Desarrollo de aplicaciones críticas en seguridad con UMLsec: una breve introducción

Jan Jürjens
Ingeniería de Software y Sistemas, Universidad Técnica de Munich (Alemania)

<juerjens@in.tum.de>

Traducción: José Luis Fernández Alemán (Departamento de Informática y Sistemas, Universidad de Murcia)

Resumen: *el desarrollo de sistemas críticos en seguridad de alta confianza es difícil y existen muchos ejemplos bien conocidos de fallos de seguridad que hacen vulnerable un sistema. Por tanto, es necesario cuanto antes obtener un método consistente que soporte el desarrollo de sistemas seguros. En este trabajo, ofrecemos una introducción sobre una extensión de UML (Unified Modeling Language) llamada UMLsec, que permite expresar información de seguridad relevante en los diagramas que especifican un sistema. Se presenta la herramienta de soporte que ha sido desarrollada para UMLsec y se explica el método a través de un ejemplo sencillo.*

Palabras clave: *evaluación de seguridad, Ingeniería del Software de Seguridad, Ingeniería de Seguridad, métodos formales en seguridad, modelos de seguridad, protocolos de Criptografía, verificación de seguridad.*

Autor

Jan Jürjens es investigador en la Universidad Tecnológica de Munich (Alemania). Es autor de un libro sobre Desarrollo de Sistemas Seguros con UML, que esta previsto que sea publicado en marzo de 2004 por Springer Verlag, y de alrededor de 30 trabajos en revistas y congresos internacionales, la mayoría sobre seguridad en computadores e Ingeniería del Software. Así mismo, en cuatro ocasiones ha sido invitado como ponente en congresos internacionales. Ha organizado e impartido un curso sobre el desarrollo de sistemas seguros en la Universidad de Oxford (Reino Unido) y cuenta con alrededor de 20 tutoriales en congresos internacionales. Es el fundador y actualmente presidente del grupo de trabajo sobre Formal Methods and Software Engineering for Safety and Security (FoMSESS) dentro de la German Society for Informatics (GI). Es miembro del comité ejecutivo de la División of Safety and Security dentro de GI, del comité ejecutivo del "QFA Modellierung" de GI, del comité consultivo del Bavarian Competence Center for Safety and Security, del grupo de trabajo sobre seguridad electrónica del gobierno regional de Baviera, y de IFIP Working Group 1.7 "Theoretical Foundations of Security Análisis and Design". Ha dirigido varios proyectos relacionados con seguridad en la industria y ha participado como revisor de proyectos de investigación de la Unión Europea. Para obtener más información se puede consultar su página web <http://www4.in.tum.de/~juerjens>.

1. Motivación

El desarrollo de sistemas críticos de alta calidad (ya sea en fiabilidad, seguridad, tiempo real, eficiencia, o sistemas híbridos) es difícil. Muchos sistemas críticos se desarrollan, se despliegan y se usan sin satisfacer sus requisitos críticos, en ocasiones con errores espectaculares.

Parte de la dificultad del desarrollo de sistemas críticos es que, con frecuencia, la corrección entra en conflicto con el coste. También, se tiende a evitar las partes de los métodos de diseño de sistemas que plantean un elevado coste en la formación de personal y en su uso.

El Lenguaje Unificado de Modelado (*Unified Modeling Language*, UML [11]) ofrece una oportunidad sin precedentes para el desarrollo de sistemas críticos de alta calidad en un contexto industrial.

- Puesto que UML se ha convertido en el estándar de-facto de modelado en la industria, un gran número de desarrolladores están formados en UML.
- En comparación con notaciones anteriores con una comunidad de usuarios de tamaño equiparable, UML está definido de forma bastante precisa.
- Se han desarrollado varias herramientas de análisis, prueba, simulación, transformación, etc., para apoyar en el trabajo diario con UML.
- Sin embargo, existen algunos desafíos que aún es preciso superar para aprovechar esta oportunidad, entre los que se incluyen los siguientes:
- Adaptación de UML a dominios de aplicación de sistemas críticos.
- El uso correcto de UML en los dominios de aplicación.
- Conflictos entre flexibilidad y precisión en el significado de la notación.
- Mejorar el soporte de herramientas para el desarrollo de sistemas críticos con UML.

Este trabajo tiene como objetivo presentar una introducción al uso de UML para el desarrollo de sistemas críticos y contribuir a superar estos desafíos. Mostramos cómo se pueden usar estereotipos, etiquetas, restricciones para encapsular conocimiento sobre el diseño de sistemas críticos y lo ponemos a disposición de desarrolladores sin suponer una experiencia en este tipo de sistemas. Asimismo, esbozamos cómo se puede comprobar si, en una especificación dada, se cumplen las restricciones asociadas a los estereotipos, y en caso de que se requiera, mediante un análisis mecánico. De esta forma, uno puede encontrar fallos en el diseño, antes de desplegar o incluso implementar el sistema (la corrección de errores en los requisitos durante las últimas fases de desarrollo puede ser hasta 200 veces más costosa que en las primeras fases [1]).

Los problemas en el desarrollo de sistemas críticos aparecen, a menudo, cuando la independencia conceptual del software de la capa física subyacente no resulta ser una abstracción fiel (por ejemplo, en contextos tales como sistemas en tiempo real o, generalmente, sistemas críticos de seguridad frente a terceros, ver [12]). Puesto que UML permite al modelador describir diferentes vistas sobre un sistema, incluyendo la capa física, parece prometedor intentar usar UML para resolver estos problemas mediante el modelado de interdependencias entre el sistema y su entorno físico.

El enfoque también puede utilizarse para analizar puntos débiles del código empleando pruebas de aspectos críticos basadas en modelos. En este caso, las secuencias de pruebas se generan a partir de una especificación abstracta del sistema para proporcionar confianza en la corrección de una implementación. En sistemas críticos es especialmente difícil encontrar pruebas para detectar posibles errores o vulnerabilidades, puesto que éstas normalmente implican escenarios de ejecución ingeniosos y complejos (y algunas veces la consideración de conceptos específicos del dominio tales como criptografía y números aleatorios).

2. Presentación mediante un caso de estudio de pago de peajes

UML ofrece tres mecanismos de extensión 'ligeros': estereotipos, valores etiquetados y restricciones. A través de estereotipos, se pueden definir nuevos tipos de elementos de modelado, los cuales extienden la semántica de los tipos existentes en el metamodelo de UML. Se representan empleando dobles paréntesis angulares. Un valor etiquetado es un par valor-nombre entre llaves, asociando datos con elementos en el modelo. Además, también se pueden incluir restricciones en los modelos.

En esta sección, para mostrar nuestras ideas, emplearemos un caso de estudio simple y

Los casos de uso también pueden emplearse para capturar requisitos de seguridad

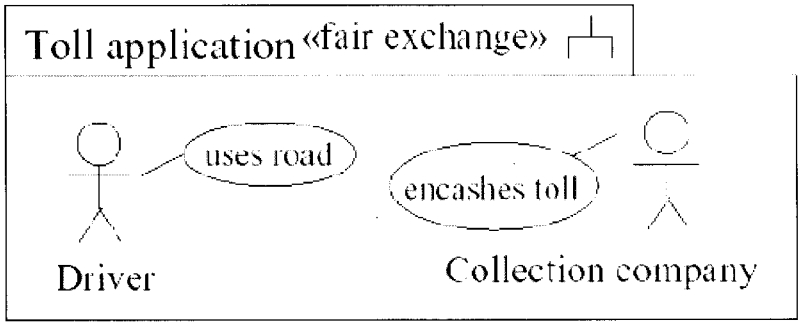

Figura 1. Diagrama de casos de uso para una aplicación de cobro de peaje (*Toll Application*).

ficticio que describe un sistema automático de cobro de peajes en una red de carreteras. Aquí, nos centraremos en los requisitos críticos de seguridad. En [7] el lector puede encontrar ejemplos para requisitos críticos de seguridad frente a terceros.

2.1. Identificación de requisitos de seguridad con Diagramas de Casos de Uso

Durante la obtención de requisitos, es frecuente emplear casos de uso para describir las interacciones típicas entre un usuario y el sistema. Los casos de uso también pueden emplearse para capturar requisitos de seguridad. Comenzaremos con nuestro ejemplo de un sistema de cobro de peajes: un conductor tiene que pagar un peaje a una empresa para acceder a una red de carreteras. Con este objetivo, cuando el conductor entra en la carretera, se registra en un puesto de peaje. Después, durante su viaje, cruza por varios puestos de peaje. Cuando el conductor abandona la carretera efectúa un abono en función del tramo de carreteras por el que circuló. El pago debería realizarse de manera que impida que cualquiera de las dos partes hagan trampa. Para el diagrama de casos de uso de la figura 1, este requisito se incluye mediante el estereotipo `<<fair exchange>>` que se encuentra asociado al subsistema[1] que contiene este diagrama.

2.2. Proceso de negocio seguro con Diagramas de Actividades

Los diagramas de actividades pueden emplearse para modelar *workflow* y explicar casos de uso con mayor detalle. Estos diagramas son también muy útiles para conseguir requisitos de seguridad más precisos.

Para continuar con nuestro ejemplo, la figura 2 explica con más detalle el caso de uso empleando una diagrama de actividades. Se incluye el requisito `<<fair exchange>>` mencionado antes. Las acciones listadas en las etiquetas `{start}` y `{stop}` deberían estar unidas en el sentido de que si se ejecuta una de las primeras acciones entonces también lo hace una de la últimas (de hecho, esta propiedad puede comprobarse mecánicamente).

Esto significa que, una vez que el conductor ha pagado su peaje, o bien consigue un recibo por su desembolso, o es capaz de rechazar el pago en ese momento.

2.3. Seguridad física empleando diagramas de despliegue

En la notación de UML, los diagramas de despliegue se utilizan para describir la capa física de un sistema. Nosotros los utilizamos para comprobar si los requisitos de seguridad establecidos sobre el nivel lógico del sistema son forzados por el nivel de seguridad física, o si deben emplearse mecanismos de seguridad adicionales (tales como la encriptación).

Volviendo a nuestro ejemplo, cuando el conductor llega a un puesto de peaje, se establece una conexión inalámbrica. La transacción del pago conlleva que la transmisión de los datos debe mantenerse secreta. Los datos privados no deberían ser visibles a un tercero. En el modelo UML de la figura 3 se refleja esta información sobre la capa física y los requisitos de seguridad. Nosotros, por tanto, empleamos el estereotipo `<<secure links>>` para expresar la restricción de que la capa física satisface los requisitos de seguridad en la comunicación. Para ser más exactos, por cada dependencia *d* estereotipada `<<secrecy>>`, entre subsistemas o clases sobre diferentes nodos *n*, *m*, y cualquier enlace de comunicación *l* entre *n* y *m* con algún estereotipo *s*, el escenario que surge del estereotipo *s* con respecto a un

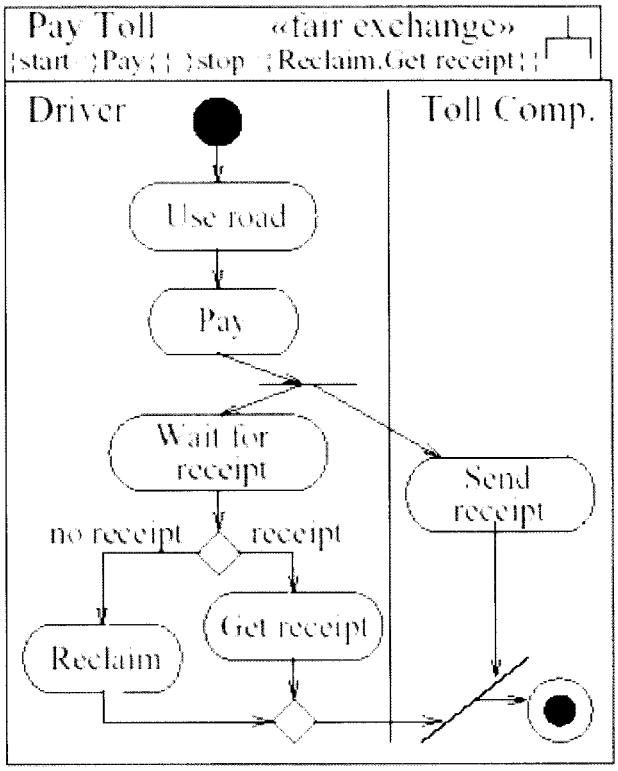

Figura 2. Diagrama de actividades para el pago del peaje.

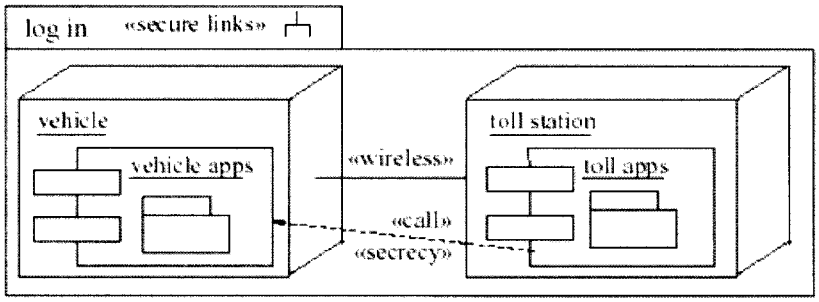

Figura 3. Nivel físico de la aplicación de cobro de peaje.

tercero que suponga una determinada amenaza para el sistema, no viola los requisitos de seguridad sobre los datos comunicados. En el diagrama mostrado, la restricción asociada al estereotipo `<<secure links>>` se viola si se consideran a los atacantes habituales, ya que las conexiones inalámbricas pueden capturarse fácilmente y, por tanto, los datos comunicados no se mantienen en secreto.

Por consiguiente, para este tipo de atacantes, el estereotipo se aplica *fuertemente* al subsistema. Esto también puede comprobarse automáticamente empleando herramientas de soporte apropiadas (se remite al lector a la discusión que aparece más adelante).

2.4. Interacción con aspectos de seguridad críticos mediante diagramas de secuencias

Los diagramas de secuencias se utilizan para especificar la interacción entre diferentes partes de un sistema. Se pueden extender vinculando requisitos de seguridad a la interacción.

Con respecto a nuestro ejemplo, especificamos la parte más importante de un protocolo de cobro de peaje empleando un diagrama de secuencias (la figura 4; observar que este protocolo sólo se utiliza aquí para demostrar nuestro método, ni mucho menos se insinúa que necesariamente tenga que ser un protocolo útil). En primer lugar, el conductor envía una solicitud para pagar en un nuevo puesto de peaje, junto con los datos recibidos del anterior puesto por el que cruzó: su nombre D, el nombre `TSNo'` del puesto actual, el nombre `TSNo` del último puesto, un contador con el número de puestos que ha cruzado desde que entró en la carretera, y el momento en el que pasó. Los últimos tres datos habían sido *firmados* en el puesto anterior con la clave privada de la empresa. Todos los datos son encriptados con la clave pública del puesto de peaje. El puesto de peaje envía la cantidad de dinero que el conductor ha pagado y sus propios datos tal y como se describió antes, encriptados con la clave pública del conductor. El conductor confirma la validez de los datos firmando y enviando estos al puesto de peaje, de nuevo encriptados empleando la clave pública de la empresa. Aquí, K_D y K_D^{-1} denotan la clave pública y la clave privada, respectivamente, del conductor, mientras que K_T y K_T^{-1} representan las correspondientes clave pública y privada de la empresa.

Para incluir requisitos de seguridad sobre los datos involucrados, de nuevo podemos acudir a los estereotipos. En este caso, el estereotipo `<<critical>>` etiqueta clases que contienen datos confidenciales y tiene asociado las etiquetas `{secrecy}` e `{integrity}` para denotar los respectivos requisitos de seguridad sobre los datos. Para exigir que estos requisitos se cumplan respecto al modelo de un determinado ata-

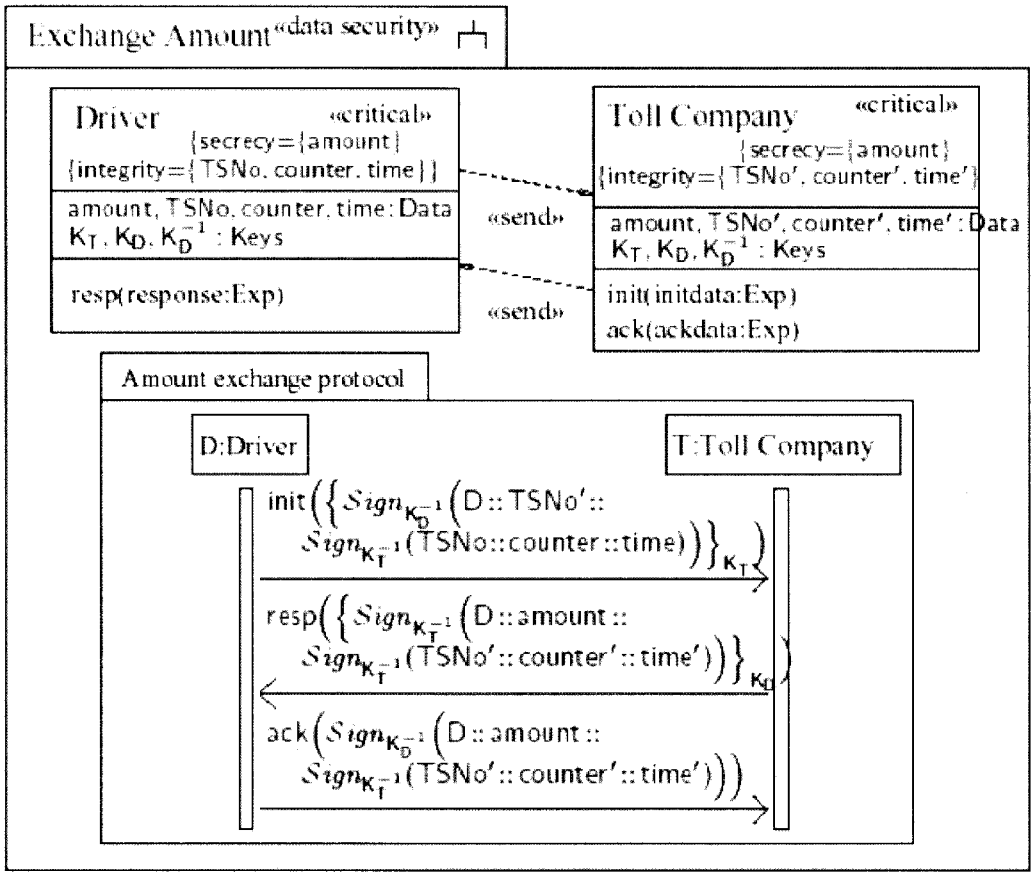

Figura 4. Protocolo para el pago.

> UML ofrece una oportunidad sin precedentes para al desarrollo de sistemas críticos de alta calidad en un contexto industrial

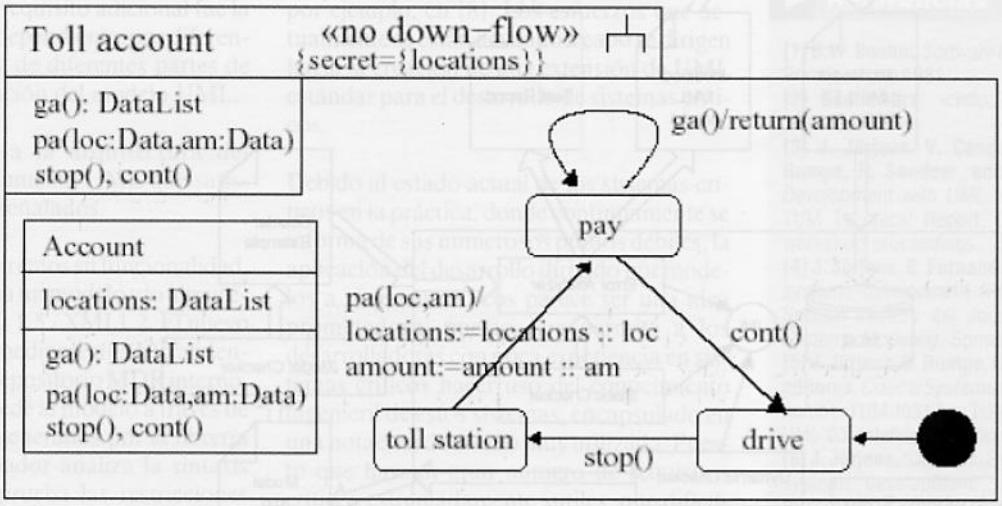

Figura. 5. Diagrama de estados para el objeto `Account`.

cante, la restricción `<<data security>>` se asocia con el modelo. Asumimos que el atacante estándar no es capaz de descifrar la encriptación, pero sí que es capaz de explotar cualquier error en el protocolo empleando el ataque *"man-in-the-middle"*. Técnicamente, la restricción impone que no hay ataques de ese tipo. En este punto, diremos que no es trivial ver si la restricción se cumple para un protocolo dado. No obstante, es posible verificar esto automáticamente, empleando conceptos bien fundados a partir de métodos formales aplicados a la seguridad de los computadores.

2.5. Estados seguros empleando diagramas de estado

Los diagramas de estado muestran los cambios en el estado de un objeto. Pueden emplearse para especificar requisitos de seguridad sobre las secuencias de estados resultantes, así como sobre la interacción con el entorno del objeto.

Continuando con el ejemplo, el conductor circula por la carretera, y cada vez que pasa por un puesto de peaje, algunos datos se añaden en dos listas. Primero, la lista *locations* se rellena con los puestos por los que ha pasado y, segundo, la lista *amount* se rellena con el correspondiente peaje. Además, asumimos que las listas se encuentran ordenadas secuencialmente, de tal forma que el i-ésimo elemento de la lista *locations* se corresponde con la i-ésima cantidad de la lista *amount*. La primera entrada de las listas es el primer puesto de peaje por el que ha pasado, la segunda entrada el segundo puesto, y así sucesivamente.

En la figura 5, podemos observar que la empresa sólo tiene acceso a esta lista de cantidades a través de la operación `ga()`. Por razones de privacidad, se prohíbe la devolución de la lista de localizaciones a la empresa. Puesto que, normalmente, las distancias entre puestos de peaje son diferentes, uno puede inferir, a partir de las cantidades pagadas por peaje incluidas en la lista, los tramos de carretera por los que ha pasado. Por tanto, las diferentes localizaciones se filtran implícitamente.

En el diagrama de estado empleamos el estereotipo `<<no down-flow>>` para indicar que el objeto no debería difundir ninguna información acerca de los datos secretos (en este caso las localizaciones). Desafortunadamente, la especificación dada no satisface el requisito (como se puede comprobar formalmente de forma precisa). Por tanto, el modelo presenta el estereotipo de forma no legítima. De nuevo, esto puede comprobarse automáticamente.

2.6. Herramientas de soporte

Para facilitar la aplicación de nuestro enfoque en la industria es necesario disponer de herramientas que automaticen el análisis de modelos UML empleando la semántica sugerida.

A continuación, describimos un *framework* que incorpora varios verificadores desarrollados en la Universidad Tecnológica de Munich (Alemania).

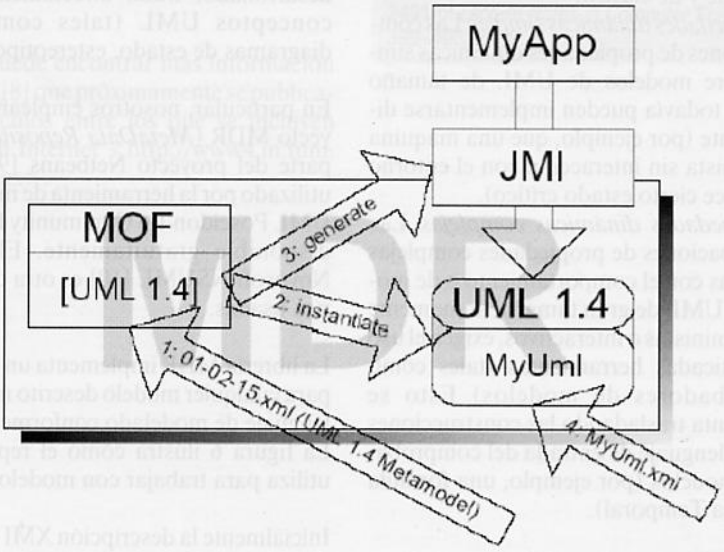

Figura 6. Utilización de la librería MDR.

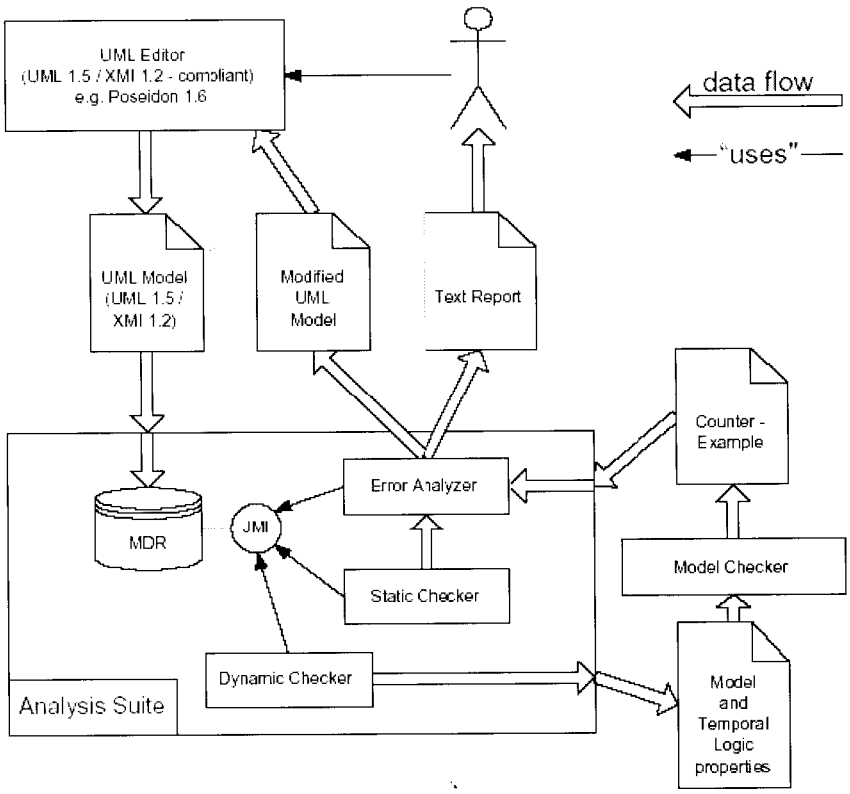

Figura 7. Conjunto de herramientas UML.

Funcionalidad: hay tres etapas consecutivas al implementar la funcionalidad de verificación completa para los modelos UML formalizados. Actualmente, existe una herramienta de verificación para las dos primeras etapas, el desarrollo de la tercera etapa se encuentra en marcha.

- *Propiedades estáticas.* Se pueden implementar directamente comprobadores de propiedades estáticas (por ejemplo, una comprobación de tipos como aplicación de los niveles de seguridad en los diagramas de despliegue y de clases).
- *Propiedades dinámicas simples.* Las comprobaciones de propiedades dinámicas simples sobre modelos de UML de tamaño limitado todavía pueden implementarse directamente (por ejemplo, que una máquina determinista sin interacción con el entorno no alcance cierto estado crítico).
- *Propiedades dinámicas complejas.* Las comprobaciones de propiedades complejas relacionas con el comportamiento o de modelos de UML de gran tamaño, sumamente no deterministas o interactivos, exigen el uso de sofisticadas herramientas (tales como comprobadores de modelos) Esto se implementa trasladando las construcciones UML al lenguaje de entrada del comprobador de modelos (por ejemplo, una fórmula en Lógica Temporal).

Acceso a los modelos UML: en el plano técnico, la cuestión fundamental es cómo adquirir y procesar modelos UML. La mayoría de las herramientas de edición de UML existentes pueden almacenar el modelo en formato XMI 1.2 (*XML Metadata Interchange*). Sin embargo, el procesamiento de un modelo UML directamente en el nivel XMI ofrece al desarrollador una representación muy abstracta del modelo. Por tanto, se han desarrollado librerías que proporcionan la representación de un archivo XMI de UML en el nivel de abstracción de un modelo UML, y por tanto permiten al desarrollador tratar directamente con los conceptos UML (tales como clases, diagramas de estado, estereotipos, etc.).

En particular, nosotros empleamos el proyecto MDR (*MetaData Repository*) que es parte del proyecto Netbeans [9], también utilizado por la herramienta de modelado de UML Poseidon 1.6 Community Edition [2], disponible gratuitamente. El proyecto Novosoft NSUML [10] es otra de las librerías usadas.

La librería MDR implementa un repositorio para cualquier modelo descrito mediante un lenguaje de modelado conforme con MOF. La figura 6 ilustra cómo el repositorio se utiliza para trabajar con modelos UML.

Inicialmente la descripción XMI del lenguaje de modelado se emplea para adaptar MDR para que trabaje con un tipo de modelo particular, UML en éste caso (paso 1 en la figura 6).

La definición XMI de UML 1.4 fue publicada por el OMG (*Object Management Group*). El siguiente paso es crear un almacenamiento adaptado al tipo de modelo dado (paso 2). Además, basándose en la especificación XMI del lenguaje de modelado, la librería MDR crea la implementación JMI (*Java Metadata Interface*) para acceder al modelo (paso 3). Esto permite a la aplicación manipular directamente el modelo en el nivel conceptual de UML. El modelo UML se carga en el repositorio (paso 4). Ahora, se puede acceder desde una aplicación Java a través de los interfaces JMI proporcionados. El modelo se puede leer, modificar y, más tarde, grabar de nuevo en un archivo XMI.

Debido al nivel de abstracción adicional implementado por la librería MDR, emplearlo en el marco de UML debería facilitar la actualización a nuevas versiones de UML, y favorece la mayor compatibilidad posible con el estándar.

Arquitectura: por su diseño, nuestro *framework* UML proporciona un entorno de programación para los desarrolladores de diferentes módulos de verificación (*tools*). Por tanto, un desarrollador de un *tool* se concentra en la lógica de verificación y no en

Es necesario disponer de herramientas que automaticen el análisis de modelos UML

las tareas secundarias como el manejo de la entrada/salida. Un requisito adicional fue la implementación independiente, por diferentes desarrolladores, de diferentes partes de la lógica de verificación del modelo UML.

La figura 7 ilustra la arquitectura del *framework* de herramienta UML que satisface los requisitos señalados.

Brevemente describiremos su funcionalidad. El desarrollador crea un modelo y lo almacena en formato UML 1.5 / XMI 1.2. El nuevo archivo se importa mediante una herramienta y se guarda en el repositorio MDR interno. La herramienta accede al modelo a través de los interfaces JMI generados por la librería MDR. El comprobador analiza la sintaxis del modelo y comprueba las restricciones asociadas con los estereotipos. Los resultados describiendo los problemas encontrados se entregan al desarrollador en un informe textual. Además, se genera un modelo UML modificado donde se resaltan los estereotipos cuyas restricciones no se han cumplido.

3. Experiencia y perspectivas de futuro

El método propuesto aquí se ha aplicado satisfactoriamente en proyectos de sistemas críticos, por ejemplo, en la evaluación del *Common Electronic Purse Specifications* bajo el desarrollo de Visa International y otros, en el proyecto *FairPay* financiado por el Ministerio Alemán de Economía; en proyectos con un gran banco alemán; y en el proyecto *Verisoft* financiado por el Ministerio Alemán de Ciencia y Tecnología.

En particular, estas experiencias indican que el enfoque es adecuado para aplicarlo en la práctica. Se pueden encontrar enfoques alternativos, por ejemplo, en [3][4][5]. En este artículo debemos omitir información sobre trabajos relacionados debido a las restricciones de espacio, pero se puede encontrar, por ejemplo, en [8]. Los esfuerzos que actualmente se están llevando a cabo se dirigen hacia la creación de una extensión de UML estándar para el desarrollo de sistemas críticos.

Debido al estado actual de los sistemas críticos en la práctica, donde continuamente se informa de sus numerosos puntos débiles, la aplicación del desarrollo dirigido por modelos a sistemas críticos parece ser una idea prometedora, puesto que permite a los desarrolladores con poca experiencia en sistemas críticos hacer uso del conocimiento ingenieril de estos sistemas, encapsulado en una notación de diseño muy utilizada. Puesto que hay un gran número de requisitos críticos extremadamente sutiles, que difícilmente pueden verificarse a simple vista, incluso expertos en sistemas críticos podrían beneficiarse de este enfoque. Aunque el enfoque explicado aquí se centra en los errores que se presentan en el nivel de diseño, podría extenderse al análisis de código empleando pruebas basadas en modelos de aspectos críticos.

Para que estas ideas sean útiles en la práctica, es importante tener herramientas de soporte inteligentes que ayuden a utilizarlas. Como se indicó previamente, las herramientas se encuentran actualmente en desarrollo y uno puede utilizarlas para comprobar mecánicamente las restricciones ilustradas antes. Las herramientas soportan el enfoque ahorrando tiempo y evitando errores cuando se analiza un modelo con el objeto de identificar fallos en el diseño del sistema crítico.

El lector puede encontrar más información en un libro [8] que próximamente se publicará, en artículos entre los que se incluyen [6][7], y en Internet <http://www4.in.tum.de/~juerjens>.

Referencias

[1] B.W. Boehm. *Software Engineering Economics.* Prentice-Hall, 1981.

[2] Gentleware: <http://www.gentleware.com, 2003>.

[3] J. Jürjens, V. Cengarle, E. Fernandez, B. Rumpe, R. Sandner, editores. *Critical Systems Development with UML*, número TUM-I0208 en TUM Technical Report, 2002. UML'02 satellite workshop proceedings.

[4] J. Jürjens, E. Fernandez, R. Sandner. "Critical systems development with UMLlike languages". Special section del *Journal on Software and Systems Modeling*, Springer-Verlag, 2004.

[5] J. Jürjens, B. Rumpe, R. France, E. Fernandez, editores. *Critical Systems Development with UML*, number TUM-I0317 en TUM technical report, 2003. UML'03 satellite workshop proceedings.

[6] J. Jürjens. "UMLsec: Extending UML for secure systems development". En J.-M. Jézéquel, H. Hussmann, y S. Cook, editores, UML 2002 – The Unified Modeling Language, Vol. 2460 de *Lecture Notes in Computer Science*, pp. 412–425, Dresden, Sept. 30 – Oct. 4, 2002. Springer-Verlag.

[7] J. Jürjens. "Developing safety-critical systems with UML". En P. Stevens, editor, UML 2003 – *The Unified Modeling Language*, Vol. 2863 of *Lecture Notes in ComputerScience*, pp. 360–372, San Francisco, Oct. 20–24, 2003. Springer-Verlag. 6th International Conference.

[8] J. Jürjens. *Secure Systems Development with UML*. Springer-Verlag, March 2004. Pendiente de publicación.

[9] Netbeans project. *Open Source*. <http://mdr.netbeans.org>, 2003.

[10] Novosoft NSUML Project: <http://nsuml.sourceforge.net/>, 2003.

[11] Object Management Group. *OMG Unified Modeling Language Specification v1.5: Revisions and recommendations*, March 2003. Version 1.5. OMG Document formal/03-03-01.

[12] B. Selic. "Physical programming: Beyond mere logic". En A. Sangiovanni-Vincentelli, J. Sifakis, editores, *Embedded Software Second International Conference (EMSOFT 2002)*, Vol. 2491 of *Lecture Notes in Computer Science*, pp. 399–406, 2002.