Generación de Entornos de Modelado Avanzados Mediante Técnicas de Transformación de Grafos

Esther Guerra, Paloma Díaz, Juan de Lara

***Resumen*—En este artículo mostramos nuestro enfoque para la generación automática de entornos de modelado que soporten Lenguajes Visuales de Dominio Específico (LVDEs) con múltiples vistas. El enfoque está basado en la herramienta de meta-modelado AToM^3. La herramienta es capaz de generar un entorno de modelado a partir de un meta-modelo del LVDE y sus vistas. La consistencia entre las vistas se consigue mediante la generación automática de reglas de gramáticas de grafos triples. Éstas construyen un único modelo que combina los distintos modelos de las vistas. El diseñador del LVDE puede especificar métricas y rediseños de manera visual para el entorno generado. Valores extremos de estas métricas pueden disparar la aplicación de rediseños, los cuales se han formalizado mediante reglas de gramáticas de grafos. Mostramos la aplicación de estos conceptos en el área del diseño hipermedia mediante la generación de un entorno de modelado para el LVDE Labyrinth.**

***Palabras Clave*—Lenguajes Visuales de Dominio Específico, Meta-Modelado, Transformación de Grafos, Métricas, Rediseño.**

I. Introducción

El uso de diagramas en el ámbito de la ingeniería del software está ampliamente extendido durante las fases de planificación, análisis y diseño de software con el propósito de especificar, comprender y razonar sobre el sistema. Los LVDEs, diseñados para ser utilizados en áreas de conocimiento particulares, también se están haciendo muy populares. Al proporcionar construcciones de alto nivel adaptadas a las tareas que deben realizar los diseñadores, pueden ayudar a incrementar su productividad. Además, puesto que en dichos LVDEs subyace conocimiento del dominio de aplicación, también se puede incrementar la calidad de los diseños. Uno de tales LVDEs es Labyrinth [5], un conjunto de notaciones para el análisis y el diseño de aplicaciones hipermedia/web.

Debido a la complejidad que están adquiriendo los sistemas software, resulta de gran ayuda una descripción separada de cada una de sus características. Por esta razón son muchos los LVDEs compuestos por un conjunto de diagramas (o vistas), donde cada uno de ellos se utiliza para la descripción de un aspecto distinto del sistema. Por ejemplo, en la especificación de UML2.0, encontramos diversos diagramas para la descripción de la estructura de un sistema (diagramas de clases y objetos) así como para su comportamiento (diagramas de transición de estados, de comunicación, etc.).

El uso de métodos formales para la definición de LVDEs formados por múltiples vistas podría reportar grandes ventajas, proporcionando mecanismos que garantizasen la consistencia entre las distintas vistas y que permitiesen analizar el sistema descrito, así como estudiar propiedades de interés.

Ya que los diagramas se pueden representar como grafos tipados atribuidos, la transformación de grafos [6] es un modo natural para su manipulación. Su principal atractivo es que se trata de una técnica gráfica, formal, declarativa y de alto nivel para expresar computaciones. Su teoría puede utilizarse para analizar propiedades de las transformaciones, como son su terminación o convergencia. Así, usamos transformación de grafos para la transformación de modelos (a una notación diferente), la simulación de modelos, la consistencia entre vistas y la optimización.

En este artículo proponemos el uso de meta-modelado para describir la sintaxis de los LVDEs y de sus diferentes vistas. Desde esta descripción de alto nivel se genera automáticamente un entorno de modelado personalizado que soporta la definición de múltiples vistas. La consistencia entre las vistas se garantiza mediante la construcción de un modelo único que aglutina todos los modelos de vistas. Llamaremos *repositorio* a tal modelo único, el cual debe ser conforme al meta-modelo completo del LVDE. Un conjunto de gramáticas de grafos automáticamente generadas a partir de la información almacenada en el meta-modelo se encarga de construir el *repositorio*, así como de propagar cambios entre vistas. Recientemente hemos implementado estas ideas en las herramienta de meta-modelado AToM^3 [2] (que sólo soportaba la definición de LVDE con una única vista).

Junto a la descripción del LVDE y de sus vistas, hemos incorporado a AToM^3 la posibilidad de definir optimizaciones y rediseños [8] que mejoren ciertas propiedades de los modelos. Los rediseños se definen a nivel de meta-modelado, ya que son restricciones o heurísticas sobre la sintaxis que el meta-modelo especifica. Se han formalizado en la forma de

Este trabajo ha sido financiado en parte por el Ministerio de Ciencia y Tecnología con los Proyectos TIC2002-01948 y TIC2003-03654.

E. Guerra y P. Díaz trabajan en el grupo DEI, Escuela de Informática, Universidad Carlos III de Madrid, Avda. de la Universidad 30, 28911 Leganés (e-mail: {eguerra, pdiaz}@inf.uc3m.es).

J. de Lara trabaja en la Escuela Politécnia Superior, Universidad Autónoma de Madrid, Campus de Cantoblanco, 28049 Madrid (e-mail: jdelara@uam.es).

reglas de transformación de grafos (en el estilo de [13]), lo que permite analizar sus propiedades. En el entorno generado los rediseños previamente definidos se aplican sobre los modelos concretos. Un ejemplo de rediseño en un LVDE con herencia podría consistir en mover características comunes de diferentes clases a una superclase común. Adicionalmente también hemos incorporado a AToM^3 la posibilidad de definir métricas estáticas [7] que puedan disparar rediseños para mejorar el diseño en aquellas partes del modelo donde las métricas alcanzan valores extremos (no deseables).

Las aportaciones novedosas de este trabajo son, por tanto, la definición de un marco para definir LVDEs con múltiples vistas y asegurar su consistencia de manera formal (mediante gramáticas de grafos triples); la creación de una herramienta para la definición de métricas para un LVDE dado, así como de rediseños (mediante gramáticas de grafos); y por último la definición y generación de un entorno de modelado para el LVDE Labyrinth. Todos estos conceptos se han implementado sobre la herramienta AToM^3.

El resto del artículo está organizado como sigue. La sección II presenta informalmente algunos conceptos básicos sobre meta-modelado y transformación de grafos. La sección III muestra nuestra propuesta, basada en AToM^3, para la especificación y generación de entornos de modelado con múltiples vistas. La sección IV detalla el proceso para la especificación de métricas y rediseños. La sección V presenta un ejemplo no trivial de generación de entorno de modelado para Labyrinth (un LVDE que tiene 16 vistas). La sección VI realiza una comparativa con trabajos relacionados. Finalmente, la sección VII termina con las conclusiones y trabajos futuros.

II. Meta-modelado y Transformación de Grafos

El meta-modelado consiste en la descripción de un conjunto de modelos válidos mediante la construcción de otro modelo que los describe, denominado meta-modelo. Para ello suelen utilizarse notaciones tales como diagramas entidad-relación o diagramas de clases. Para restringir el número de modelos permitidos pueden utilizarse restricciones adicionales sobre el meta-modelo. Éstas, a veces, se expresan mediante un lenguaje textual de restricciones (como por ejemplo OCL [19]). El meta-modelado es una técnica común para la descripción de notaciones diagramáticas y para la generación de sus correspondientes entornos de modelado. Por ejemplo, UML está definido mediante una meta-arquitectura de cuatro capas. El meta-modelo de UML está en el tercer nivel. En la capa del nivel inferior se encuentran los modelos UML definidos por el usuario. En la capa superior se encuentra la descripción del lenguaje en el que UML está escrito, el cual es un subconjunto de los diagramas de clases (o MOF). Las notaciones de la cuarta meta-capa pueden describirse a sí mismas.

Para especificar un LVDE se debe describir tanto su *sintaxis abstracta* (conceptos del lenguaje) como su *sintaxis concreta* (representación gráfica de los conceptos). Para el primer tipo de sintaxis podemos utilizar un meta-modelo. Para el segundo existen varias aproximaciones. En la herramienta AToM^3 esto se realiza mediante una clase especial de atributos gráficos asociados a las clases y relaciones del meta-modelo. A partir de esta especificación de alto nivel, AToM^3 puede generar una herramienta de modelado personalizada para el LVDE. La funcionalidad de dicha herramienta puede ser enriquecida mediante la definición de transformaciones de grafos que manipulen los modelos. Los conceptos básicos de esta técnica se explican a continuación.

La transformación de grafos [6] es un mecanismo formal para la manipulación de grafos basado en reglas. En analogía a las gramáticas de Chomski sobre cadenas de caracteres, las reglas de grafos están formadas por una parte izquierda y una parte derecha (LHS y RHS respectivamente) que contienen grafos. Intuitivamente, para poder aplicar una regla a un grafo (llamado *grafo anfitrión*) debemos encontrar primero un morfismo de correspondencia entre la LHS de la regla y el grafo. Si tal morfismo se encuentra, la regla puede aplicarse sustituyendo la imagen de dicho morfismo en el grafo por la RHS de la regla. Este proceso se denomina *paso de derivación*. En la Fig. 1 se muestra un ejemplo de aplicación de regla a un grafo. La regla simplemente borra una relación "permission" de un nodo de tipo *Subject* si tal relación ya está definida en otro *Subject* que compone al primero.

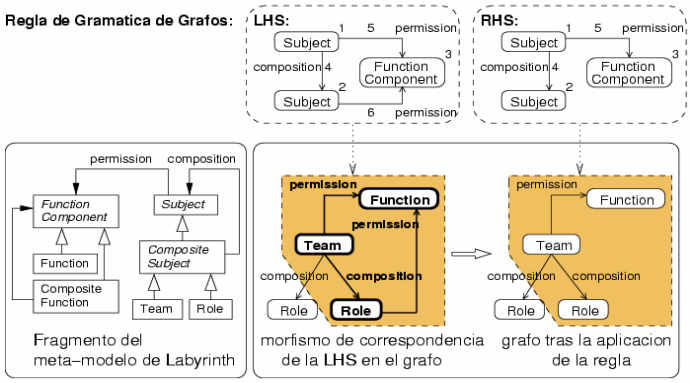

Fig. 1. Aplicación de una regla

El meta-modelado y la transformación de grafos se pueden combinar para permitir que en las reglas aparezcan nodos abstractos [1]. En esos casos los nodos abstractos pueden hacerse corresponder con cualquier nodo que sea instancia de una de sus subclases. Así, en el ejemplo previo podemos establecer un morfismo de correspondencia entre los nodos abstractos de tipo *Subject* que aparecen en la regla, y los nodos concretos *Team* y *Role* que aparecen en el grafo. Esto es así porque *Team* y *Role* son subclases de *Subject*, como se muestra en el meta-modelo. Del mismo modo podemos establecer un morfismo de correspondencia entre *Function Component* y cualquiera de sus subclases *Function* o *Composite Function*.

La aplicabilidad de una regla puede restringirse mediante el uso de condiciones de aplicación. Una condición de aplicación está formada por un grafo premisa X, un conjunto de grafos consecuencia Y_i, y morfismos y_i a cada Y_i: $c = \{X, X \xrightarrow{y_i} Y_i\}$. Si existe un morfismo de correspondencia entre X y el grafo

anfitrión, entonces también debe existir un morfismo de correspondencia entre cada grafo consecuencia Y_i y el grafo anfitrión para que la regla pueda ser aplicada. Si una condición no tiene grafos consecuencia, entonces encontrar un morfismo de correspondencia de *X* en el grafo prohibe la aplicación de la regla. Este último caso es un tipo de condición especial denominada *condición de aplicación negativa* (NAC).

El enfoque Double Pushout (DPO) [6] seguido en este trabajo utiliza teoría de categorías para formalizar las reglas de grafos y su aplicación. En DPO las reglas se representan como tres grafos y dos morfismos inyectivos: $L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R$. *K* se denomina grafo interfaz y contiene los elementos que la regla preserva. *L-K* y *R-K* son los elementos borrados y añadidos por la regla, respectivamente. La aplicación de una regla se modela mediante dos "pushouts" en la categoría **Grafo**. El enfoque categórico tiene la ventaja de que la teoría es independiente de la categoría concreta utilizada (siempre que la categoría cumpla ciertas condiciones).

En el caso particular de entornos con múltiples vistas estamos interesados en la reescritura no de grafos simples, sino de grafos triples formados por los grafos de dos vistas ($VIEW_i$) más un grafo intermedio (*LINK*) cuyos nodos relacionan los objetos de las vistas. Esto es, cada nodo en el grafo *LINK* tiene un par de morfismos: uno a un objeto del grafo $VIEW_1$ y otro a un objeto del grafo $VIEW_2$. Así, un grafo triple se anota como $VIEW_1 \overset{v_1}{\leftarrow} LINK \overset{v_2}{\rightarrow} VIEW_2$. En este trabajo $VIEW_1$ será el modelo del repositorio y $VIEW_2$ cualquiera de las vistas parciales del sistema dibujadas por el usuario. Para reescribir grafos triples usamos gramáticas de grafos triples [16] (GGTs). Con ellas especificamos cómo los diferentes modelos de las vistas se aglutinan en el repositorio único. En [11] definimos las GGTs usando el enfoque DPO y las usamos en combinación con meta-modelado.

III. Generación de Entornos de Modelado con Múltiples Vistas

En el enfoque de meta-modelado un LVDE se especifica por medio de un meta-modelo, a partir del cual se genera un entorno de modelado automáticamente. Si el LVDE tiene múltiples vistas, en el caso más sencillo sus meta-modelos están incluidos en el meta-modelo del LVDE completo. Si un mismo elemento aparece en más de una vista pueden surgir algunos problemas. Por ejemplo, un usuario puede intentar asignar valores distintos al mismo atributo del elemento en las distintas vistas. Para poder detectar ese tipo de inconsistencias resultaría útil tener una perspectiva del sistema completo y no sólo de las vistas aisladas. En este trabajo solucionamos estos problemas construyendo un modelo único – el *repositorio* – mediante la unión de los distintos modelos de vistas a través de los elementos comunes.

Hemos implementado estas ideas en la herramienta de meta-modelado AToM^3, que anteriormente sólo soportaba la generación de entornos para LVDEs con una sola vista. Para ello hemos utilizado sus características de generación de código. Así, definimos el núcleo de la nueva herramienta para la especificación de vistas de un LVDE mediante el meta-modelo de la Fig. 2. El meta-modelo está compuesto de una única clase llamada *View*, y de la relación *view_consistency*.

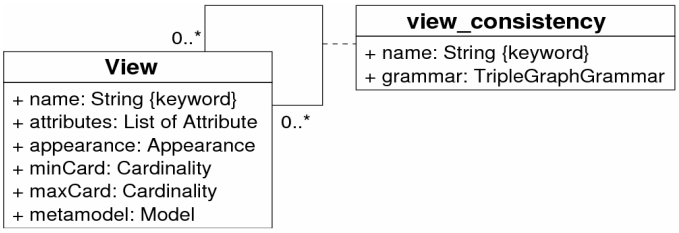

Fig. 2 Meta-modelo de la herramienta para la definición de vistas.

Los atributos de la clase son un nombre único (*name*), una lista de atributos generativos (*attributes*) [2] (que se muestran y rellenan en el entorno generado, esto es, se instancian en el meta-nivel inferior), una apariencia gráfica (*appearance*), el número mínimo y máximo de instancias permitidas para la vista (*minCard* y *maxCard*), y un meta-modelo (*metamodel*). La relación *view_consistency* expresa una relación de consistencia entre dos vistas (normalmente distintas). Sus atributos son un nombre (*name*) y una gramática de grafos triple (*grammar*).

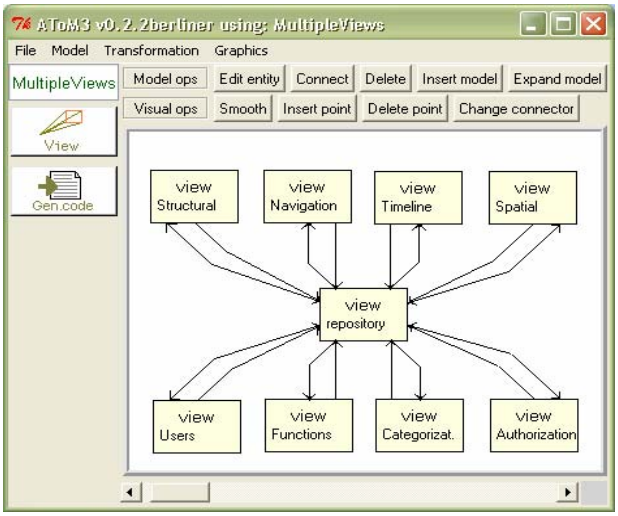

Fig. 3 Herramienta para la definición de vistas.

Con este meta-modelo, se generó automáticamente una herramienta que fue integrada con el meta-metamodelo de los diagramas de clases. Este es el meta-formalismo que utilizamos para definir los LVDEs. De esta manera, una vez que se ha definido el meta-modelo del LVDE completo, se puede abrir la herramienta para la definición de sus vistas. La herramienta se muestra en la Fig. 3. En ésta se aprecian algunas de las vistas de Labyrinth. Por defecto siempre se incluye una vista llamada repositorio. Ésta contiene el meta-modelo completo del LVDE y se usa para mantener un único modelo con la unión de las vistas creadas por el usuario final. De manera adicional, el diseñador del LVDE pueden añadir

tantas vistas como sea necesario utilizando esta herramienta. Por defecto, las nuevas vistas contienen el meta-modelo completo del LVDE, que puede editarse posteriormente. Hemos seguido un enfoque restrictivo, de tal manera que sólo es posible borrar clases y relaciones, o atributos y restricciones locales a éstas. Además, podemos especificar para cada vista su apariencia gráfica, una lista de atributos y su cardinalidad.

A. Consistencia

En la herramienta para la definición de vistas se puede asegurar la consistencia entre vistas por medio de la relación *view_consistency*. La consistencia se modela mediante GGTs. Usamos reglas triples porque permiten relacionar de manera fácil e intuitiva los meta-modelos de dos vistas a través de un grafo de correspondencia. Los cambios realizados en una vista dispararán la ejecución de las gramáticas triples definidas en las relaciones *view_consistency* de salida.

La herramienta genera automáticamente las relaciones de consistencia entre cada vista y el repositorio, incluyendo las gramáticas de grafo triples asociadas. Las gramáticas contienen reglas que aseguran la consistencia en caso de que los usuarios creen, borren o editen elementos de una vista. Estas reglas se generan para cada clase y relación concreta en el meta-modelo de la vista. No generamos reglas para elementos abstractos, ya que el usuario no puede instanciarlos.

Las reglas generadas automáticamente que conciernen a la creación de objetos, relacionan el objeto creado con una única copia de sí mismo que se guarda en el repositorio. Esta copia del repositorio puede ser referenciada varias veces si el mismo objeto aparece en varias vistas. Las reglas para el borrado eliminan del repositorio los elementos que no se referencian en ninguna vista. Por último, las reglas de edición copian el valor de los atributos de los objetos de las vistas a los objetos del repositorio, si son diferentes. A continuación el nuevo valor del atributo debe ser propagado a los elementos relacionados en otras vistas (si los hay). Una GGT en la dirección contraria realiza esta tarea. Este mecanismo se muestra gráficamente en la Fig. 4 y es similar al patrón Model View Controller. La sección V.A muestra un ejemplo detallado de estas reglas en el contexto del LVDE Labyrinth.

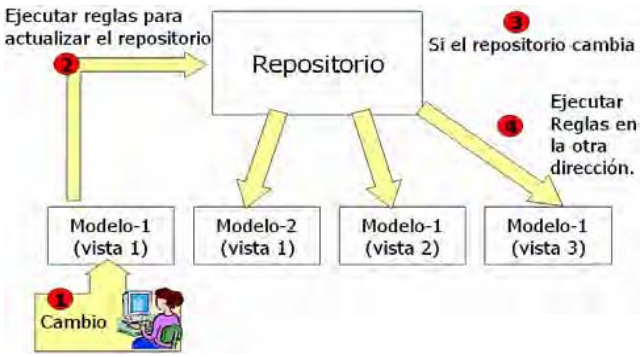

Fig. 4 Esquema de propagación de cambios con GGTs.

Además de las relaciones generadas de manera automática, los diseñadores de LVDEs pueden crear sus propias relaciones de consistencia para implementar restricciones adicionales de la semántica estática del LVDE. En [11] mostramos un ejemplo basado en UML en el que hay que definir reglas adicionales si la sintaxis concreta (la de las vistas) es muy distinta de la sintaxis abstracta (la del repositorio).

B. Análisis y Manipulación del Repositorio

Gracias a que construimos un modelo único (el repositorio) con la suma de todos los modelos de las vistas, podemos usar gramáticas de grafos normales sobre él para manipularlo con diversos propósitos. Por un lado, podemos transformarlo a otros dominios semánticos para su simulación y verificación funcional [10]. En estas transformaciones el modelo almacenado en el repositorio no cambia, sino que simplemente se le muestran al usuario los resultados de la simulación o el análisis. Por otro lado podemos realizar manipulaciones donde la estructura del modelo sí se modifique con el fin de mejorar la calidad del diseño software (rediseño) [8][13]. A veces estos rediseños sólo son aplicables y/o detectables mediante métricas sobre el modelo completo, pero no sobre las vistas aisladas. En esos casos los cambios se deben aplicar en el repositorio, y propagar a continuación a los diferentes modelos de las vistas para mantener la consistencia. Por supuesto, si los cambios estructurales afectan a una única vista se pueden aplicar directamente sobre ella, y dejar que las reglas de consistencia sean las que recuperen la consistencia del sistema completo. En la siguiente sección veremos cómo la definición de métricas y rediseños sobre modelos ha sido integrada en la herramienta de meta-modelado AToM^3, pudiendo utilizarse sobre LVDEs con una o varias vistas.

IV. Métricas y Rediseños

Las métricas de producto miden características de un sistema software con el fin de controlar y mejorar su calidad [7]. Así, ofrecen una medida de ciertas propiedades como son la complejidad, la fiabilidad, la mantenibilidad o la usabilidad. Las métricas estáticas miden el producto sin necesidad de ejecutarlo, de manera que es posible tomar métricas sobre documentos o diagramas. Por otro lado los rediseños (llamados refactorizaciones [8] cuando se aplican sobre código) son modificaciones hechas a la estructura interna del software con el fin de hacerlo más comprensible o mantenible, sin que esto afecte a su comportamiento externo. El uso de métricas puede ayudar a detectar algunas anomalías, que la aplicación de rediseños localizados puede corregir.

Se han propuesto un gran número de métricas estáticas [7][17][18], algunas de ellas de dominio específico [3]. No obstante, para utilizar métricas sobre un LVDE, siempre es necesario adaptarlas a las particularidades del lenguaje y de su dominio de aplicación. Por esta razón hemos incorporado en AToM^3 una nueva herramienta para la definición de métricas específicas para el LVDE que se está diseñando. Al igual que en el caso de múltiples vistas, una gran parte de la nueva herramienta se ha generado automáticamente a partir de un meta-modelo construido en AToM^3, donde se ha integrado posteriormente. El meta-modelo para la definición de métricas puede verse en la Fig. 5.

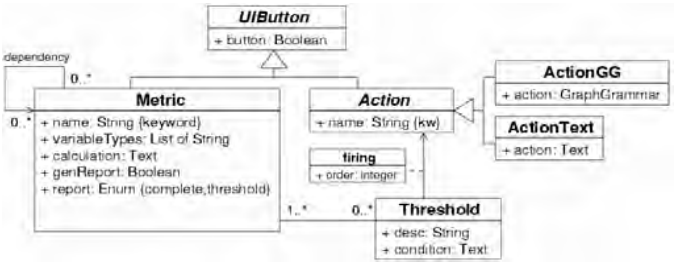

Fig. 5 Meta-Modelo de métricas.

La clase *Metric*, que representa cualquier métrica, tiene como atributos un nombre (*name*); una lista (*variableTypes*) con los tipos del LVDE que constituyen el dominio de la métrica, es decir, los tipos para los que la métrica se va a calcular; un método escrito en Python que calcula la métrica para un determinado valor del dominio (*calculation*), y un par de indicadores para la generación de informes con el resultado obtenido o sólo con aquellos valores que alcanzaron cierto valor umbral (*genReport* y *report*). Se pueden definir diversos valores umbral para una métrica mediante la clase *Threshold*, donde el atributo *condition* permite especificar una expresión booleana (en código Python) que se compara con el resultado de la métrica para detectar valores críticos. La clase abstracta *Action* representa una acción (posiblemente un rediseño) aplicable a un modelo, y la implementan las clases concretas *ActionText* y *ActionGG* para especificar la acción en código Python o con una gramática de grafos respectivamente. Cada valor umbral puede disparar una lista de acciones (relación *firing*) en un orden dado. La relación *dependency* permite expresar relaciones de dependencia entre métricas, lo cual es útil cuando una métrica se calcula en función de otras. Finalmente, el atributo *button* de la clase abstracta *UIButton* indica para qué métricas o acciones se generará automáticamente un botón que las calcule o ejecute en la interfaz de usuario de la herramienta generada. De este modo en los entornos generados podremos calcular métricas (que opcionalmente disparen acciones), pero también podremos ejecutar acciones que no sean guiadas por el valor de métricas.

La herramienta generada a partir de este meta-modelo se ha integrado en AToM^3. Así, una vez que el diseñador del LVDE ha definido su meta-modelo, puede abrir la herramienta para definir métricas y rediseños específicos para él. La herramienta se muestra en la Fig. 6, donde vemos la definición de algunas métricas y de una acción para Labyrinth. En este caso no se ha especificado un valor umbral para las métricas. Sí que existen relaciones de dependencia entre la métrica *PermissionInheritanceFactor* y las métricas *RoleInheritedPermissions, TeamInheritedPermissions, RolePermissions* y *TeamPermissions*, puesto que la primera se calcula a partir de las segundas. Aunque no se muestra, el atributo *button* de todas las métricas y acciones definidas se ha seleccionado, lo que indica que se añadirá un botón para cada una de ellas en el entorno generado. En la sección V.B se analizará este ejemplo más en profundidad.

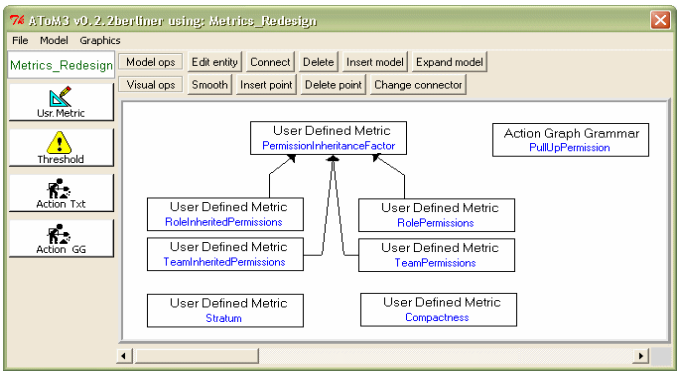

Fig. 6 Herramienta para la definición de métricas y rediseños.

V. Una Aplicación a la Hipermedia

Labyrinth [5] es un conjunto de notaciones para el diseño de aplicaciones hipermedia independientes de la plataforma. Permite la descripción tanto de la estructura estática, como del comportamiento dinámico de este tipo de aplicaciones. Un extracto muy pequeño de su meta-modelo se mostró en la parte izquierda de la Fig. 1. Con esta notación, los sistemas hipermedia se describen como un conjunto de nodos donde se pueden incluir contenidos (textos o imágenes, por ejemplo). Los enlaces (links) entre nodos y contenidos establecen de qué manera los usuarios pueden navegar por el sistema. Además, los usuarios pueden asumir roles y tomar parte en distintos equipos, que le proporcionan un conjunto de permisos.

El método de desarrollo Ariadne [4] utiliza Labyrinth en el proceso de construcción de aplicaciones hipermedia. El método propone una serie de artefactos que son en realidad vistas del meta-modelo de Labyrinth. Por ejemplo, uno de los productos es el *Diagrama de Usuarios*, que se utiliza para especificar jerarquías de roles y equipos. El meta-modelo para este artefacto contiene sólo las clases y relaciones necesarias para expresar esa información (por ejemplo, la clase *Role* está incluida, pero la clase *Function* no). En total, Ariadne define 16 artefactos de diseño, o lo que es lo mismo, existen 16 vistas en el meta-modelo de Labyrinth.

Utilizando AToM^3 se ha generado un entorno que soporta los productos de Ariadne. Para ello, se empezó definiendo el meta-modelo completo de Labyrinth en la herramienta. Después, se extrajeron las 16 vistas del meta-modelo, una para cada producto. Por defecto, AToM^3 permite a los usuarios un número arbitrario de modelos (instancias) de una vista en sus proyectos. No obstante, se tuvo que restringir la cardinalidad de alguna de las vistas a una sola instancia, y otras a al menos una para adecuar el entorno a las características y restricciones del método Ariadne. De manera adicional, cada vista se completó con un nombre y una descripción. No hubo que modelar relaciones de consistencia sintácticas, ya que sólo fueron necesarias las generadas automáticamente por la herramienta.

La Fig. 7 muestra el entorno generado. Los botones con fondo blanco a la izquierda de la ventana del fondo permiten añadir al proyecto un nuevo modelo de una cierta vista. El nuevo modelo que se añade se muestra como un icono en la ventana principal. Si el usuario selecciona el icono, un cuadro de diálogo permite editar los atributos de la vista (el

"nombre", la "descripción" y el modelo). La ventana del frente en la misma figura muestra la edición del modelo correspondiente al Diagrama de Usuarios.

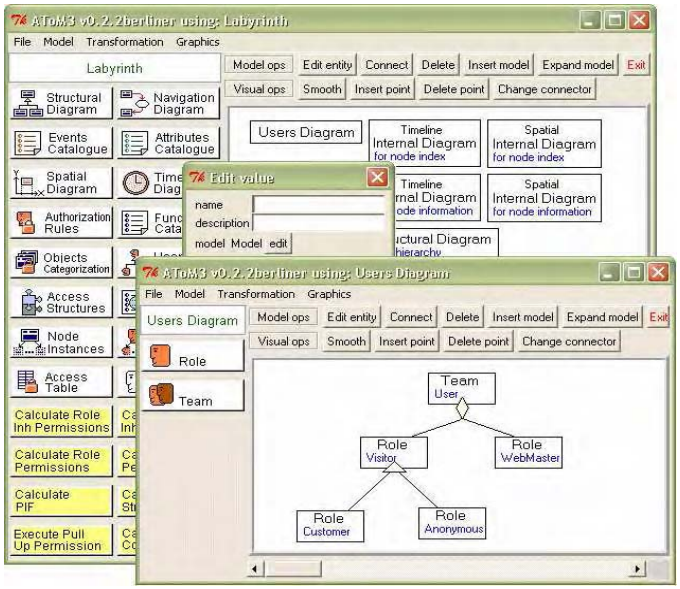

Fig. 7 Entorno generado para Labyrinth.

La próxima sección muestra alguna de las reglas triples generadas para asegurar la consistencia entre las vistas de Labyrinth.

A. Consistencia

Tal y como se explicó en la subsección III.A, la herramienta provee un mecanismo de coherencia automático entre las vistas. Con este propósito, una relación de consistencia se genera automáticamente entre cada vista del meta-modelo y la vista del repositorio (ver por ejemplo la Fig. 3). Estas relaciones contienen gramáticas triples que contruyen un único modelo de repositorio a partir de las distintas vistas. Cuando el usuario termina de modificar alguna vista, se le pregunta si desea ejecutar las gramáticas triples pertinentes. Éstas contienen un conjunto de reglas triples para cada clase y relación concreta en el meta-modelo de la vista, y gestionan cada acción posible del usuario (creación, edición y borrado de objetos gráficos). Como ejemplo, vamos a mostrar las reglas generadas automáticamente para la clase *Role*. Las reglas triples (ver Fig. 8, 9 y 10) siempre relacionan elementos del repositorio (que se muestran en el área superior de las reglas), con algún elemento de la vista (área inferior) a través de un grafo de correspondencia (área intermedia).

La Fig. 8 muestra las reglas generadas para gestionar la creación de instancias de *Role* en las vistas. La regla superior crea un *role* en el repositorio si detecta un *role* nuevo en la vista. La NAC garantiza que el *role* no existe en el repositorio, lo que significa que es nuevo. La regla inferior relaciona un nuevo *role* en la vista con uno ya existente en el repositorio que tiene el mismo identificador. El atributo "refcount" del *role* en el repositorio cuenta cuántas referencias tiene el *role*, esto es, en cuántas vistas aparece. Cuando se crea un *role* en el repositorio su valor se pone a 1. El contador se incrementa cada vez que se incluye una nueva referencia al *role* en una vista (e.d. un *role* con el mismo identificador).

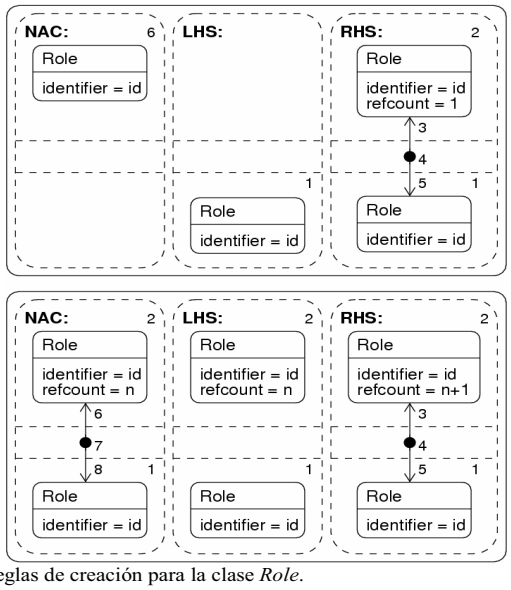

Fig. 8 Reglas de creación para la clase *Role*.

La Fig. 9 muestra las reglas generadas para la edición de instancias de *Role* en las vistas. La primera regla propaga el valor de algún atributo del *role* de la vista al repositorio, si son diferentes. La segunda regla pertenece a una gramática distinta, que está en una relación de consistencia en el sentido opuesto (del repositorio a las vistas, ver Fig. 4). Se ejecuta una vez que las gramáticas previas terminan y propaga cambios del repositorio a las vistas. Así, cada cambio de una instancia de *role* en una vista se propaga automáticamente a todas las vistas donde esa misma instancia aparece.

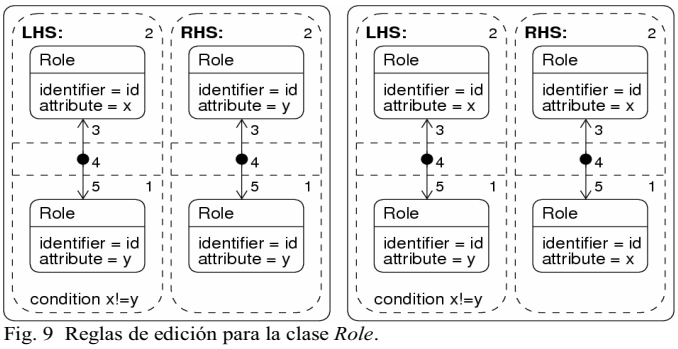

Fig. 9 Reglas de edición para la clase *Role*.

La Fig. 10 muestra las reglas generadas para el borrado de instancias de *Role* en las vistas. La primera de ellas decrementa el atributo "refcount" en el *role* del repositorio, y elimina su relación de correspondencia. La NAC comprueba que no hay una instancia con el mismo identificador en la vista, luego el *role* se ha eliminado. La segunda regla elimina un *role* del repositorio si su atributo "refcount" es 0, lo que significa que no queda ninguna instancia en ninguna vista.

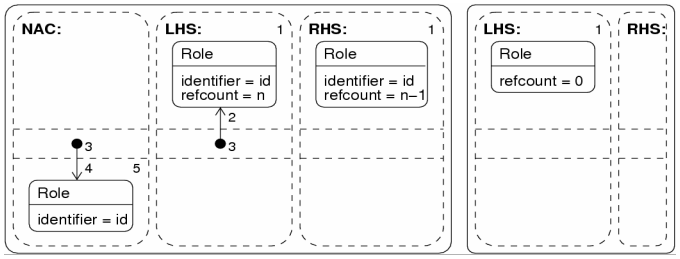

Fig. 10 Reglas de borrado para la clase *Role*.

B. *Métricas y Rediseños*

Utilizando nuestro entorno para la definición de métricas y rediseños hemos especificado algunas métricas y acciones aplicables sobre diseños Labyrinth. Respecto a métricas, se han añadido algunas específicas del dominio hipermedia (en particular *stratum* y *compactness* [3]). También se han adaptado métricas de herencia del dominio de la orientación a objetos [18]. Por ejemplo, en Labyrinth se permite que roles y equipos de una aplicación ejecuten ciertas funciones a través de una relación *permission* entre ambos (vease el meta-modelo de la Fig. 1). Como hemos dicho antes, roles y equipos pueden anidarse en estructuras jerárquicas mediante la relación *composition*. Los permisos se heredan a través de esta jerarquía. Así, hemos definido la métrica *PermissionInheritanceFactor* (PIF) que calcula la proporción de permisos heredados en nuestro sistema, siendo un indicador del nivel de reutilización. Ésta es una particularización de las métricas Proporción de Métodos y Atributos Heredados (MIF y AIF respectivamente) en orientación a objetos. Se calcula como la suma de todos los permisos heredados en roles y equipos dividido entre el número total de permisos (locales más heredados) definidos en roles y equipos. Como mostramos en la Fig. 6, en Labyrinth hemos definidos las métricas *RoleInheritedPermissions*, *TeamInheritedPermissions*, *RolePermissions* y *TeamPermissions* para calcular los factores de esta división. Así, la proporción de permisos heredados (PIF) se calcula a partir ellas.

En la parte inferior de la ventana del fondo de la Fig. 7 pueden verse los botones generados (en amarillo) para el cálculo de métricas (y acciones) sobre modelos Labyrinth, a partir de la definición mostrada en la Fig. 6. La ejecución de una métrica sobre un modelo genera un documento *pdf* donde se muestra el resultado obtenido en forma de tabla. En la Fig. 11 puede verse el informe generado al aplicar la métrica *RolePermissions* sobre cierto modelo. La primera columna contiene los distintos roles existentes en el modelo, y la segunda contiene el valor de la métrica para cada rol (es decir el número de permisos heredados y definidos).

Metrics: RolePermissions

Role	result
identifier: Visitor	4
identifier: WebMaster	9
identifier: Customer	6
identifier: Anonymous	4

Fig. 11 Informe generado para una métrica.

También hemos modelado algunas acciones con gramáticas de grafos para la optimización de diseños en Labyrinth. Como ejemplo, mostramos una gramática que implementa un rediseño similar al "pull up method" [13] (promocionar método). Este rediseño mueve un método (en nuestro caso una relación) a una clase, si éste está definido en todas sus subclases. La gramática se aplica sobre el repositorio, ya que no existe una vista que contenga todas las clases y relaciones implicadas en la optimización estructural. Posteriormente los cambios se propagan a las vistas correspondientes.

Esta mejora de diseño actúa sobre las relaciones *permission* salientes de roles y equipos. Si todos los hijos de un equipo o role puede ejecutar cierta función, podemos mover el permiso de ejecución a su padre. De esta manera, la regla de la Fig. 12 crea el correspondiente permiso para que un cierto equipo ejecute una función si el equipo no la tiene ya (condición de aplicación *X1/Y1*), si el equipo tiene algún hijo (condición *X2/Y2*) y el permiso que se va a mover se ha definido en todos los hijos (condición *X3/Y3*). Entonces, la regla en la Fig. 1 elimina dicho permiso de todos los hijos. No obstante, la última regla es útil por sí sola, ya que elimina permisos redundantes en nodos hijos cuando han sido definidos en alguno de sus padres.

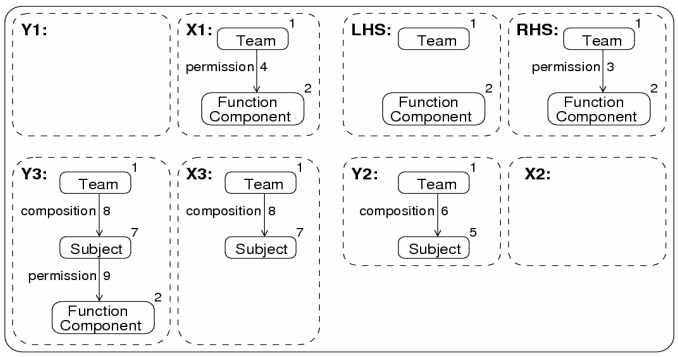

Fig. 12 Una de las reglas de rediseño para mover permisos.

VI. Trabajo Relacionado

Diversas aproximaciones para el chequeo de la consistencia en entornos para LVDEs con múltiples vistas implementan la idea de un repositorio común donde los modelos de las vistas están de algún modo relacionados. El modo en que esas relaciones se construyen puede especificarse mediante una notación textual o gráfica. Ejemplos del primer caso incluyen MetaEdit+[12] y Pounamu [20]. Ejemplos del segundo incluyen JComposer [9] y AToM^3. En JComposer los *grafos de respuesta y propagación de cambios* permiten asociar visualmente eventos a los elementos del modelo. En cualquier caso, las acciones ejecutadas en respuesta a esos eventos se especifica utilizando un lenguaje textual. En AToM^3 utilizamos gramáticas de grafos triples para garantizar la consistencia y propagar cambios. Pensamos que las gramáticas de grafos triples son un modo gráfico y natural de expresar relaciones entre modelos. Además el hecho de ser formal permite que la teoría de gramáticas de grafos [6] se pueda utilizar para su análisis.

Existen varios enfoques para la implementación de refactorizaciones y rediseños [8][14]. Por ejemplo, en [13] también se propone el uso de transformación de grafos. Nosotros seguimos un enfoque similar (aunque las condiciones de aplicación de nuestras reglas son más expresivas), pero aplicado a la mejora del diseño, y en entornos de múltiples vistas (por lo que han de ser aplicadas en el repositorio). De manera adicional, las transformaciones pueden ser guiadas mediante métricas. Estas pueden ser definidas y personalizadas para el LVDE específico. Esta es una característica única de nuestro enfoque.

Próximamente el OMG aprobará un lenguaje para la descripción de consultas, vistas y transformaciones (QVT) [15] sobre modelos. Este lenguaje incluye aspectos declarativos, similares a las reglas de GGTs, si bien estas últimas tienen una base más formal.

VII. Conclusiones

En este trabajo hemos presentado una aproximación formal a la generación automática de entornos para LVDEs con múltiples vistas. El enfoque está basado en técnicas de meta-modelado y transformación de grafos. La consistencia entre vistas se obtiene mediante la construcción de un único modelo aglutinando todos los modelos definidos para cada uno de los distintos tipos de diagramas (vistas). Este modelo único, llamado repositorio se crea mediante la ejecución de GGTs, generadas automáticamente por la herramienta. Una vez que obtenemos este modelo único, se pueden aplicar validaciones funcionales y mejoras estructurales usando gramáticas de grafos normales. Estas ideas se han implementado en la herramienta de meta-modelado AToM^3. También hemos extendido la herramienta con la posibilidad de especificar métricas y rediseños para los LVDEs. Como prueba de concepto, hemos mostrado un ejemplo no trivial, basado en la construcción de un entorno para la notación Labyrinth. La creación manual de dicho entorno habría sido sin duda más costosa que mediante meta-modelado. Este enfoque tiene también ventajas en el mantenimiento, ya que la herramienta se puede mantener a nivel de modelos y no de código.

En la actualidad, estamos trabajando en la integración de mecanismos de análisis para transformación de grafos, en el modelado de aspectos avanzados de la interfaz de usuario generada, así como en la transformación del modelo del repositorio a dominios semánticos para verificación funcional.

Referencias

[1] R. Bardohl, H. Ehrig, J. de Lara, y G. Taentzer. 2004. "Integrating Meta Modelling with Graph Transformation for Efficient Visual Language Definition and Model Manipulation". LNCS 2984, pp. 214-228. Springer.

[2] J. de Lara y H. Vangheluwe. 2002. "AToM3: A Tool for Multi-Formalism Modelling and Meta-Modelling". LNCS 2306, pp. 174 - 188. Springer. Página web de AToM3 en: http://atom3.cs.mcgill.ca

[3] D. Dhyani, W. Keong, y S. S. Bhowmick. 2002. "A Survey of Web Metrics". En ACM Computing Surveys (CSUR), 34(4), pp. 469-503.

[4] P. Díaz, S. Montero, e I. Aedo. 2005. "Modelling hypermedia and web applications: the Ariadne Development Method". Information Systems, Vol. 30, No. 8, pp. 649-673.

[5] P. Díaz, I. Aedo, y F. Panetsos. 2001. "Modeling the Dynamic Behavior of Hypermedia Applications". IEEE Transactions on Software Engineering, 27 (6), pp. 550-572.

[6] H. Ehrig, H.-J. Kreowski, U. Montanari, y G. Rozenberg. (eds). 1999. "Handbook of Graph Grammars and Computing by Graph Transformation. Vol 1, 2 and 3". World Scientific.

[7] N. E. Fenton y S. L. Pfleeger. "Software Metrics: A Rigorous Approach". Chapman&Hall.

[8] M. Fowler. 1999. "Refactoring: Improving the Design of Existing Code". Addison Wesley.

[9] J. C. Grundy, W. B. Mugridge, y J. G. Hosking. 1998. "Visual Specification of MultiView Visual Environments". IEEE Symposium on Visual Languages, pp. 236-243.

[10] E. Guerra y J. de Lara. 2003. "A Framework for the Verification of UML Models. Examples using Petri Nets". Proc. JISBD'03. pp. 325-334.

[11] E. Guerra y J. de Lara. 2004. "Event-Driven Grammars: Towards the Integration of Meta-Modelling and Graph Transformation". LNCS 3256, pp. 54-69. Springer

[12] http://www.metacase.com/

[13] T. Mens, S. Demeyer, y D. Janssens. 2002. "Formalising Behaviour Preserving Program Transformation". LNCS 2505, pp. 286-301. Springer.

[14] T. Mens, T. Tourwé, y F. Muñoz. 2003. "Beyond the Refactoring Browser: Advanced Tool Support for Software Refactoring", Proc. IWPSE'03, pp. 39-44, IEEE Computer Society.

[15] Propuesta del grupo VT-Merge para QVT: http://www.omg.org/docs/ad/05-03-02.pdf.

[16] A. Schürr. 1994. "Specification of Graph Translators with Triple Graph Grammars". LNCS 903, pp. 151-163. Springer

[17] F. Simon, S. Löffler, y C. Lewerentz. 1999. "Distance Based Cohesion Measuring". En Proc. del 2nd European Software Measurement Conference (FESMA) 99, Technologisch Institut Amsterdam, pp. 69-83.

[18] P. J. Vázquez, M. N. Moreno, y F. J. García. 2001. "Métricas Orientadas a Objetos". Informe técnico DPTOIA-IT-2001-02. Universidad de Salamanca.

[19] J. Warmer y A. Kleppe. "The Object Constraint Language: Precise Modeling with UML". Addison-Wesley. 1998.

[20] N. Zhu, J. C. Grundy, y J. G. Hosking. 2004. "Pounamu: a meta-tool for multi-view visual language environment construction". Proc. 2004 VL/HCC, pp. 254-256.

Esther Guerra es Profesora Ayudante en el Departamento de Informática de la Universidad Carlos III de Madrid. Sus areas de investigación incluyen el Meta-Modelado, la Transformación de Grafos, y su aplicación en la generación automática de entornos para Lenguajes Visuales de Dominio Específico.

Paloma Díaz es Catedrática del Departamento de Informática de la Universidad Carlos III de Madrid. Es responsable del grupo de investigación DEI, cuyas actividades se centran en cuatro líneas básicas: ingeniería de la web, interacción persona-ordenador, tele-educación y aplicaciones para la sociedad de la información. Es coautora de varios libros y artículos de investigación y pertenece a diversas asociaciones profesionales, siendo Senior Member de IEEE, miembro de ACM y vicepresidenta del IEEE Learning Technology Technical Comitee.

Juan de Lara es Profesor Titular de Universidad en la Escuela Politécnica Superior de la Universidad Autónoma de Madrid. Es doctor en Ingeniería Informática por la misma universidad desde el año 2000. Sus areas de investigación incluyen el Meta-Modelado, la Transformación de Grafos, Modelado y Simulación, en particular basada en Agentes.