Modelado de la Usabilidad en Entornos de Desarrollo Dirigidos por Modelos[1]

José Ignacio Panach[1], Francisco Valverde[1], Óscar Pastor[1], Natalia Juristo[2]

[1]Centro de Investigación en Métodos de Producción de Software
Universidad Politécnica de Valencia
Camino de Vera s/n, 46022 Valencia
{jpanach, fvalverde, opastor}@pros.upv.es
[2]Universidad Politécnica de Madrid
Campus de Montegancedo, 28660 Boadilla del Monte
natalia@fi.upm.es

Abstract. Los métodos *Model-Driven Development (MDD)* permiten representar, mediante un Modelo Conceptual, un sistema de forma abstracta. A partir del Modelo Conceptual y mediante reglas de transformación, un Compilador de Modelos puede generar el código que implemente el sistema. El objetivo de este trabajo es el de definir un método para incorporar mecanismos de usabilidad desde el Modelo Conceptual hasta la generación de código en cualquier entorno MDD. Por un lado se deben añadir nuevas Primitivas al Modelo Conceptual que representen los atributos de usabilidad. Por otro lado, el Compilador de Modelos se debe modificar para ser capaz de reconocer las nuevas Primitivas y generar el código conforme a lo representado en ellas. El método propuesto está basado en mecanismos de usabilidad extraídos de la literatura. Como ejemplo, se ha aplicado el método propuesto a OO-Method, un entorno MDD capaz de generar aplicaciones totalmente funcionales a partir de un Modelo Conceptual.

Keywords: Usabilidad, Desarrollo Dirigido por Modelos, Modelo Conceptual

1 Introducción

La comunidad Interacción Persona Ordenador ha propuesto recomendaciones para mejorar la usabilidad de los sistemas. Muchas de estas recomendaciones no sólo implican cambios en la interfaz, sino que afectan a la arquitectura por tener implicaciones funcionales (Ej. la acción deshacer). Estas recomendaciones implican cambios en la arquitectura si se incluyen una vez ésta ha sido diseñada. Para evitar estos cambios, Bass [3] y Folmer [8] han propuesto la incorporación de características de usabilidad desde las primeras fases de desarrollo. Sin embargo, esas propuestas tienen algunas desventajas: (1) Mayor complejidad en el desarrollo; (2) Errores en la captura de requisitos arrastran el problema hasta la implementación [12]; (3) Requisitos variables [9]. Para evitar estos problemas, proponemos la incorporación de características de usabilidad en un proceso de desarrollo dirigido por modelos (*Model-Driven Develop-*

[1] Este trabajo se ha desarrollado con el soporte del MEC mediante el proyecto SESAMO TIN2007-62894 y cofinanciado por FEDER.

ment, MDD [13]). Un entorno MDD parte de un Modelo Conceptual donde se presenta el sistema de forma abstracta. Al Modelo Conceptual se pueden aplicar reglas de transformación para generar el código que implemente lo especificado en el Modelo Conceptual (Figura 1). Estas transformaciones se pueden automatizar mediante un Compilador de Modelos. En este caso, el proceso MDD se conoce con el nombre de *Tecnología de Transformación de Modelos (TTM).* La contribución de este trabajo es la de definir un método para incluir características de usabilidad dentro de una TTM. Este método se ha denominado MIMAT *(Método de Incorporación de Mecanismos de usAbilidad a una TTM).* Al tratar la usabilidad en una TTM, los cambios producidos por la variabilidad de los requisitos o por la detección tardía de errores sólo implicarán cambios en el Modelo Conceptual, ya que el código se genera automáticamente.

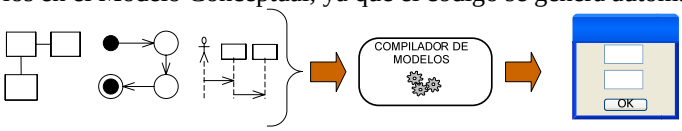

Fig. 1. Proceso de generación de código en una Tecnología de Transformación de Modelos

Para extraer las propiedades de usabilidad a incluir en el Modelo Conceptual, nos hemos basado en trabajos existentes de la literatura, especialmente en los trabajos de Juristo et al [10][11], en los que se han identificado un conjunto de características de usabilidad que incluyen aspectos funcionales. A estas características se les conoce como *Functional Usability Features (FUF).* Cada FUF tiene un objetivo principal que se puede especializar en objetivos más detallados llamados *mecanismos de usabilidad.* Por ejemplo, el FUF *Retroalimentación* está formado por varios mecanismos que persiguen el objetivo de proporcionar retroalimentación al usuario: retroalimentación del estado del sistema y retroalimentación de progreso. Se han elegido estos mecanismos por dos de los elementos con los que se definen:

- **Guías para la captura de requisitos:** cada mecanismo de usabilidad va acompañado de una lista de preguntas para capturar requisitos de usabilidad. Se han utilizado estas guías para extraer las propiedades de los mecanismos que debe especificar el analista. Estas propiedades darán lugar a nuevas Primitivas Conceptuales.
- **Patrones de usabilidad:** cada mecanismo de usabilidad va acompañado de un patrón de usabilidad. Estos patrones esbozan cómo implementar los mecanismos. Se han utilizado estos patrones para identificar los cambios que habría que realizar en el Compilador de Modelos para generar el código que implemente los mecanismos.

La estructura del documento es la siguiente. La sección 2 resume trabajos relacionados de la literatura. La sección 3 introduce el concepto de desarrollo dirigido por modelos. La sección 4 presenta nuestra propuesta para incorporar los mecanismos de usabilidad a cualquier entorno MDD. La sección 5 aplica nuestra propuesta a OO-Method. Por último, la sección 6 presenta las conclusiones del trabajo.

2 Estado del arte

Siguiendo la propuesta de tratar la usabilidad desde las primeras fases de desarrollo promovida por Bass [3] y Folmer [8], varios autores han propuesto la incorporación

de la usabilidad desde la fase de captura de requisitos. Entre estos trabajos se encuentran los de Adikari [2]. Este autor propone dos modelos para representar los requisitos de usabilidad: (1) *Modelo de Usuarios*: modela los atributos del usuario a tener en cuenta, como experiencia o factores culturales. (2) *Modelo de Usabilidad*: representa los atributos de usabilidad. El principal inconveniente que hemos encontrado en este trabajo es que no se describe de manera detallada cómo construir ambos modelos.

Otros autores como Cysneiros [6] han aplicado la notación i* [21] para la captura de requisitos de usabilidad. Cysneiros ha definido en i* un catálogo que guíe a los analistas a través de las distintas posibilidades de usabilidad. Los inconvenientes de esta propuesta se derivan del uso de la notación i* [7]: ambigüedad; especificación lejos del lenguaje natural; pueden aparecer contradicciones en un mismo modelo.

Las propuestas que incorporan la usabilidad en la fase de captura de requisitos, como las de Adikari y Cysneiros, tienen como principal inconveniente que errores en la fase de captura de requisitos o la variación de requisitos hacen que se propaguen errores a las siguientes fases. En nuestra propuesta, aunque existan errores y variaciones, éstos sólo afectan al Modelo Conceptual, ya que la fase de generación de código se realiza de manera automática por el Compilador de Modelos.

Por otro lado, hay autores que proponen el uso de patrones de usabilidad. En este grupo de autores cabe destacar, entre otros, el trabajo de Tidwell [19], que propone una serie de patrones de interacción para conseguir interfaces usables. En el mismo ámbito que Tidwell están los patrones de usabilidad de Perzel [16], cuyo trabajo está más orientado a resolver problemas de usabilidad específicos de la Web. Otro de los autores que ha definido patrones para mejorar la usabilidad de los sistemas es Welie [20]. Los patrones de Welie son similares a los de Tidwell y Perzel pero con la diferencia de que Welie hace una distinción explícita entre la perspectiva del *usuario* y la perspectiva del *diseño*. La principal diferencia entre los patrones de usabilidad de Tidwell, Perzel y Welie y nuestra propuesta es el nivel de abstracción. Los patrones proponen incorporar la usabilidad a nivel de diseño, mientras que nuestra propuesta es a nivel de análisis. Además, el uso de patrones de usabilidad implica siempre una fase de implementación, mientras que en nuestra propuesta sólo hay que incluir las reglas de generación del mecanismo de usabilidad en el Compilador de Modelos y éste generará el código automáticamente.

En cuanto a trabajos que proponen incorporar la usabilidad en un entorno de desarrollo MDD, se encuentran los trabajos de Abrahao [1]. Abrahao propone un Modelo de Usabilidad a partir del cual se puede evaluar y mejorar la usabilidad en entornos MDD. Cada uno de los atributos de usabilidad se representa con al menos una Primitiva Conceptual que mide ese atributo. El principal problema es que muchos de los atributos que componen el Modelo de Usabilidad son subjetivos y dependen del criterio del usuario, por lo que no se pueden evaluar con Primitivas Conceptuales.

Otro de los autores que también ha tratado la usabilidad desde un entorno MDD es Seffah [18]. En su trabajo propone cómo representar patrones de interacción y de usabilidad en entornos de desarrollo basados en las transformaciones entre modelos. Nuestra propuesta difiere de la de Seffah en la manera de representar la usabilidad. Seffah propone el uso de unos modelos específicos, mientras que en nuestra propuesta, la usabilidad se incluiría añadiendo Primitivas Conceptuales nuevas a los Modelos Conceptuales ya existentes. Por tanto, los cambios a realizar en el entorno MDD serían mínimos ya que no habría que añadir nuevos modelos, sino nuevas Primitivas.

3 Desarrollo de Sistemas Dirigido por Modelos (MDD)

Dentro de un entorno de desarrollo MDD [13], se entiende por *modelo o vista* a un conjunto de elementos formales que describen algo construido para un propósito concreto. Cada uno de los modelos representan un aspecto distinto del sistema, por ejemplo, puede haber un modelo para representar la persistencia, otro para representar la funcionalidad y otro para representar la interfaz. Los modelos están compuestos por un conjunto de *Primitivas Conceptuales*, que son los ladrillos utilizados para su construcción. Más formalmente, se pueden definir las Primitivas Conceptuales como un elemento del lenguaje de modelado que permite representar de forma abstracta algún aspecto del sistema. Las Primitivas Conceptuales están agrupadas por modelos.

El conjunto de modelos forman el *Modelo Conceptual***.** El Modelo Conceptual se utiliza para representar las actividades que elicitan y describen el conocimiento general que un sistema particular debe ofrecer [14]. El Modelo Conceptual está compuesto por varios modelos y cada modelo representa una vista diferente del sistema. Los modelos tienen relaciones entre sí que permiten la transformación de un modelo a otro de un nivel menos abstracto. A estas relaciones se les conoce con el nombre de *mapeos*, y hacen posible que en cada nivel de abstracción se expresen diferentes aspectos del sistema, manteniendo sincronizados los modelos de cada nivel. El mapeo que transforma los modelos al código de un lenguaje de programación se denomina *Compilación de Modelos.* La Figura 2 muestra un esquema con todos los conceptos de MDD.

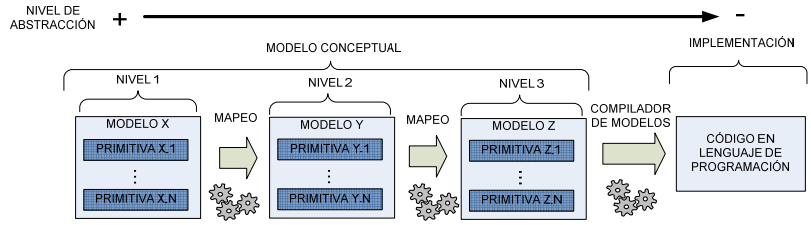

Fig. 2. Vista esquemática de los componentes de un Modelo Conceptual

Cuando el proceso MDD se automatiza, aparece el concepto de *Tecnología de Transformación de Modelos (TTM)*. TTM se define como la capacidad de tomar un modelo abstracto y transformarlo de forma automática en otro modelo (o código).

4 Método de Incorporación de Mecanismos de Usabilidad a una TTM: MIMAT

Esta sección muestra en detalle el método propuesto para introducir mecanismos de usabilidad en una TTM. Este método se ha denominado con el nombre de MIMAT (Método de Incorporación de Mecanismos de usAbilidad a una TTM). La idea es partir de los mecanismos de usabilidad descritos en la literatura [10] e incorporarlos en el Modelado Conceptual de cualquier TTM. El método MIMAT parte de estos mecanismos de usabilidad, dividiéndose en cuatro pasos:

1. **Definición de Formas de Uso:** el propósito que se pretende conseguir con un determinado mecanismo de usabilidad se puede alcanzar mediante distintas representaciones. Cada una de estas representaciones es lo que se ha denominado *Forma de Uso*. Las Formas de Uso se han extraído de las guías de captura de requisitos de usabilidad definidos por Juristo [11]. Por ejemplo, el mecanismo de usabilidad *System Status Feedback* tiene como objetivo el informar al usuario sobre el estado en el que se encuentra el sistema en todo momento. Este objetivo se puede conseguir mediante las siguientes Formas de Uso: (1) Informar del éxito o fracaso en la ejecución de sistemas; (2) Mostrar el estado de la información almacenada en el sistema; (3) Mostrar el estado de las acciones visibles; (4) Informar de falta de recursos. Las distintas Formas de Uso no son excluyentes entre sí.
2. **Identificación de Propiedades:** las guías de captura de requisitos también incluyen preguntas para configurar las distintas aplicaciones del mecanismo de usabilidad. Estas preguntas dan lugar a las *Propiedades*. Las Propiedades de las Formas de Uso son las distintas posibilidades de configuración que tiene la Forma de Uso para adaptarse a los requisitos de usabilidad. Hay dos tipos de Propiedades: Configurables y No Configurables. Los valores óptimos de las Propiedades Configurables dependen de los requisitos del usuario para un caso particular y por tanto es el analista el que debe especificarlos. Por otro lado, las No Configurables son Propiedades que o bien no presentan alternativa en la guía o bien la guía recomienda que su configuración sea siempre la misma en todos los sistemas desarrollados. Este tipo de Propiedades no deben ser especificadas por el analista, ya que su configuración debe ser la misma en todos los sistemas. Por ejemplo, la Forma de Uso *Informar del éxito o fracaso en la ejecución del servicio* tiene dos Propiedades:
   - Selección del servicio: esta Propiedad se utiliza para indicar qué servicios mostrarán si su ejecución ha finalizado o no correctamente. Esta Propiedad es No Configurable, ya que según el criterio ergonómico de Bastien y Scapin llamado *Immediate feedback* [4], debería estar presente en todos los servicios.
   - Visualización del mensaje: esta Propiedad establece el aspecto visual de la información que indica si el servicio se ha ejecutado o no correctamente. Es Configurable porque el analista debe decidir entre las opciones de configuración.
3. **Definición de Primitivas Conceptuales:** este paso es dependiente de la TTM seleccionada para incorporar los mecanismos de usabilidad, ya que los Modelos Conceptuales de cada TTM son exclusivos de la TTM. En este paso se verifica si para cada una de las Propiedades Configurables, existe ya alguna Primitiva Conceptual que la represente en el Modelo Conceptual de la TTM elegida. En caso de que no exista una Primitiva Conceptual o alguna de las posibilidades de configuración de una Propiedad no se pueda representar, es necesario añadir nuevas Primitivas. Por ejemplo, la Propiedad Configurable *Visualización del mensaje* implicaría cambios en la TTM si ésta no tuviera ya alguna Primitiva para representar la forma de visualizar los mensajes de error o de éxito.
4. **Cambios necesarios en el Compilador de Modelos:** el último paso del método para incorporar mecanismos de usabilidad a una TTM consiste en describir los cambios que se deben aplicar al Compilador de Modelos. Este paso, al igual que el de la definición de Primitivas Conceptuales, es dependiente de la TTM seleccionada porque cada Compilador de Modelos es específico de la TTM. Los cambios a incorporar al Compilador de Modelos de la TTM seleccionada se derivan de:

- Las nuevas Primitivas Conceptuales: el Compilador de Modelos debe ser capaz de reconocer las nuevas Primitivas Conceptuales y generar su código.
- Las Propiedades No Configurables: el Compilador de Modelos debe incorporar, de manera transparente para el analista, las Propiedades No Configurables.

La Figura 3 muestra un resumen gráfico de los cuatro pasos que componen el método MIMAT. Una vez se han definido las distintas Formas de Uso (paso 1) y se han identificado sus Propiedades Configurables y No Configurables (paso 2) usando las guías para la captura de requisititos de los mecanismos, el siguiente paso es el de identificar los cambios que se deben realizar en la TTM. Los cambios en el Modelo Conceptual de la TTM se realizan para incorporar Primitivas Conceptuales que modelen las Propiedades Configurables (paso 3). Por último, los cambios en el Compilador de Modelos se hacen para generar el código que representa las opciones modeladas mediante las nuevas Primitivas Conceptuales y además aquellas Propiedades No Configurables (paso 4). Los cambios en el Compilador de Modelos van guiados por los patrones arquitecturales incluidos en la definición de los mecanismos de usabilidad.

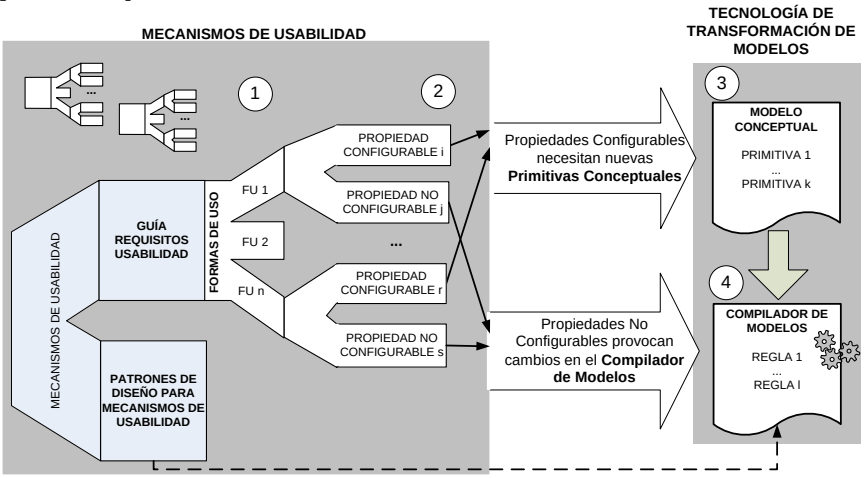

Fig. 3. Proceso en cuatro pasos de MIMAT

La siguiente sección muestra la aplicación práctica de este método al entorno de desarrollo MDD llamado OO-Method [15]. Mediante este ejemplo se pretende ilustrar sobre cómo aplicar el método y además demostrar que la propuesta puede funcionar en un entorno de desarrollo MDD industrial, como es OO-Method.

5 Aplicación de MIMAT a OO-Method

Como ejemplo de TTM [15] para aplicar MIMAT, se ha elegido OO-Method (versión industrial en la herramienta OLIVANOVA [5]). La elección de OO-Method ha sido por sus dos principales ventajas: (1) Genera aplicaciones totalmente funcionales a partir de Modelos Conceptuales; (2) Sus Modelos Conceptuales son lo suficientemente abstractos para facilitar la inclusión de nuevos elementos que representen la usabili-

dad. Como ejemplo hemos tomado el mecanismo de usabilidad *Structured Text Entry* [11], que se utiliza para especificar elementos de la interfaz que facilitan la entrada de datos que requieren un formato específico. Es un ejemplo sencillo pero muy útil para clarificar ideas. A continuación se aplican los pasos de MIMAT para ese mecanismo:

- **Definición de Formas de Uso:** la Tabla 1 presenta las tres Formas de Uso extraídas de la guía de captura de requisitos definidas por Juristo et al [11]. Para cada Forma de Uso se especifica la pregunta de la guía de la que se ha extraído y el objetivo que se persigue con esa Forma de Uso.

Tabla 1. Formas de Uso para el mecanismo de usabilidad *Structured Text Entry*

Forma de Uso	Pregunta de la Guía	Objetivo
Especificar el tipo de visualización del campo de entrada	¿Cuál es el formato de los argumentos de entrada para el usuario?	Especificar el formato del campo de entrada para ayudar al usuario
Definición de máscaras	¿Cómo guiar al usuario para que introduzca los datos en un formato?	Evitar que el usuario introduzca datos con un formato no válido
Valores por defecto	¿Cómo guiar al usuario para que introduzca los datos en un formato?	Guiar al usuario para que sepa cuál es el formato que debe utilizar

- **Identificación de Propiedades:** la Tabla 2 presenta la única Propiedad de la Forma de Uso *Especificar el tipo de visualización del campo de entrada*, la pregunta de la guía de la que se deriva y el objetivo que se pretende conseguir con ella.

Tabla 2. Propiedad de Especificar el tipo de visualización del campo de entrada

Propiedades	Pregunta de la Guía	Objetivo
Tipo de campo de entrada	¿Cuál es el formato de los argumento de entrada?	Determinar el formato del argumento de entrada

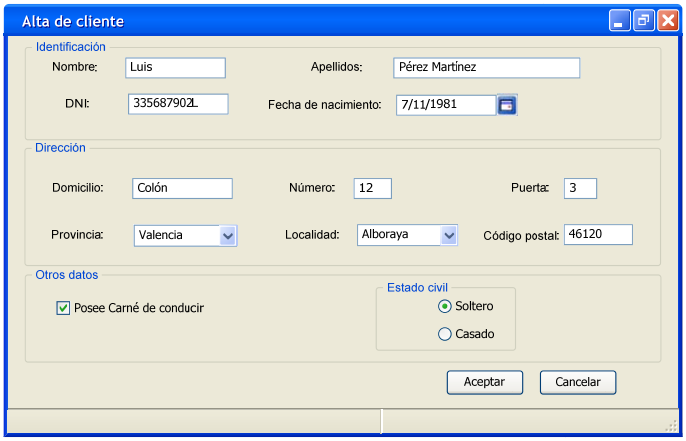

Fig. 4. Ejemplo de pantalla para dar de alta a un nuevo cliente

Como ejemplo práctico de las Propiedades, nos vamos a basar en un sistema de gestión de una empresa de alquiler de vehículos. En concreto, vamos a utilizar el servicio *Alta de cliente*, utilizado por un trabajador desde una de las oficinas de la empresa para dar de alta un nuevo cliente (Figura 4). En la Figura 4 se puede apreciar que la Propiedad *Tipo de campo de entrada* toma el valor de un simple campo de texto en todos los campos excepto en las *fechas* (componente fecha), *listados* (ListBox), *estado civil* (radiobutton) y posee *carné de conducir* (checkbox). La elección de cada uno de los tipos de campos de entrada las habría hecho el analista.

La Tabla 3 presenta las Propiedades de la Forma de Uso *Definición de máscaras*.

Tabla 3. Propiedades de la Forma de Uso *Definición de máscaras*

Propiedades	Pregunta de la Guía	Objetivo
Campo al que se aplicará la máscara	¿En qué campos se deben introducir los datos con un formato específico?	Determinar los argumentos en los que hay que definir una máscara
Expresión regular	¿Cuál es el formato de los argumentos de entrada?	Definir el formato de los datos a introducir con máscara

En el ejemplo de la Figura 4 se ha definido una máscara para el DNI y otra para el código postal. La Propiedad *Campo al que se aplicará la máscara* toma el valor *DNI* y *Código postal*, mientras que la Propiedad *Expresión regular* toma el valor ########[A-Z] para el DNI y el valor ##### para el código postal. La Tabla 4 presenta las Propiedades de la Forma de Uso *Valores por defecto*

Tabla 4. Propiedades de la Forma de Uso *Valores por defecto*

Propiedades	Pregunta de la Guía	Objetivo
Selección del campo	¿En qué campos se deben introducir los datos con un formato específico?	Seleccionar argumentos que mostrarán valores por defecto
Definición del valor por defecto	¿Cuál es el formato de los argumentos de entrada?	Definir el valor por defecto

En el ejemplo de la Figura 4 se ha definido un valor por defecto para el argumento *Posee carné de conducir* (por defecto estaría a *Verdadero*). En este caso, la Propiedad *Selección del campo* tomaría el valor *Posee carné de conducir* y la Propiedad *Definición del valor por defecto* tomaría el valor *Verdadero*. El resto de pasos de MIMAT son dependientes de la TTM elegida (OO-Method en este caso). A pesar de la dependencia de la TTM, los ejemplos sirven para clarificar ideas y actúan a modo de muestra para aplicar MIMAT a cualquier otra TTM.

- **Definición de Primitivas Conceptuales**: de las tres Formas de Uso en las que se divide el mecanismo de usabilidad *Structured Text Entry*, la única que incorpora cambios a OO-Method es *Especificar el tipo de visualización del campo de entrada*. El resto de Formas de Uso ya están soportadas. Esta Forma de Uso tiene una Propiedad Configurable (*Tipo de campo de entrada)*, por lo tanto, es necesario incorporar cambios en el Modelado Conceptual. La Propiedad indica la manera en la que se visualizarán los campos de entrada en la interfaz, por tanto se debería mode-

lar en la vista del Modelo Conceptual de OO-Method dedicado a representar cómo se visualizarán las interfaces: el *Modelo de Interacción Concreto*.

En este modelo se deben crear nuevas Primitivas de Modelado con las cuales especificar los aspectos visuales de los campos de entrada de información de las interfaces. Los posibles valores que pueden tomar estas Primitivas dependerían del tipo de argumento que se espera recibir en cada campo. A continuación se presenta la lista de nuevas Primitivas Conceptuales para cada campo dependiendo del tipo de argumento: para argumentos Booleanos se podría elegir entre Radiobuttons y Checkbox; para argumentos Enumerados se podría elegir entre ListBox, ComboBox y Radiobuttons; para todos los tipos se podría usar un campo de texto editable.

- **Modificación del Compilador de Modelos:** los cambios realizados a nivel conceptual por la Forma de Uso *Especificar el tipo de visualización del campo de entrada*, implicarían cambios en la estrategia de generación de código. La Propiedad de esta Forma de Uso sólo contiene aspectos visuales de la interfaz y por lo tanto, siguiendo el patrón arquitectural, no habría que añadir nueva funcionalidad al código. Los cambios van enfocados a incluir nuevos componentes gráficos al código generado para dar soporte a las distintas posibilidades de representación de los campos de entrada de datos. Una vez hecho este cambio, todas las generaciones de código podrían implementar la Forma de Uso.

6 Conclusiones y Trabajo Futuro

Este trabajo ha presentado el método MIMAT para incorporar mecanismos de usabilidad descritos en la literatura [11] a cualquier TTM. El método está basado en las guías de captura de requisitos y en los patrones de diseño incluidos en la definición de los mecanismos de usabilidad. Por un lado, de las guías se han extraído las Propiedades de los mecanismos que se deberían representar a nivel conceptual. Por otro lado, los patrones arquitecturales han sido útiles para definir los cambios a aplicar en el Compilador de Modelos para generar el código que implemente las Propiedades. Como ejemplo, en este trabajo se han explicado los cambios que habría que aplicar a OO-Method [15]. Esta descripción es útil porque sirve de muestra para aplicar los conceptos a cualquier otra TTM, además de poner de manifiesto la aplicabilidad de la propuesta.

Aunque se ha demostrado que se puede aplicar el método propuesto a un entorno de desarrollo MDD como OO-Method, se va a realizar como trabajo futuro una evaluación empírica para medir su eficacia. Para ello, pensamos hacer una evaluación empírica que verifique la mejora en usabilidad de las aplicaciones generadas con OO-Method. Se harán dos grupos de usuarios. Por un lado, un grupo interactuará con una aplicación generada con OO-Method sin mecanismos de usabilidad. Por otro lado, otro grupo de usuarios interactuará con la misma aplicación pero incorporando de manera manual la funcionalidad de los mecanismos de usabilidad. De esa manera se puede estimar la mejora proporcionada por dichos mecanismos antes de llevar a cabo los cambios en el Modelo Conceptual y en el Compilador de Modelos de OO-Method.

Actualmente, el método propuesto se ha aplicado a la mitad de los mecanismos de usabilidad propuestos por Juristo et al y se está trabajando en aplicar el método al re-

sto. Otro de los trabajos futuros es el de ampliar el método con otros mecanismos de usabilidad de la literatura distintos a los de Juristo et al. Esto podría enriquecer el método y abrir la puerta a otro tipo de aplicaciones, como por ejemplo, aplicaciones multimedia. En la actualidad, el tipo de aplicaciones en el que se centra nuestra propuesta es el de aplicaciones de gestión.

Referencias

1. Abrahao, S., Insfrán, E.: Early Usability Evaluation in Model Driven Architecture Environments. Sixth International Conference on Quality Software (QSIC'06), Beijing, China (2006) 287-294
2. Adikari, S., McDonald, C.: User and Usability Modeling for HCI/HMI: A Research Design. Information and Automation, 2006. ICIA 2006 (2006) 151-154
3. Bass, L., Bonnie, J.: Linking usability to software architecture patterns through general scenarios. The journal of systems and software 66 (2003) 187-197
4. Bastien, J.M., Scapin, D.: Ergonomic Criteria for the Evaluation of Human-Computer Interfaces. Rapport technique de l'INRIA (1993) 79
5. CARE Technologies S.A. www.care-t.com . Última visita: Marzo 2009.
6. Cysneiros, L.M., Kushniruk, A.: Bringing Usability to the Early Stages of Software Development. International Requirements Engineering Conference. IEEE (2003) 359- 360
7. Estrada, H., Martínez, A., Pastor, Ó. and Mylopoulos, J.: An empirical evaluation of the i* framework in a model-based software generation environment: CAISE 2006. Lecture Notes in Computer Science 4001, Springer Verlag (2006) 513-527
8. Folmer, E., Bosch, J.: Architecting for usability: A Survey. Journal of Systems and Software, Vol. 70 (1) (2004) 61-78
9. Goguen, J.A., Linde, C.: Techniques for Requirements Elicitation. In: Fickas, S., Finkelstein, A. (eds.): Requirements Engineering (1993) 152-164
10. Juristo, N., Moreno, A.M., Sánchez, M.I.: Analysing the impact of usability on software design. Journal of Systems and Software, Vol. 80 (2007) 1506-1516
11. Juristo, N., Moreno, A.M., Sánchez, M.I.: Guidelines for Eliciting Usability Functionalities. IEEE Transactions on Software Engineering, Vol. 33 (2007) 744-758
12. Kozima, A., Kiguchi, T., Yaegashi, R., Kinoshita, D., Hayashi, Y., Hashiura, H., Komiya, S.: A System To Guide Interview-Driven Requirements Elicitation Work: Domain -- Specific Navigation Using The Transition Pattern Of Topics. Journal of Integrated Design & Process Science 9 (2005) 27-39
13. Mellor, S.J., Clark, A.N., Futagami, T.: Guest Editors' Introduction: Model-Driven Development. IEEE Software, Vol. 20 (2003) 14-18
14. Olivé, A.: Conceptual Modeling of Information Systems. Springer (2007)
15. Pastor, O., Molina, J.: Model-Driven Architecture in Practice. Springer, Valencia (2007)
16. Perzel, K., Kane, D.: Usability Patterns for Applications on the World Wide Web. PloP'99 Conference (1999)
17. Potts, C.: Software-Engineering Research Revisited. IEEE Software, Vol. 10 (1993) 19-28
18. Seffah, A., Ashraf, G.: Model-Based User Interface Engineering with Design Patterns. The journal of systems and software 80 (2007) 1408-1422
19. Tidwell, J.: Designing Interfaces. O'Reilly Media (2005)
20. Welie, M.v., Traetteberg, H.: Interaction Patterns in User Interfaces. 7th. Pattern Languages of Programs Conference, Illinois, USA (2000)
21. Yu, E.: Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering. In: IEEE (ed.): IEEE Int. Symp. on Requirements Engineering (1997) 226-235