"Generacion des interfaces de usuario a partir de wireframes"

Sanchez Ramon, Oscar ; Sanchez Cuadrado, Jesus ; Garcia Molina, Jesus ; Vanderdonckt, Jean

Abstract

En los ultimos anos han aparecido un gra numero de herramientas de wireframing destinadas especialmente al desino de sitis y paginas web. Un wireframe es una de las representaciones usadas en el diseno de interfaces graficas de usuario (GUI) que muestra la estructura de la interfaz (que controles se utilizan y como se disponen en el area de visualizacion) sin atender a detalles come colores o imagenes. En este trabajo presentamos una solucion basada en tecnicas de Ingenieria de Software Dirigida por Modelos (MDE) para generar codigo final de la implementacion de una GUI a partir de un wireframe. Las tecnicas MDE nos han proporcionado varias ventajas importantes como i) disponer de una representacion de lato nivel del conocimiento extradio en el proceso de ingenieria inversa aplicado a los wireframes, ii) independencia de este proceso de la herramienta wireframing usada y del toolkit de GUI destino, iii) no es necesario que la alineacion de los controls sea precisa y iv) la calidad de...

Document type : *Article de périodique (Journal article)*

novática
Revista de la Asociación de Técnicos de Informática

Novática, revista fundada en 1975 y decana de la prensa informática española, es el órgano oficial de expresión y formación continua de **ATI** (Asociación de Técnicos de Informática), organización que edita también la revista **REICIS** (Revista Española de Innovación, Calidad e Ingeniería del Software).

<http://www.ati.es/novatica/>
<http://www.ati.es/reicis/>

ATI es miembro fundador de **CEPIS** *(Council of European Professional Informatics Societies)* y es representante de España en **IFIP** *(International Federation for Information Processing)*; tiene un acuerdo de colaboración con **ACM** *(Association for Computing Machinery)*, así como acuerdos de vinculación o colaboración con **AdaSpain**, **AI2**, **ASTIC**, **RITSI** e **Hispalinux**, junto a la que participa en **ProInnova**.

Coordinación Editorial, Redacción Central y Redacción ATI Madrid
Plaza de España 6, 2ª planta, 28008 Madrid
Tlfn.914029391; fax.913093685 <novatica@ati.es>
Composición, Edición y Redacción ATI Valencia
Av. del Reino de Valencia 23, 46005 Valencia
Tlfn. 963740173 <novatica_prod@ati.es>
Administración y Redacción ATI Cataluña
Calle Avila 50, 3a planta, local 9, 08005 Barcelona
Tlfn.934125235; fax 934127713 <secregen@ati.es>
Redacción ATI Aragón
Lagasca 9, 3-B, 50006 Zaragoza.
Tlfn./fax 976235181 <secreara@ati.es>
Redacción ATI Andalucía <secreand@ati.es>
Redacción ATI Galicia <secregal@ati.es>
Suscripción y Ventas <novatica.subscripciones@atinet.es>
Publicidad Plaza de España 6, 2ª planta, 28008 Madrid
Tlnf.914029391; fax.913093685 <novatica@ati.es>
Imprenta: Derra S.A., Juan de Austria 66, 08005 Barcelona.
Depósito legal: B 15.154-1975 -- ISSN: 0211-2124; CODEN NOVAEC
Portada: "Heuristically" - Concha Arias Pérez / © ATI
Diseño: Fernando Agresta / © ATI 2003

sumario

Nº 226, noviembre-diciembre 2013, año XXXIX

Tema del próximo número:
"Eficiencia energética en centros de procesos de datos"

Generación de interfaces de usuario a partir de *wireframes*

Óscar Sánchez Ramón[1], Jesús Sánchez Cuadrado[2], Jesús J. García Molina[1,4], Jean Vanderdonckt[3]

[1]Universidad de Murcia; [2]Universidad Autónoma de Madrid; [3]Catholic University of Louvain, Louvain-La-Neuve, Bélgica; [4]Coordinador de la sección técnica "Tecnología de Objetos" de Novática

<osanchez@um.es>, <jesus.sanchez.cuadrado@uam.es>, <jmolina@um.es>, <jean.vanderdonckt@uclouvain.be>

Resumen: *En los últimos años han aparecido un gran número de herramientas de* wireframing *destinadas especialmente al diseño de sitios y páginas web. Un* wireframe *es una de las representaciones usadas en el diseño de interfaces gráficas de usuario (GUI) que muestra la estructura de la interfaz (qué controles se utilizan y cómo se disponen en el área de visualización) sin atender a detalles como colores o imágenes. En este trabajo presentamos una solución basada en técnicas de Ingeniería del Software Dirigida por Modelos (MDE) para generar código final de la implementación de una GUI a partir de un* wireframe*. Las técnicas MDE nos han proporcionado varias ventajas importantes como i) disponer de una representación de alto nivel del conocimiento extraído en el proceso de ingeniería inversa aplicado a los* wireframes*, ii) independencia de este proceso de la herramienta* wireframing *usada y del* toolkit *de GUI destino, iii) no es necesario que la alineación de los controles sea precisa, y iv) la calidad del código generado mediante la obtención de interfaces flexibles.*

Palabras clave: *Inferencia de* layout*, ingeniería del software dirigida por modelos, ingeniería inversa, interfaces de usuario gráficas,* mockups*,* wireframing*.*

Autores

Óscar Sánchez Ramón es miembro del grupo Modelum de la Universidad de Murcia desde 2006 habiendo participado en proyectos de investigación relacionados con la aplicación de la ingeniería dirigida por modelos en diversas áreas como la formalización y ejecución de procesos software y la ingeniería de requisitos de seguridad. En la actualidad se encuentra finalizando su tesis doctoral en la que ha abordado la ingeniería inversa de interfaces de usuario gráficas y ha publicado resultados en revistas y congresos de las áreas de ingeniería inversa e ingeniería dirigida por modelos.

Jesús Sánchez Cuadrado es Doctor en Informática por la Universidad de Murcia desde 2009. Actualmente es Profesor Ayudante Doctor en la Universidad Autónoma de Madrid, y sus líneas de investigación principales se centran en los lenguajes de transformación de modelos, los lenguajes específicos del dominio y la ingeniería de modelos.

Jesús J. García Molina es profesor de la Facultad de Informática de la Universidad de Murcia desde 1984 en la que ha sido decano. Su docencia e investigación siempre ha estado centrada en el área de la tecnología del software. Ha sido pionero en nuestro país en la programación orientada a objetos y en la ingeniería de software dirigida por modelos, tanto en la enseñanza universitaria como en investigación. Dirige el grupo de investigación Modelum desde 2005. Tiene unas cuarenta publicaciones en revistas JCR y congresos de primer nivel y ha participado en varios proyectos de transferencia de tecnología a las empresas. Desde 2002, es coordinador de la sección técnica "Tecnología de Objetos" de ***Novática***.

Jean Vanderdonckt es profesor de la Universidad Católica de Louvain (Bélgica) y lidera el *Louvain Interaction Laboratory* desde 1998. Es coordinador del *UsiXML Consortium*, co-editor jefe de la serie *Springer Human-Computer Interaction Series* (HCI) y Presidente de la "*Louvain School of Management Research Institute*". Su investigación siempre ha estado centrada en la *Human-Computer Interaction*, área en la que tiene más de 300 publicaciones (h-index=30) y ha dirigido unas 20 tesis doctorales. Además ha participado en la puesta en marcha de dos empresas *spin-off* y ha participado en un buen número de proyectos con empresas.

1. Introducción

La creación de interfaces gráficas de usuario (GUI) de sistemas interactivos es una tarea crucial y compleja del desarrollo de aplicaciones que conlleva la consideración de una diversa variedad de aspectos como son la funcionalidad, accesibilidad y usabilidad.

El diseño de GUIs involucra a usuarios y desarrolladores, y también es habitual que participen especialistas en diseño gráfico y usabilidad. Estos participantes normalmente aplican un proceso iterativo en el diseño de la interfaz, a lo largo del cual se construyen prototipos de la interfaz final.

Estos prototipos de la GUI facilitan la comunicación entre los participantes y posibilitan experimentar con la estructura y comportamiento de la interfaz. Tienen diferente naturaleza según su objetivo, pudiéndose destacar cuatro tipos, cada uno de los cuales es un refinamiento del anterior: *sketches* que muestran la idea que se tiene sobre la interfaz que verá el usuario, *wireframes*[1] que muestran los componentes visuales (*widgets*) y su *layout*, *mockups*[2] que refinan un *wireframe* con detalles como color e imágenes, y finalmente un *prototipo* ejecutable para una determinada plataforma GUI.

Los *wireframes* y los *mockups* son creados con herramientas específicas y pueden ser utilizados como entrada de generadores que producen automáticamente el código de la GUI final para una o más plataformas. Por ejemplo, la herramienta Reify[3] genera código a partir de *mockups* creados con Balsamiq[4].

Como es lógico, con este tipo de herramientas se puede reducir de forma significativa el tiempo dedicado por los desarrolladores a la implementación de una GUI. Sin embargo, estos generadores están ligados a *layouts* concretos y sin anidamientos [11] o están ligados a plataformas concretas, como interfaces web con CSS [12], o incluso son herramientas todavía inmaduras como es el caso de Reify.

En la década pasada, la ingeniería del software basada en modelos (*Model-driven engineering*, MDE) se ha consolidado como una nueva visión del desarrollo de software en la que la utilización sistemática de modelos a lo largo del ciclo de vida promueve un mayor nivel de abstracción y un aumento del grado de automatización en la construcción de software [1]. Aprovechando nuestra experiencia en la implementación de una solución MDE para migrar interfaces de usuario de aplicaciones RAD (*Rapid Application Development*) a nuevas plataformas como Java [2], hemos abordado el problema de la generación de GUIs de calidad a partir de *wireframes*, preocupándonos especialmente de la inferencia del *layout* a partir de la disposición de los elementos en el *wireframe*.

En este artículo presentamos el enfoque MDE ideado para realizar ingeniería inversa de *wireframes* con el propósito de extraer el *layout* implícito, es decir, las relaciones entre los elementos que forman la ventana mostrada por el *wireframe*.

> “La creación de una interfaz de usuario de calidad que satisfaga las necesidades de los usuarios es una tarea compleja y costosa en la que los participantes en el diseño suelen aplicar un proceso iterativo para conseguir el resultado final a través de una serie de refinamientos”

Mostraremos cómo a través del uso de modelos hemos obtenido los siguientes beneficios: 1) la solución se integra con herramientas de *wireframing* existentes, de modo que los desarrolladores y usuarios pueden crear los *wireframes* con las herramientas que habitualmente utilizan; 2) la solución es independiente de la herramienta de *wireframing* y del *toolkit* GUI destino; 3) el usuario no debe preocuparse de alinear con precisión los elementos de la ventana cuando crea el *wireframe*, dado que la alineación es inferida; y 4) el código generado es de calidad ya que se han seguido buenas prácticas de programación de GUIs, como las que propone el *Responsive Web Design* [3] para crear interfaces flexibles cuyo *layout* se adapta automáticamente al tipo de dispositivo de visualización (PC, tableta, teléfonos inteligentes, etc.). Para validar nuestro enfoque hemos creado un prototipo que genera páginas web ZK[5] para *wireframes* creados con la herramienta WireframeSketcher[6].

Hemos organizado este artículo del siguiente modo. En la siguiente sección se aclaran los conceptos de *sketching*, *wireframe* and *mockup* que suelen confundirse y se establece el contexto en que se usará nuestra herramienta. En la **sección 3** se describe cómo se ha llevado a cabo la inferencia del *layout*. En la **sección 4** se discute otro trabajo relacionado. Finalmente, se comenta brevemente la implementación del prototipo que hemos construido antes de presentar las conclusiones.

2. Técnicas en el diseño GUI y generación de código

Como hemos indicado, la creación de una interfaz de usuario de calidad que satisfaga las necesidades de los usuarios es una tarea compleja y costosa en la que los participantes en el diseño suelen aplicar un proceso iterativo para conseguir el resultado final a través de una serie de refinamientos. En cada etapa se crea un prototipo GUI por medio de diferentes técnicas, siendo el *sketching*, el *wireframing* y la creación de *mockups* tres de las más comunes y que se suelen confundir.

El *sketching* se utiliza para plasmar la idea concebida para la interfaz en forma de boceto poco elaborado e informal que puede realizarse a mano o con ayuda de una herramienta (por ejemplo [4] y [5]).

El *wireframing* se suele aplicar con herramientas específicas, como Balsamiq o WireframeSketcher, que permiten producir *wireframes* y *mockups*. Un *wireframe* muestra qué *widgets* (por ejemplo, botones, listas de selección y campos de texto) componen la interfaz y cuál es su disposición (*layout*) pero sin considerar detalles como colores, imágenes o logos. En la **figura 1** puede verse un *wireframe* para una ventana de registro de una aplicación “miApp” que ha sido creado con la herramienta WireframeSketcher. Este *wireframe* será utilizado como ejemplo en la siguiente sección para ilustrar nuestro proceso de ingeniería inversa.

Si un *wireframe* es enriquecido con todos los detalles necesarios se obtiene un *mockup* que representa un diseño final de la interfaz de usuario de la aplicación, a partir del cual se puede implementar un prototipo ejecutable. En el *sketching* y el *wireframing* se pretende que el usuario no se distraiga con detalles como los colores y se centre en los aspectos esenciales de la interfaz: estructura y comportamiento. En cambio, con un *mockup* se pretende obtener una representación visual de alta fidelidad que sea útil para demos y tests de usabilidad.

Figura 1. *Wireframe* de una ventana de registro creado con WireframeSketcher.

Un diseñador podría combinar de diferentes formas las técnicas anteriores, por ejemplo podría ignorar el *sketching* y construir el prototipo ejecutable a partir de un *mockup* creado como refinamiento de un *wireframe*, o bien podría crear directamente un *mockup* sin crear *sketches* ni *wireframes*.

Como se sugiere en [6], las tres técnicas pueden ser combinadas. Primero, una etapa de *sketching* permite plasmar las ideas iniciales en el diseño de una GUI, entonces se refina la idea a partir de una secuencia de *wireframes*, y finalmente a partir del último *wireframe* se crea un *mockup* que sería utilizado en la creación del prototipo ejecutable.

Tanto *mockups* como *wireframes* pueden ser utilizados como modelos de entrada a una herramienta que genere código final de la interfaz de usuario para uno o más *toolkits* de GUI. En el caso de los *wireframes* es posible generar el diseño de la interfaz final, es decir los componentes ordenados según algún tipo de *layout* (por ejemplo, *FlowLayout*, *BorderLayout* o *GridBagLayout*), mientras que los *mockups* tienen información suficiente para generar todo el código de la interfaz final.

Con la finalidad de generar código del diseño (*layout*) de la interfaz a partir de *wireframes* creados con cualquier herramienta de *wireframing*, hemos ideado el enfoque y herramienta presentados en este artículo y que serán descritos en las siguientes secciones.

El fichero de un *wireframe* sería la entrada a nuestra herramienta que identificaría los *widgets* e inferiría el *layout* y que generaría el código que implementa la GUI con el *toolkit* elegido. El código generado podría ser manualmente modificado para completar la implementación de la interfaz y una vez estuviese listo sería integrado con el código del sistema en construcción.

Tanto nuestra herramienta como la herramienta de *wireframing* podrían estar integradas con el entorno de desarrollo utilizado. Nótese que el enfoque sirve tanto para *wireframes* como para *mockups*, pues en ambas representaciones se tiene un diseño de la interfaz, no siendo así en los *sketches*.

3. Inferencia del *layout*

Se han aplicado técnicas MDE en la implementación de la herramienta de generación de código GUI a partir de *wireframes*.

Una solución MDE consiste de una cadena de transformaciones de modelos que incluye una transformación modelo-a-texto (m2t) final que genera código de los artefactos de la aplicación (en nuestro caso el código UI) a

partir de algún modelo. Esta transformación m2t suele ir precedida de una o más transformaciones modelo-a-modelo (m2m) que son pasos intermedios en los que el modelo inicial se va transformando en otros de menor nivel de abstracción con el fin de facilitar la transformación final m2t.

Cada modelo conforma con un metamodelo definido con un lenguaje de metamodelado (por ejemplo Ecore si se usa el *framework* Eclipse/EMF [7]) y existen diferentes lenguajes para escribir las transformaciones m2m y m2t, siendo ATL[7] y QVT[8] de los más usados entre los primeros y MofScript[9] y Acceleo[10] entre los segundos.

La **figura 2** muestra la cadena de transformación que hemos aplicado en el caso de la herramienta aquí presentada, la cual consta de cuatro transformaciones m2m que preceden a la transformación final m2t que genera el código.

Las transformaciones m2m implementan el proceso de ingeniería inversa que infiere el *layout* del *wireframe*. Con el fin de favorecer la reutilización, el primer paso consiste en transformar el modelo que representa un *wireframe* para una plataforma específica en un modelo genérico de *wireframes* que independiza el resto de la cadena de herramientas concretas de *wireframing*.

A continuación, se identifican las regiones o zonas visuales de la interfaz (modelo de regiones) y se obtiene el posicionamiento relativo entre *widgets* (modelo de relaciones espaciales). A partir del modelo con información sobre las regiones y el posicionamiento se infiere el *layout* (modelo de *layout*) que se utiliza para generar el código de la UI.

A continuación, vamos a explicar cada una de estas cinco etapas.

3.1. Normalización del modelo fuente

El propósito de la primera etapa es convertir la representación de *wireframe* obtenida con la herramienta de *wireframing* en un modelo que conforma a un metamodelo genérico de *wireframes* que incluye conceptos de GUI básicos como ventana, panel, botón o lista de selección. Por lo tanto, esta etapa "normaliza" el *wireframe* de entrada de modo que nuestro proceso de inferencia de *layout* sea independiente de la herramienta de *wireframing* usada.

La implementación del proceso de normalización se lleva a cabo por medio de una transformación m2m cuya fuente debe ser un modelo que conforme a un metamodelo definido con Ecore (ya que nuestra implementación se basa en EMF), que represente los *wireframes* de la herramienta de *wireframing*.

Algunas herramientas como WireframeSketcher representan los *wireframes* con modelos Ecore, pero la mayoría utilizan otros formatos como esquemas XML (por ejemplo, Balsamiq). En estos casos la normalización requiere un proceso de inyección, previo a la ejecución de la transformación m2m, que genere un modelo del *wireframe* origen.

La implementación del proceso de inyección dependerá del formato original del *wireframe*. Si se trata de archivos XML, EMF ofrece una herramienta que realiza la inyección generando el modelo y metamodelo destino. En el caso de archivos con texto conforme a una gramática se pueden crear inyectores ad-hoc o utilizar el lenguaje Gra2MoL [8] que permite definir transformaciones texto-a-modelo (t2m) a través del establecimiento de correspondencias (*mappings*) entre los elementos de la gramática y el metamodelo destino.

En el caso del *wireframe* de la **figura 1**, sólo sería necesario implementar la transformación m2m que genera el modelo genérico de *wireframes* a partir del modelo Ecore del *wireframe* proporcionado por WireframeSketcher. Esta transformación es muy sencilla de implementar dado que para la mayoría de elementos del metamodelo fuente existe una correspondencia uno a uno con los elementos del metamodelo genérico.

3.2. Extracción de regiones

Una vista (por ejemplo, una ventana o una página web) es un agregado implícito o explícito de partes (*widgets* u otras vistas) que determina su estructura. A partir de ahora nos referiremos a estas partes como *regiones*.

Esta fase tiene un doble objetivo:

- *Identificación de regiones*. Deben identificarse todos los elementos que agrupen *widgets*, por ejemplo un panel con borde que agrupe una lista de botones que permiten elegir una opción entre varias.
- *Hacer explicita la contención entre regiones*. En algunos casos, la relación de contención entre un elemento y su contenedor no es explícita aunque visualmente sí pueda considerarse. Por ejemplo, podríamos tener un panel con borde que visualmente contuviese varios *widgets* pero realmente tanto el panel como los *widgets* son elementos de la ventana al mismo nivel sin ningún anidamiento en la implementación.

La identificación de regiones y el explicitar todas las relaciones de contención entre regiones permite simplificar en gran medida los algoritmos de ingeniería inversa del *layout* de una vista debido a que se establece una correspondencia entre la estructura visual y el *layout* "físico" realmente implementado.

Esta etapa tiene como entrada el modelo de *wireframe* y obtiene como resultado un modelo de regiones. Este modelo añade información al modelo de *wireframe* para explicitar las relaciones entre *widgets*.

En un modelo de regiones: 1) cada *widget* de la GUI es representado como una región que es delimitada por un área rectangular definida por medio de las coordenadas de un par de puntos del área de la interfaz; 2) existen regiones adicionales para agrupar *widgets* espacialmente relacionados; 3) en un mismo nivel de anidamiento todas las regiones incluidas son o bien *widgets* contenedores o bien no contenedores; 4) no se permiten regiones superpuestas. Una explicación detallada de la extracción de regiones se describe en [2].

En la **figura 3** pueden verse las tres regiones identificadas para el *wireframe* de la ventana de registro. El borde que rodea los controles de la parte central compone los límites de la región R2. Los controles que se sitúan entre el borde superior y el límite superior de la ventana forman la región R1. Del mismo modo, los controles que se emplazan debajo del borde inferior de R2 y el límite inferior de la ventana forman la región R3.

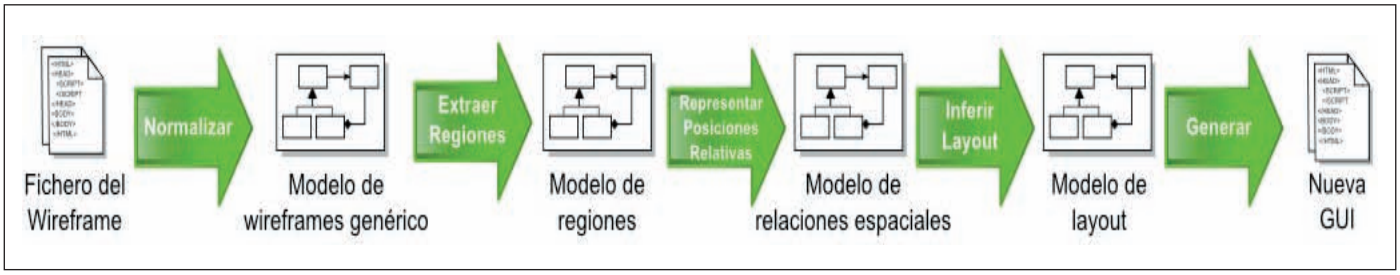

Figura 2. Cadena de transformaciones para obtener una nueva GUI a partir de un *wireframe*.

> “La identificación de regiones y el explicitar todas las relaciones de contención entre regiones permite simplificar en gran medida los algoritmos de ingeniería inversa del *layout* de una vista”

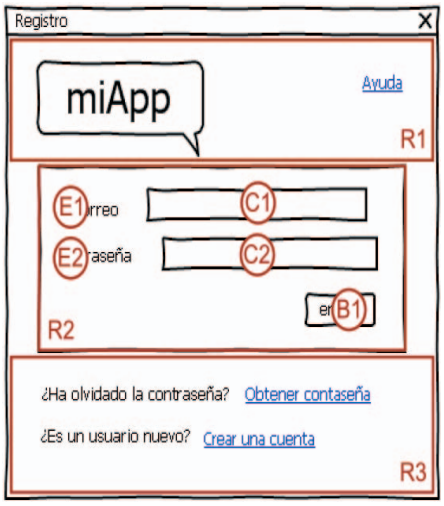

Figura 3. Regiones en el *wireframe* de la ventana de registro.

3.3. Obtención del posicionamiento relativo

Un posicionamiento basado en coordenadas no es deseable dado que los sistemas de coordenadas son dependientes de la plataforma y las vistas no se visualizan correctamente cuando la aplicación se ejecuta para una plataforma diferente a aquella para la cual fue creada. Además, el uso de coordenadas no favorece las acciones relacionadas con la adaptación de la GUI como, por ejemplo, el cambio de su tamaño durante la ejecución.

Por lo tanto, esta etapa está destinada a transformar el modelo de regiones en un modelo de relaciones espaciales mediante la conversión de las coordenadas en posiciones relativas que representan relaciones espaciales de posición entre los elementos (por ejemplo, expresar que un *widget* está situado encima de otro y alineado a la izquierda de éste).

Este modelo representa la estructura de una vista como un grafo dirigido acíclico y atribuido en el que los nodos representan elementos de la GUI (regiones de varios elementos o *widgets*) y los arcos representan la posición relativa entre un par de elementos. Este grafo puede contener otros grafos anidados ya que cuando un nodo representa a una región contiene otro grafo que corresponde a los elementos que la forman. Los nodos también registran información sobre qué porcentaje de anchura y altura de su región contenedora ocupan y la distancia a los elementos vecinos medida como un porcentaje con respecto a la anchura o altura de su región contenedora.

El álgebra de los intervalos de Allen [9] se puede utilizar para representar las relaciones espaciales entre elementos. Un intervalo Allen define la relación entre un par de segmentos en una dimensión. Por ejemplo, dado un primer segmento definido por *ab* (con origen en *a* y destino en *b*) y un segundo segmento *cd*, el intervalo MEETS indica que el final del primer segmento coincide con el inicio del segundo, es decir $a<b<d$ y $b=c$; un intervalo CONTAINS indica que el segundo segmento está estrictamente contenido en el primero, es decir $a<c<d<b$.

Dado que los elementos se sitúan en un plano (2 dimensiones), utilizamos 2 intervalos Allen para definir la posición entre dos elementos. Los dos nodos que componen un arco se ordenan de arriba abajo y de izquierda a derecha (se trata de un grafo dirigido), y esto permite interpretar los intervalos Allen para los arcos sin ambigüedad.

Por lo tanto, hemos etiquetado cada arco con tres informaciones: los intervalos Allen para las dimensiones X e Y, y el valor de proximidad. Este valor es una estimación discreta de la separación entre los dos elementos que representan los nodos conectados por el arco.

Vamos a mostrar un ejemplo del modelo de relaciones espaciales (grafo de posiciones relativas) para la región R2 de la **figura 3**.

La **figura 4** muestra las posiciones relativas entre los *widgets* de la región R2 expresadas como un grafo. Los vértices del grafo representan los *widgets* que han sido etiquetados con las etiquetas asociadas en la **figura 3**.

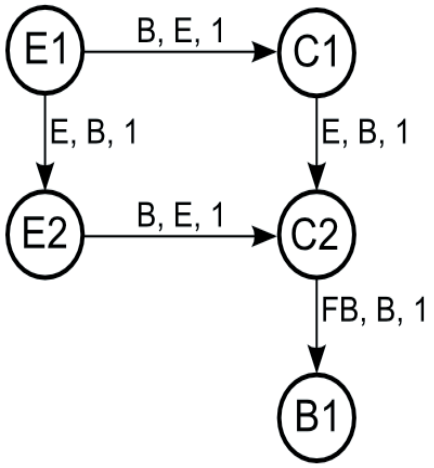

Figura 4. Modelo de posiciones para la región R2 de la ventana de registro.

Los dos primeros valores de los arcos son los intervalos Allen, que han sido representados con la siguiente leyenda: B=Before, E=Equals, FB=FinishedBy. Por ejemplo, si nos fijamos en el arco entre C2 y B1 vemos que la proyección horizontal de C2 es más larga que la de B1 y ambas finalizan más o menos en el mismo punto, y por esto se etiquetan como FB (FinishedBy). Además, si nos fijamos en las proyecciones verticales de ambos *widgets* veremos que C2 está situado antes (más arriba) que B1, y por eso se etiqueta como B (Before).

El tercer atributo de las aristas es el nivel de cercanía, que en todos es 1 debido a que todos los *widgets* están situados a una distancia relativamente similar.

3.4. Obtención del *layout*

Hemos definido un algoritmo de inferencia de *layout* que obtiene un modelo de *layout* a partir de un modelo de relaciones espaciales. Este algoritmo se basa en una estrategia de vuelta atrás (*backtracking*) que aplica una técnica de reescritura de grafos.

El algoritmo genera todas las posibles permutaciones de un conjunto predefinido de patrones de *layout* y para cada permutación se aplica la secuencia de patrones sobre el grafo de relaciones espaciales para comprobar si es una solución. Una permutación se considera una solución si es posible encajar todos los nodos del grafo en sucesivas aplicaciones de los patrones establecidos en la secuencia. Cuando se encuentra que un conjunto de nodos encaja con un patrón, dicho conjunto de nodos es sustituido por un nodo que representa el patrón, y este nodo a su vez puede ser encajado en una posterior aplicación de ese patrón u otro diferente.

Una vez obtenidas todas las soluciones, se valora cada una de ellas con una función de idoneidad (*fitness*) que calcula un valor que depende de los tipos de *layouts* de la secuencia y de la forma en que se combinan. La mejor solución será aquella que tenga un valor la función *fitness* más alto. La estrategia aplicada tiene la ventaja de ofrecer varias soluciones alternativas, lo que permitiría considerar varias opciones de implementación y guiar manualmente el posterior proceso de reestructuración.

Se han considerado cuatro patrones de *layout* que corresponden a tipos de *layout* muy comunes y que son incorporados en la mayoría de *toolkits* de GUI, como es el caso de Java Swing.

Los cuatro patrones considerados son los siguientes (entre paréntesis se indica el nombre comúnmente usado en los *toolkits* de GUI):

■ *Flujo Horizontal / Vertical* (*FlowLayout*): selecciona una secuencia de nodos que son conectados sólo por un arco de salida con los intervalos de Allen X e Y igual a BEFORE o MEETS.

■ *Rejilla* (*GridBagLayout*): Se buscan recursivamente subgrafos de 2×2 nodos conectados de manera que formen una rejilla (*grid*) de n×m nodos. Además, se debe cumplir la restricción de que sólo los nodos de los bordes de la rejilla pueden estar conectados a otros nodos que no forman parte de la rejilla.

■ *Borde* (*BorderLayout*): Se buscan subgrafos que encajen con las cinco zonas de una topología en estrella: *Norte*, *Sur*, *Centro*, *Este* y *Oeste*. No es necesario identificar las cinco áreas, sino que están también permitidas las siguientes cuatro configuraciones: que falte norte o sur, que falte este y oeste, y que falte norte, sur y centro.

■ *Formulario* (*FormLayout*). Primero se busca un flujo vertical compuesto de una lista de flujos horizontales de *widgets*. Después se comprueba que al menos dos de los *widgets* están alineados.

El proceso de aplicar (*matching*) los patrones de *layout* de una secuencia (permutación) sobre el grafo se realiza de acuerdo a los niveles de separación entre los nodos. El algoritmo maneja un nivel límite de separación que se usa para limitar el número de arcos considerados en el *matching*: sólo son candidatos los arcos con un nivel de separación igual o menor que el nivel límite actual.

Al principio, el nivel límite es el más bajo y si no se encuentra un subgrafo que encaje con el patrón entonces el nivel límite se incrementa y se vuelve a aplicar el *matching* sobre los arcos que tengan el nivel límite más bajo o el siguiente nivel, y así sucesivamente. De esta forma un grafo se particiona en subgrafos cuyos nodos están conectados por arcos que tienen un nivel de separación igual o menor que el nivel límite. Cuando un patrón ha sido probado con todos los niveles de separación y no se encuentra un grafo que encaje, entonces el algoritmo deja de probar la secuencia ya que no se puede hallar una solución con esa permutación.

Después de probar todas las secuencias, el algoritmo obtiene un modelo de *layout* que representa la lista de soluciones, cada una de las cuales es una composición de *layouts* que expresa la estructura de la vista.

Continuando con el ejemplo del *wireframe* de la ventana de registro, se obtienen varias soluciones, y la mejor solución (mejor valor de la función *fitness*) es el FormLayout.

Como muestra la **figura 5**, este diseño está compuesto de tres filas (indicadas con rectángulos) y dos columnas (delimitadas por líneas discontinuas). Las cajas de texto se alinean a la derecha de la columna, las cajas de texto a ambos lados y el botón a la derecha. Nótese que se admite cierto grado de flexibilidad en las comparaciones y por esta razón los *widgets* se consideran alineados aunque no estén perfectamente alineados.

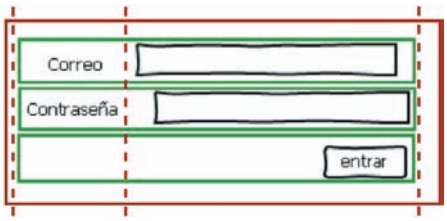

Figura 5. Modelo de *layout* para los *widgets* representados en la región R2 del wireframe de la ventana de registro.

3.5. Generación de código GUI

Un modelo de *layout* contiene información suficiente para generar el *layout* para cualquier plataforma. Además, este modelo hace posible que la cadena de transformaciones sea independiente de la plataforma destino: para cada nueva plataforma sólo será necesario implementar el paso de generación de código que puede ser implementado de tres formas:

■ *Generación directa de código*. Se implementa una transformación m2t que genera el código directamente a partir del modelo de *layout*.

■ *Uso de una transformación intermedia*. Para reducir el salto semántico entre el código GUI y el modelo de *layout* y facilitar la escritura de la transformación final m2t se define un metamodelo de la plataforma destino y se implementa una transformación m2m que transforma el modelo de *layout* en un modelo de la plataforma y entonces se implementaría una transformación m2t para generar código GUI a partir del modelo de la plataforma.

■ *Uso de una herramienta de terceros*. Una variante de la opción anterior es que la transformación m2m genere una representación de un tercero, por ejemplo un metamodelo de un Lenguaje de Definición de Interfaces de Usuario (*User Interface Description Language*, UIDL), y entonces aprovechar los generadores de código existentes para dicho lenguaje. Por ejemplo, se podría aplicar una transformación m2m para generar un modelo UsiXML[11] a partir del modelo *layout*.

Este paso de generación de código suele ser relativamente sencillo comparado con la complejidad de la inferencia del *layout*.

4. Trabajo relacionado

El enfoque presentado se basa en los resultados obtenidos en un trabajo previo [2] en el que se ideó una estrategia de ingeniería inversa para extraer el *layout* de GUIs de aplicaciones RAD (*Rapid Application Development*). Ahora se ha aplicado el mismo algoritmo de identificación de regiones pero la inferencia del *layout* es más sofisticada. Mientras que en el anterior trabajo se aplicaron heurísticas simples, ahora se usa un algoritmo exploratorio que obtiene todos los posibles *layouts* de una vista.

Otro trabajo relacionado con la inferencia de *layout* de una GUI es [10], en el que se usa un modelo matemático basado en programación lineal para expresar las restricciones que determinan el *layout* de una GUI. En vez de expresiones matemáticas, nosotros usamos un modelo de objetos que explícitamente representa el *layout* como un árbol de tipos de *layouts*, y además capturamos las estructuras visuales.

Un enfoque para generar interfaces web de *mockups* es presentado en [11]. Los autores presentan un modelo GUI intermedio que es independiente de las plataformas origen y destino y que representa la estructura lógica de la GUI. El reconocimiento de las relaciones de contención entre elementos se basa en una estrategia similar a la que nosotros aplicamos para detectar las regiones. La principal diferencia de su enfoque es que el usuario debe indicar el tipo de *layout* a aplicar y no se realiza una inferencia del *layout* como la que se ha descrito para obtener la combinación de *layouts*.

En [12] se presenta un método para generar interfaces con *layouts* fluidos y elásticos a partir de *mockups* extraídos de un editor WYSIWYG. Primero se infiere el *layout* como una jerarquía de cajas horizontales y verticales y entonces implementa el *layout* con reglas HTML/CSS. Los autores usan un algoritmo de *backtracking* que guarda alguna similitud con el nuestro. La principal diferencia con nuestro enfoque es su alta dependencia del sistema de *layout* HTML/CSS, pues realmente ellos sólo consideran un tipo de *layout*: flujo horizontal/vertical de elementos.

MockupDD [13] es una metodología ágil basada en modelos para conseguir una participación activa de los usuarios en el desarrollo GUI. Los usuarios y desarrolladores crean *mockups* que son convertidos en un modelo de interfaz de usuario abstracto del cual se pueden generar modelos de presentación y prototipos GUI dependientes de una tecnología.

Las herramientas de *wireframing* normalmente ofrecen facilidades para generar *mockups* y prototipos. Una herramienta para diseñar

> “El enfoque presentado se basa en los resultados obtenidos en un trabajo previo en el que se ideó una estrategia de ingeniería inversa para extraer el *layout* de GUIs de aplicaciones RAD”

GUIs a diferentes niveles de fidelidad es presentada en [5], la cual soporta la creación de prototipos. Esta herramienta obtiene una descripción de la GUI en términos de un UIDL denominado UsiXML para el cuál existen generadores que producen la GUI final para diferentes plataformas (por ejemplo, *smartphones* o aplicaciones web).

5. La herramienta

Se ha desarrollado un prototipo de la herramienta usando el *Eclipse Modeling Framework*[12] (EMF) de modo que la implementación está basada en modelos creados con el lenguaje de metamodelado Ecore.
La herramienta es un plugin Eclipse que se ha integrado en WireframeSketcher que es la herramienta usada para crear *wireframes*. Este plugin ofrece dos facilidades:

- Generar un modelo *layout* de *wireframes*.
- Generar código para alguna plataforma GUI. En la implementación actual se puede generar código para ZK (páginas web ZUML) y Swing, y está previsto incorporar otras plataformas en el futuro.

WireframeSketcher ayuda a la creación rápida de *mockups* para aplicaciones de escritorio, web y móviles. Hemos elegido esta herramienta debido a que los *mockups* son almacenados como modelos que conforman a un metamodelo Ecore que está disponible con la distribución de la herramienta.

En la **figura 1** mostramos el *wireframe* creado con WireframeSketcher que nos ha servido como ejemplo.

El modo de empleo de la herramienta[13] sería el siguiente:

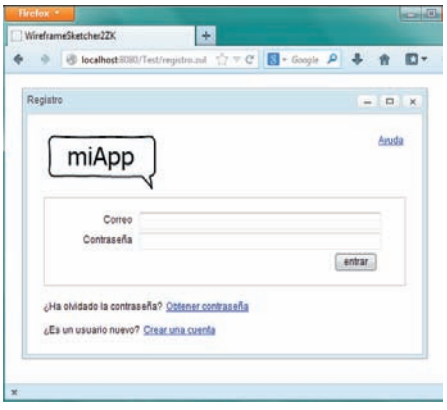

Figura 6. Ventana de registro generada para ZK.

Los desarrolladores crean *wireframes* con WireframeSketcher y discuten los diseños con los usuarios. Tras varias iteraciones, los *stakeholders* validan un *mockup*.

Entonces el modelo que representa este *mockup* podría ser añadido a un proyecto Eclipse y nuestra herramienta sería ejecutada para la generación automática del código que implementa la ventana que corresponde al *mockup*. Nótese que aunque *wireframes* y *mockups* son conceptualmente distintos, nuestra herramienta los trata indistintamente.

En la **figura 6** se muestra la página web ZK que se generaría a partir del modelo de *layout* inferido para el *wireframe* de la **figura 3**. Nótese que el código generado respeta el *layout*, la separación entre *widgets* y el alineamiento con respecto a la ventana principal.

6. Conclusiones

El uso de herramientas de *wireframing* está cada vez más extendido y los desarrolladores pueden ver reducido el esfuerzo de implementación a través de generadores de código que aprovechen la información contenida en los *wireframes* y *mockups*.

En este trabajo hemos presentado un enfoque basado en técnicas MDE cuya principal novedad es el algoritmo para la inferencia del *layout*. A diferencia de los enfoques existentes, nuestra herramienta puede ser integrada con cualquier herramienta de *wireframing* existente y puede generar código para cualquier plataforma GUI.

En la actualidad, estamos extendiendo la herramienta para soportar más plataformas origen y destino, y estamos experimentando con usuarios para validar la validez de las soluciones propuestas.

Referencias

[1] Marco Brambilla, Jordi Cabot, Manuel Wimmer. *Model-Driven Software Engineering in Practice*. Synthesis Lectures on Software Engineering. Morgan & Claypool Publishers, 2012.
[2] Óscar Sánchez Ramón, Jesús Sánchez Cuadrado, Jesús J. García Molina. *Model-driven reverse engineering of legacy graphical user interfaces*. Automated Software Engineering, 2013, (DOI) 10.1007/s10515-013-0130-2.
[3] Ethan Marcotte. *Responsive Web Design,* A Book Appart, 2011.
[4] James A. Landay, Brad A. Myers. Sketching Interfaces: Toward More Human Interface Design. *IEEE Computer Volume 34 Issue 3*, pp. 56-64, marzo 2001.
[5] Adrien Coyette, Suzanne Kieffer, Jean Vanderdonckt. *Multi-fidelity prototyping of user interfaces*. En INTERACT'07, LNCS 4662, pp. 150-164, 2007.
[6] UX Movement. *Why it's important to sketch before you wireframe*. Publicado el 27/08/2012, <http://uxmovement.com/wireframes/why-its-important-to-sketch-before-you-wireframe/>.
[7] Dave Steinberg, Frank Budinsky, Marcelo Paternostro, Ed Merks. *Eclipse Modeling Framework*, 2nd Edition, Addison-Wesley, 2008.
[8] Javier L. Cánovas, Jesús García Molina. *Extracting models from source code in software modernization*. Software and Systems Modeling, septiembre 2012, DOI:10.1007/s10270-012-0270-z.
[9] James Allen. Maintaining Knowledge about Temporal Intervals. *Communications of the ACM, volume 26*, pp. 832-843, noviembre 1983.
[10] Christof Lutteroth. *Automated reverse engineering of hard-coded GUI layouts*. En *AUIC'08, volume 76*, pp. 65-73, ACS, 2008.
[11] José Matías Rivero, Gustavo Rossi, Julián Grigera, Juan Burella, Esteban Robles Luna, Silvia E. Gordillo. *From Mockups to User Interface Models: An Extensible Model Driven Approach*. ICWE Workshops 2010: pp. 13-24.
[12] N. Sinha, R. Karim. Compiling mockups to flexible UIs. *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2013*, ACM, 2013, pp. 312–322.
[13] José Matías Rivero, Julián Grigera, Gustavo Rossi, Esteban Robles Luna, Nora Koch. Towards Agile Model-Driven Web Engineering. *CAiSE Forum 2011*: pp. 142-155.

Notas

[1] Un *wireframe* para un sitio web, también conocido como un esquema de página o plano de pantalla, es una guía visual que representa el esqueleto o estructura visual de un sitio web. El *wireframe* esquematiza el diseño de página u ordenamiento del contenido del sitio web, incluyendo elementos de la interfaz y sistemas de navegación, y cómo funcionan en conjunto. <http://es.wikipedia.org/wiki/Wireframe_(Diseño_web)‎> Último acceso: 10 de febrero de 2014.
[2] En la manufactura y diseño, un *mockup*, o maqueta es un modelo a escala o tamaño real de un diseño o un dispositivo, utilizado para la demostración, evaluación del diseño, promoción, y para otros fines. Un *mockup* es un prototipo si proporciona al menos una parte de la funcionalidad de un sistema y permite pruebas del diseño. <http://es.wikipedia.org/wiki/Wireframe_(Diseño_web)‎> Último acceso: 10 de febrero de 2014.
[3] <http://www.smartclient.com/product/reify.jsp>.
[4] <http://balsamiq.com/>.
[5] <http://www.zkoss.org/>.
[6] <http://wireframesketcher.com/>.
[7] <http://www.eclipse.org/atl/>.
[8] <http://www.omg.org/spec/QVT/1.1/>.
[9] <http://www.eclipse.org/gmt/mofscript/>.
[10] <http://www.eclipse.org/acceleo/>.
[11] <http://www.usixml.org/en/home.html?IDC=221>.
[12] <http://www.eclipse.org/modeling/emf/>.
[13] La herramienta puede ser descargada en <http://www.modelum.es/guizmo>.