XV JENUI. Barcelona, 8-10 de julio de 2009
ISBN: 978-84-692-2758-9
http://jenui2009.fib.upc.edu/

CUCKOO: Una plataforma web para la verificación de modelos UML

Santi Caballé, Jordi Cabot, Robert Clarisó, Jordi Conesa, Elena Planas, Daniel Riera
Estudis d'Informàtica, Multimèdia i Telecomunicació
Universitat Oberta de Catalunya
Rambla del Poblenou, 156
08018 Barcelona
{scaballe, jcabot, rclariso, jconesac, eplanash, drierat}@uoc.edu

Resumen

En este artículo presentamos CUCKOO (**QU**ality che**CK**ing of **O**bject **O**riented designs) una plataforma web para la verificación de diagramas de clases UML. Esta plataforma está orientada a facilitar el aprendizaje de las fases de análisis y diseño dentro de las asignaturas de Ingeniería del Software.

1. Motivación

La capacidad para realizar correctamente el análisis y el diseño de un sistema software es sin duda una de las competencias más importantes que debe adquirir todo estudiante como parte de las asignaturas del ámbito de la Ingeniería del Software.

En la actualidad, la notación más utilizada para modelar un sistema software es UML (Unified Modeling Language). Por ese motivo, UML y su aplicación al modelado de software forma parte de la gran mayoría (por no decir todos) de currículums de Ingeniería en Informática. El aprendizaje de esta materia se realiza típicamente a través de la propuesta por parte del profesor de diversos ejercicios de modelado. En la gran mayoría de los casos, dichas actividades son corregidas manualmente por el profesor, quien revisa las propuestas de los estudiantes para comprobar la calidad, corrección y completitud de sus modelos. Normalmente, los ejercicios de modelado son difíciles de corregir por su carácter "abierto" y por el hecho de existir diversas soluciones correctas a un mismo problema.

La falta de exhaustivos repositorios de ejercicios resueltos y las dificultades que supone una autoevaluación en este contexto (es mucho más difícil "probar" un modelo de software que la implementación del software, donde muchas veces hay la opción de compilarlo y ejecutarlo para "ver si funciona") hacen que los estudiantes dependan en exceso del profesor (y su dedicación) para avanzar en su aprendizaje.

En este artículo, proponemos un primer paso hacía una solución efectiva a esta situación mediante el uso de una herramienta Web que permite dar un feedback inmediato a ejercicios de modelado con UML. Más concretamente, nuestra herramienta CUCKOO permite, dado un diagrama de clases UML (complementado, opcionalmente, con un conjunto de restricciones textuales descritas mediante el lenguaje Object Constraint Language (OCL)) comprobar automáticamente ciertas propiedades de corrección del modelo (por ej. *satisfactibilidad*, es decir, que sea posible instanciar el modelo definido). Creemos que esta evaluación preliminar de la calidad del modelo ayuda en gran medidad al progreso del aprendizaje del estudiante.

2. Objetivos docentes

Como se ha comentado anteriormente, el principal objetivo de nuestra herramienta es facilitar la práctica del modelado de sistemas software con UML. De esta manera, el estudiante puede profundizar más en su aprendizaje a través del seguimiento de un conjunto de problemas de modelado (predefinidos o no por el

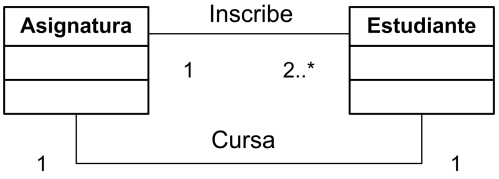

Figura 1: Ejemplo de modelo incorrecto

profesor en la propia herramienta) que puede realizar de forma autónoma. En este sentido creemos que la herramienta puede ser útil como recurso adicional en cursos de Ingeniería del Software, especialmente en los cursos que se focalizen en las fases de análisis y modelado conceptual. Después de introducir un nuevo tipo de elemento de modelado (p.ej. clases asociativas) el profesor podría preparar algunos ejercicios que hicieran incapié en el uso de esa construcción para obligar a los estudiantes a practicar con ella.

Aunque la herramienta no informa acerca de lo buena que es la solución del estudiante (es decir, lo parecida que es con la solución que tenia en mente el profesor) sí proporciona un primer feedback en base a un análisis de las propiedades de corrección del modelo. Por ejemplo, si el profesor propone un ejercicio para modelar la relación de "Inscribe" y "Cursa" entre asignaturas y estudiantes, y el estudiante propone una solución como la de la figura 1, la herramienta le dirá inmediatamente que su propuesta es incorrecta ya que no es ni siquiera instanciable. Es decir, no se puede crear ningún objeto de tipo "Estudiante" o "Asignatura" sin violar las restricciones de cardinalidad en alguna de las asociaciones: según "Inscribe", cada asignatura necesita al menos dos estudiantes distintos, mientras que según "Cursa" hay sólo un estudiante por cada asignatura. Al recibir este feedback, el estudiante puede empezar en seguida a repensar su modelo.

Este análisis nos lleva al segundo objetivo docente. Creemos que una herramienta como la nuestra (u otras similares, ver la sección de trabajo relacionado) permite abordar con más probabilidades de éxito la verificación de modelos software, un tema que pese a su gran importancia sigue sin estar presente en el currículum de muchas universidades españolas. Esta importancia viene del uso creciente de los métodos de desarrollo de software dirigido por modelos (conocidos como Model Driven Development) donde el código del sistema se genera (parcialmente) a partir del modelo inicial, de forma automática, con lo que la corrección del modelo tiene un efecto directo en la calidad del código generado.

Introducir aspectos de corrección de modelos es una cuestión siempre complicada en el entorno docente. Primero, para el estudiante y su típica aversión a los aspectos formales (y más en asignaturas de análisis y diseño donde se espera un grado elevado de formalización) y, segundo, por la propia complejidad del problema en sí (verificar un modelo es un problema indecidible en el caso general) [3, 5]. En este sentido, una herramienta que permita "explorar" estos conceptos y probar ejemplos de modelos correctos e incorrectos puede favorecer el auto-aprendizaje de forma significativa.

Finalmente, la necesidad de usar una herramienta de soporte al desarrollo de software con modelado UML (input de nuestra herramienta de verificación) siempre es un hecho positivo ya que obliga al estudiante a familiarizarse con un tipo de herramientas que va a utilizar con toda seguridad en su práctica profesional. Es bueno que desde un primer momento pueda ver las ventajas y limitaciones de este tipo de herramientas (ej. problemas de importación/exportación del modelo, detección de errores sintácticos, limitaciones de expresividad en los diagramas, etc.).

3. Descripción de la plataforma

3.1. Presentación

El objetivo de CUCKOO es ofrecer funcionalidades a cuatro perfiles de usuario:

- El *invitado* o *anónimo* puede verificar modelos cualesquiera sin necesidad de darse de alta en el sistema.
- El *estudiante* dispone de una colección de ejercicios sobre modelado que puede re-

solver, enviar las soluciones y recibir un feedback inmediato.

- El *profesor* puede editar los ejercicios de la colección y revisar los ejercicios resueltos por los estudiantes.
- El *administrador* puede añadir y eliminar usuarios al sistema.

Todas estas tareas pueden realizarse a través de una interfaz Web con cualquier navegador y no requieren la instalación de ningún componente adicional. De esta forma, se pretende evitar uno de los grandes inconvenientes de algunas herramientas de verificación: la necesidad de instalar y configurar software adicional (solvers, librerías externas, demostradores de teoremas, ...) en el ordenador del estudiante.

3.2. Verificación de modelos

Nuestra herramienta CUCKOO puede verificar propiedades de corrección sobre un modelo UML anotado con restricciones OCL. Estas propiedades se centran en la *satisfactibilidad* del modelo (es posible crear objetos que cumplan todas las restricciones del modelo?), la *redundancia* de las restricciones (pueden eliminarse restricciones sin cambiar la semántica del modelo?) y la *corrección de las operaciones OCL* (están bien definidas las precondiciones y postcondiciones de las operaciones?). Existe una lista predefinida de propiedades a verificar, y para cada ejercicio, el profesor define qué propiedades deben comprobarse. Por otro lado, los estudiantes y usuarios anónimos también pueden analizar un modelo cualquiera que no se corresponda con ningún ejercicio, y en ese caso pueden seleccionar ellos mismos qué propiedades van a analizar. La figura 2 muestra la página de selección de propiedades a verificar (vease [5, 6] para una descripción formal de cada propiedad).

Internamente, CUCKOO utiliza un solver de programación con restricciones llamado ECL$^{\text{i}}$PS$^{\text{e}}$[1] para analizar el modelo y detectar si una propiedad se cumple. Este solver se utiliza para buscar *ejemplos* y *contraejemplos* de las propiedades de forma totalmente automática. Estos ejemplos y contraejemplos son instancias del modelo, es decir diagramas de objetos creados a partir de las clases definidas en el diagrama de clases UML. Por ejemplo, cualquier diagrama de objetos válido demuestra la satisfactibilidad del modelo, y un diagrama de objetos donde se cumplen todas las restricciones del modelo excepto una restricción R demuestra que R no es redundante.

Para encontrar ejemplos y contraejemplos, CUCKOO traduce el modelo UML, las restricciones OCL y la propiedad de corrección a verificar, a un *problema de satisfacción de restricciones* [11]: un conjunto de variables, cada una con un dominio de valores posibles, y una serie de restricciones sobre dichas variables. Resolver el problema consiste en encontrar una asignación de valores a las variables que satisface todas las restricciones. En nuestro caso, las variables representan los objetos y atributos de nuestro modelo, y los dominios de las variables representan el número de objetos que intentaremos crear de cada tipo y los posibles valores para cada atributo. Así pues, la asignación resultante definirá un conjunto de objetos concretos. En caso de que no pueda encontrarse ninguna asignación válida, podremos concluir, por ejemplo, que el diagrama de clases no es satisfactible o que la restricción que buscamos es redundante *dentro los dominios que hemos proporcionado*. Es posible que existan asignaciones factibles fuera de estos dominios para las variables, por ejemplo, usando más objetos. Sin embargo, la inexistencia de soluciones en un dominio pequeño puede ser darnos alguna información: por ejemplo, si un modelo sencillo no puede instanciarse usando 10 objetos o menos, puede ser una señal de inconsistencia en el modelo.

El proceso de traducción y verificación es totalmente automático [5] y se realiza de forma totalmente transparente al usuario. El feedback que se retorna al usuario es una representación gráfica del diagrama de objetos, como el que puede verse en la Figura 3 para un modelo que define "Átomos", "Moléculas" y la relación "Contiene" entre ambos. En caso que no haya podido encontrarse ningún diagrama

[1] `http:///www.eclipse-clp.org`

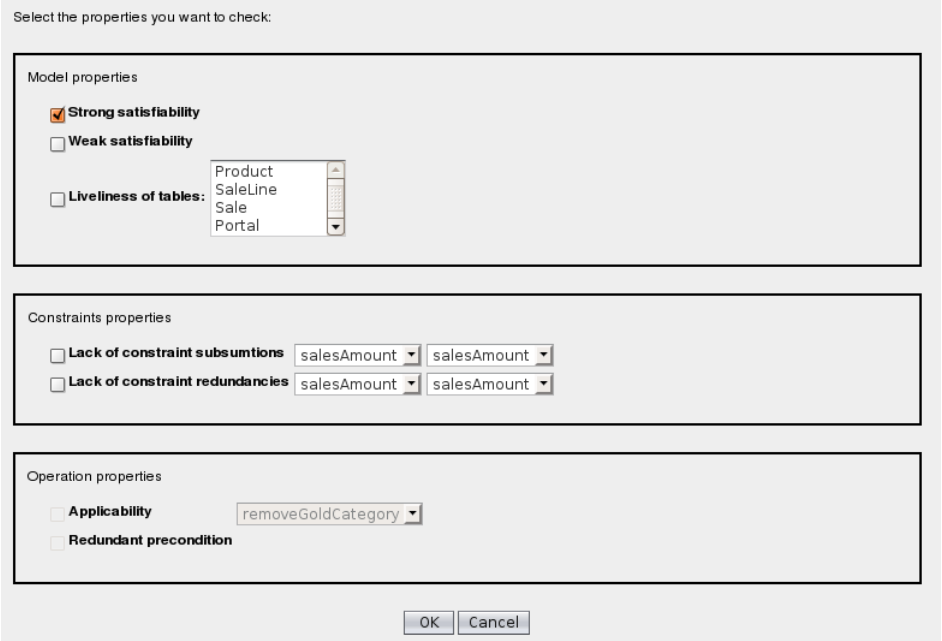

Figura 2: Selección de propiedades a verificar.

de objetos (por ejemplo, porque el modelo es incorrecto), se informa al estudiante y se le anima a enviar un modelo corregido.

Respecto a la eficiencia del proceso, en caso peor la búsqueda tiene un coste exponencial respecto al tamaño del modelo (número de clases y asociaciones). A efectos prácticos, el tiempo de respuesta depende del tipo de modelo que se proporcione como entrada. Los modelos UML sin invariantes OCL se verifican de forma prácticamente instantánea (menos de un segundo) incluso en modelos con centenares de clases. Esto permite comprobar las cardinalidades y jerarquías de herencia de modelos de gran tamaño. Respecto a los modelos con invariantes OCL, depende en gran medida del modelo y del grado de complejidad de las restricciones: hay modelos con un centenar de clases que han podido analizarse, mientras que otros con una decena de clases tienen un tiempo de verificación excesivo. Por otro lado, la verificación resulta más eficiente para los modelos satisfactibles (detenemos la búsqueda al

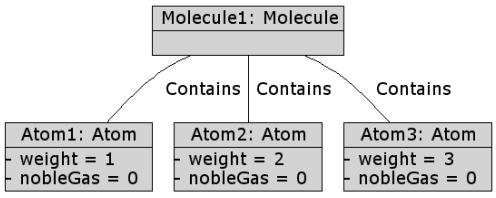

Figura 3: Diagrama de objetos mostrado como feedback para un modelo satisfactible.

hallar una solución factible) que para los insatisfactibles (debe recorrerse todo el espacio de búsqueda).

3.3. Arquitectura

La Figura 4 muestra la arquitectura general del entorno CUCKOO. El usuario realiza todas las tareas a través de su navegador Web: autenticarse, gestionar colecciones de ejercicios, administrar usuarios, seleccionar ejercicios a validar, enviar propuestas de soluciones

y mostrar el feedback de respuesta. Para ello, el entorno proporciona una interfaz Web como se ve en la Figura 5 donde se muestra la página de entrada al sistema y la Figura 6 muestra la pantalla de selección de ejercicios a verificar.

Internamente, el sistema mantiene una base de datos de los usuarios y de las colecciones de ejercicios existentes. Además, el sistema guarda un repositorio de los modelos enviados por los estudiantes como solución, que permite al profesor monitorizar el progreso de los estudiantes. El resto de componentes del entorno están relacionados con la lectura del diagrama de clases UML (NetBeans MDR) y las restricciones OCL (Dresden OCL toolkit), la traducción a un problema de satisfacción de restricciones (UMLtoCSP) y la demostración de las propiedades a verificar (ECL^iPS^e). Todos estos componentes actúan de forma transparente al usuario (ver Figura 4).

4. Trabajo relacionado

El trabajo de aquellas competencias que requieren de alta dedicación práctica y una constante realimentación en el control de la validez de los resultados obtenidos es una preocupación creciente en el mundo de la docencia de la informática. Es por esto que hace ya algún tiempo, los docentes han intentado proporcionar herramientas a los estudiantes para facilitarles dicha tarea cuando no están en presencia de un profesor.

Este tipo de herramientas pretenden, de un modo u otro, validar automática o semiautomáticamente esquemas conceptuales realizados por los estudiantes. La herramienta presentada se encuadra en este grupo. La mayoría de herramientas similares, de las que hablaremos seguidamente, están concebidas para la investigación o su comercialización profesional. Aun así, es interesante ver las propiedades y limitaciones de cada una de ellas.

Una primera diferencia que se encuentra, es el ámbito de aplicación: las hay que validan esquemas ER, otros aceptan UML, y algunos permiten incorporar a este último, restricciones OCL.

Antes de introducir las diferentes herramientas existentes, es importante tener en cuenta las dos principales problemáticas que se presentan: intentar dar respuesta a un problema no decidible, y garantizar un tiempo de respuesta razonable. A cambio, tienen alguna/s de las siguientes limitaciones: no finalización asegurada, expresividad limitada y/o semi-automatización.

Las aproximaciones del mundo docente suelen validar esquemas conceptuales sintácticamente [1]. En [12] se define junto al problema, un conjunto de esquemas que representan distintas soluciones válidas del mismo. La solución enviada por el estudiante se compara con el conjunto de soluciones válidas y en caso de no concordar se rechaza. Este sistema es claramente incompleto. La mayoría de herramientas de validación en línea siguen este paradigma.

En cambio, los métodos que estudian el análisis de modelos UML / OCL en un contexto de investigación [2, 3, 4, 10, 13, 14], utilizan una herramienta off-line que debe ser descargada por el estudiante. Respecto a su funcionamiento interno, estos métodos transforman el diagrama UML en un formalismo sobre el que se pueda resolver eficientemente o que disponga de un demostrador de teoremas automático. Esta tambíen es la base de CUCKOO. Así, HOL-OCL [4] utiliza lógica de orden superior (HOL), pero es semi-automático. UMLtoAlloy [2] traduce automáticamente el diagrama a Alloy [9], y éste trabaja a partir de la transformación del problema a SAT. Su principal limitación surge al traducir restricciones OCL de manera semi-automática. Otras herramientas [8] utilizan description logics y aceptan operaciones y razonamientos sobre éstas. Sin embargo, requieren restringir las posibles construcciones aceptadas para asegurar que el razonamiento a seguir siga siendo decidible. Una herramienta cercana, dedicada tanto al modelado como la validación de diagramas UML es MOVA [7]. MOVA permite la comprobación de la escritura y ejecución de queries en OCL, pero solamente sobre instancias del diagrama UML. Finalmente, comentar que enfoques anteriores basados en programación con restricciones [10] no admiten expre-

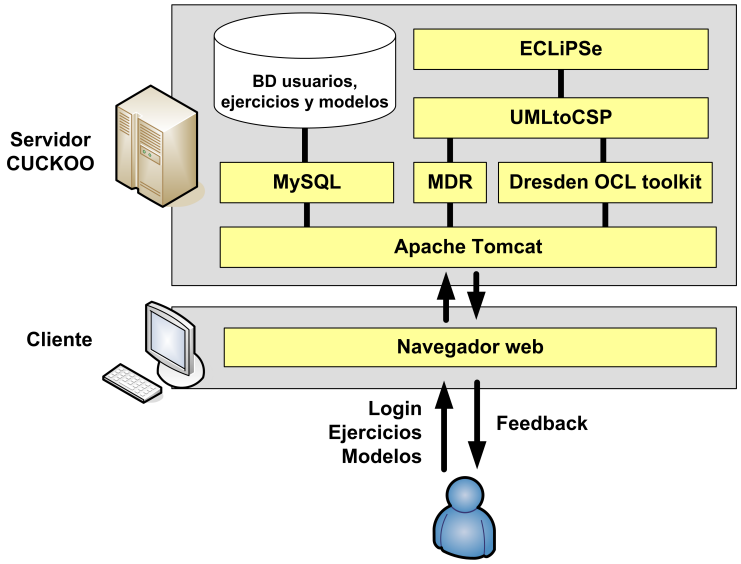

Figura 4: Arquitectura del entorno CUCKOO.

siones OCL, mientras que el método utilizado por CUCKOO [5, 6] permite validar UML con invariantes OCL y operaciones declarativas.

En resumen, una ventaja de CUCKOO es que intenta aproximar la potencia de las herramientas de análisis UML/OCL al contexto del aula, ofreciendo una interfície online que simplifica la puesta en marcha para el estudiante. Esto representa un cambio de filosofía: se pasa de "comparar un modelo con la solución oficial" a "buscar errores en los modelos".

5. Evaluación preliminar

Uno de los objetivos principales de CUCKOO es ofrecer a los estudiantes un entorno de análisis de modelos UML sin necesidad de una instalación en la propia máquina. Esto facilita el estudio y la prueba desde cualquier ordenador con conexión a Internet. A partir de aquí, y dado que CUCKOO es una herramienta actualmente en una fase bastante inicial, se pretende conocer qué recursos pueden ser más interesantes y qué carencias puede tener el sistema para el estudiante. Por esto, se ha realizado recientemente una pequeña prueba piloto con estudiantes de un postgrado en Ingeniería del Software en la UOC. La prueba piloto ha consistido en la realización de cinco ejercicios de modelado UML/OCL dentro de una actividad de evaluación continua. El resultado es que, aunque valoran la herramienta positivamente como apoyo a su aprendizaje, han puesto de manifiesto algunas carencias:

- Necesidad de una retroalimentación más completa
- Soluciones más inteligibles
- Adaptación a todos los navegadores existentes

Está claro a partir de los comentarios que una mejora importante para nuestra herramienta es la usabilidad, mejor en algunas otras propuestas existentes [12, 1, 15]. Algunos aspectos a mejorar, en los que ya estamos trabajando, son una descripción más detallada de los errores encontrados, una mayor trazabilidad de la actividad del estudiante, o la creación de unas instancias que tengan unos valores más parecidos a los reales, tratando de simular valores o nombres reales, entre otros.

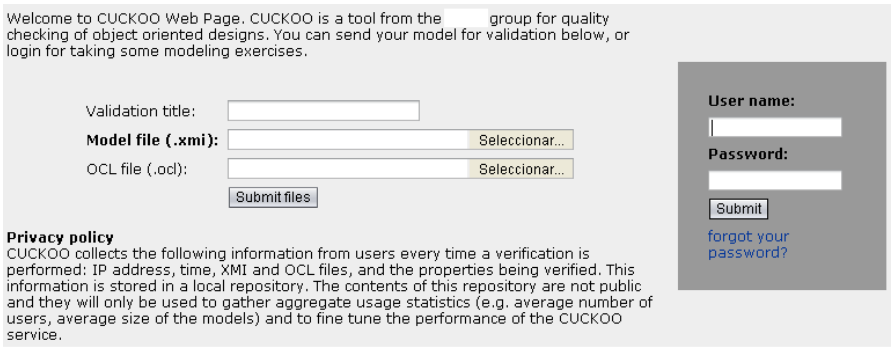

Figura 5: Entrada al entorno CUCKOO.

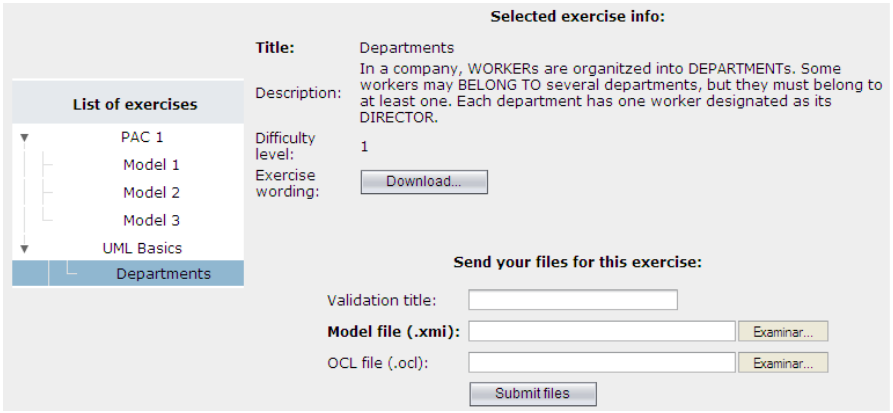

Figura 6: Presentación de un ejercicio.

6. Conclusión

En este artículo hemos presentado la herramienta CUCKOO para la verificación de modelos de software formales descritos en UML, como un primer paso para resolver una carencia específica de los cursos del área de Ingeniería del Software. Nuestra principal motivación ha sido proporcionar a los estudiantes la posibilidad de potenciar significativamente su aprendizaje a partir de autoevaluar constantemente su propio progreso, de forma automática e inmediata, sin esperar la intervención directa del profesor. Un análisis de los primeros datos empíricos obtenidos demuestra que el estudiante aprovecha en gran medida esta ventaja para practicar asiduamente con ejercicios de modelado, mucho más que sin la ayuda de CUCKOO. Creemos que este resultado confiere, sin duda, un paso importante hacia una autoevaluación efectiva y continua y con ello una mejora del progreso del aprendizaje. Otro objetivo planteado ha sido ayudar a los estudiantes a enfrentarse a la siempre temida formalización existente durante el modelado de software, mediante una aproximación altamente práctica. Finalmente, el continuo uso y familiarización con una herramienta de desarrollo de software prepara mejor al estudiante en su vertiente tecnológica y profesional.

Ésta es en definitiva una iniciativa para la consecución de los objetivos mencionados. Sin embargo, la herramienta CUCKOO necesita más rodaje y evaluación, así como resolver pro-

blemas de usabilidad detectados durante la experimentación, especialmente con ciertos navegadores, además de incorporar y mejorar ciertas funcionalidades, como el seguimiento de la actividad de los estudiantes. Aún así, los primeros resultados, aunque no concluyentes debido a su naturaleza exploratoria, son prometedores y nos animan a trabajar en la resolución de estos problemas mediante nuevas versiones de la herramienta.

Agradecimientos

Este trabajo ha sido financiado parcialmente por un Proyecto de Innovación Docente de la UOC y el proyecto TIN2008-00444 del Ministerio de Ciencia y Tecnología. Queremos agradecer a David Gañán el trabajo realizado en la implementación y a Anna Queralt y los estudiantes del curso de postgrado en Ingeniería del Software OO con UML de la UOC la colaboración en la implantación y evaluación de CUCKOO.

Referencias

[1] A. Abelló, M. E. Rodríguez, T. Urpí, X. Burgués, M. J. Casany, C. Martí, and C. Quer. LEARN-SQL: Automatic assessment of SQL based on IMS QTI specification. *IEEE Int. Conf. Advanced Learning Technologies*, 0:592–593, 2008.

[2] K. Anastasakis, B. Bordbar, G. Georg, and I. Ray. UML2Alloy: A challenging model transformation. In *MODELS'07*, volume 4735 of *LNCS*, pages 436–450, 2007.

[3] D. Berardi, D. Calvanese, and G. D. Giacomo. Reasoning on UML class diagrams. *Artificial Intelligence*, 168:70–118, 2005.

[4] A. D. Brucker and B. Wolff. The HOL-OCL book. Technical Report 525, ETH Zurich, 2006.

[5] J. Cabot, R. Clarisó, and D. Riera. Verification of UML/OCL class diagrams using constraint programming. In *Workshop on Model Driven Engineering, Verification and Validation (MoDeVVa'08)*, pages 73–80, 2008.

[6] J. Cabot, R. Clarisó, and D. Riera. Verifying UML/OCL operation contracts. In *iFM'09*, volume 5423 of *LNCS*, pages 40–45, 2009.

[7] M. Clavel, M. Egea, and V. T. da Silva. Mova: A tool for modeling, measuring and validating uml class diagrams. Technical report, Universidad Complutense de Madrid, 2007.

[8] C. Drescher and M. Thielscher. Integrating action calculi and description logics. In J. Hertzberg, M. Beetz, and R. Englert, editors, *KI*, volume 4667 of *LNCS*, pages 68–83. Springer, 2007.

[9] D. Jackson. Alloy: a lightweight object modelling notation. *ACM Transactions on Software Engineering and Methodology*, 11(2):256–290, 2002.

[10] H. Malgouyres and G. Motet. A UML model consistency verification approach based on meta-modeling formalization. In *SAC'2006*, pages 1804–1809. ACM Press, 2006.

[11] K. Marriott and P. J. Stuckey. *Programming with Constraints: An Introduction*. MIT Press, Cambridge, Massachussetts, 1998.

[12] F. Prados, I. Boada, J. Soler, and J. Poch. A web-based tool for entity-relationship modeling. In M. L. Gavrilova, O. Gervasi, V. Kumar, C. J. K. Tan, D. Taniar, A. Laganà, Y. Mun, and H. Choo, editors, *ICCSA (1)*, volume 3980 of *LNCS*, pages 364–372. Springer, 2006.

[13] A. Queralt and E. Teniente. Reasoning on UML class diagrams with OCL constraints. In *ER'06*, volume 4215 of *LNCS*, pages 497–512, 2006.

[14] R. V. D. Straeten, T. Mens, J. Simmonds, and V. Jonckers. Using description logic to maintain consistency between UML models. In *UML'03*, volume 2863 of *LNCS*, pages 326–340. Springer, 2003.

[15] P. Suraweera and A. Mitrovic. KERMIT: A constraint-based tutor for database modeling. In *Proc. 6 th Int. Conf on Intelligent Tutoring Systems ITS 2002*, pages 377–387, 2002.