Diseño de reglas de adaptación y transformación para interfaces de usuario

Designing adaptation and transformation rules for user interface

Víctor López-Jaquero, PhD., Francisco Montero, PhD.
Departamento de Sistemas Informáticos. Universidad de Castilla-La Mancha, España
{victor,fmontero}@dsi.uclm.es

Recibido para revisión 8 de diciembre de 2009, aceptado 10 de febrero de 2010, versión final 24 de febrero de 2010

Resumen—La especificación de las reglas necesarias para la adaptación de la interfaz de usuario, así como de la reglas de transformación necesarias para la generación de una interfaz de usuario basada en modelos, es un tema de gran importancia para la comunidad de desarrollo de interfaces de usuario, ya que cada día aparecen más y más aproximaciones basadas en modelos, y es además hacia donde las tendencias de desarrollo de software en general están actualmente encaminadas. La ubiquidad en la interacción y los distintos perfiles de usuario son dos de los retos más importantes actualmente en el desarrollo de software. Además, el contexto de uso evoluciona a lo largo del tiempo. Por lo tanto, existe una necesidad real de proporcionar una serie de reglas de adaptación que permitan a la interfaz de usuario cambiar de acuerdo a la evolución del contexto de uso. En este artículo se presenta un metamodelo para la definición de reglas de adaptación de una manera sistemática, para tratar de alcanzar una visión ingenieril del diseño del proceso de adaptación. Finalmente, también se presenta una herramienta, denominada T:XML que permite la especificación de las reglas de adaptación de una manera visual, simplificando el proceso de diseño de adaptaciones para entornos basados en modelos.

Palabras Clave—Entorno de Desarrollo de Interfaces de Usuario Basado en Modelos, Adaptación de la Interfaz de Usuario, T:XML, ISATINE Framework.

Abstract—In a model-based user interface development environment, user interface generation or adaptation rules are required in order to either generate the user interface or adapt it. The design of these rules is becoming more and more important, since the current trends in the development of software in general, and user interface design is no exception, are leading to model-based environments. Ubiquity in interaction and the different user profiles making use of a single application pose a great challenge for software developers, which should design a user interface usable in all the situations and by all the users. Furthermore, the context of use where interaction takes place evolves along time, so the consideration of the context of use should be tackled also at run-time. Thus, the specification of a set of adaptation rules that support the evolution of the user interface according to the evolution of the context of use is required. In this paper a metamodel for the specification of adaptation rules in a systematic manner is introduced. It pursues getting closer to engineering the adaptation design. Finally, a T:XML tool is described. This tool supports the specification of adaptation rules in a visual manner, making easier the design of adaptation for model-based user interfaces.

Keywords—Model-Based User Interface Development, User Interface Adaptation, T:XML, ISATINE Framework.

I. INTRODUCCIÓN

Con la llegada de la ubiquidad a la interacción y los rápidos avances de la tecnología están apareciendo muchos dispositivos heterogéneos. Además, se puede observar como los contextos de uso en que dicho dispositivos están siendo usados van más allá del escritorio.

Por otro lado, el perfil actual de los usuarios de las aplicaciones se está desplazando cada vez más hacia un tipo de usuario menos avanzado. El perfil tradicional de usuario, que incluía incluso algunas capacidades de programación, se está reduciendo drásticamente en porcentaje de la masa global de usuarios.

Esta situación, donde una gran cantidad de factores influyen en la interacción, supone importantes retos para los desarrolladores de interfaces de usuario. Muchos usuarios demandan aplicaciones que puedan ser ejecutadas en distintas

plataformas y entornos. Los usuarios piden aplicaciones que sean capaces de permitir la realización de una misma tarea con distintos dispositivos y en distintos lugares. Sin embargo, la creación de aplicaciones capaces de cubrir esos requisitos es compleja y costosa, ya que en el peor de los escenarios, una interfaz de usuario distinta tendría que ser creada para cada uno de los contextos de usos soportados.

Para solventar este problema, entre otros, el desarrollo de interfaces de usuario basado en modelos [8] se usa en el desarrollo de interfaces de usuario. Este tipo de desarrollo persigue la sistematización del desarrollo de las interfaces de usuario a partir de la creación de una serie de modelos. Estos modelos son más tarde transformados en el código que realmente ejecutará el usuario. Sin embargo, son necesarias una serie de reglas que describan cómo se debe generar la interfaz de usuario a partir de los modelos para cada uno de los contextos de uso soportados.

En la mayoría de los casos, el código final para cada contexto de uso destino no puede ser generado a priori en las etapas de diseño, ya que el propio contexto de uso evoluciona a lo largo del tiempo. Un ejemplo de este tipo de entornos es un sistema de e-Learning, donde el usuario (el conocimiento que él/ella tiene) evoluciona conforme el aprendizaje va avanzando. La aproximación más común para resolver este problema es facilitar un conjunto de reglas de adaptación. Dichas reglas de adaptación facilitan los artefactos necesarios para que la interfaz de usuario evolucione de acuerdo a la evolución del usuario.

De esta manera, las reglas de adaptación son realmente necesarias en las actuales prácticas de desarrollo de interfaces de usuario. Sin embargo, en la mayoría de los sistemas con capacidades de adaptación, dichas reglas son diseñadas de una manera ad-hoc. Por lo tanto, la mayoría de estos sistemas utilizan lo que nosotros denominamos adaptación hardcoded. En este tipo de adaptación, el código necesario para realizar las adaptaciones está totalmente mezclado con el resto del código de la aplicación, haciendo que la reutilización de las capacidades de adaptación sea prácticamente imposible. El camino que conduce hacia la utilización de un nivel de abstracción mayor en el diseño de las adaptaciones es lo que nosotros denominamos adaptación ingenieril [4].

En este artículo se presenta un metamodelo para la especificación de reglas de adaptación de propósito general y reutilizables. Este metamodelo contribuye a mejorar el proceso ingenieril de adaptación. Finalmente, también se describe una herramienta que permite el diseño de la adaptación de la interfaz de usuario producida por dichas reglas.

II. DISEÑO DE LA ADAPTACIÓN DE INTERFACES DE USUARIO

Como se ha mencionado anteriormente, las aproximaciones basadas en modelos para el desarrollo de interfaces de usuario hacen uso de una serie de modelos para especificar el sistema. Los modelos más abstractos son transformados en modelos más concretos, hasta llegar al código final. Para ello son necesarias un conjunto de reglas que especifiquen cómo unos modelos se transforman en otros.

En la literatura se pueden encontrar distintas aproximaciones para la transformación de unos modelos en otros, o en código. La mayoría de dichas aproximaciones son de propósito general, pero algunas de ellas son específicas (o han sido adaptadas) para su uso en el desarrollo de interfaces de usuario.

De acuerdo a [1], las siguientes aproximaciones genéricas para la transformación de modelos se usan actualmente: (1) aproximaciones basadas en la manipulación directa, (2) aproximaciones dirigidas por la estructura, (3) aproximaciones relacionales, (4) aproximaciones basadas en la transformación de grafos, (5) aproximaciones basadas en plantillas, (6) aproximaciones operacionales, (7) aproximaciones híbridas, (8) otras aproximaciones: como por ejemplo XSLT (eXtensible Stylesheet Transformation Language) [11], que no cae dentro de ninguna de las categorías anteriores, pero que se usa bastante habitualmente para realizar transformación de modelos.

Por otro lado, existen algunas aproximaciones en entornos de transformación de modelos para interfaces de usuario que adaptan algunas de las técnicas anteriormente mencionadas. Por ejemplo, en [2], una adaptación de la técnica de transformación de grafos es propuesta para el desarrollo de interfaces de usuario. En esta aproximación, el modelo de interfaz de usuario es expresado en el lenguaje de descripción de interfaces de usuario UsiXML [10], que es transformada utilizando gramáticas de grados atribuidos. Para ello se usa el API de la herramienta AGG [9].

De acuerdo a [5], existen una serie de características deseables que una aproximación basada en modelos debiera cubrir:

- Reglas de transformación: toda aproximación debe facilitar un lenguaje que permita la especificación de reglas de transformación.
- Control de la aplicación de las reglas: es necesario un mecanismo que controle qué reglas se debe aplicar, ya que no todas las reglas son aplicables dado un contexto de uso concreto.
- Organización de las reglas: la reutilización de las reglas es un factor clave. Mediante la reutilización de las reglas no estamos simplemente reutilizando el tiempo necesario para crearlas, sino que también se reutiliza la experiencia adquirida durante el desarrollo.
- Relación origen-destino: es la capacidad de una transformación de generar varios ficheros de salida. Esta característica es importante en la adaptación, ya que normalmente es necesario generar varios ficheros de salida. Por ejemplo, si deseamos generar una presentación en HTML a partir de una serie de modelos, normalmente la presentación de la página web se realizada a partir de un

fichero CSS (Cascade Style Sheet) aparte.

- Incrementalmente: ya que se pueden producir cambios tanto en los requisitos como en el contexto de uso, es deseable la posibilidad de poder actualizar un modelo existente. Esta característica es especialmente importante en la adaptación, ya que es necesario actualizar los modelos para realizar las adaptaciones.
- Direccionabilidad y trazabilidad: idealmente, las transformaciones debieran ser bidireccionales, y un registro de qué transformaciones han sido aplicadas se debería mantener. Esta característica daría soporte a un problema muy habitual en los sistemas con capacidades de adaptación: deshacer una adaptación. Cualquier sistema con capacidades de adaptación debiera trabajar de acuerdo al principio de que puede que las adaptaciones aplicadas no le gusten al usuario. Por lo tanto, capacidades para deshacer una adaptación debieran estar presentes en cualquier entorno de adaptación.

Este conjunto de características deseables son aplicables tanto a la generación como a la adaptación de interfaces de usuario.

Actualmente, la única aproximación que soporta todas estas características es QVT [6]. Sin embargo, la manera en que QVT especifica las transformaciones es compleja, y requiere conocimiento adicional del lenguaje OCL (Object Constraint Language). Además, la aplicación de las transformaciones diseñadas usando los motores actuales de QVT implica el uso de una cantidad considerable de recursos de computación.

Nosotros hemos estado trabajando utilizando una aproximación basada en grafos durante los últimos años [3, 4]. Aunque la manera en que se especifican las transformaciones usando la herramienta AGG [9] es bastante visual, la cantidad de recursos necesarios para la transformación de una interfaz de usuario media es demasiado lenta para su uso en tiempo real. Esto es especialmente cierto cuando hablamos de dispositivos móviles, cuyas capacidades de computación son mucho más escasas que en los sistemas de escritorio. Además, algunos de estos dispositivos móviles no permiten la utilización del Java Run-Time Environment necesario para ejecutar el motor de AGG. La transformación basada en grafos soporta todas las características deseables para una aproximación de transformación de modelos, excepto la direccionabilidad y la trazabilidad.

Por otro lado, XSLT es una aproximación muy popular entre los desarrolladores de software, y existen muchas herramientas que permiten el diseño y ejecución de XSLT. XSLT es una herramienta versátil para expresar las transformaciones necesarias para las reglas de adaptación. Sin embargo, la organización de las reglas, el control de aplicación de las reglas o la incrementalidad deben ser implementadas por el desarrollador. Además, no soporta direccionalidad ni trazabilidad.

A partir de nuestra experiencia en diseño de reglas de adaptación usando una aproximación basada en la transformación de grafos, hemos trabajado en una aproximación que cubra todos nuestros requisitos. Uno de los requisitos que normalmente no se contempla es la portabilidad. La portabilidad es de especial importancia en la adaptación de interfaces de usuario, ya que debe funcionar para todas las aproximaciones para las que el software fue diseñado (PCs, PDA, teléfonos móviles, etc). Este requisito puede ser cubierto usando una arquitectura cliente/servidor, pero hace que los dispositivos tengan que contar constantemente con una conexión de red, que puede finalmente agotar las baterías de los dispositivos portátiles.

Por lo tanto, en nuestra aproximación usamos transformación basada en grafos para el diseño de las transformaciones visualmente, pero internamente dichas transformaciones son convertidas automáticamente en transformaciones XSLT. Dicho código puede ser ejecutado en prácticamente cualquier plataforma, usando las implementaciones basadas en Java o Javascript disponibles, como AJAXSLT.

Sin embargo, mediante el uso de esta aproximación no se soporta directamente la trazabilidad y la direccionabilidad. Para soportar la direccionabilidad hemos añadido una capa de "deshacer" sobre XSLT. Esta capa aprovecha la información sobre las trasformaciones almacenada en los modelos transformados. Este tipo de información se almacena utilizando correspondencias (mappings) o enlaces (links) [10], que expresan las relaciones entre los elementos en el modelo original y los elementos de los modelos transformados. Estas correspondencias pueden ser representadas usando lenguajes de descripción de interfaces de usuario (UIDL), como UsiXML [10].

El requisito de relación origen-destino de las aproximaciones de transformación anteriormente comentado puede ser implementado mediante la utilización de XSLT, en sus versiones 1.1 o 2.0. En dichas versiones varios ficheros de salida pueden ser producidos a partir de una sola transformación.

La definición de una regla de adaptación no es sólo la especificación de la transformación que produce sobre los modelos, sino que también incluye el contexto en el que es aplicable y los eventos que disparan la regla de adaptación. Desafortunadamente, no existe ningún estándar en el diseño de interfaces de usuario para especificar cómo se deben describir las reglas de adaptación. Por lo tanto, la reutilización de las reglas de adaptación de una aplicación en otra es un trabajo tedioso, puesto que las reglas deben ser convertidas de un metamodelo a otro. A continuación se describe el metamodelo que se propone para la definición de reglas de adaptación.

III. METAMODELO DE REGLAS DE ADAPTACIÓN DE INTERFACES DE USUARIO

En la figura 1 se puede observar el metamodelo para las reglas de adaptación propuesto. En este metamodelo, una regla de adaptación se especifica en términos de los eventos del contexto

de uso que disparan la regla de adaptación, los sensores que producen dichos eventos, los datos que a los que la regla de adaptación necesita acceder (leer/escribir), la transformación que debe producirse como resultado de la aplicación de la regla, y la precondición del contexto. La regla de adaptación es disparada por eventos del contexto. Estos eventos representan los distintos cambios en el contexto de uso detectados por los sensores. Un evento del contexto puede ser producido por más de un sensor, de manera que la misma regla puede ser disparada fácilmente por distintos eventos de entrada.

En la figura 1 (parte inferior derecha), se pueden apreciar los detalles del metamodelo de sensores para las reglas de adaptación. Los sensores pueden ser software o hardware. Los sensores hardware están integrados en la plataforma hardware donde se ejecuta la aplicación. Por otro lado, los sensores software son programas incluidos en las aplicaciones que capturan información. Por ejemplo, un sensor software se podría añadir para capturar el tiempo ocioso del usuario cuando usa la aplicación. Tanto los sensores software como los sensores hardware puede especializarse para detectan cambios en la plataforma, el usuario o el entorno físico. Uno de los atributos más importantes de un sensor es el tipo de datos (dataType). Este atributo almacena el tipo de datos de la información que captura el sensor.

Cada regla tiene una precondición del contexto. La precondición del contexto específica las condiciones necesarias que el contexto de uso actual debe cumplir para que la regla de adaptación sea aplicable. Por ejemplo, se puede expresar que el valor del parámetro capturado por el sensor debe ser mayor que un valor específico.

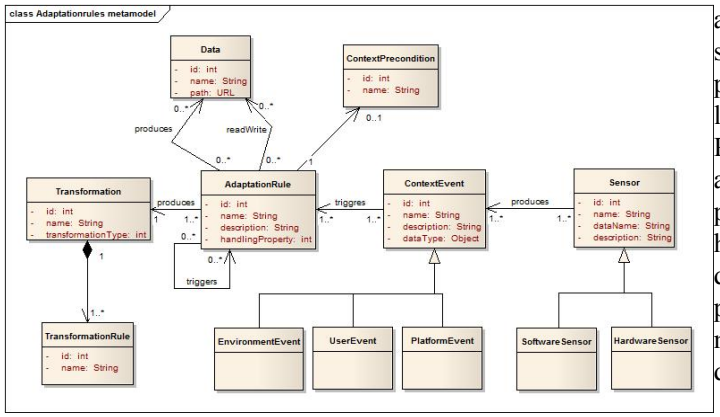

Figura 1. Metamodelo reglas de adaptación, sensores y eventos del contexto

La clase AdaptationRule contiene la información genérica de la regla. Incluye su nombre, una descripción de lo que hace la regla, para que sea más sencillo entender su propósito y una prioridad. El primer nivel del control de selección de reglas es elegir aquellas que tengan una prioridad menor. Para la evaluación de las precondiciones del contexto y la ejecución de la regla de adaptación, es necesario acceder a ciertos modelos y datos. En la clase data se almacenan los recursos necesarios para la ejecución/evaluación de la regla. Esto es especialmente importante cuando se usan sistemas multi-agente [3] o distribuidos en general para realizar el proceso de adaptación, ya que los recursos están distribuidos y deben ser compartidos por distintas entidades.

La clase transformation representa la especificación de la propia adaptación de la interfaz de usuario. Contiene un conjunto de transformaciones de modelos. Estas transformaciones pueden ser expresadas en cualquier lenguaje de transformación de modelos. El atributo transformationType almacena en qué lenguaje se encuentran dichas transformaciones.

IV. T:XML: UNA HERRAMIENTA PARA EL DISEÑO DE REGLAS DE ADAPTACIÓN

T:XML es una herramienta que permite la especificación, organización y validación de las reglas de adaptación. Esta herramienta utiliza una aproximación para la especificación de reglas inspirada en la herramienta AGG [9]. Sin embargo, la notación visual usada en la herramienta ha sido adaptada para hacer más sencilla la especificación de las reglas.

Actualmente, la herramienta permite la definición de reglas de adaptación que transforman modelos expresados en UsiXML. T:XML organiza las transformaciones en carpetas de proyectos para poder manejar fácilmente las reglas y los modelos. Cada regla se corresponde con un proyecto, y cada proyecto puede contener varias reglas de transformación. El orden en que estas reglas de transformación deben ser aplicadas puede ser cambiado también.

El objetivo de la herramienta es la generación de las adaptaciones para distintos lenguajes destino, como pueden ser XSLT [11] o QVT [6]. Mediante la generación de código para distintos lenguajes de transformación se pretende maximizar la reutilización y portabilidad de las adaptaciones diseñadas. Para comprobar la viabilidad del proyecto, la herramienta permite actualmente la generación de código XSLT automáticamente a partir de las adaptaciones diseñadas visualmente. Además, la herramienta permite la generación del código final de la interfaz de usuario en el lenguaje OpenLaszlo [7], para poder previsualizar y depurar las adaptaciones diseñadas. De esta manera, el diseñador puede descubrir fácilmente efectos no deseados o errores en las adaptaciones diseñadas.

Hay cuatro zonas en el espacio de diseño de la aplicación: (1) la barra de herramientas principal, que está situada en la parte superior de la ventana. Dicha barra permite acceder a las funciones de manejo de ficheros (guardar, cargar, etc). (2) en la parte izquierda de la ventana se sitúa el gestor de proyectos. El gestor de proyectos estructura los proyectos y las reglas de adaptación en carpetas usando una estructura de árbol. En este árbol el diseñador puede navegar o seleccionar las reglas de adaptación y las reglas de transformación. (3) el espacio de diseño de modelos. En esta zona el diseñador puede visualizar

los modelos importados o crear un nuevo modelo desde cero. Para mantener más ordenados los modelos creados, el diseñador pueden expandir y contraer las ramas del modelo que no están actualmente en uso. Las ramas que han sido contraídas son marcadas con el signo "+" (véase el elemento head de la figura 2). (4) zona de edición de atributos. En esta área el diseñador puede modificar los atributos de los elementos del modelo. En la figura 2 dicha área no es mostrada, puesto que sólo se muestra al pulsar con el botón derecho sobre un nodo. Algunos nodos tienen un marca blanca (tick) que indica que dicho elemento tiene un valor asociado en el modelo de recursos (véase el botón OK en la figura 2). El modelo de recursos es usado en UsiXML para separar los contenidos de su presentación, permitiendo así su internacionalización. Los recursos pueden ser modificados en la pestaña Resources.

Cuando el usuario muestra el diseñador de transformaciones aparece una nueva área de trabajo que contiene 4 partes. Tiene una barra de herramientas con botones y 3 áreas para la especificación de reglas de transformación. Para ilustrar cómo se especifican las reglas de transformación se ha utilizado un ejemplo sencillo. En el ejemplo se desea diseñar una regla de adaptación que se ejecutará cuando el espacio de pantalla asignado a nuestra aplicación se reduzca. La idea general de esta adaptación es que se reemplacen todos los grupos de botones de radio por listas desplegables (comboBox) para reducir el espacio necesario para representar tareas de selección en la interfaz de usuario.

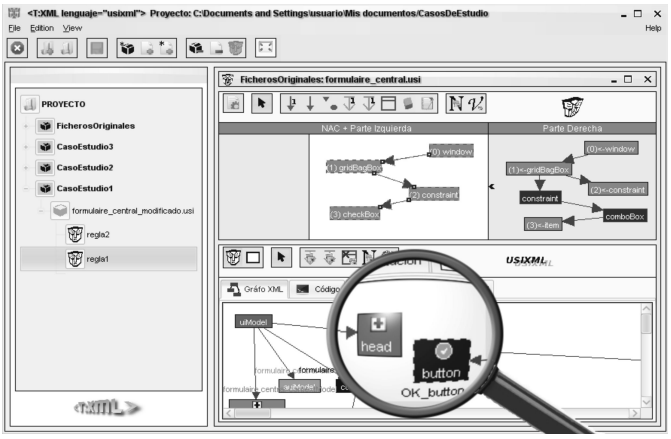

Figura 2. Espacio de trabajo de T:XML

El área central está dedicada a la especificación de la parte izquierda de la transformación. En esta área el diseñador define el patrón que debe ser buscado en el modelo. En nuestro ejemplo estamos buscando grupos con botones de radio (véase la figura 3a). La parte de la derecha está dedicada a la especificación de la parte derecha de la transformación. En esta área el diseñador especifica cómo transformar aquellos nodos del modelo que coincidan con el patrón especificado en la parte izquierda. Aquellos elementos de la parte izquierda que no aparezcan en la parte derecha serán eliminados del modelo. En nuestro ejemplo se ha arrastrado la parte izquierda a la parte derecha para su posterior edición (véase la figura 3b). Los números que parecen en los elementos y que coinciden en parte izquierda y la derecha representan una correspondencia entre aquellos nodos que tienen el mismo número. Como en esta primera transformación deseamos reemplazar los botones de radio por los elementos (ítems) de una lista desplegable, en la parte derecha el tipo del elemento radioButton ha sido cambiado a item.

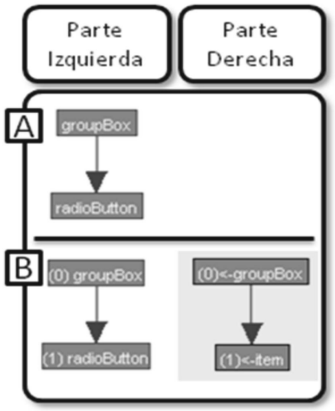

Figura 3. Ejemplo de regla de transformación

V. DISCUSIÓN

Cuando se crea una regla de transformación de modelos, un aspecto importante a considerar es la diferencia entre aquellas transformaciones que transforman unos modelos en otros modelos (model-to-model transformation) y las que producen código (model-to-code transformation). Las transformaciones de modelos a código en el desarrollo de interfaces de usuario son normalmente usadas en el nivel final, cuando se desea generar la interfaz de usuario que será presentada al usuario. Por ejemplo, cuando se genera código Java a partir del lenguaje UIML. Nuestra herramienta permite realizar transformaciones modelo a modelo, cubriendo todos los modelos habitualmente usados en un entorno de desarrollo basado en modelos (tareas, dominio, contexto, presentación, etc). Sin embargo, el desarrollo de generadores de código para un lenguaje basado en XML (como XHTML) también sería posible.

Otro de los puntos abiertos en el desarrollo de adaptaciones para interfaces de usuario es el momento en que los modelos son transformados en código final para que el usuario pueda manipularlos. Debemos llegar a un equilibrio entre las adaptaciones generadas a partir de reglas y el generador de código. Si el generador de código es demasiado simple, todas las adaptaciones tendrán que ser creadas por los diseñadores de las aplicaciones. Por otro lado, si el generador de código es demasiado complejo, será complicado personalizar las capacidades de adaptación, al estar estas integradas casi totalmente en el generador de código.

Cuando el diseñador crea reglas de adaptación es importante tener en cuenta que si las reglas de adaptación son demasiado

detalladas será difícil volver a usarlas en otras aplicaciones. Una buena aproximación sería añadir una capa extra para el control preciso de las adaptaciones usando una aproximación clásica basada en reglas, como pueden ser una herramienta para el desarrollo de sistemas expertos, como CLIPS.

VI. CONCLUSIONES Y TRABAJO FUTURO

La especificación de reglas de transformación y adaptación es un tema de gran interés para la comunidad de desarrolladores de interfaces de usuario, ya que cada vez van apareciendo más aproximaciones basadas en modelos.

En este artículo se aporta un metamodelo para la especificación de reglas de adaptación. Dicho modelo permite la especificación de las transformaciones de la interfaz de usuario, así como los eventos que disparan las reglas y los sensores que capturan la información del contexto de uso donde la aplicación se ejecuta.

Además, se presenta una herramienta (T:XML) que permite la especificación de las reglas de adaptación para entornos basados en modelos. La definición de las reglas se simplifica en gran manera al proporcionar una notación gráfica que oculta la complejidad detrás de la creación de reglas de adaptación en un lenguaje de programación. La herramienta genera automáticamente el código en XSLT necesario para la aplicación de las transformaciones diseñadas visualmente.

Como trabajo futuro se desea añadir otros lenguajes de descripción de interfaces de usuario, como UIML o XIML. Además, para mejorar la versatilidad de la herramienta se desea permitir la generación para otros lenguajes de transformación. Actualmente, estamos trabajando en la generación para el lenguaje QVT, ya que es el más potente de los lenguajes analizados, al soportar transformaciones bidireccionales y trazabilidad de las transformaciones aplicadas.

AGRADECIMIENTOS

Este trabajo ha sido parcialmente financiado por el proyecto TIN2008-06596-C02-01 del Gobierno de España y los proyectos PII2I09-0146-8894 y PEII09-0054-9581 de la Junta de Comunidades de Castilla-La Mancha

BIBLIOGRAFÍA

[1] K. Czarnecki, S. Helsen. Feature-Based Survey of Model Transformation Approaches. IBM Systems Journal, 45(3), 2006, pp. 621-645

[2] Q. Limbourg, J. Vanderdonckt, B. Michotte, L. Bouillon, V. López-Jaquero: USIXML: a Language Supporting Multi-Path Development of User Interfaces. In: Proc. of 9th IFIP Working Conf. on Engineering for Human-Computer Interaction jointly with 11th Int. Workshop on DSVIS EHCI-DSVIS'2004 (Hamburg, July 11-13, 2004). LNCS, Vol. 3425, Springer-Verlag, Berlin (2005) 200-220.

[3] V. López-Jaquero, F. Montero, J.P. Molina, P. González, A. Fernández-Caballero. 'A Multi-Agent System Architecture for the Adaptation of User Interfaces'. 4th International Central and Eastern European Conference on Multi-Agent Systems. 15-17 September 2005, Budapest, Hungary. In Multi-Agents Systems and Applications IV. M. Pechoucek, P. Petta, L. Zsolt Varga (Eds.) LNAI 3690, Springer-Verlag, Germany, 2005..

[4] V. López Jaquero, J. Vanderdonckt, F. Montero, P. González. Towards an Extended Model of User Interface Adaptation: the ISATINE framework. Proc. of Engineering Interactive Systems 2007 (IFIP WG2.7/13.4 10th Conference on Engineering Human Computer Interaction jointly organized with IFIP WG 13.2 1st Conference on Human Centred Software Engineering and DSVIS - 14th Conference on Design Specification and Verification of Interactive Systems) EIS'2007 (Salamanca, 22-24 March 2007). M.B. Harning, J. Gulliksen (eds.), Springer-Verlag, Berlin, 2007

[5] E.Navarro. ATRIUM: Architecture traced from requirements by applying a unified methodology. Tesis doctora.l Universidad de Castilla-La Mancha, 2007.

[6] QVT. MOF Query/Views/Transformations final adopted specification. OMG, 2005

[7] OpenLaszlo. Rich Internet Applications framework. http://www.openlaszlo.org

[8] A.R. Puerta. A Model-Based Interface Development Environment. IEEE Soft. 14, 4. 1997.

[9] Taentzer, G. AGG: A Tool Environment for Algebraic Graph Transformation. Proc. of the Int. Workshop on Applications of Graph Transformations with industrial Relevance. LNCS 1779, 2000.

[10] UsiXML. USer Interface eXtensible Markup Language. http://www.usixml.org

[11] XSL Transformations (XSLT), version 1.0. http://www.w3c.org/TR/xslt

Víctor López-Jaquero, Ostenta el grado de Doctor en Informática por la Universidad de Castilla-La Mancha desde el año 2005. Es profesor del Departamento de Sistemas Informáticos de la Universidad de Castilla-La Mancha desde año 2000, e imparte cursos relacionados con bases de datos y la Interacción Persona-Ordenador. Su actividad investigadora se centra en la Interacción Persona-Ordenador, y especialmente en la generación automática de interfaces de usuario adaptables y la utilización de sistemas multi-agente para proporcionar las capacidades de adaptación. Es autor de más de 40 artículos en congresos relevantes y revistas en temas relacionados con el diseño de interfaces de usuario, incluyendo, pero no limitado a, descripción de interfaces de usuario basado en XML y la generación de interfaces de usuario para distintas plataformas destino siguiendo una aproximación basada en modelos, modelos de calidad y usabilidad. Es autor de más de 40 artículos en congresos relevantes y revistas en temas relacionados con el diseño de interfaces de usuario, incluyendo, pero no limitado a, descripción de interfaces de usuario basado en XML y la generación de interfaces de usuario para distintas plataformas destino siguiendo una aproximación basada en modelos, sistemas multi-agente y modelos de adaptación. Víctor López es miembro de la ACM y de AIPO.

Francisco Montero. Ostenta el grado de Doctor en Informática por la Universidad de Castilla-La Mancha desde el año 2005. Es profesor del Departamento de Sistemas Informáticos de la Universidad de Castilla-La Mancha desde año 2000, e imparte cursos relacionados con el diseño de software y la Interacción Persona-Ordenador. Su actividad investigadora se centra en la Interacción Persona-Ordenador, y especialmente en la generación automática de interfaces de usuario integrando en su desarrollo modelos de calidad para mejorar la usabilidad y accesibilidad global de los sistemas. Es autor de más de 40 artículos en congresos relevantes y revistas en temas relacionados con el diseño de interfaces de usuario, incluyendo, pero no limitado a, descripción de interfaces de usuario basado en XML y la generación de interfaces de usuario para distintas plataformas destino siguiendo una aproximación basada en modelos, modelos de calidad y usabilidad. Francisco Montero es miembro de la IEEE, de la ACM y de AIPO.