquetas, un campo de introducción de texto, un campo de introducción de contraseña y un botón. Para la definición de la interfaz hay dos posibilidades: escribir de cero la especificación atendiendo a la sintaxis UsiXML ó hacer uso de alguna herramienta que de soporte como

GrafiXML. La solución elegida va a ser una mezcla de ambas, si bien arrastraremos de forma sencilla los componentes, la especificación de los detalles se harán a mano en el fichero resultante.

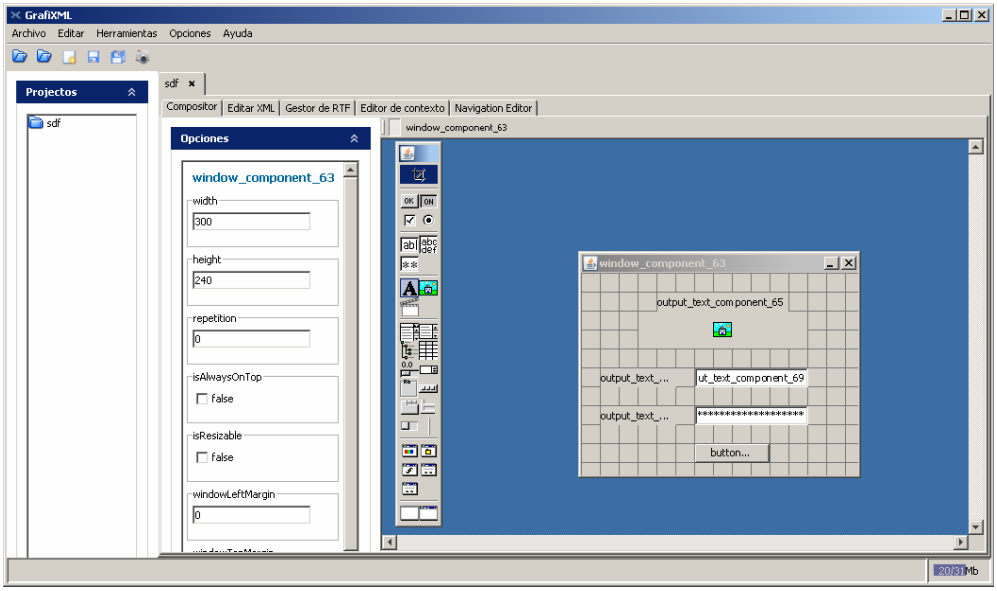


Figura 4.2 Modelo de petición de usuario y contraseña en GrafiXML

Una vez esta hecha la disposición (Fig. 4.2), se obtiene el fichero en *xml* de la especificación de la interfaz diseñada:

```
<uiModel>
    <cuiModel id="GMail-cui_23" name="GMail-cui">
        <window id="window1" name="window1" width="300" height="240">

            <outputText id="output_text_component_3"
                name="output_text_component_3" isVisible="true"/>
            <imageComponent id="image_component_4"
                name="image_component_4" isVisible="true"/>
            <outputText id="output_text_component_5"
                name="output_text_component_5" isVisible="true"/>
            <outputText id="output_text_component_6"
                name="output_text_component_6" isVisible="true"/>
            <inputText id="input_text_component_7"
                name="input_text_component_7" isVisible="true"
                isEnabled="true" textColor="#000000"
                maxLength="50" numberOfColumns="15" isEditable="true"/>
            <inputText id="input_text_component_8"
                name="input_text_component_8" isVisible="true"
                isEnabled="true" maxLength="50" numberOfColumns="15"
                isPassword="true" isEditable="true"/>
            <button id="button_component_9"
                name="button_component_9" isVisible="true" isEnabled="true"/>
        </window>
   </cuiModel>
</uiModel>
```

Se procede a la modificación de algunas partes del archivo. Se modifican los nombres, identificadores, etc. y se eliminan los atributos que no sean relevantes, como por ejemplo el color del texto o la propiedad de ser visible (es obvio que se desean componentes visibles), haciendo cambios como el siguiente:

```
<outputText id="output_text_component_6"
      name="output_text_component_6" isVisible="true"
      isEnabled="true" isBold="true" textColor="#000000"/>
```

```
<outputText id="etiquetaPassword" name="Password"
          isEnabled="true"/>
```

Lo siguiente será añadir contenedores gráficos para que se conserve el orden deseado, en este ejemplo básicamente se desean cinco contenedores horizontales, los primeros y el último alojarán un único componente; etiqueta, imagen y botón respectivamente; los otros dos (tercero y cuarto) contendrán una etiqueta y un campo de introducción de texto; en el primero de ellos será de tipo estándar y en el segundo un campo de introducción de password. Como ejemplo se muestra a continuación el primer contenedor:

```
<outputText id="etiqueta1"
     name="Registrese a Gmail con su" isEnabled="true"/>
```

```
<box id="box_1" name="box_1" type="horizontal">
      <outputText id="etiqueta1"
          name="Registrese a Gmail con su" isEnabled="true"/>
</box>
```

Con las modificaciones comentadas anteriormente se obtendrá el siguiente fichero:

```
<uiModel>
  <cuiModel id=" GMail-cui_23" name=" GMail-cui">
      <window id="ventana" name="Gmail login" width="300" height="240">

          <box id="box_1" name="box_1" type="horizontal">
            <outputText id="etiqueta1"
               name="Registrese a Gmail con su" isEnabled="true"/>
          </box>

          <box id="box_2" name="box_2" type="horizontal">
            <imageComponent id="image1" name="C:\\google.bmp" isEnabled="true"/>
          </box>

          <box id="box_3" name="box_3" type="horizontal">
            <outputText id="et2" name="Nombre de usuario:" isEnabled="true"/>
            <inputText id="introduceUser" name="introduceUser"
               isEnabled="true" maxLength="50" numberOfColumns="15"
               isEditable="true"/>
          </box>

          <box id="box_4" name="box_4" type="horizontal">
            <outputText id="etiqueta3" name="Contrasena:" isEnabled="true"/>
            <inputText  id="introducePass" name="introducePass" isEnabled="true"
               maxLength="50" numberOfColumns="15" isPassword="true"
               isEditable="true"/>
          </box>

          <box id="box_5" name="box_5" type="horizontal">
            <button id="botonOk" name="Entrar" isEnabled="true"/>
          </box>
      </window>
   </cuiModel>
</uiModel>
```

Este proceso que es llevado a cabo cada vez que se desea crear un nuevo modelo no es absolutamente imprescindible, ya que se podría prescindir de él, pero sí es necesario si se quiere respetar los ordenes relativos de los componentes en contenedores y dar nombres y etiquetas a los componentes. Cabe la posibilidad de automatizar el proceso, bien mediante una transformación declarativa similar a la estudiada, o bien mediante algún tratamiento de la imagen resultante que detecte los contenedores y, si es posible, las etiquetas. En el momento del desarrollo del proyecto final de carrera la herramienta *GrafiXML* no disponía de la posibilidad de tratar con '*Layouts*' o posiciones relativas, en fechas próximas a la publicación del presente trabajo el desarrollador de *GrafiXML* lanzó una nueva versión donde contempló esta posibilidad incluyendo diversas formas de gestionar estas posiciones relativas de forma muy similar a las usadas en Java, la modificación de las etiquetas sigue siendo una tarea manual.

Una vez se ha llegado a este punto, con la especificación UsiXML al completo, el siguiente paso será introducir esta especificación a la herramienta TicXML. Bastará con especificar como entrada el archivo que acabamos de crear, así como el directorio donde se alojarán los ficheros resultantes, en este caso se adopta el directorio por defecto que será el mismo que el del archivo origen, además habrá que especificar la implementación de salida deseada, en este caso se ha escogido Java(Fig. 4.3).

Figura 4.3 Parámetros de entrada para el primer caso de estudio

Como resultado se obtiene el archivo de texto con la implementación final indicada (Fig. 4.4), atendiendo a lo especificado en el archivo origen y atendiendo a las particularidades del lenguaje de implementación final seleccionado. El archivo obtenido tendrá la extensión acorde a su tipo, por ejemplo en el caso de Java se obtendrá un archivo .java.

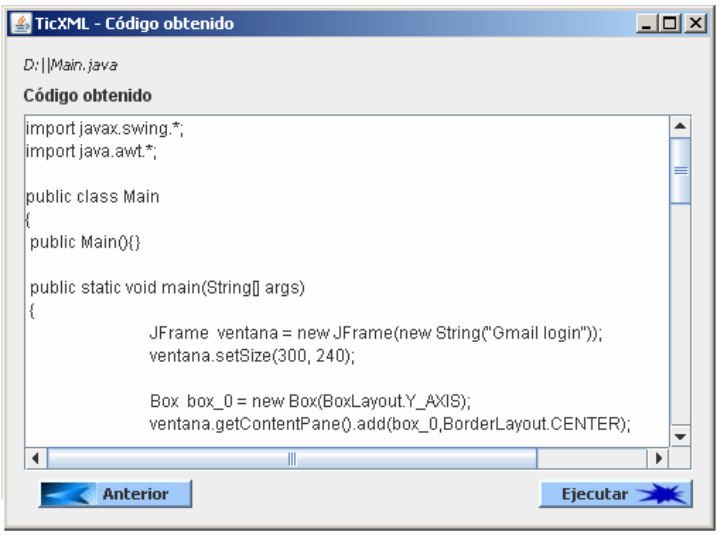


Figura 4.4 Resultado obtenido para la interfaz de petición de usuario y contraseña

Finalmente, se aprovecha la capacidad de TicXML para ejecutar en el mismo momento de la obtención del archivo, al pulsar el botón 'Ejecutar' se lanza el programa a través de la plataforma correcta, en este caso el proceso consiste en compilar el archivo obtenido y ejecutarlo con la máquina virtual de Java (Fig. 4.5).

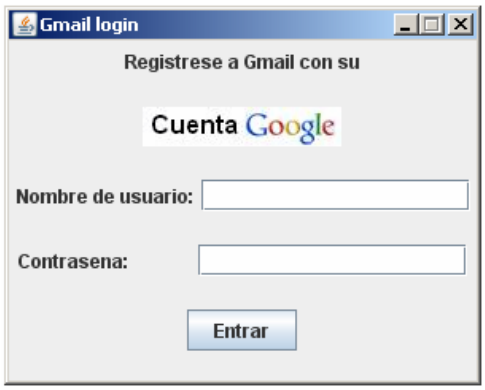


Figura 4.5 Implementación en Java de la interfaz de petición de usuario y contraseña

A modo ilustrativo se va a mostrar la misma interfaz para otra implementación, en este caso HTML, el resultado sería el siguiente (Fig. 4.6 y 4.7):

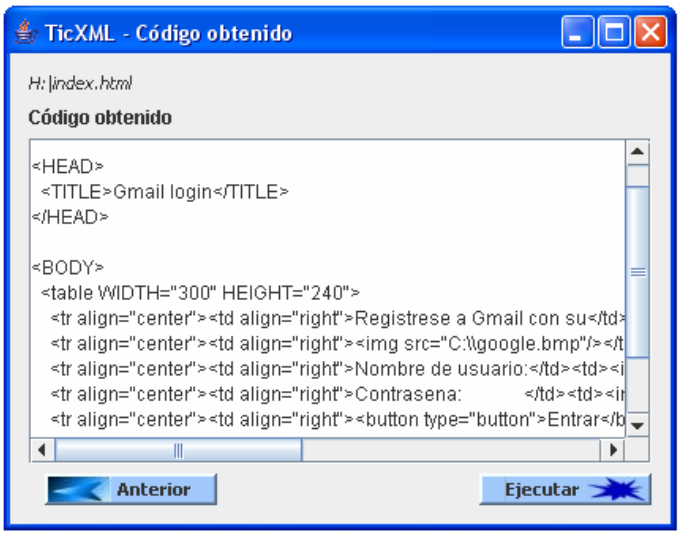


Figura 4.6 Archivo HTML obtenido para la interfaz de petición de usuario y contraseña

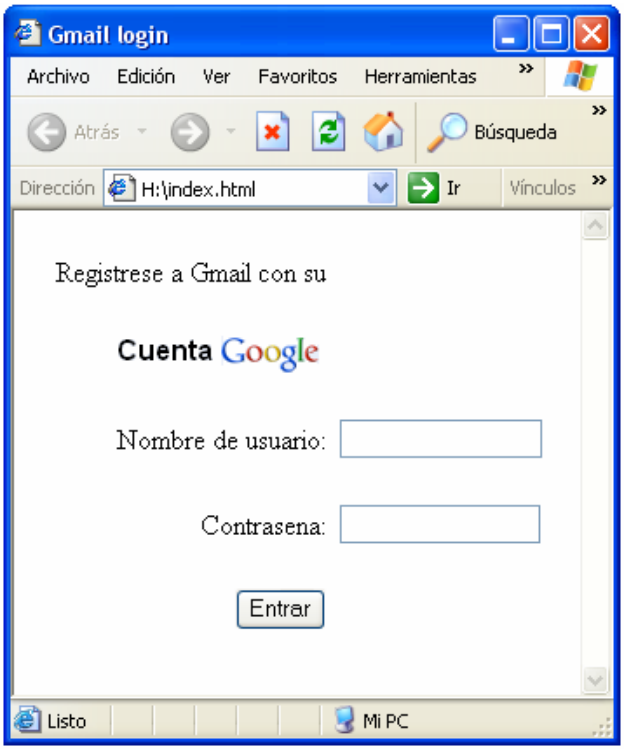


Figura 4.7 Implementación en HTML para la interfaz de petición de usuario y contraseña

4.2 Ejemplo 2: Compra de billetes de avión

En este segundo caso de estudio se va a abordar una interfaz de usuario común a los procesos de compra de billetes de avión o cualquier otro transporte, así como reserva de hoteles y de forma similar a como se hace en diversos formularios para diversos propósitos.

Este caso presenta más complejidad que el anterior, servirá para probar la potencia de la solución desarrollada, aunque no es posible probarla en su completitud ya que algunos componentes como barras de progreso, spinners, etc… no tienen cabida en este ejemplo.

El caso concreto elegido ha sido la interfaz de reserva de billetes de la compañía aérea ClickAir (Fig. 4.8).

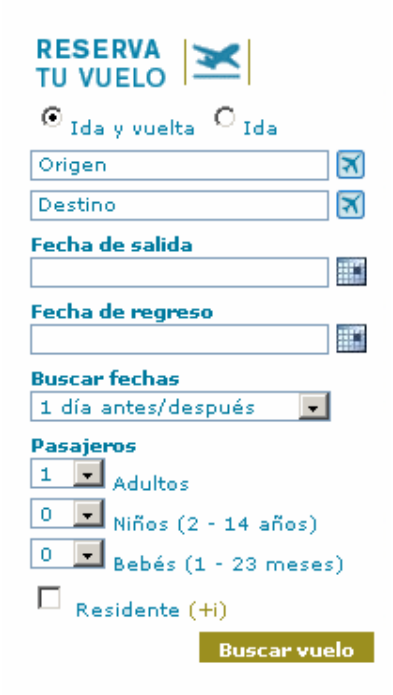


Figura 4.8 Ejemplo de interfaz de compra de billetes de avión

De la misma manera que se hizo en el anterior caso, el primer caso consistirá en introducir el modelo de la interfaz al programa GrafiXML para obtener el modelo con el correspondiente fichero en UsiXML. El modelo se muestra en la siguiente figura.

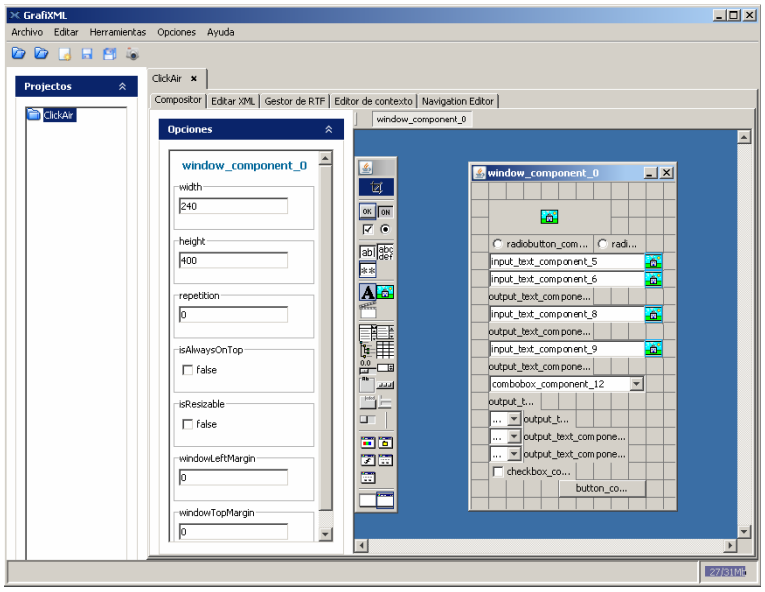


Figura 4.9 Modelo de compra de billetes de avión en GrafiXML

Ahora hay que reformatear el texto obtenido por la herramienta UsiXML de la misma manera que se ha venido comentando anteriormente, una vez hecho

esto se contará con la especificación UsiXML, en este punto se introduce la misma a la herramienta TicXML (Fig 4.10).

Figura 4.10 Parámetros de entrada para el segundo caso de estudio

Como consecuencia se obtiene el código (Fig. 4.11), en Java en este caso, más adelante se mostrará la situación análoga en HTML, y la posterior ejecución del programa (Fig. 4.12).

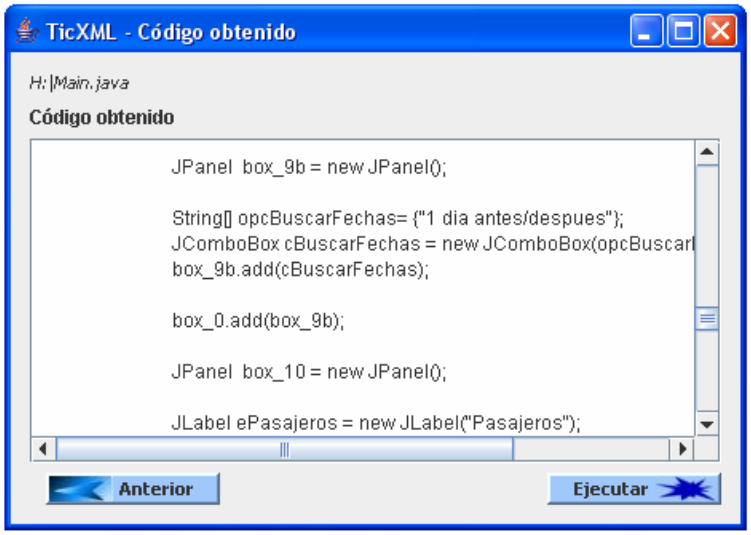


Figura 4.11 Java obtenido para la interfaz de compra de billetes de avión

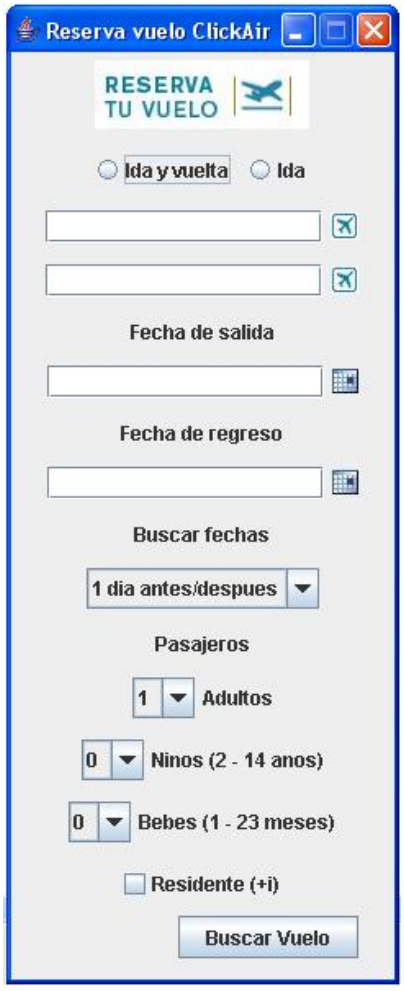


Figura 4.12 Implementación en Java de la interfaz de compra de billetes de avión

El caso análogo del resultado en HTML se muestra en código (Fig. 4.13) y en ejecución (Fig. 4.14).

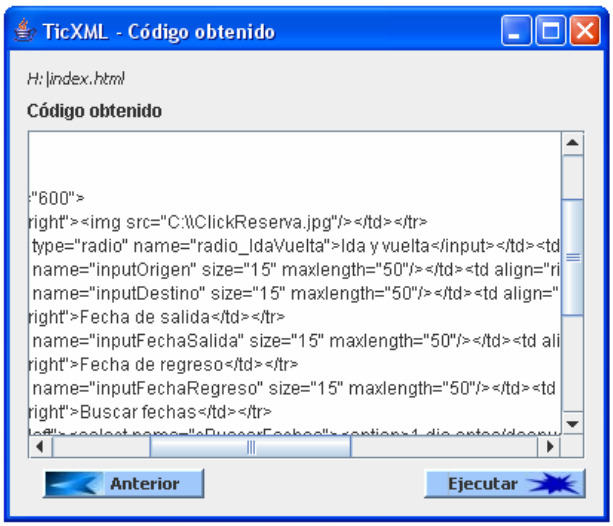


Figura 4.13 Código HTML obtenido para la interfaz de compra de billetes de avión

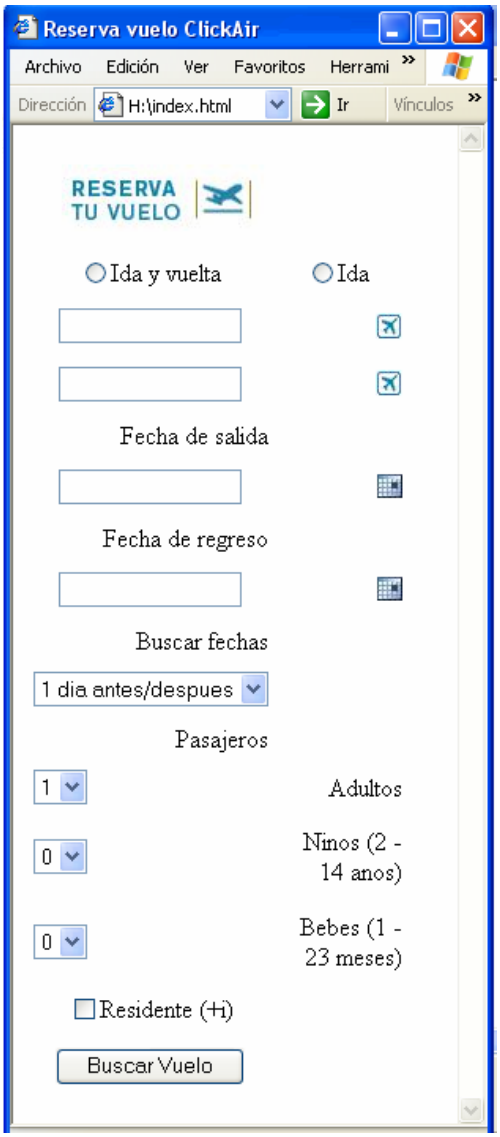


Figura 4.14 Implementación en HTML de la interfaz de compra de billetes de avión

4.3 Conclusiones y comentarios finales

Al considerar modelos de interfaces de usuario, es posible especificar la disposición, los controles y la posición relativa que estos ocupan; sin tener en cuenta la implementación final que le dará forma.

Con los casos de estudio citados se pone en evidencia que es posible tener esta separación y que ésta se realice de forma automática. Ambos casos se tratan de dos ejemplos que muestran de forma clara las características de la solución presentada, si bien es claro que la potencia de la misma es muy amplia ya que se podría realizar cualquier combinación posible con los componentes contemplados.

La gestión de las posiciones relativas abre diversas posibilidades, es posible utilizar ‘cajas’ contenedoras colocadas en orden, opción escogida por este trabajo, aunque también es posible otras organizaciones como situar en coordenadas bidimensionales (x,y), o posiciones relativas a la ventana como por ejemplo en las esquinas o en el centro.

Aunque no se ha considerado, es interesante tener en cuenta el comportamiento y la funcionalidad, es decir, cómo asociar las acciones correspondientes a los componentes introducidos. Para ello se puede atender a los modelos de dominio y CTT, definiendo la asociación que se crea entre ellos. En este sentido se abre una nueva posibilidad a tener en cuenta ya que resulta de gran interés la incorporación de estas asociaciones para la obtención de implementaciones finales más completas.

A la vista de los resultados obtenidos es posible afirmar que la integración y ampliación de este tipo de sistemas resultaría muy ventajosa al mundo de la ingeniería de interfaces de usuario, y en extensión a la ingeniería del software.

Capítulo 5 : CONCLUSIONES Y PROPUESTAS

5.1 Conclusiones

Con el trabajo realizado en el presente proyecto final de carrera se puede concluir que se han conseguido los objetivos que se marcaron inicialmente. El trabajo ha comprendido labores de investigación, análisis, diseño e implementación.

En primer lugar fue necesario familiarizarse con la forma de desarrollar las interfaces de usuario. Para ello se estudiaron diversas interfaces, fruto de ello se pudo llegar a la conclusión de que muchas de las interfaces disponibles utilizan una manera similar de expresar las acciones. Esta tarea de búsqueda ocupó la primera línea de investigación, estos aspectos han quedado reflejados en el inicio capítulo 1, donde se expresa la contextualización del proyecto.

Seguidamente la idea de realizar transformaciones entre modelos fue tomando forma, para ello se consultaron referencias sobre este tema. La filosofía de MDA da luz en este aspecto y sirvió de guía y apoyo para las siguientes fases del proyecto. Estas ideas quedan recogidas en los dos primeros puntos del segundo capítulo.

Una vez se sabía que la tarea era la transformación de modelos, hubo que evaluar los modelos existentes en torno a las interfaces de usuario, quedando recogido en el punto 2.3. En este sentido existen diversas propuestas, con distintos grados de desarrollo, cobertura y herramientas que las acompañan, cabe destacar que se estudiaron en profundidad las propuestas de UsiXML, XIML, TERESA, Thinlet y Laszlo; estas propuestas se comentaron en el apartado 2.4. Se escogió la primera de las citadas, en concreto se trabajó con el modelo de interfaz de usuario a nivel concreto, que sirvió como base para el futuro desarrollo del sistema, en el apartado 3.2 se recogen las particularidades del mismo.

Del estudio de varios lenguajes de implementación de interfaces de usuario, se pensó que resultaría interesante abordar dos ideas de código diferenciadas, en este caso, de lenguajes de texto plano (como java) y de lenguajes de etiquetas (como html), recogidas en el punto 3.3. Para la familiarización se concluyó que la mejor manera era observar varias implementaciones en ambos lenguajes, y además, el estudio de algún entorno de desarrollo donde se encuentren los componentes disponibles.

La tarea más compleja, en lo que a desarrollo se refiere, consistió en abordar la forma de realizar las transformaciones, para ello se estudiaron en profundidad los orígenes y los destinos de las mismas, poniendo especial énfasis en las partes comunes y a replicar. Paralelamente se ideó la manera de hacerlo en términos de ahorro de código y eficiencia. Las ideas y técnicas básicas llevadas a cabo se encuentran recogidas en el punto 3.4.

Para abordar la conversión de modelos en implementaciones se ha concluido que el lenguaje de transformación XSLT era el más conveniente, por trabajar de forma cómoda con XMLs y por su naturaleza declarativa, resultando de forma más sencilla al desarrollador. Dicho lenguaje es comentado en el apartado 3.5.

La mejor forma de automatizar el proceso anteriormente descrito, es estimó que era la construcción de una herramienta que integre todo ello y muestre al usuario de forma sencilla los pasos a realizar en forma de asistente. La herramienta queda comentada en el punto 3.6.

Finalmente, se han presentado dos casos de estudio que pusieron a prueba tanto el sistema desarrollado como las ideas e investigaciones presentadas. En el primer caso se mostró una interfaz sencilla y en el segundo se abordó algo más complejo para que quedase de manifiesto la potencia de la solución presentada. Ambos casos se encuentran comentados en el capítulo cuarto.

5.2 Trabajos futuros

Con la realización de este proyecto fin de carrera se abren varias líneas de investigación, que dan lugar a posibles trabajos futuros a desarrollar, a continuación se señalarán algunos de ellos:

- Desarrollar transformaciones para más implementaciones, en la misma línea y de forma similar a las ya realizadas. Por ejemplo se podrían desarrollar transformaciones para otros lenguajes de programación como C, Pascal, PHP, Basic…

- Estudio de un sistema similar al presentado cambiando el punto de partida, es decir, optar por otro modelo inicial de interfaz de usuario a nivel concreto (CUI), por ejemplo se podría abordar para el modelo de aplicaciones de dispositivos móviles (veáse la figura 2.2).

- Realización de un sistema complementario que se ocupe de hacer el paso inverso, es decir, partiendo de implementaciones generadas con este

sistema en Java o HTML, que el sistema a desarrollar se ocupe de realizar la transformación a código UsiXML. En el contexto de las herramientas en UsiXML se encontraría situada en la misma categoría que *ReversiXML* (Fig. 2.4).

- Integración de las ideas presentadas a mayor escala, es decir, posibilidad de interactuar con patrones y acciones de forma integrada. Idealmente sería interesante que se integrara con todo el proceso de la ingeniería del software, no sólo con el diseño de la interfaz de usuario. Esta línea de investigación se estima como la más importante y más prometedora, ya que si alguna empresa de desarrollo lo utilizara tendría ante sí la posibilidad de portar las herramientas a cualquier lenguaje de los desarrollados de forma totalmente o parcialmente automática, ahorrando así mucho tiempo y, en consecuencia, dinero.

- Consideración del comportamiento/funcionalidad que debe ofrecer la aplicación.

Capítulo 6 : Bibliografía

Montero, F. Tesis doctoral: Integración de calidad y experiencia en el desarrollo de interfaces de usuario dirigido por modelos. (Julio, 2005)

Brown, A. An introduction to Model Driven Architecture. *http://www-128.ibm.com/developerworks/rational/library/3100.html* . (Febrero, 2004)

XML. Extensible Markup Language (XML) 1.0 (Fourth Edition). *http://www.w3.org/TR/REC-xml/* . (Septiembre, 2006)

Paternò, F. Model Based Design and Evaluation of Interactive Applications, Springer-Verlag, London, 1999.

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

UsiXML. *USer Interface eXtensible Markup Language*. *http://www.usixml.org*. Consultado en 2007

GrafiXML. GrafiXML graphical tool. *http://www.usixml.org/index.php?view=page&idpage=10* . Consultado en 2007

FlashiXML. Rendering engine FlashiXML. *http://www.usixml.org/index.php?view=page&idpage=15* . Consultado en 2007

ReversiXML. ReversiXML tool *http://www.usixml.org/index.php?view=page&idpage=17* . Consultado en 2007

IdealXML. Pattern-oriented tool IdealXML. *http://www.usixml.org/index.php?view=page&idpage=34* . Consultado en 2007

XIML. XIML provides a universal specification for interaction data and knowledge that enables a level of control over user interfaces never befote possible. 1990-2004. *http://www.ximl.org/* . Consultado en 2007

Mori et al. Design and Development of Multidevice User Interfaces through MultipleLogical Descriptions IEEE Transactions on Software Engineering (August 2004, pp.507-520)

TERESA. Transformation Enviroment for inteRactivE Systems representAtions. *http://giove.isti.cnr.it/teresa.html* . Consultado en 2007

Thinlet Thinlet tool. *http://www.thinlet.com/* . Consultado en 2007

Laszlo. Laszlo systems. *http://www.laszlosystems.com/* . Consultado en 2007

Java. Java programing language. *http://java.sun.com/* . Consultado en 2007

Eckel, B. Piensa en Java**,** Editorial: Prentice may, Año: 2002

JavaAlmanac. The Java Developers Almanac 1.4. *http://www.javaalmanac.com* . Consultado en 2007

NetBeans. NetBeans IDE. *http://www.netbeans.org* . Consultado en 2007

HTML. Guía de HTML. *http://gias720.dis.ulpgc.es/Gias/Cursos/Tutorial_html/indice.*htm . Consultado en 2007

XSLT. XSLT language. *http://www.w3.org/TR/xslt* . Consultado en 2007

Dom4j. The flexible XML framework for Java. *http://www.dom4j.org* . (Mayo, 2005)