VRU: Un método para validar requisitos y generar interfaces de usuario multiplataforma

Juan Sánchez[1], Jorge Belenguer[1], Pablo Belenguer[2], David Pascual[2]

Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
Valencia – España
[1] {jsanchez, jorbefa}@dsic.upv.es;
[2] {pabbefa, dapasser}@inf.upv.es

Resumen. En el artículo se presenta un método de desarrollo de software que permite por una parte validar requisitos de usuario mediante prototipación automática y por otra generar interfaces de usuario. La generación de las interfaces utiliza como lenguaje intermedio el lenguaje de marcado UIML. Una interfaz en UIML puede ser traducida a diversas plataformas, lenguajes y sistemas operativos. El método es consistente con cualquier método de producción de software que utilice una capa de casos de uso y alguna variante de los diagramas de secuencia para mostrar las interacciones internas en el sistema. La ventaja de la propuesta radica en la utilización de una única definición de interfaz de usuario, que luego puede ser traducida y animada en diversos entornos. Las interfaces, con pequeñas modificaciones, pueden formar parte del producto software final.

1 Introducción

Con la proliferación de dispositivos móviles y con el incremento de las necesidades de acceso a la información desde diversos puntos (Internet, redes locales, redes locales inalámbricas, teléfonos móviles, etc.), una misma aplicación (o parte de su funcionalidad) puede ser invocada desde un ordenador de escritorio, un ordenador móvil, un asistente digital personal (PDA), o bien un teléfono con características WAP. Los usuarios exigen el acceso, desde cualquier ubicación, a la información y funcionalidad que poseen.

Los desarrolladores de las aplicaciones ahora también deben tener en cuenta qué funcionalidad ha de ejecutarse y visualizarse sobre los diversos dispositivos que pueden hacer uso de la aplicación. Si el proceso de construcción de la interfaz se aborda manualmente, lo habitual es que se empleen diversos lenguajes para programar la interfaz, siendo en el peor de los casos un lenguaje diferente por cada tipo de dispositivo.

Otra cuestión, que nosotros relacionaremos en este trabajo con las interfaces multiplataforma o multicanal, radica en el hecho de que la validación de los requisitos de usuario se realice de forma apropiada. Cuando se concibe un sistema informático,

una de las etapas cruciales, por el efecto que tiene en el resto del ciclo de desarrollo, reside en captar y mostrar de modo apropiado los requisitos de usuario. Esta actividad recibe el nombre de *Ingeniería de Requisitos*, habiendo sido reconocida como crucial ([15], [4]) dentro del proceso de desarrollo.

La etapa de captura de requisitos se puede abordar utilizando técnicas de escenarios. Un escenario se define como una descripción parcial del comportamiento de un sistema en una situación particular [5]. Un proceso de ingeniería de requisitos, basado en escenarios ([24]) posee dos tareas principales. En la primera, se genera una especificación, a partir de los escenarios, para describir el comportamiento del sistema. En la segunda, se validan éstos con el usuario, mediante simulación o prototipación. Ambas tareas son difíciles de manejar si no están asistidas por alguna herramienta automática o semiautomática.

Para abordar de un modo apropiado el proceso de captura y validación de requisitos, consideramos que es necesario definir una aproximación metodológica que permita derivar automáticamente interfaces a partir de requisitos de usuario. Por ello, estamos interesados en la validación de escenarios mediante la generación automática de prototipos de interfaces de usuario de la aplicación y la ejecución simbólica de los mismos.

En este artículo se presenta la propuesta metodológica y la herramienta que le da soporte, dentro del campo de la ingeniería de requisitos. La aproximación se basa en el Lenguaje Unificado de Modelado (UML [18]), extendido con la introducción de *Message Sequence Charts*[1] (MSCs [13]) enriquecidos con información referente a la interfaz de usuario. Dado que los MSCs se consideran una extensión de los diagramas de secuencia de UML, añadiendo los correspondientes estereotipos, la propuesta puede considerarse, desde el punto de vista notacional, consistente con UML.

El proceso es iterativo e incremental y permite derivar, como ya hemos comentado, los prototipos de la aplicación. Además, se genera una especificación formal del sistema que se representa mediante diagramas de transición entre estados. Estos diagramas describen el comportamiento de los objetos de interfaz y de control presentes en cada MSC. Una importante contribución del método es que genera de forma automática un modelo de navegación entre formularios, basado en las relaciones existentes, incluye y extiende, dentro del modelo de casos de uso. Esta característica de navegación permite la utilización de los prototipos generados dentro de entornos web.

En este artículo presentamos una extensión a nuestra propuesta inicial de validación de requisitos de usuario mediante prototipación automática ([21], [22], [23]) que utiliza una única definición de interfaz en el lenguaje de marcado UIML (User Interface Markup Language [12]), a partir de la cual mediante un proceso de traducción, se pueden crear y animar interfaces en diversos lenguajes y plataformas. En el trabajo haremos mayor hincapié en los nuevos aspectos incluidos en la propuesta: la utilización de UIML y la traducción de la interfaz al lenguaje destino.

El artículo está estructurado de la siguiente forma: la sección 2 contiene un resumen de la propuesta de generación con el proceso que guía los diferentes pasos.

[1] Utilizaremos indistintamente MSC ó diagrama de secuenciación de mensajes.

La sección tercera está dedicada a presentar los diagramas de secuenciación de mensajes (MSCs), que juegan un rol determinante en el proceso de generación. En la sección cuarta describimos las características principales de UIML, el lenguaje de marcado que hemos seleccionado para almacenar definiciones de interfaces de usuario. La sección quinta describe detalladamente el proceso de creación de la interfaz en UIML y las actividades de traducción a la plataforma destino. En la sección 6 se utiliza un caso de estudio para ilustrar el proceso de generación de la interfaz. Por último la sección 7 contiene las conclusiones y los trabajos futuros.

2 El método de generación

En este apartado describiremos el método que permite generar por una parte una especificación del comportamiento del sistema y por otra un conjunto de prototipos de interfaz de usuario. Utilizaremos el acrónimo *VRU* (Validación de Requisitos de Usuario) para referirnos al mismo. VRU puede integrarse dentro de cualquier método de producción de software que utilice una capa de casos de uso para describir los requisitos funcionales y alguna variante de los diagramas de secuencia para describir las interacciones internas dentro del sistema.

VRU está formado por un componente proceso, una arquitectura de modelos y una notación. El componente proceso describe mediante flujos de trabajo las distintas fases del ciclo de desarrollo que cubre VRU, desde la verificación de requisitos hasta la generación de las interfaces y la validación de los requisitos iniciales. El componente de arquitectura de modelos refleja los artefactos o modelos a partir de los cuales se genera la interfaz de usuario, así como sus dependencias y conexiones. Finalmente, el componente de notación describe el perfil específico de UML que está incluido dentro de VRU. En las siguientes secciones únicamente comentaremos el proceso de VRU y los diagramas MSCs. Para obtener más detalles puede consultarse ([21]).

2.1 El proceso de VRU

El proceso de VRU define los pasos que implementan el método dentro de una organización de desarrollo de software. Aunque el proceso cubre las fases iniciales de desarrollo de software, puede extenderse fácilmente para hacerlo compatible con cualquier método orientado a objetos, como por ejemplo el Proceso Unificado de Desarrollo (USDP) de I. Jacobson ([14]).

La tabla 1 contiene los componentes principales de VRU, las fases del ciclo de vida que cubre, los flujos de trabajo en cada fase, las actividades incluidas en los mismos y el conjunto de entregables o artefactos que se generan en cada actividad. La primera columna de la tabla contiene el nombre de las fases del ciclo de desarrollo cubiertos por la propuesta: análisis y validación. La segunda, los flujos de trabajo que gobiernan cada una de las etapas. En la tercera y cuarta se incluyen, respectivamente, las actividades y los entregables.

Table 1. Flujos de trabajo, actividades y entregables de VRU.

Etapa del ciclo de vida	Flujo de Trabajo	Actividades de VRU	Entregables
Análisis preeliminar	Análisis Externo	Interiorizar Proyecto	Diccionario
			Requisitos Informales
		Crear perfil de usuarios	Mapa de roles de usuario
		Elaborar ámbito sistema	Modelo de dominio
			Diagrama de contexto
Análisis detallado	Análisis Funcional	Crear modelo funcional	Actores
			Casos de Uso
			Servicios Funcionales
	Análisis Interno	Modelización de objetos	Diagrama de clases
		Síntesis de casos de uso	Diagramas MSC
		Etiquetar MSCs	Diagramas MSC etiquetados
	Generación	Generar interfaz	Modelo de presentación
		Generar dinámica	Diagramas de transición
Validación	Animación	Ejecución interfaz	

La fase de análisis de requisitos, o análisis preeliminar de requisitos, supone el arranque del proyecto. En esta etapa participan dos tipos de trabajadores o están presentes dos roles. El primero, el experto en el dominio del problema, representa un profesional con experiencia en las prácticas y en la terminología utilizada por la organización para la cual se desarrollará el sistema. Se encarga de proporcionar la entrada inicial al proceso de desarrollo: el documento inicial de requisitos. El segundo, el ingeniero de requisitos, representa un profesional cuyo campo de experiencia es la construcción de modelos de requisitos de acuerdo a la propuesta descrita en este artículo. Debe ser capaz de analizar el documento informal de requisitos y construir los modelos de análisis (explicados después).

Esta etapa comprende tres actividades secuenciales: *interiorización del proyecto*, creación del *perfil de los usuarios*, y *elaboración del ámbito del sistema*. En la actividad de *interiorización* del proyecto un experto del dominio del problema se encarga de elaborar un documento textual, que contiene por una parte un *diccionario* con la terminología del dominio, y por otra una descripción de alto nivel de la funcionalidad o servicios que ofertará el futuro sistema (requisitos informales). Los artefactos utilizados se engloban en el *documento inicial de requisitos*. Tanto el diccionario como los requisitos informales deben estar elaborados en un lenguaje comprensible tanto por los clientes/usuarios como por los desarrolladores. Esta documentación textual no forma parte de ninguno de los modelos propuestos por UML.

Los perfiles de usuario, o el modelo de roles de usuario, describe los roles que juegan los usuarios cuando interactúan con el sistema para ejecutar la funcionalidad contenida en él mismo. Este modelo está basado en la propuesta de mapa de roles de usuario de Constantine y Lockwood ([6]), donde los roles representan tipos de usuarios que muestran el mismo interés, comportamiento, responsabilidades y expectativas en su relación con el sistema. Constantine define tres tipos de relaciones entre roles: afinidad, clasificación y composición. Estas ideas manteniendo la semántica original definida en "*Usage-Centered Design*", las hemos representado mediante los mecanismos de extensión de UML. Nuestro modelo de roles contiene actores y tres relaciones estereotipadas: afinidad, herencia e inclusión.

La actividad *elaborar ámbito del sistema* pone en contexto al mismo dentro de la organización en la cual se encontrará inmerso. Crea el modelo de dominio y el diagrama de contexto. El modelo de dominio es similar al modelo de dominio propuesto por I. Jacobson [14], que define de la siguiente forma: "*el modelo de dominio representa los tipos de objetos más importantes dentro del contexto del sistema. Los objetos del dominio representan entidades o cosas que existen en el entorno en el cual operará el sistema*". El diagrama de contexto define el ámbito del sistema (qué queda fuera y qué queda dentro del mismo), similar a los diagramas de contexto de E. Yourdon aunque con la notación del modelo de casos de uso de UML.

El objetivo de la fase de análisis detallado es rescribir el documento de requisitos inicial bajo la forma de modelos gráficos y textuales consistentes con UML. La fase de análisis detallado de requisitos crea, a partir de los modelos generados en la fase de análisis preeliminar, una especificación de requisitos del sistema. Ésta está gobernada por los siguientes flujos de trabajo: análisis funcional, análisis interno y generación.

El flujo de trabajo de análisis funcional es similar al propuesto por USDP con la salvedad de que no se crea manualmente un prototipo de la interfaz de usuario y de que se introduce un modelo adicional, la descomposición de servicios funcionales del sistema (macro servicios, que explicaremos a continuación).

La figura 1 contiene el flujo de trabajo funcional de VRU, en USDP este flujo recibe el nombre de flujo de trabajo de requisitos. Con respecto a los artefactos generados (óvalos inferiores de la figura), según lo comentado anteriormente, son similares a los propuestos por UML. El modelo de actores contiene tanto roles como los actores no humanos. Existen tres modelos, dos gráficos y uno textual, para reflejar los requisitos funcionales del sistema: el diagrama de casos de uso, la descripción de cada caso y el modelo estructurado. La descripción de casos de uso contiene plantillas textuales que muestran la comunicación actor-sistema. El diagrama de casos de uso refleja gráficamente los actores y los distintos casos; mientras que el modelo estructurado contiene las relaciones estereotipadas de UML: <<include>>, <<extend>> y <<generalization>> entre los diversos casos de uso.

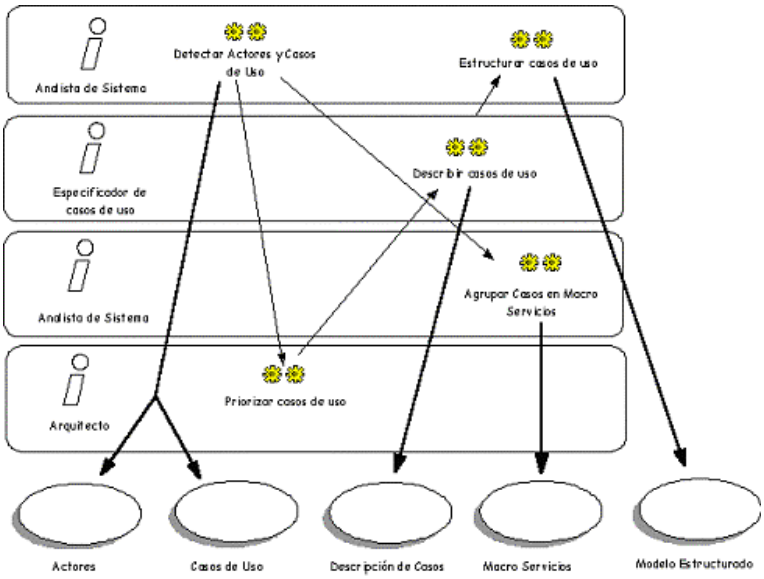

Fig. 1. Flujo de trabajo de VRU.

El concepto de macro servicio permite descomponer jerárquicamente el modelo de casos de uso. Para ello definimos un *servicio* del sistema, o *caso de uso elemental*, como la unidad básica de funcionalidad que puede invocar o utilizar un actor externo del sistema. Mientras que un *macro servicio* del sistema, o un *caso de uso de alto nivel*, contiene una agrupación jerárquica de servicios. Estos servicios pueden ser a su vez elementales o macro servicios. El modelo de casos de uso puede dividirse considerando los diversos tipos de usuarios y las necesidades de éstos con respecto al sistema. Podemos tener usuarios que utilicen frecuentemente las funciones ofertadas por el sistema; otros pueden utilizar el sistema ocasionalmente, mientras que una tercera clase de usuarios pueden encargarse de administrarlo. La *aproximación basada en grupos de usuarios* ([16]) utiliza los grupos de usuarios, representados en el modelo de actores, como criterio de agrupación y descomposición de los casos de uso. También puede dividirse utilizando el concepto de *situación de trabajo* ([17]). Ésta información será utilizada en el proceso de generación, para estructurar la interfaz de usuario de la aplicación.

El segundo flujo de trabajo, dentro de la fase análisis detallado, recibe el nombre de *análisis interno* y contiene las actividades llamadas modelización de objetos, síntesis de casos de uso y etiquetado de diagramas MSCs. En la Figura 2 se muestra el flujo de trabajo utilizando un diagrama de actividad de UML. Las dos actividades principales son la actividad de modelización de objetos y la actividad de síntesis de casos de uso. La primera crea un diagrama de clases del sistema, mientras que la segunda describe las interacciones internas asociadas a los casos de uso, utilizando diagramas MSCs. Éstos diagramas se enriquecen con información referente a la interfaz de usuario.

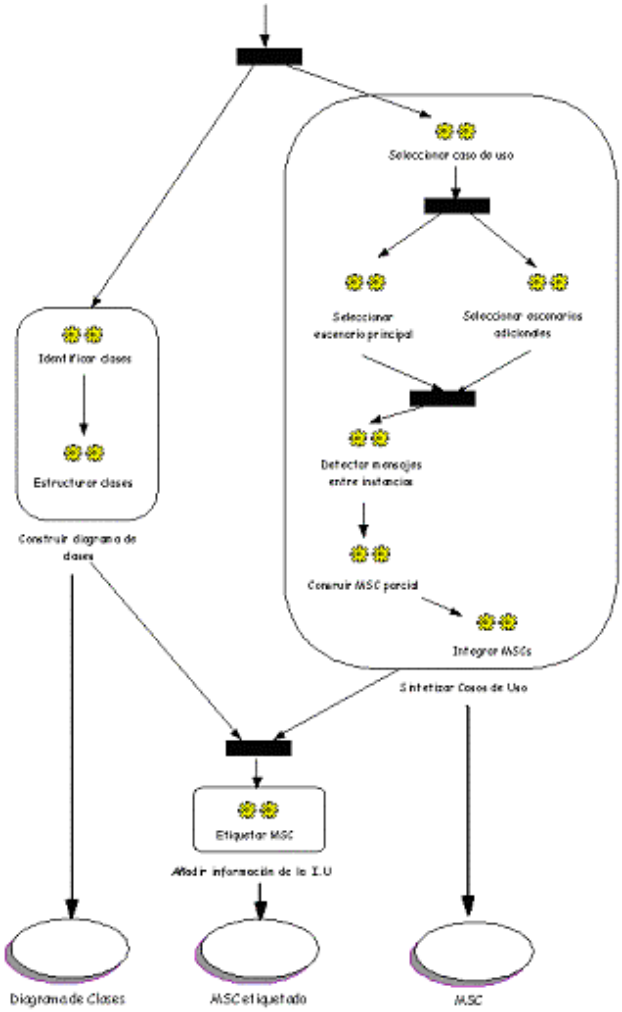

Fig. 2. Flujo de trabajo análisis interno.

El flujo de trabajo de generación contiene dos actividades automáticas que se realizan en paralelo: la generación del modelo de presentación y la generación del comportamiento dinámico del sistema. El primero será descrito en detalle en las siguientes secciones, ya que implica introducir una nueva arquitectura para la interfaz de usuario. El segundo genera un diagrama de transición entre estados para las instancias incluidas en los diagramas MSCs.

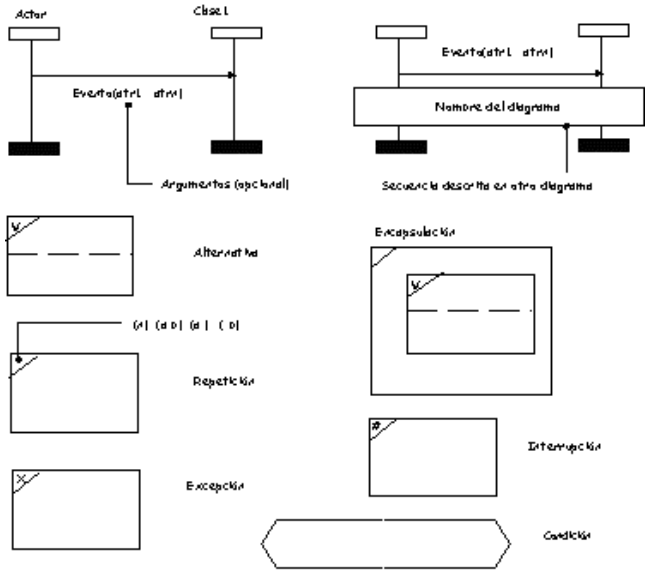

Fig. 3. Notación incluida en los diagramas MSCs.

3 Diagramas de secuenciación de mensajes (MSCs)

Los MSC se emplean ampliamente dentro del campo de las telecomunicaciones para describir el intercambio de información entre las instancias o procesos de un sistema. La notación incluida en nuestra herramienta aparece reflejada en la Figura 3. Las líneas verticales representan instancias de clases o actores del sistema, el paso de mensajes o el intercambio de información se refleja en la forma de líneas horizontales. Cada mensaje puede llevar asociado un conjunto de atributos. Éstos pueden ser tipos básicos (enteros, reales, etc.) o tipos clase. Existe una notación adicional que refleja: repetición de eventos, alternativa, excepciones, interrupciones y condiciones que pueden reflejar el estado en el que se encuentra el sistema. Los MSC se pueden descomponer jerárquicamente por niveles, de forma que en cada diagrama el analista puede utilizar el nivel de abstracción que desee. Los diagramas MSC se pueden enriquecer con información referente a la interfaz de usuario. Ésta información está formada por etiquetas textuales que aparecerán literalmente en la interfaz generada.

El tipo de los atributos de los mensajes determina durante el proceso de generación, el componente gráfico (*widget*) que le corresponde en una interfaz de usuario. Si el argumento es un tipo enumerado cuya talla es menor que 4, se le asocia un conjunto de botones de selección. Por el contrario, si la talla es mayor que 4 le corresponde una lista de selección. Los detalles de la equivalencia pueden obtenerse en [22].

4 User Interface Markup Language (UIML)

Aunque existen diversas propuestas para definir interfaces de usuario de forma declarativa y universal (AUIML [3], IML [10], XIML [7],[19]) nosotros hemos seleccionado UIML por ser un lenguaje no propietario, abierto y por la disponibilidad de herramientas de desarrollo de terceros.

User Interface Markup Language ([1], [2]) es un lenguaje declarativo basado en XML ([25]) que permite la descripción canónica de interfaces de usuario. Se trata, por tanto, de un meta-lenguaje (o lenguaje extensible) de descripción de interfaces de usuario genéricas que puede proyectarse sobre distintos lenguajes dependientes de dispositivo como pueden ser HTML ([26]), Java™ ([11]) ó WML ([28]).

Desde su génesis, UIML ha intentado mantener una separación natural entre la interfaz de usuario y la lógica de aplicación, y es por esa razón por lo que las etiquetas de UIML son independientes de cualquier metáfora de interfaz de usuario, de cualquier plataforma[2] destino o de cualquier lenguaje al que UIML pueda ser proyectado[3].

Un documento UIML típico se compone de dos partes bien diferenciadas. La primera parte es un prólogo que identifica la versión del lenguaje XML, la codificación y la localización de la DTD UIML. La segunda parte es el elemento raíz del documento, representado por medio de la etiqueta <*uiml*>. El elemento raíz contiene a su vez cuatro subelementos opcionales: head, template, *interface* y peers. El elemento cabecera *head* alberga metadatos acerca del documento. El elemento template permite la reutilización de fragmentos de UIML. El elemento *interface* describe la estructura por medio de los distintos *parts* que componen la interfaz. Por último, *peers* define la proyección de las clases y etiquetas del documento UIML a una herramienta de construcción de interfaces concreta.

En UIML, una interfaz de usuario es simplemente un conjunto de elementos, denominados *part* (‘parte’ o ‘pieza’), con los que el usuario interactúa. Cada pieza de la interfaz posee un contenido empleado para transmitir información al usuario (v.g. textos, sonidos, imágenes).

UIML representa lógicamente la estructura de una interfaz de usuario como un árbol de piezas que pueden tener asociado además de un contenido, un comportamiento y un estilo. Éste árbol virtual es proyectado a una plataforma destino de acuerdo con la especificación de la presentación oportuna y se comunica con la lógica de aplicación mediante las definiciones de la lógica. El árbol puede ser modificado de forma dinámica mediante la repetición, el borrado, la sustitución, o la mezcla de subárboles en el árbol principal.

Todo documento UIML debe hacer uso de un vocabulario (de forma análoga a XML, el cual requiere de una DTD). El vocabulario establece un conjunto lícito de

---

[2] A partir de ahora, el término ‘plataforma’ se referirá a la combinación de dispositivo, sistema operativo y lenguaje de programación de alto nivel (toolkit).

[3] El vocablo anglosajón ‘render’ se utiliza para referirnos al proceso de traducir un documento a una forma en la que pueda ser visualizado y con la que el usuario pueda interactuar. Ésta palabra ha sido traducida al castellano como ‘proyectar’.

clases de piezas y de propiedades de clases, y puede comprender desde una única plataforma hasta una familia completa plataformas. En el momento de escribir este artículo, la lista de glosarios estándar disponibles se podía encontrar en [27].

Un vocabulario UIML genérico ([8]) puede describir cualquier interfaz de usuario en cualquier plataforma de su familia. Sin embargo, un glosario genérico no es útil sin el correspondiente conjunto de transformaciones que permitan convertir la interfaz de usuario desde el plano abstracto a una implementación particular.

En la Figura 4 puede observarse un ejemplo de documento UIML en el que se ha hecho uso de un vocabulario genérico. En la ilustración también puede verse la interfaz de usuario que resultaría tras su proyección en Windows. El proceso de generación de interfaces de usuario multiplataforma mediante UIML, así como una descripción del marco de trabajo puede consultarse en [9].

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE uiml PUBLIC "-//Harmonia//DTD UIML 3.0 Draft//EN" "UIML3_0a.dtd">
<uiml>
  <interface id="Autentificación">
    <structure>
      <part id="gtc_Autentificación" class="GTopContainer">
       <style> <property name="title">Autentificación</property> </st
       <part id="ga_MainArea" class="GArea">
         <part id="gl_Usuario" class="GLabel"
           <style> <property name="text">Usuario</property> </style>
         <part/>
         <part id="gsltr_Usuario" class="GSLTextRegion" />
         <part/>
         <part id="gl_Contraseña" class="GLabel" />
           <style> <property name="text">Contraseña</property> </sty
         <part/>
         <part id="gsltr_Contraseña" class="GSLTextRegion" />
         <part/>
       </part>
       <part id="ga_ButtonsArea" class="GArea">
         <part id="gb_Aceptar" class="GButton">
           <style> <property name="text">Aceptar</property> </style>
         </part>
         <part id="gb_Cancelar" class="GButton" />
           <style> <property name="text">Cancelar</property> </style>
         </part>
       </part>
     </part>
    <structure>
  </interface>
</uiml>
```

Fig. 4. Ejemplo de documento UIML genérico.

Todo lo expuesto hasta el momento hace que UIML se perfile como un lenguaje idóneo para crear interfaces de usuario dinámicas, multiplataforma, multimodales y multilingües, siempre y cuando se utilice un vocabulario lo suficientemente universal.

5 Uso de UIML en el proceso de generación de interfaces de usuario

El objetivo de este trabajo ha sido poder definir una interfaz que sea independiente del dispositivo, de la plataforma y del sistema operativo en la cual será visualizada, construida o proyectada. Para ello se ha hecho uso del lenguaje UIML, permitiendo así la definición genérica y abstracta de las interfaces. Éste lenguaje ha sido integrado dentro de nuestra propuesta, de modo que el proceso de generación de interfaces de usuario producirá, en un primer estadio, un conjunto de especificaciones UIML

genéricas, y en última instancia, éstas serán traducidas a uno o más lenguajes de programación de alto nivel. Este hecho permite representar la interfaz de usuario sobre distintas plataformas, evitando la necesidad de modificar su definición según el sistema destino seleccionado. De este modo, el diseñador podrá realizar personalizaciones o ajustes sobre la definición de interfaz UIML a un alto nivel de abstracción, propagándose los cambios realizados hasta los lenguajes de programación destino. Todo ello redundará en una reducción considerable del tiempo y del esfuerzo requerido para el desarrollo de aplicaciones multiplataforma, así como facilitará futuros mantenimientos de las mismas, sin olvidar que se mantiene la consistencia entre capas.

5.1 Generación de la Interfaz de Usuario

La fase de generación de la interfaz dentro de VRU (véase tabla 1), emplea los constructores situados en los MSC para generar el modelo de presentación. Éste está formado por: un modelo de vistas por actor que muestra la funcionalidad que puede activar cada actor del modelo de casos de uso, un formulario por caso de uso, y finalmente un modelo de navegación que contiene los formularios y el orden en que se pueden activar.

Los diagramas de secuenciación de mensajes (MSC) se utilizan como entrada a la etapa de generación de la interfaz de usuario, la cual se descompone en 3 etapas secuenciales (Figura 5). La arquitectura de generación de interfaz que presentamos a continuación permite añadir nuevas plataformas destino de un modo sencillo y nada traumático. También permite la reutilización de *renders* de terceras personas, así como la utilización de herramientas de autor que posibiliten modificar la definición en UIML.

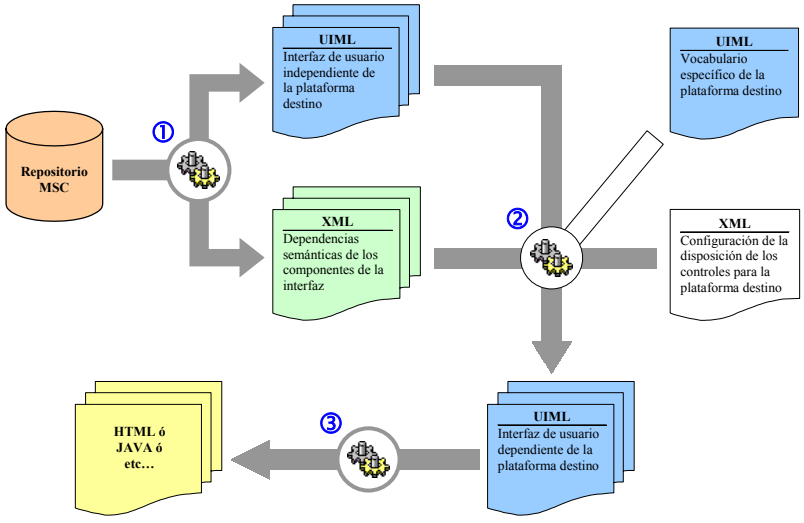

Fig. 5. Generación de la Interfaz de Usuario.

Hay que indicar que todas las actividades contenidas en la figura anterior, desde la 1 hasta la 3, son actividades automáticas. El analista únicamente debe seleccionar el MSC que describe formalmente un caso de uso y el lenguaje destino para la generación.

5.1.1 Cohesión semántica entre controles

Un documento UIML contiene la jerarquía de elementos que participan en la interfaz, donde para cada uno de éstos elementos se puede describir una serie de propiedades básicas como puede ser el texto informativo, la visibilidad o la habilitación/inhabilitación de los mismos. Sin embargo, al margen de lo que es meramente la interacción con el usuario, hay cierta información semántica que escapa del ámbito de UIML.

La información de la que hablamos es la *cohesión semántica* entre controles, o dependencia semántica entre controles, que tiene como objetivo la agrupación de los controles en unidades funcionales. De este modo, catalogaremos una cohesión semántica de acuerdo a la siguiente categorización: *fuerte*, *intermedia* y *débil*.

Una *cohesión fuerte* indica la necesidad de que dos o más elementos de la interfaz queden fuertemente ligados, de forma que sea cual fuere la traducción de dichos elementos, no se les separe en distintas unidades. La cohesión fuerte debe emplearse para exigir que pares de elementos del tipo etiqueta-valor permanezcan juntos en una interfaz (v.g. una caja de texto puede requerir de una etiqueta informativa para informar al usuario sobre el valor solicitado).

Por el contrario, una *cohesión intermedia* indica que un conjunto de elementos presenta un contexto semántico común, como puede ser el caso de varios grupos de pares etiqueta-valor que solicitan nombre, DNI y dirección para crear una instancia de trabajador. En ese caso, el traductor intentará que dichos elementos pertenezcan a la misma unidad funcional, pero en caso de no ser posible por restricciones de espacio, se separarían en unidades distintas. Este tipo de cohesión se asume como propiedad intrínseca a los elementos de agrupación (como pueden ser los marcos o grupos de controles).

El tercer tipo de cohesión, la *cohesión débil*, se emplea para relacionar las diversas partes de un mismo caso de uso, de manera que todos los elementos incluidos en el mismo participan en fases de la misma acción. Sin embargo, este tipo de enlace no se explicita sino que se presupone en cada interfaz dado que la utilización de una interfaz conlleva la funcionalidad de la cohesión débil.

Nuestra cohesión semántica actúa como un contenedor de elementos de interfaz. Pero no sólo puede contener elementos de interfaz sino también otras cohesiones, siempre y cuando se cumplan las siguientes reglas de anidamiento: una cohesión fuerte sólo puede contener elementos de interfaz; una cohesión intermedia puede contener tanto elementos de interfaz como otras cohesiones, siempre y cuando éstas sean fuertes o intermedias; y una cohesión débil puede contener tanto elementos de interfaz como otras cohesiones, excepción hecha de cohesiones débiles.

Cada documento UIML tendrá asociado un fichero, en formato XML y con una gramática elaborada *ex profeso* para ello, que contendrá la cohesión semántica

existente entre los elementos de su interfaz. En la figura 6 puede observarse un ejemplo del fichero en cuestión.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE cohesion PUBLIC "-//MSC//DTD Cohesion 1.0//EN" "Cohesion1_0a.dtd">
<cohesion>
  <interface id="Autentificación">
    <mlink>
      <part id="gl_Autentificación" />
      <mlink>
        <hlink>
          <part id="gl_Usuario" />
          <part id="gsltr_Usuario" />
        </hlink>
        <hlink>
          <part id="gl_Contraseña" />
          <part id="gsltr_Contraseña" />
        </hlink>
      </mlink>
      <mlink>
        <part id="gb_Aceptar" />
        <part id="gb_Cancelar" />
      </mlink>
    </mlink>
  </interface>
</cohesion>
```

Fig. 6. Fichero de cohesión semántica asociado al ejemplo de la figura 4.

5.1.2 Etapa 1: Obtención de una especificación UIML genérica

La primera de las etapas (Figura 5) de la generación de interfaces de usuario consiste en construir de manera automática interfaces software de usuario, de modo que éstas sean independientes de plataforma. Por esa razón, las interfaces serán expresadas mediante UIML y un vocabulario genérico. El proceso consistirá en recorrer el repositorio de MSCs e ir componiendo una interfaz por cada diagrama MSC. Como resultado de esta etapa se obtendrán tantos documentos UIML (en su forma canónica [12]) como MSCs existan, así como un fichero de cohesión semántica por cada uno de los documentos UIML.

5.1.3 Etapa 2: Traducción de la especificación UIML genérica a una plataforma concreta.

Éste paso consiste en concretizar las interfaces genéricas para la plataforma destino y tiene como entradas los documentos obtenidos en la etapa anterior (que corresponden a las interfaces UIML y a su cohesión semántica) y un vocabulario específico para la plataforma destino. Adicionalmente, para que las interfaces de usuario a generar posean la suficiente calidad estética, también se hace uso de una librería que se encargará de ubicar los elementos en la interfaz y de un fichero de configuración con detalles gráficos de los elementos concretos. El contenido de este último fichero estará descrito mediante XML y empleará una DTD específica a tal efecto.

La consecuencia de esta segunda etapa es la conversión del conjunto de interfaces UIML a una nueva representación concorde con el vocabulario específico. Así, los documentos resultantes estarán descritos en términos del lenguaje de programación de

alto nivel y en ellos estarán definidas todas las propiedades de cada uno de los *widgets* (desde la ubicación hasta el tamaño, incluyendo el color, etc.).

5.1.4 Etapa 3: Generación del código fuente del lenguaje de programación de alto nivel destino

En la última etapa de la generación, los documentos UIML específicos son enviados al módulo *render* correspondiente, quién se encargará de la traducción a código fuente del lenguaje de programación de alto nivel, pudiéndose compilar y ejecutar a continuación. Hay que indicar que a pesar de poder utilizar renders de terceras partes, en la herramienta se ha creado un vocabulario genérico propio (adaptado a los diagramas MSCs) y se han programado cada uno de ellos.

6 Un escenario de ejemplo: Gestión de Congresos

El caso de estudio seleccionado es una versión más general del problema conocido como "Conference Review System" distribuido en la conferencia OOPSLA de 1991. Una solución propuesta por J. Rumbaugh puede consultarse en la columna de análisis y diseño del *Journal of Object Oriented Programming* [20]. El propósito del sistema es dar soporte a los procesos de envío, evaluación y selección de artículos, vía Web para una conferencia o congreso. La interacción de los usuarios con el sistema se podrá llevar a cabo mediante un navegador web o bien mediante un dispositivo PDA conectado a un servidor central.

Por cuestiones de espacio únicamente trataremos parte de la funcionalidad asociada al actor autor. Los autores pueden registrarse en la conferencia para enviar un artículo, enviar un artículo, modificarlo, mandar la versión definitiva e inscribirse en la conferencia. La Figura 7 contiene, por una parte, los tres macro servicios que puede activar el autor y, por otra, los casos de uso incluidos en gestión de envíos.

La descomposición de macro servicios o la agrupación de casos de uso, permite obtener automáticamente una estructura de menú para cada actor del sistema (modelo de vistas). A cada macro servicio le corresponde un botón de activación o una entrada de menú en un formulario, en el lenguaje destino.

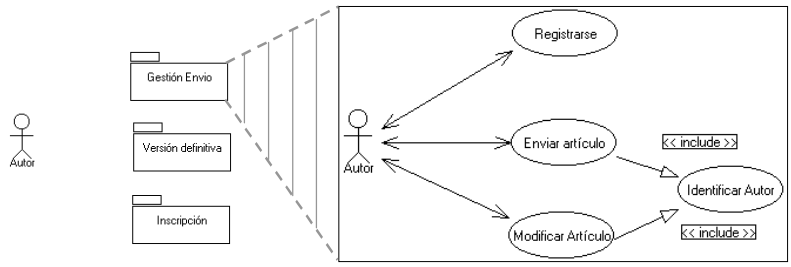

Fig. 7. Vista del actor sobre el modelo de Gestión de Congresos.

La Figura 8 muestra un ejemplo de las pantallas que simulan la navegación sobre las opciones que puede activar el actor autor. Para cada macro servicio se ha generado automáticamente un botón sobre el formulario de la Figura 8. Los botones incluidos en el formulario "Envío de artículos" activan cada uno de los casos de uso pertenecientes al macro servicio "gestión envío": registrarse, enviar artículo y modificar artículo. El caso "identificar autor" no puede activarse directamente, su activación forma parte de la activación de enviar artículo y modificar artículo.

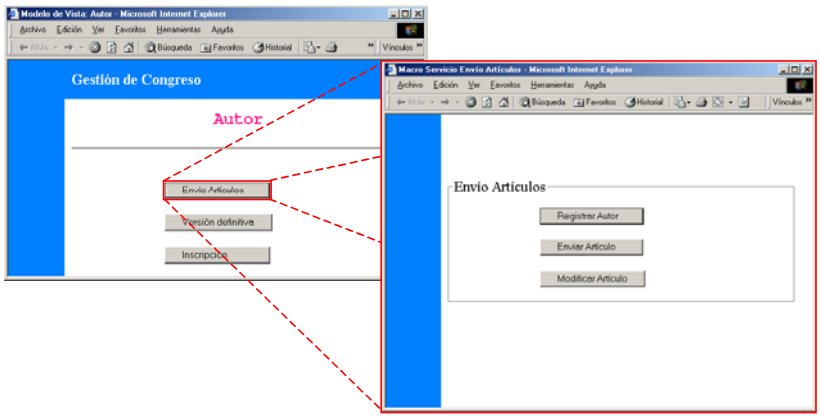

Fig. 8. Estructuración de las opciones del menú.

Para obtener el prototipo de la interfaz de usuario es necesario describir cada uno de los casos de uso, mediante un diagrama MSC. La Figura 9 muestra el diagrama de identificación en el sistema. Este caso de uso es utilizado (<<included>>) por los casos enviar artículo y modificar artículo.

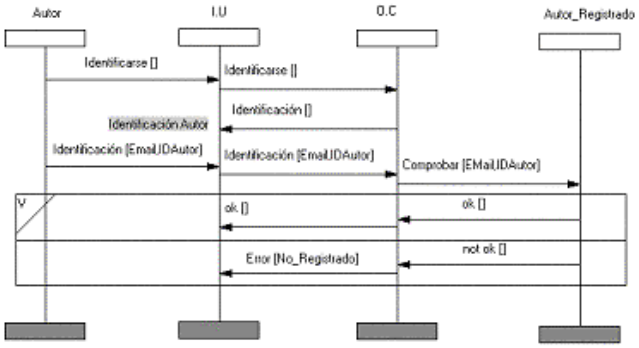

Fig. 9. MSC para identificación de autor.

La Figura 9 muestra las interacciones internas que se producen en el sistema cuando un autor debe identificarse en el mismo. La primera línea vertical representa

al actor primario del caso de uso (en este caso el autor), en cada MSC situamos un objeto de interfaz (I.U.) y un objeto de control (O.C), la cuarta línea vertical representa una instancia de Autor_Registrado. El diagrama contiene la etiqueta "Identificación Autor" que luego se utilizará en la interfaz generada. El MSC contiene los dos escenarios posibles incluidos en el caso de uso, representados por un rectángulo con la etiqueta "V", que el autor esté registrado o que no esté. Para este caso de uso se genera automáticamente, a partir de la información contenida en el MSC, una definición de interfaz en UIML. Seleccionando como lenguaje destino HTML y eMbedded Visual Basic, el proceso de *renderización* genera los formularios de la Figura 10.

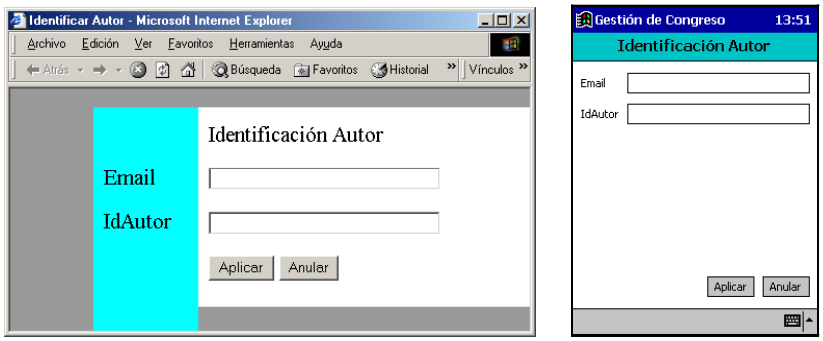

Fig. 10. Apariencia en HTML y eVB de la pantalla de identificación del autor.

El último componente del modelo de presentación, es el modelo de navegación, que habilita la navegación desde un formulario a otro de la aplicación. Para el macro servicio gestión de envíos la única navegación posible se da entre los casos de uso que utilizan a "identificación". Esto se simula en tiempo de ejecución con un botón que permite pasar de un formulario a otro.

7 Conclusiones

En el artículo se ha presentado una extensión multiplataforma a un proceso de validación de requisitos de usuario, mediante la prototipación automática de interfaces de usuario. La utilización de un lenguaje declarativo como UIML para almacenar las definiciones de interfaz de usuario, permite que las mismas sean independientes de la plataforma destino. Además de crear un vocabulario genérico, que cubre aquellos aspectos propios de la información de interfaz que puede incluirse en un diagrama MSC, se han utilizado ficheros de configuración en XML para ubicar los controles sobre la plataforma destino. La utilización de los distintos tipos de cohesión entre los componentes de la interfaz permite una ubicación conveniente de los controles. El proceso de proyección se encuentra parametrizado por estos archivos. Cambiando los archivos de configuración se puede modificar la apariencia de las interfaces

generadas. La arquitectura de generación permite la inclusión de futuras plataformas, sencillamente codificando un nuevo *render*.

En la actualidad nos encontramos estudiando la posibilidad de construir una herramienta de autor para UIML. Ésta herramienta permitiría modificar los componentes o partes de una definición en UIML con el objetivo de mejorar el proceso de proyección.

Referencias

1. Marc Abrams, Constantinos Phanouriou, Alan L. Batongbacal, Stephen M. Williams, Jonathan E. Shuster; "UIML: An Appliance-Independent XML User Interface Language"; En actas de 8th International World Wide Web Conference (Toronto, Canada, Mayo 1999). Elsevier Science Publishers, Amsterdam (1999), pp. 1695-1708.
2. Marc Abrams, Constantinos Phanouriou; "UIML: An XML Language for Building Device-Independent User Interfaces"; En actas de XML'99 (Philadelphia, December 1999). Elsevier Science Publishers, Amsterdam (1999).
3. Pedro Azevedo, Roland Merrick, Dave Roberts, "OVID to AUIML – User-Oriented Interface Modelling", En actas de Towards a UML Profile for Interactive System Development (TUPIS 2000) Workshop (York, UK, Octubre 2000).
4. Dennis W. Bennett; "Designing Hard Software: The Essential Tasks"; Manning Publications co, (1997).
5. Kevin Benner, Martin S. Feather, W.Lewis Johnson; "Utilizing Scenarios in the Software Development Process". En Information System Development Process, pp 117-134. Elsevier Science Publisher, (1993). Editores: N. Prakash, C. Rolland, y B. Percini.
6. Larry L. Constantine, Lucy A.D. Lockwood; "Software for Use: A practical Guide to the Models and Methods of Usage-Centered Design"; Addison Wesley (1999).
7. "eXtensible Interface Markup Language (XIML) Forum", http://www.ximl.org.
8. Mir Farooq Ali, Marc Abrams; "Simplifiyng Construction of Multi-Plataform User Interfaces Using UIML"; En actas de UIML'2001 (Paris, France, Marzo 2001).
9. Mir Farooq Ali, Manuel A. Pérez-Quiñones, Marc Abrahams; "A Multi-Step Process for Generating Multi-Plataform User Interfaces using UIML"; Technical Report cs.HC/0111025, Computing Research Repository, (2001).
10. Karl M. Göschka, Robert. Smeikal; "Interaction Markup Language - An Open Interface for Device Independent Interaction with E-Commerce Applications"; En actas de 34th Hawaii International Conference on Systems Sciences 2001 – HIICSS 34. IEEE Computer Society (Maui, Hawaii, Enero 2001).
11. James Gosling, Bill Joy, Guy Steele, Gilad Bracha; "The Java Language Specification, Second Edition"; http://java.sun.com/docs/books/jls/.
12. Harmonia Inc.; "UIML v3.0 Draft Specification", (Febrero 2002). Disponible a través de: http://www.uiml.org/specs/uiml3/DraftSpec.htm.
13. ITU: Recommendation Z. 120: Message Sequence Chart (MSC). ITU, Geneva, 1996.
14. Ivar Jacobson, Grady Booch, James Rumbaugh; "The Unified Software Development Process"; Addison-Wesley (1999).
15. Gerald Kotonya, Ian Sommerville; "Requirements Engineering: Process and Techniques"; John Wiley & Sons. (1998).
16. Geoff Lee; "Object-Oriented GUI Application Development"; Prentice Hall (1993).
17. Magnus Lif; "User Inteface Modelling- Adding Usability to Use Cases"; International Journal of Human-Computer Studies 50(3). March 1999: 243-262.
18. Object Management Group. UML Versión 1.4. Documento 01-09-67. Encontrado en http://www.omg.org.

19. Angel Puerta, Jacob Eisenstein; "XIML: A Universal Language for User Interfaces", 2002 Conference on Intelligent User Interfaces – IUI 2002 (San Francisco, California, Enero 2002).
20. James Rumbaugh; "Onwoard to OOPSLA"; Journal of Object Oriented Programming, 5(4): 20-24, (Julio/Agosto 1992).
21. Juan Sánchez Díaz; "Validación de requisitos de usuario mediante prototipación y técnicas de transformación de modelos"; Tesis doctoral, Universidad Politécnica de Valencia, Departamento de Sistemas Informáticos y Computación (Noviembre 2002).
22. Juan Sánchez; Juan J. Fons; Óscar Pastor; "From user requirements to user interfaces: a methodological approach". CAISE 2001. (Interlaken, Suiza Junio 2001).
23. Juan Sánchez, Óscar Pastor, Jorge Belenguer; "Generación automática de prototipos de interfaz de usuario a partir de modelos de requisitos"; VI Jornadas de Ingeniería del Software y Bases de Datos (Almagro, España, Noviembre 2001).
24. Stéphane S. Somé, Rachida Dssouli, Jean Vaucher; "Toward an Automation of Requirements Engineering using Scenarios"; Journal of Computing and Information, vol. 2,1, (1996), pp 1110-1132.
25. "eXtensible Markup Language (XML) 1.0"; World Wide Web Consortium Recommendation (Agosto 1998), http://www.w3.org/XML/
26. "HTML 4.01 Specification"; World Wide Web Consortium Recommendation (Diciembre 1999), http://www.w3.org/TR/html401/
27. User Interface Markup Language (UIML) Web, http://www.uiml.org.
28. WAP Forum; "Wireless Markup Language (WML) version 2 Specification"; (Septiembre 2001), http://www1.wapforum.org/tech/documents/WAP-238-WML-20010911-a.pdf