novática
CEPIS UPGRADE
Revista de la Asociación de Técnicos de Informática

Novática, revista fundada en 1975 y decana de la prensa informática española, es el órgano oficial de expresión y formación continua de **ATI** (Asociación de Técnicos de Informática), organización que edita también la revista **REICIS** (Revista Española de Innovación, Calidad e Ingeniería del Software). ***Novática*** edita asimismo **UP**GRADE, revista digital de **CEPIS** (*Council of European Professional Informatics Societies*), en lengua inglesa, y es miembro fundador de **UP**ENET (**UP**GRADE *European* ***NET****work*).

<http://www.ati.es/novatica/>
<http://www.ati.es/reicis/>
<http://www.upgrade-cepis.org/>

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

Consejo Editorial
Joan Batlle Montserrat, Rafael Fernández Calvo, Luis Fernández Sanz, Javier López Muñoz, Alberto Llobet Batllori, Gabriel Martí Fuentes, Josep Molas i Bertran, José Onofre Montesa Andrés, Olga Pallás Codina, Fernando Piera Gómez (Presidente del Consejo), Ramon Puigjaner Trepat, Miquel Sarries Griñó, Adolfo Vázquez Rodríguez, Asunción Yturbe Herranz

Coordinación Editorial
Llorenç Pagés Casas <pages@ati.es>
Composición y autoedición
Jorge Llácer Gil de Ramales
Traducciones
Grupo de Lengua e Informática de ATI <http://www.ati.es/gt/lengua-informatica/>, Dpto. de Sistemas Informáticos - Escuela Superior Politécnica - Universidad Europea de Madrid
Administración
Tomás Brunete, María José Fernández, Enric Camarero, Felicidad López

Secciones Técnicas - Coordinadores
Acceso y recuperación de la información
José María Gómez Hidalgo (Optenet), <jmgomezh@yahoo.es>
Manuel J. Maña López (Universidad de Huelva), <manuel.mana@diesia.uhu.es>
Administración Pública electrónica
Francisco López Crespo (MAE), <flc@ati.es>
Arquitecturas
Enrique F. Torres Moreno (Universidad de Zaragoza), <enrique.torres@unizar.es>
Jordi Tubella Morgadas (DAC-UPC), <jordit@ac.upc.es>
Auditoría SITIC
Marina Touriño Troitiño, <marinatourino@marinatourino.com>
Manuel Palao García-Suelto (ASIA), <manuel@palao.com>
Derecho y tecnologías
Isabel Hernando Collazos (Fac. Derecho de Donostia, UPV), <ihernando@legaltek.net>
Elena Davara Fernández de Marcos (Davara & Davara), <edavara@davara.com>
Enseñanza Universitaría de la Informática
Cristóbal Pareja Flores (DSIP-UCM), <cpareja@sip.ucm.es>
J. Ángel Velázquez Itúrbide (ESCET-URJC), <a.velazquez@escet.urjc.es>
Entorno digital personal
Alonso Alvarez García (TID), <aag@tid.es>
Diego Gachet Páez (Universidad Europea de Madrid), <gachet@uem.es>
Estándares Web
Encarna Quesada Ruiz (Oficina Española del W3C) <equesada@w3.org>
José Carlos del Arco Prieto (TCP Sistemas e Ingeniería) <jcarco@gmail.com>
Gestión del Conocimiento
Joan Baiget Solé (Cap Gemini Ernst & Young), <joan.baiget@ati.es>
Informática y Filosofía
José Angel Olivas Varela (Escuela Superior de Informática, UCLM) <joseangel.olivas@uclm.es>
Karim Gherab Martín (Harvard University) <kgherab@gmail.com>
Informática Gráfica
Miguel Chover Sellés (Universitat Jaume I de Castellón), <chover@lsi.uji.es>
Roberto Vivó Hernando (Eurographics, sección española), <rvivo@dsic.upv.es>
Ingeniería del Software
Javier Dolado Cosín (DLSI-UPV), <dolado@si.ehu.es>
Luis Fernández Sanz (PRIS-EI-UEM), <lufern@dpris.esi.uem.es>
Inteligencia Artificial
Vicente Botti Navarro, Vicente Julián Inglada (DSIC-UPV)
<{vbotti, vinglada}@dsic.upv.es>
Interacción Persona-Computador
Julio Abascal González (FI-UPV), <julio@si.ehu.es>
Lengua e Informática
M. del Carmen Ugarte García (IBM), <cugarte@ati.es>
Lenguajes informáticos
Andrés Marín López (Univ. Carlos III), <amarin@it.uc3m.es>
Óscar Belmonte Fernández (Univ. Jaime I de Castellón), <belfern@lsi.uji.es>
Lingüística computacional
Xavier Gómez Guinovart (Univ. de Vigo), <xgg@uvigo.es>
Manuel Palomar (Univ. de Alicante), <mpalomar@dlsi.ua.es>
Mundo estudiantil y jóvenes profesionales
Federico G. Mon Trotti (RITSI) <gnu.fede@gmail.com>
Mikel Salazar Peña (Area de Jovenes Profesionales, Junta de ATI Madrid), <mikeltxo_uni@yahoo.es>
Profesión informática
Rafael Fernández Calvo (ATI), <rfcalvo@ati.es>
Miquel Sàrries Griñó (Ayto. de Barcelona), <msarries@ati.es>
Redes y servicios telemáticos
José Luis Marzo Lázaro (Univ. de Girona), <joseluis.marzo@udg.es>
Germán Santos Boada (UPC), <german@ac.upc.es>
Seguridad
Javier Areitio Bertolín (Univ. de Deusto), <jareitio@eside.deusto.es>
Javier López Muñoz (ETSI Informática-UMA), <jlm@lcc.uma.es>
Sistemas de Tiempo Real
Alejandro Alonso Muñoz, Juan Antonio de la Puente Alfaro (DIT-UPM),
<{aalonso,jpuente}@dit.upm.es>
Software Libre
Jesús M. González Barahona, Pedro de las Heras Quirós (GSYC-URJC),
<{jgb,pheras}@gsyc.escet.urjc.es>
Tecnología de Objetos
Jesus García Molina (DIS-UM), <jmolina@um.es>
Gustavo Rossi (LIFIA-UNLP, Argentina), <gustavo@sol.info.unlp.edu.ar>
Tecnologías para la Educación
Juan Manuel Dodero Beardo (UC3M), <dodero@inf.uc3m.es>
César Pablo Córcoles Briongo (UOC), <ccorcoles@uoc.edu>.
Tecnologías y Empresa
Didac López Viñas (Universitat de Girona), <didac.lopez@ati.es>
Francisco Javier Cantais Sánchez (Indra Sistemas), <fjcantais@gmail.com>
TIC y Turismo
Andrés Aguayo Maldonado, Antonio Guevara Plaza (Univ. de Málaga)
<{aguayo, guevara}@lcc.uma.es>

Las opiniones expresadas por los autores son responsabilidad exclusiva de losmismos. ***Novática*** permite la reproducción, sin ánimo de lucro, de todos los artículos, a menos que lo impida la modalidad de © o *copyright* elegida por el autor, debiéndose en todo caso citar su procedencia y enviar a ***Novática*** un ejemplar de la publicación.

Coordinación Editorial, Redacción Central y Redacción ATI Madrid
Padilla 66, 3º, dcha., 28006 Madrid
Tlfn.914029391; fax.913093685 <novatica@ati.es>
Composición, Edición y Redacción ATI Valencia
Av. del Reino de Valencia 23, 46005 Valencia
Tlfn./fax 963330392 <secreval@ati.es>
Administración y Redacción ATI Cataluña
Via Laietana 46, ppal. 1ª, 08003 Barcelona
Tlfn.934125235; fax 934127713 <secregen@ati.es>
Redacción ATI Andalucía
Isaac Newton, s/n, Ed. Sadiel,
Isla Cartuja 41092 Sevilla, Tlfn./fax 954460779 <secreand@ati.es>
Redacción ATI Aragón
Lagasca 9, 3-B, 50006 Zaragoza.
Tlfn./fax 976235181 <secreara@ati.es>
Redacción ATI Asturias-Cantabria <gp-astucant@ati.es>
Redacción ATI Castilla-La Mancha <gp-clmancha@ati.es>
Suscripción y Ventas <http://www.ati.es/novatica/interes.html>, ATI Cataluña, ATI Madrid
Publicidad
Padilla 66, 3º, dcha., 28006 Madrid
Tlnf.914029391; fax.913093685 <novatica@ati.es>
Imprenta: Derra S.A., Juan de Austria 66, 08005 Barcelona.
Depósito legal: B 15.154-1975 -- ISSN: 0211-2124; CODEN NOVAEC
Portada: "Razón áurea" - Concha Arias Pérez / © ATI
Diseño: Fernando Agresta / © ATI 2003

Nº 192, marzo-abril 2008, año XXXIV

sumario

editorial
La elaboración de normas en el ámbito de las TIC > 02
en resumen
El corazón del problema > 02
Llorenç Pagés Casas
IFIP
Reunión del TC1 (Foundations of Computer Science) > 03
Michael Hinchey, Joaquim Gabarró Vallés
Task Force de IFIP para estudiar el nuevo modelo de afiliación > 03
Ramon Puigjaner Trepat

monografía
Desarrollo de Software Dirigido por Modelos
(*En colaboración con* **UP**GRADE)
Editores invitados: *Jean Bézivin, Antonio Vallecillo Moreno, Jesús García Molina y Gustavo Rossi*
Presentación. Siete años de MDA®: pasado, presente y futuro > 04
Jean Bézivin, Antonio Vallecillo Moreno, Jesús García Molina, Gustavo Rossi
Una breve historia de MDA > 09
Andrew Watson
Manifestaciones sobre MDA > 13
Bran Selic
Entornos de desarrollo integrados específicos de dominio > 17
Steve Cook, Stuart Kent
Inteligencia de modelos: un enfoque para guiar el modelado > 21
Jules White, Douglas C. Schmidt, Andrey Nechypurenko, Egon Wuchner
Diferencias entre modelos en Eclipse EMF > 28
Cédric Brun, Alfonso Pierantonio
Arquitectura dirigida por modelos en Eclipse > 33
Richard C. Gronback, Ed Merks
Ingeniería Web dirigida por modelos > 37
Nora Koch, Santiago Meliá Beigbeder, Nathalie Moreno Vergara, Vicente Pelechano Ferragud, Fernando Sánchez Figueroa, Juan Manuel Vara Mesa

secciones técnicas

Arquitecturas
Microbots: Fundamentos y aplicaciones > 42
José María Rodríguez Corral, Arturo Morgado Estévez, Francisco Cordón González, Rafael González Chacón, Ignacio García Vargas
Enseñanza Universitaria de la Informática
Preguntas frecuentes y nuestras respuestas favoritas sobre la pertinencia de los métodos docentes centrados en el estudiante para adaptar una asignatura al EEES > 48
Miguel Valero-García, Juan José Navarro Guerrero
Estándares Web
DENEB: Una plataforma para el desarrollo y ejecución de procesos Web dinámicos > 51
Javier Fabra Caro, Pedro Álvarez Pérez-Aradros, José Ángel Bañares Bañares, Joaquín Ezpeleta Mateo
Redes y servicios telemáticos
Una radiografía del estado de Internet > 56
José Luis Gahete Díaz, Natalia Fernández Gallego, Ana María Garzo Ortega, Gonzalo Martín Villaescusa
Referencias autorizadas > 62

sociedad de la información

Comunidades y trabajo colaborativo
Prácticas científicas y Sociedad del Conocimiento: el ejemplo de las comunidades FLOSS > 68
Roberto Feltrero Oreja
Programar es crear
Voto Útil (CUPCAM 2007, problema B, enunciado) > 73
Manuel Freire Morán, Julio Mariño Carballo
Buscando orejas (CUPCAM 2007, problema C, enunciado) > 74
Dolores Lodares González
Polígonos Escalera (CUPCAM 2007, problema A, solución) > 75
Manuel Abellanas Oar, Manuel Freire Morán

asuntos interiores

Coordinación Editorial / Programación de Novática / Socios Institucionales > 77

Monografía del próximo número: "*El futuro de la tecnología educativa*"

Jules White[1], Douglas C. Schmidt[1], Andrey Nechypurenko[2], Egon Wuchner[2]
[1]*Vanderbilt University, Nashville, Tennessee (USA);* [2]*Siemens AG, Munich (Alemania)*

<jules.white@gmail.com>,
<d.schmidt@vanderbilt.edu>,
<andrey.nechypurenko@siemens.com>,
<egon.wuchner@siemens.com>

Inteligencia de modelos: un enfoque para guiar el modelado

Traducción: José E. Rivera (Dpto. Lenguajes y Ciencias de la Computación, Universidad de Málaga <rivera@lcc.uma.es>)

Resumen: *la Ingeniería Dirigida por Modelos (Model-Driven Engineering, MDE) facilita la creación de soluciones en muchos dominios de aplicaciones empresariales a través del uso de abstracciones y de restricciones específicas del dominio. Una cualidad importante de los enfoques MDE es su capacidad para comprobar una solución para requisitos específicos del dominio, como por ejemplo las restricciones de seguridad, que son difíciles de evaluar cuando se siguen desarrollos tradicionales centrados en el código fuente. Sin embargo, en muchos dominios empresariales el desafío no es simplemente comprobar la corrección de una solución, sino encontrar soluciones válidas. Así, en estos dominios surge la necesidad de aplicar lo que se denomina "inteligencia de modelos" que usa las restricciones del dominio para guiar a los modeladores hacia las soluciones válidas que satisfagan las restricciones impuestas. Este artículo muestra como las técnicas existentes de especificación y comprobación de restricciones, tales como OCL (Object Constraint Language), pueden ser adaptadas y aprovechadas para guiar a los usuarios hacia soluciones correctas usando señales visuales.*

Palabras clave: *comprobación de restricciones, guías de modelado, ingeniería dirigida por modelos, modelado específico de dominio, razonamiento con restricciones.*

Autores

Jules White es estudiante de doctorado en el *Department of Electrical Engineering and Computer Science* (EECS) de la Universidad de Vanderbilt. Su investigación se centra en el uso de técnicas de optimización con restricciones para guiar el modelado, y en el ensamblado automático basado en restricciones y optimización de aplicaciones que combinan componentes, desarrollo dirigido por modelos y sistemas distribuidos Java. Es el desarrollador responsable del *Generic Eclipse Modeling System* (GEMS) <http://www.eclipse.org/gmt/gems>, que forma parte del proyecto Eclipse GMT. Antes de pertenecer al grupo DOC, trabajó para el Cambridge Innovation Center de IBM y estuvo trabajando con modelado de restricciones y sistemas basados en reglas.

Douglas C. Schmidt es *Full Professor* en el *Department of Electrical Engineering and Computer Science* (EECS), *Associate Chair* del Programa *Computer Science and Engineering*, y *Senior Research Scientist* en el *Institute for Software Integrated Systems* (ISIS) de la Universidad de Vanderbilt, Nashville (Tennessee). Durante las últimas dos décadas, ha realizado investigación pionera en patrones, técnicas de optimización y análisis empíricos para *frameworks* orientados a objetos y basados en componentes y para herramientas de desarrollo dirigido por modelos que facilitan el desarrollo de *middleware* y sistemas distribuidos. Es un experto en patrones de computación distribuida y *frameworks* de *middleware*, y ha publicado más de 350 artículos científicos y 9 libros que cubren un amplio rango de temas que incluyen sistemas software de comunicaciones de altas prestaciones, procesamiento paralelo para protocolos de red de alta velocidad, computación distribuida orientada a objetos y de tiempo real, patrones orientados a objetos para sistemas distribuidos y concurrentes, y herramientas de desarrollo dirigido por modelos.

Andrey Nechypurenko es ingeniero de software senior en Siemens AG Corporate Technology (CT SE2). Ofrece servicios de consultoría a las unidades de negocio de Siemens sobre sistemas embebidos y sistemas distribuidos de tiempo real. También participa en actividades de investigación relacionadas con el desarrollo dirigido por modelos y computación paralela. Antes de incorporarse a Siemens AG, trabajó en Ucrania en sistemas distribuidos de altas prestaciones en el dominio de las telecomunicaciones.

Egon Wuchner trabaja como investigador y consultor en Siemens AG Corporate Technology SE2 de Munich (Alemania). Es un experto en arquitectura de software y sistemas distribuidos. Su investigación actual se centra en conceptos, tecnología y herramientas para mejorar el desarrollo de grandes sistemas distribuidos, por ejemplo, su manejo de requisitos operacionales, su mantenimiento y su comprensión. Sus últimos trabajos de investigación se han centrado en Desarrollo de Software Dirigido por Aspectos y Desarrollo Dirigido por Modelos.

1. Introducción

La Ingeniería Dirigida por Modelos *(Model-Driven Engineering, MDE)* [1] ha surgido como un enfoque muy eficaz para la construcción de sistemas empresariales complejos. MDE permite a los desarrolladores construir soluciones usando abstracciones, tales como las que proporcionan los lenguajes visuales adaptados al dominio de la solución. Por ejemplo, en el dominio de despliegue de componentes software a servidores de datos, los desarrolladores pueden manipular diagramas visuales mostrando cómo son asignados los componentes software a los nodos individuales, tal y como se muestra en la **figura 1**.

Un beneficio importante proporcionado por los enfoques MDE es que se pueden capturar e incluir en las propias herramientas las restricciones del dominio. Estas restricciones son propiedades, como por ejemplo los requisitos de memoria de un componente software que precisa de un servidor, que no pueden ser comprobadas de forma sencilla por un compilador u otra herramienta de un lenguaje de programación de tercera generación. Las restricciones del dominio sirven como un compilador de soluciones del dominio que pueden mejorar de forma significativa el grado de confianza en la corrección de una solución. El lenguaje de especificación de restricciones más ampliamente usado es OCL (*Object Constraint Language*) [2].

Aunque MDE puede mejorar la corrección de las soluciones y capturar errores que previamente eran difíciles de identificar, en muchos dominios el principal reto es *derivar* la solución correcta, y no solamente comprobar la corrección de la solución. Por ejemplo, en el despliegue de componentes software a servidores (nodos), cada componente puede tener asociadas numerosas restricciones funcionales (como por ejemplo el requerir que un conjunto específico de otros componentes estén instalados en el mismo servidor que él) y no funcionales (como requerir que un nodo disponga de cortafuegos), que hacen que el desarrollo de un modelo de despliegue sea difícil. Cuando nos encontramos con grandes modelos industriales con

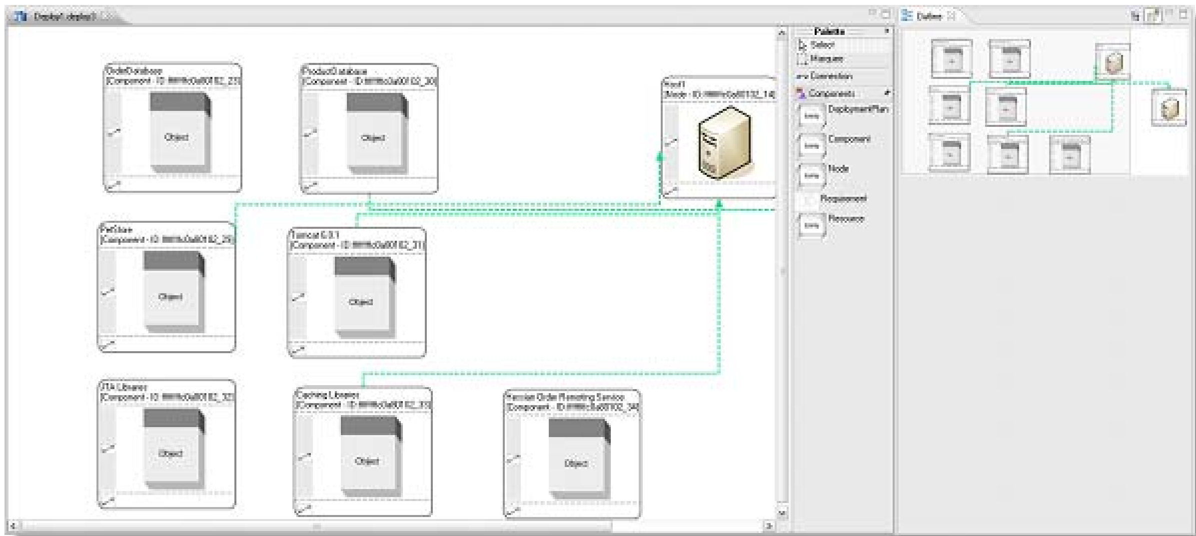

Figura 1. Modelo de despliegue de un *DataCenter*.

decenas, centenas, o millares de elementos y múltiples restricciones por cada elemento, la construcción y la validación de modelos de forma manual no son procesos fácilmente escalables.

Los modelos industriales también pueden contener restricciones globales, como por ejemplo estipular que los componentes asignados a un nodo no excedan su RAM disponible, que complican aún más el modelado. Aunque se pueden usar lenguajes como OCL para validar una solución, éstos no facilitan encontrar la solución correcta. Al revés, los desarrolladores deben primero construir modelos de forma manual, y luego invocar la comprobación de los requisitos para observar si se ha cometido algún error.

A continuación mostramos un conjunto de propiedades de los modelos industriales que dificultan su construcción:

1. Los modelos industriales son normalmente grandes y pueden contener varias vistas, haciendo difícil o imposible observar toda la información necesaria para tomar una decisión de modelado compleja.
2. Los requisitos en los sistemas industriales normalmente integran aspectos funcionales y no funcionales que están repartidos por múltiples vistas o aspectos, y que son difíciles de resolver manualmente.
3. Se puede requerir que las soluciones de modelado sean óptimas o que satisfagan restricciones globales complejas, lo que conlleva a evaluar un gran número de potenciales modelos solución.

Las técnicas actuales de construcción de modelos son procesos manuales en gran medida. La dificultad de comprender y abordar un gran modelo industrial de forma completa, junto con la necesidad de encontrar y evaluar un gran número de soluciones potenciales, hace que el modelado industrial sea complejo.

Para motivar la necesidad de herramientas que ayuden a los modeladores a deducir

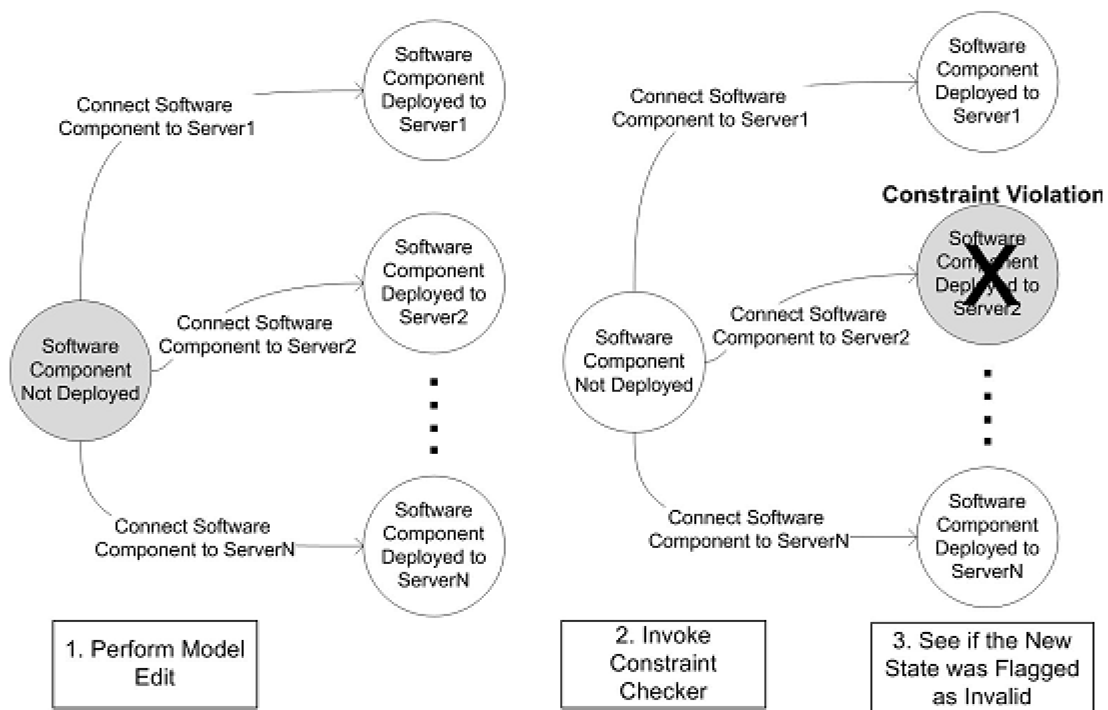

Figura 2. Edición de modelos y comprobación de restricciones.

soluciones conformes a las restricciones del dominio, introduciremos un ejemplo de aplicación de modelado del despliegue de componentes software sobre los servidores de una empresa. De forma ideal, cuando un desarrollador hiciera clic en un componente software para desplegarlo, la infraestructura de la herramienta subyacente usaría las restricciones del dominio para obtener los nodos válidos para dicho componente. Denominaremos "inteligencia de modelos" a estos mecanismos que guían a los modeladores hacia soluciones correctas.

2. Limitaciones de los actuales enfoques de comprobación de restricciones

Para motivar los retos de utilizar la tecnología existente para restricciones, tal como OCL, como un mecanismo de guía, evaluaremos una restricción simple de despliegue de un componente software sobre un servidor: "para cada componente, el nodo en el que se despliega debe tener el sistema operativo (OS) para el cual se compila el componente".

Esta restricción se puede expresar en OCL de la siguiente forma:

```
Context SoftwareComponent
  inv:  self.hostingServer.OS  =
self.requiredOS
```

Después de que un componente software (`SoftwareComponent`) haya sido desplegado a un servidor, la restricción anterior comprueba que el nodo (almacenado en la variable `hostingServer`) tenga el sistema operativo requerido por el componente. Como se muestra en la **figura 2**, para utilizar esta restricción, el modelador tiene que realizar primero un cambio al modelo (paso 1), después invocar el comprobador de restricciones (paso 2) y por último observar si se ha alcanzado un estado de error (paso 3). El problema radica en que el modelador no puede predecir por adelantado si el modelo va a alcanzar un estado inválido, sino que el hecho de que un estado tenga errores sólo se comprueba cuando el estado ya ha sido alcanzado.

Una manera de intentar subsanar esta incapacidad de comprobar la restricción antes de que al nodo le haya sido asignado el `SoftwareComponent` es usar las precondiciones OCL como "guardas" en las transiciones. Una precondición OCL es una expresión que debe satisfacerse antes de ejecutarse una operación. Sin embargo, el principal problema de usar precondiciones OCL como guardas es que son diseñadas para especificar el correcto comportamiento de una operación realizada por la *implementación* del modelo. Este hecho obliga a que usar una precondición OCL como guarda durante el modelado requiera definir un requisito en términos de la operación realizada por la herramienta de modelado y no por el modelo en sí.

Por ejemplo, la precondición que debe imponerse para comprobar que el sistema operativo es el adecuado es un requisito sobre una operación (como puede ser la creación de una conexión entre dos elementos) realizada por la herramienta de modelado, y no por el modelo. Por lo tanto, para definir una precondición OCL, los desar-rolladores deben definir las restricciones OCL en términos de la definición de la operación de la herramienta de modelado, que puede no usar la misma terminología que el modelo. Además, definir una restricción como una precondición de una operación realizada por la herramienta de modelado requiere que los desarrolladores creen una restricción duplicada para comprobar si un estado del modelo existente es correcto o no.

Sin estas dos restricciones (una para comprobar la corrección de la acción de la herramienta de modelado y otra para comprobar la corrección de un estado ya construido) es imposible identificar los extremos (origen y destino) de la operación y asegurar la consistencia del modelo. Por lo tanto, el enfoque basado en precondiciones OCL añade una notable complejidad al requerir que los desarrolladores mantengan definiciones separadas (y no necesariamente idénticas) de la restricción, que además pueden no estar sincronizadas. Además, este enfoque también acopla la restricción a una única plataforma de modelado, ya que la precondición se define en términos de la operación de conexión que la herramienta (y no el modelo) provee.

3. Inteligencia de modelos: un enfoque para guiar el modelado

Una herramienta de modelado puede implementar la "inteligencia de modelos" usando restricciones para obtener estados finales válidos a la hora de editar un modelo, *antes* de que registre el cambio en el modelo. Los mecanismos tradicionales de especificación de restricciones asocian una restricción a objetos (por ejemplo, `SoftwareComponent`) en lugar de a las asociaciones entre ellos (por ejemplo, la relación de despliegue entre un `SoftwareComponent` y un `Server`). Por lo tanto, para determinar si es válida la relación entre dos objetos, la relación debe ser creada y efectuada en el modelo para que las restricciones existentes sobre los dos objetos asociados a la relación puedan ser comprobadas.

Las transiciones en el diagrama de estados de la **figura 2** corresponden a la creación de relaciones entre objetos. Para que una herramienta soporte cierta "inteligencia de modelos" debe usar restricciones del dominio para comprobar la corrección de una modificación de una relación entre objetos en un modelo antes de que la modificación se registre en el modelo. Si las restricciones se asocian a las relaciones en lugar de a los objetos, las herramientas pueden usar dichas restricciones no sólo para obtener estados finales válidos, sino también para sugerir transiciones al modelador.

3.1. Restricciones sobre relaciones

Las relaciones entre objetos son los arcos en el grafo de objetos subyacente de un modelo. Cada arco tiene un objeto origen y un objeto destino. De esta manera, pueden crearse restricciones que especifiquen la corrección de una relación en términos de las propiedades de los elementos origen y destino de una transición. Por ejemplo, si el despliegue de un componente software a un servidor se representa como una relación de *despliegue*, podríamos aplicarle una restricción especificándola en términos de las propiedades del elemento origen (p.ej., un `Software Component`) y el elemento destino (p.ej., un `Server`):

```
context Deployment
  inv: source.requiredOS = target.OS
```

Una propiedad clave a la hora de asociar restricciones y especificarlas en términos de los elementos origen y destino de la relación, es poder usar una restricción para comprobar la corrección de la creación de una relación *antes* de que la relación se registre en el modelo. Así, antes de crear la relación en cuestión, se pueden establecer los elementos origen y destino a los que se refiere la expresión de la restricción, ejecutar esta expresión y comprobar si se cumple o no. En caso de que se cumpla, la correspondiente relación puede ser creada en el modelo.

En la **sección 2** mostramos que si para guiar el modelado se usan las técnicas y herramientas OCL existentes se requiere mantener especificaciones separadas de cada restricción. Si las restricciones se asocian a las relaciones y se expresan en términos de los elementos origen y destino de una relación, podemos usarlas para comprobar la validez de una acción de modelado *antes* de que sea registrada en el modelo. Además, la misma restricción se puede usar también para comprobar las relaciones existentes entre elementos de modelado, lo que no puede ser realizado con las prácticas estándares de OCL.

3.2. Derivación de los extremos de una relación

Un modelo puede considerarse como una base de conocimiento. En ese caso, los elementos del modelo definirían hechos sobre la solución. El objetivo de la inteligencia de modelos es ejecutar consultas sobre la base del conocimiento y obtener los extremos

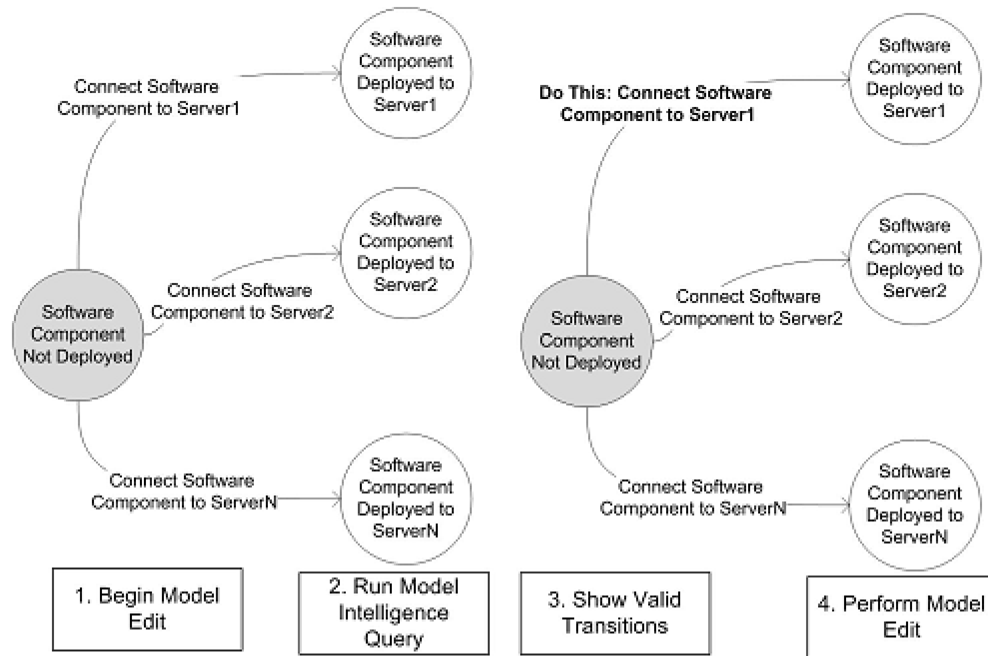

Figura 3. Secuencia de edición de un modelo para la inteligencia de modelos.

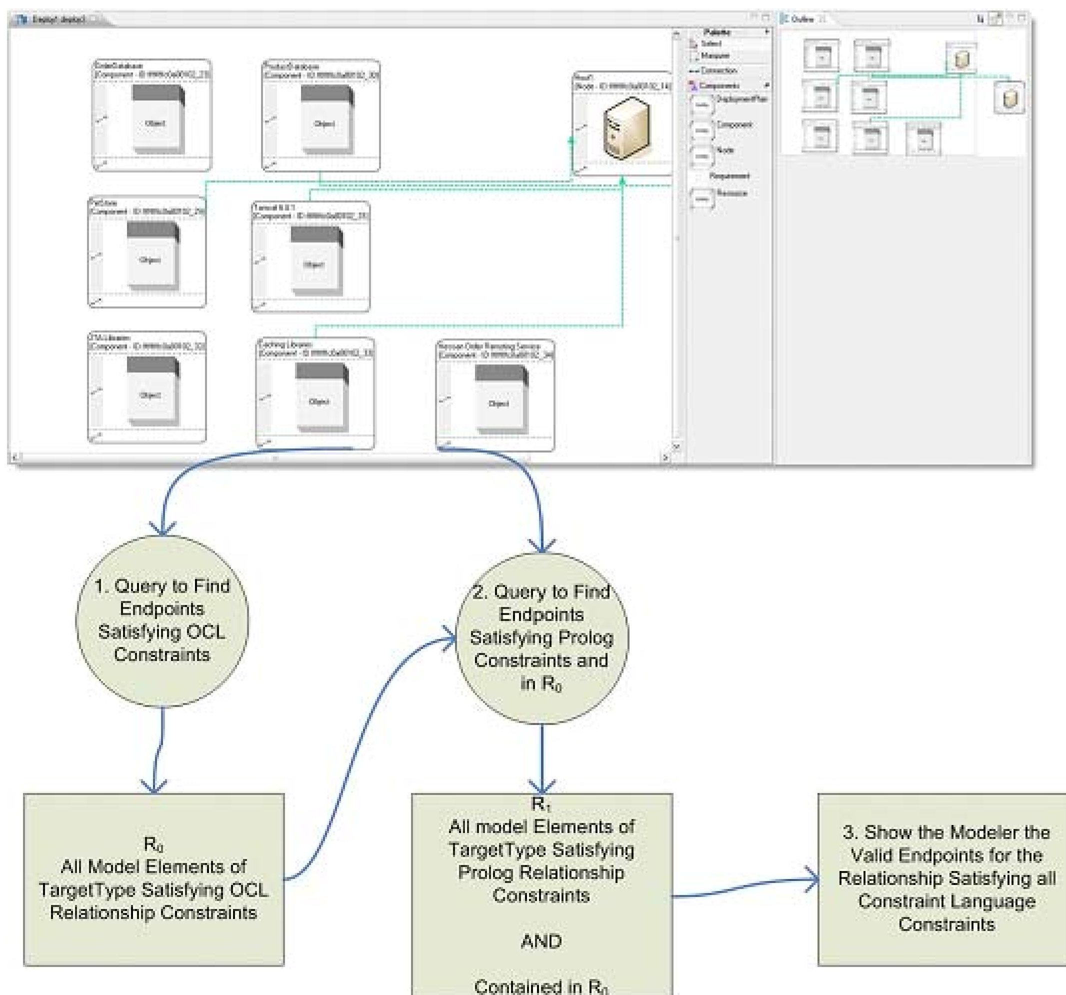

Figura 4. Consultas de inteligencia de modelos sobre múltiples lenguajes de restricciones.

válidos de una relación (por ejemplo los nodos válidos para un determinado componente) que está siendo creada por un modelador. En un diagrama de estados en el que se detalle un escenario de edición de modelos, como el mostrado en la **figura 3**, las consultas sirven para obtener los estados válidos a los que el modelo puede transitar.

La creación de una relación (que se corresponde a una posible transición válida del modelo) empieza cuando un modelador selecciona un tipo de relación (por ejemplo, una relación de despliegue) y un extremo para la nueva relación (por ejemplo, un `SoftwareComponent`).

La inteligencia de modelos usa primero el tipo de la relación para determinar las restricciones que deben cumplirse para esa relación, y después usa dichas restricciones para realizar consultas que busquen en la base de conocimiento los extremos válidos para crear la relación, tal y como se muestra en el paso 2 de la **figura 3**. Los extremos válidos determinan los estados válidos a los que el modelo puede transitar. Como se muestra en el paso 3 de la **figura 3**, a los modeladores pueden sugerirse las transiciones que conducen a estados válidos, de forma que puedan seleccionarlas como posibles formas válidas de completar la edición del modelo en curso.

Cada tipo de relación tiene un conjunto de restricciones asociadas. Una vez que la inteligencia de modelos conoce el objeto origen y las restricciones OCL de la relación que está siendo modificada, se puede ejecutar una consulta para encontrar extremos válidos para completar la relación. Por ejemplo, si usáramos la restricción de despliegue del sistema operativo de la **sección 2**, la consulta para encontrar los extremos de una relación de despliegue quedaría de la siguiente forma:

```
Server.allInstances()->collect(target | target.OS = source.OS);
```

En este ejemplo, la inteligencia de modelos especificaría al motor de OCL que la variable `source` se ha ligado al `SoftwareComponent` que ha sido establecido como el elemento origen de la relación de despliegue. La consulta entonces devolvería la lista de todos los servidores que tuvieran el sistema operativo adecuado para el componente. Para cualquier relación con una restricción `Constraint` entre elementos `Source` y `Target` de tipos `SourceType` y `TargetType`, respectivamente, se puede construir una consulta que obtenga los extremos válidos. Asumiendo que una relación tiene un conjunto de elementos origen `Source`, una consulta para encontrar los valores potenciales de los destinos Target se construiría de la siguiente forma:

```
TargetType->allInstances()->collect(target | Constraint);
```

donde `Constraint` es una expresión booleana sobre las variables origen y destino. De forma más general, la consulta se puede expresar como: *Encuentra todos los elementos del tipo* `TargetType` *para los que la restricción* `Constraint` *se cumpla siendo* `Source` *el elemento origen.*

3.3. Obtención de los extremos con múltiples lenguajes de restricciones

Aunque hasta ahora nos hayamos centrado en OCL, la definición de la consulta generalizada de la **sección 3.2** también puede adaptarse a otros lenguajes de restricciones. En un trabajo previo [4], implementamos la inteligencia de modelos usando OCL, Prolog, BeanShellm y Groovy. Por ejemplo, Prolog define de forma natural una base de conocimiento como un conjunto de hechos definidos usando la lógica de predicados. Las consultas se pueden llevar a cabo en la base de conocimiento de Prolog especificando restricciones que tienen que ser asociadas a los hechos obtenidos.

La inteligencia de modelos también puede usarse para obtener soluciones que satisfagan un grupo de restricciones definidas en múltiples lenguajes heterogéneos. En ese caso, se puede usar un proceso de filtrado iterativo para obtener los extremos que satisfagan las restricciones de los distintos lenguajes, como se muestra en la **figura 4**.

Inicialmente, la inteligencia de modelos lanza una consulta para obtener las soluciones potenciales con respecto al conjunto de restricciones de un lenguaje de restricciones específico. El resultado de esta consulta se

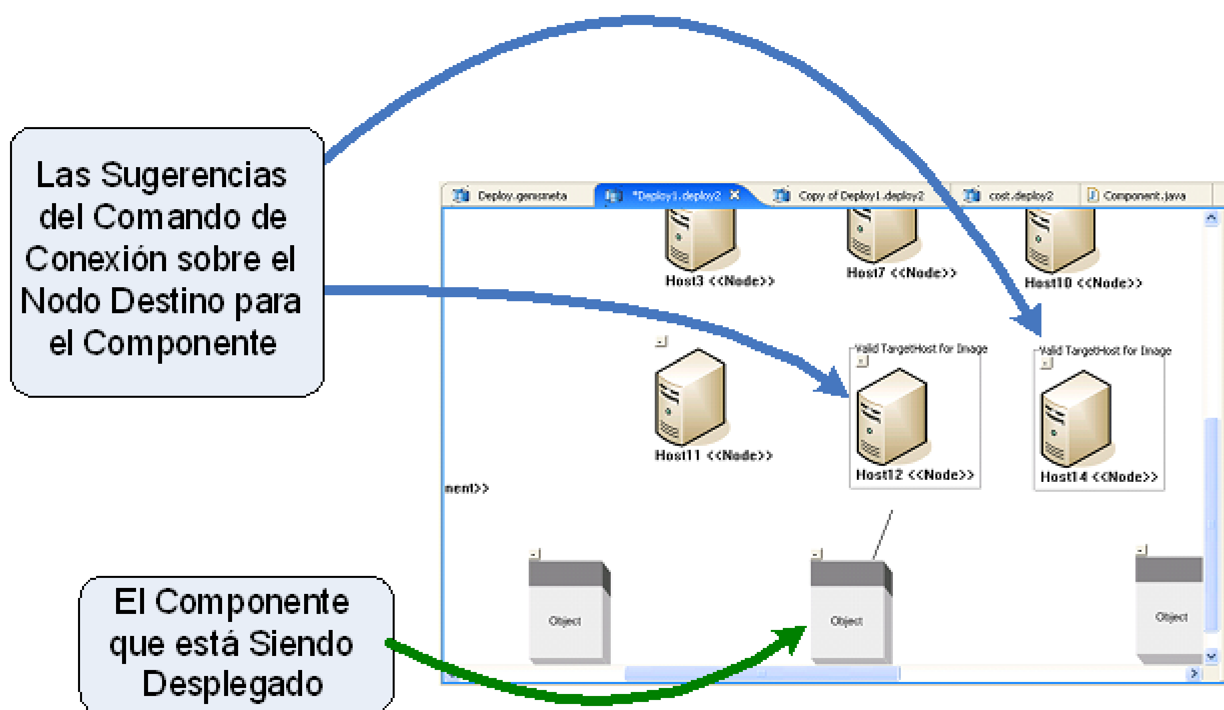

Figura 5. El comando de despliegue mostrando los extremos válidos obtenidos a través de la inteligencia de modelos.

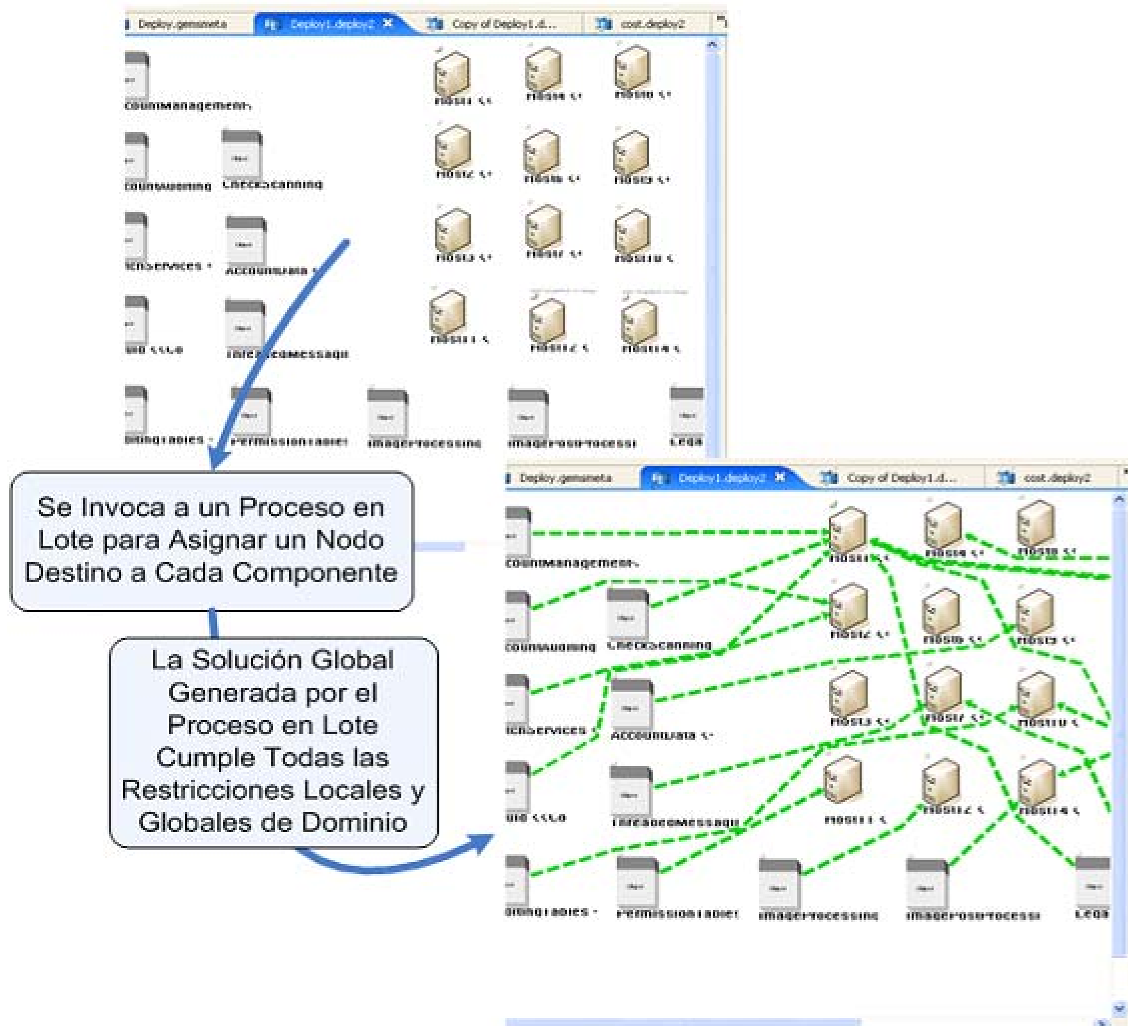

Figura 6. Un proceso en lote de inteligencia de modelos para asignar un nodo a cada componente.

almacena en el conjunto R_0. Para cada lenguaje de restricciones C_i, los resultados de la consulta que satisfagan las restricciones del lenguaje se almacenan en R_i. Para cada lenguaje de restricciones C_i, donde $i > 0$, la inteligencia de modelos lanza una consulta usando una versión modificada del formato de consulta definido en la **sección 3.2**: *Encuentra todos los elementos del tipo*

`TargetType` *en los que se cumpla la restricción* `Constraint`, *sea* `Source` *el elemento origen y sean miembros del conjunto* R_{i-1}.

La versión modificada de la consulta introduce una nueva restricción en la solución obtenida: todos los elementos obtenidos como resultado tienen que ser miembros del anterior conjunto de resultados. Una manera simple de especificar conjuntos de resultados es asociando un identificador único a cada elemento de modelado, capturando los resultados de las consultas como una lista de esos identificadores. Así, las consultas modificadas pueden definirse comprobando tanto que se cumple el conjunto de restricciones como que el identificador de cada elemento de modelado obtenido está contenido en el conjunto anterior de resultados.

4. Integración de la inteligencia de modelos en el patrón *Command*

Existen numerosas aplicaciones de la inteligencia de modelos, tales como la ejecución automática de un proceso en lote autónomo de operaciones de edición sobre modelos, y la de proveer a los modeladores retroalimentación visual. En esta sección mostramos cómo se puede integrar la inteligencia de modelos con el *patrón Command* [3] para proporcionar indicaciones visuales que ayuden a los modeladores a completar las acciones de modelado correctamente.

El patrón *Command* encapsula una acción, y los datos que ésta necesita, en un objeto. Este patrón se usa en varios entornos de modelado gráfico, como por ejemplo en el Eclipse Graphical Editor Framework [5]. A medida que los modeladores editan un modelo, los "comandos" se van creando y ejecutando sobre el modelo para realizar las acciones que desea el modelador.

Las plataformas de modelado proporcionan herramientas, como por ejemplo una herramienta de conexión, que un modelador usa para manipular un modelo. Cada herramienta está soportada por un determinado objeto *command*, como por ejemplo un *command* "conexión". Cuando un modelador elige una herramienta, se crea una instancia de la clase del correspondiente *command*. Las posteriores acciones: apuntar, hacer clic y teclear algunos caracteres, serán las que establezcan los argumentos con los que opera el *command* (por ejemplo, los extremos de la conexión). Cuando los argumentos del *command* queden especificados completamente (por ejemplo, cuando se establezcan ambos extremos del *command* de conexión), el *command* se ejecuta.

En la **sección 3** se describió cómo era posible resaltar visualmente las localizaciones de despliegue válidas de un componente software después de que un modelador hiciera

clic sobre él para iniciar la conexión de despliegue. Esta funcionalidad puede lograrse combinando la inteligencia de modelos con el *command* de conexión de despliegue. Después de establecerse su argumento inicial (el origen), el *command* puede usar la inteligencia de modelos para consultar las localizaciones de despliegue válidas. Si sólo existe un servidor que pueda albergar al componente, el *command* puede elegirlo de forma autónoma como localización del despliegue y puede pasar a ejecutarse.

Si hay más de un nodo potencialmente válido, puede mostrarlos todos para ayudar al modelador a elegir el argumento final del *command*, como se muestra en la **figura** 5.

5. Conclusiones

Nuestra experiencia en el desarrollo de modelos para dominios empresariales nos ha enseñado que no basta con determinar si un modelo es correcto o no. Hemos aprendido que el uso de restricciones para verificar la corrección en las relaciones entre objetos (en lugar de sobre los estados de los objetos individuales) permite a las herramientas de modelado guiar hacia soluciones correctas, sugiriendo distintas maneras de completar las acciones de edición. Además, pueden construirse procesos *en lote* sobre los mecanismos de sugerencia que permitan a las herramientas completar de forma autónoma conjuntos de acciones de modelado. Por ejemplo, se puede crear un proceso en lote para desplegar un gran conjunto de componentes software, obtener los conjuntos de nodos válidos para cada componente, y elegir de forma inteligente un nodo de cada conjunto, tal y como se muestra en la **figura 6**.

En otros trabajos [4], hemos usado la inteligencia de modelos como base para la creación de procesos en lote de modelado que usan motores de resolución de restricciones para automatizar grandes conjuntos de acciones de modelado y seleccionar de forma óptima los extremos de relaciones que permitan satisfacer las restricciones globales, u objetivos de optimización.

La inteligencia de modelos propuesta en este artículo ha sido implementada para el Eclipse Modeling Framework [6] bajo el nombre de *GEMS EMF Intelligence*, y es un proyecto de código abierto disponible en <www.eclipse.org/gmt/gems>.

Referencias

[1] J. Bézivin. "En búsqueda de un principio básico para la Ingeniería Guiada por Modelos". *Novatica nº 168 (marzo-abril 2004)*, pags. 18-20.

[2] J.B. Warmer, A.G. Kleppe. *The Object Constraint Language: Getting Your Models Ready for MDA*. Addison-Wesley Professional, New York, NY, USA (2003). ISBN: 0321179366.

[3] E. Gamma, R. Helm, R. Johnson, J. Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software*, Addison-Wesley, Boston, MA, USA (1995). ISBN: 0201633612.

[4] J. White, A. Nechypurenko, E. Wuchner, D.C. Schmidt. "Reducing the Complexity of Designing and Optimizing Large-scale Systems by Integrating Constraint Solvers with Graphical Modeling Tools, en "*Designing Software-Intensive Systems: Methods and Principles*, editado por Dr. Pierre F. Tiako, Langston University, Oklahoma, USA, (2008).

[5] Graphical Editor Framework, <www.eclipse.org/gef>.

[6] F. Budinsky, S.A. Brodsky, E. Merks. Eclipse Modeling Framework. Pearson Education, Upper Saddle River, NJ, USA, (2003).

INNOVATIVE CITIES FOR THE NEXT GENERATION CONFERENCE

26 June, Barcelona

¿Cómo será la ciudad del futuro?

Ahora tiene la oportunidad de saberlo el próximo 26 de Junio en Barcelona
Contaremos con la presencia de expertos europeos en eGovernment, miembros de la *European Network of Living Labs*, del proyecto *U-City* coreano y del *MIT SENSEable City La*b estadounidense.

El registro para el evento es gratuito pero las plazas son limitadas. Si desea consultar más información puede hacerlo a través de la página:

www.epractice.eu/workshop/icing

icing