

Propuesta para pruebas dirigidas por modelos usando el perfil de pruebas de UML 2.0

Beatriz Pérez Lamancha¹, Pedro Reales Mateo², Ignacio García-Rodríguez de Guzmán², Macario Polo Usaola²

¹ Centro de Ensayos de Software (CES), Instituto de Computación,
Universidad de la República de Uruguay
Julio Herrera y Reissig 565, 11300, Montevideo, Uruguay
bperez@fing.edu.uy

² Grupo de Investigación ALARCOS, Departamento de Tecnologías y Sistemas de
Información, Universidad de Castilla-La Mancha
Paseo de la Universidad/4, 13071, Ciudad Real, España
{pedro.reales, ignacio.grodriguez, macario.polo}@uclm.es

Resumen. En la actualidad, UML se presenta como uno de los metalenguajes más utilizados para la especificación de sistemas mediante distintos tipos de modelos. Junto con UML 2, OMG ha propuesto un perfil para la realización de pruebas para que éstas puedan ser especificadas no como un artefacto final, sino como modelos que van anotando los modelos UML 2 generados en el desarrollo. En este artículo se presenta una propuesta para pruebas en el contexto de la Ingeniería dirigida por modelos. A partir de los modelos de diseño del sistema en UML, se propone realizar transformaciones a modelos de prueba basados en el perfil de pruebas de UML. Para que la generación de los casos de prueba sea automática, se define una extensión del metamodelo de UML de forma que se puedan anotar los diagramas de secuencia con información que, luego, pueda ser utilizada para generar el oráculo de pruebas. Esta información es anotada en OCL como pre y postcondiciones en el diagrama.

Palabras Clave: Ingeniería dirigida por modelos, Pruebas, Pruebas dirigidas por modelos, oráculo de pruebas.

1 Introducción

Entre otros elementos, la ingeniería dirigida por modelos (Model-Driven Engineering o MDE) combina lenguajes de modelado específicos del dominio y motores de transformación y generadores, que analizan ciertos aspectos de los modelos y que sintetizan varios tipos de artefactos, como código fuente, simulación de entradas u otras alternativas de modelado [1].

La transformación de modelos es el proceso de convertir un modelo en otro para el mismo sistema, usándose un lenguaje para describir dicha transformación [2]. Una transformación establece un conjunto de reglas que describen cómo un modelo expresado en un lenguaje origen puede ser transformado en un modelo en un lenguaje destino [3]. Para realizar las transformaciones, OMG (*Object Management Group*) ha definido un estándar para la definición de transformaciones llamado QVT (Query/View/Transformation) [4].

El estándar de OMG de Arquitectura dirigida por modelos (Model-Driven Architecture o MDA) es el ejemplo más conocido de MDE. MDA especifica tres vistas del sistema: 1) Vista independiente de la computación (CIM), representa los requisitos del sistema; 2) Vista independiente de la plataforma (PIM), representa la operación del sistema sin tener en cuenta detalles de una plataforma particular; y 3) Vista específica de la plataforma (PSM), que combina la vista independiente de la plataforma con los detalles de su uso en un plataforma específica [2].

Dado que MDA se basa en modelos, es posible utilizar el perfil de pruebas para UML (UML Testing Profile, UMLTP), también de OMG, que define un lenguaje para diseñar, visualizar, especificar, analizar, construir y documentar los artefactos de un sistema de pruebas. El perfil de pruebas extiende UML con conceptos específicos de pruebas (componentes de prueba, veredictos, etc.). El perfil de pruebas de UML se basa en el metamodelo de UML y reutiliza su sintaxis definiendo conceptos para observar el comportamiento de las pruebas y las actividades durante las pruebas, arquitectura de las pruebas, datos de pruebas y tiempo [5].

En este artículo se presenta una propuesta para la generación automática de casos de prueba en el contexto de MDA, basada en el metamodelo de UML 2 y el perfil de pruebas de UML, realizando transformaciones desde los modelos UML al modelo de pruebas, utilizando como modelo de descripción de comportamiento del sistema el diagrama de secuencia de UML. Dentro de la propuesta se aborda la generación automática de los oráculos de las pruebas ya que éstos son dependientes del dominio de la aplicación. Este hecho hace necesario anotar los modelos con información adicional durante el diseño para poder generar los oráculos. Dado que la propuesta de este artículo está basada en MDA, se presenta una extensión al metamodelo del diagrama de secuencia de UML para representar la información dependiente del dominio como pre y postcondiciones anotadas usando OCL que servirá posteriormente para generar los oráculos de las pruebas. Los diagramas de secuencia extendidos con pre y postcondiciones son transformados posteriormente en modelos de prueba que son instancias del UMLTP.

La sección 2 presenta los principales trabajos relacionados en los que se basa esta investigación, en la sección 3 se describe la propuesta para pruebas dirigidas por modelos usando el perfil de pruebas de UML, en la sección 4 se muestra un ejemplo práctico de dicha propuesta y por último en la sección 5 se presentan las conclusiones y trabajo a futuro.

2 Antecedentes

En esta sección se describen aquellos trabajos que componen la base de nuestra investigación, la cual se centra en la definición automática de un oráculo para las pruebas, derivando pruebas en el contexto de la ingeniería dirigida por modelos y generando un modelo de pruebas basado en el perfil de pruebas de UML.

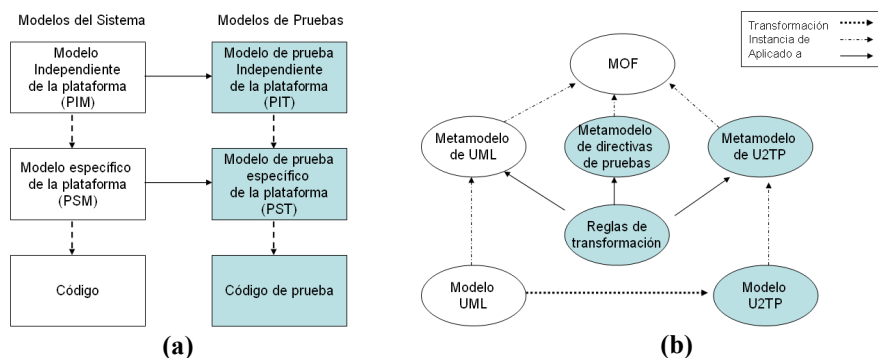


Figura 1. Transformación de modelos de pruebas[7]

El oráculo es el mecanismo de que se dota a cada caso de prueba para determinar, tras su ejecución, si el sistema que se está probando supera o no el caso. Una de las principales dificultades con las que se encuentra la investigación en el área del testing, y que, de acuerdo con Bertolino [6], representa un gran obstáculo para avanzar en su automatización, es la descripción del oráculo. Bertolino hace referencia al oráculo ideal, que describe como “un método mágico [sic] que proporciona las salidas para cada caso de prueba, aunque en la práctica se implemente como un motor o heurística que emite un veredicto de paso o fallo sobre las salidas observadas”.

Puede aplicarse el perfil de pruebas de UML a un modelo de diseño en UML para derivar un modelo de pruebas. La idea de MDA puede aplicarse al modelado de las pruebas, los modelos de pruebas pueden ser independientes de la plataforma o específicos de la plataforma en forma previa a generar el código de pruebas ejecutable [7]. En la Figura 1(a) se muestra el enfoque propuesto por Dai siguiendo el enfoque MDA para los modelos de pruebas. En la Figura 1(b) se muestran las transformaciones para las pruebas basadas en los metamodelos. El metamodelo fuente es el de UML y el metamodelo destino es el metamodelo de UML Testing Profile (U2TP). La transformación entre modelos es especificada por reglas de acuerdo a la especificación QVT [7]. Esta propuesta es teórica y no ha sido desarrollada.

3 Propuesta para pruebas dirigidas por modelos

La definición del proceso de derivación de pruebas dirigido por modelos propuesto en este trabajo sigue la propuesta de Dai [7] resumida en la sección 2 de este trabajo. En las pruebas dirigidas por modelos resulta imprescindible poder generar casos de prueba completos en forma automática. Esto incluye representar la información dependiente del dominio para las pruebas en los modelos UML y poder transformarla luego en el oráculo de pruebas en los distintos modelos de prueba.

Nuestra propuesta añade una pequeña extensión al metamodelo de UML para agregar la información requerida para el oráculo de las pruebas en los diagramas de secuencia: puesto que un diagrama de secuencia se corresponde con un escenario relevante que debe ser probado, el diagrama se anota de forma que incluya información sobre el resultado esperado y el estado inicial del mismo.

En la Figura 2 se muestra la extensión del metamodelo de UML definida, donde en la clase *InteractionFragment* se le agrega una *Constraint* de OCL, y dos relaciones, una que representa la precondición del diagrama de secuencia y otra que representa sus postcondiciones. Se agrega también una restricción que indica que dichas pre y postcondiciones no se aplican especializaciones *InteractionFragment* de tipo *StateInvariant*.

Un *InteractionFragment* es un fragmento de interacción, por lo que la semántica de nuestra extensión es que cualquier segmento de una interacción puede tener una pre y postcondición a ser utilizada como oráculo para las pruebas. Esto nos permite poder derivar casos de prueba a distintos niveles (dado que los segmentos se anidan): unitario, de integración ó funcionales.

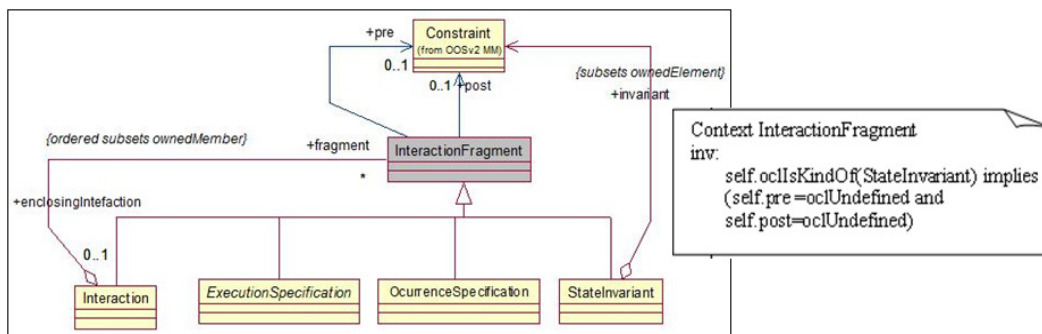


Figura 2. Extensión al metamodelo de UML para representar pre y postcondiciones para las pruebas

En la Figura 3 (a) se pueden observar los distintos puntos donde se pueden poner pre y postcondiciones. Por ejemplo, la pareja pre1 y pos 1, tienen la información necesaria para construir el oráculo para todo el diagrama, lo que se conoce como una prueba funcional. La pareja pre2 y pos2 dan la información necesaria para construir el oráculo para probar la interacción entre B y C, mientras que la pareja pre3 y pos3 nos permiten realizar pruebas unitarias de C. La Figura 3 (b) muestra cómo empleando transformaciones QVT sobre el diagrama de secuencia extendido se pueden generar modelos de pruebas unitarias, de integración y funcionales. Siguiendo un enfoque MDA, primero se

obtiene el modelo de pruebas en el nivel deseado y luego dicho modelo es refinado según la plataforma, en caso de que el lenguaje sea Java, podríamos derivar casos de prueba en JUnit. Las transformaciones serán realizadas utilizando QVT [8, 9].

A partir del diagrama de clases (que representa la arquitectura del sistema) y de los diagramas de secuencia extendidos con pre y postcondiciones (que representan el comportamiento), se realizan las transformaciones para llegar al modelo de pruebas. Primero se construye la arquitectura del modelo de pruebas a partir de la arquitectura del sistema y de los modelos que representan el comportamiento (en nuestro caso diagramas de secuencia). Una vez construida la arquitectura de sistema de pruebas se genera el comportamiento asociado a cada uno de los casos de prueba mediante un diagrama de secuencia.

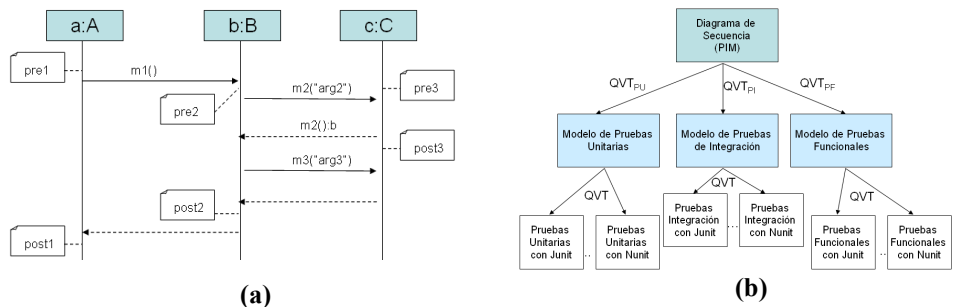


Figura 3. Información para el oráculo según el nivel de prueba

4 Ejemplo de aplicación de pruebas dirigidas por modelos

En esta sección se presenta un ejemplo de la propuesta definida en la sección 3 de este artículo, centrándonos en el nivel de pruebas funcionales. Para este ejemplo se usará una parte de un sistema en la que se modela el escenario de ejecución principal de la funcionalidad de autenticación de un usuario en el sistema. Aunque un entorno real aparecerían múltiples funcionalidades con múltiples escenarios, para este ejemplo nos centraremos en un escenario concreto. La Figura 4 (a) muestra el diagrama de clases con la arquitectura que implementa esta funcionalidad y la Figura 4 (b) representa el diagrama de secuencia del comportamiento del sistema durante el escenario de ejecución principal.

Las pre y postcondiciones que extienden el diagrama para este ejemplo están definidas en OCL y se describen para el ejemplo en la Tabla 1. Obsérvese que, el contexto de la restricción OCL que se ha añadido al diagrama está centrado en un elemento del propio diagrama, de manera que esta restricción solo es aplicable en este escenario de ejecución.

En la propuesta, los modelos de diseño en UML serán transformados al modelo de prueba basado en el perfil de pruebas de UML utilizando QVT. En la Figura 5 (a) se muestra la arquitectura de pruebas para este ejemplo una vez realizada dicha transformación. Si bien la transformación en QVT no ha sido implementada aún, sí se cuenta con el pseudocódigo que realiza la correspondencia entre ambos modelos. Obsérvese que las diferentes clases están estereotipadas según el metamodelo de pruebas de UMLTP y que diversos elementos se nombran de manera que hacen referencia a lo que se está probando, tanto la funcionalidad, como su escenario de ejecución. También se pueden observar las diferentes relaciones entre el sistema de pruebas y el sistema.

El diagrama de la Figura 5(b) muestra el comportamiento del caso de prueba testCaseValidLogin(). El diagrama está compuesto de tres partes: la primera es la obtención de los datos de prueba, donde el dataPool tendrá que tener en cuenta las precondiciones anotadas en OCL del diagrama de secuencia origen (ver Tabla 1) para generar los datos de prueba del testContext.

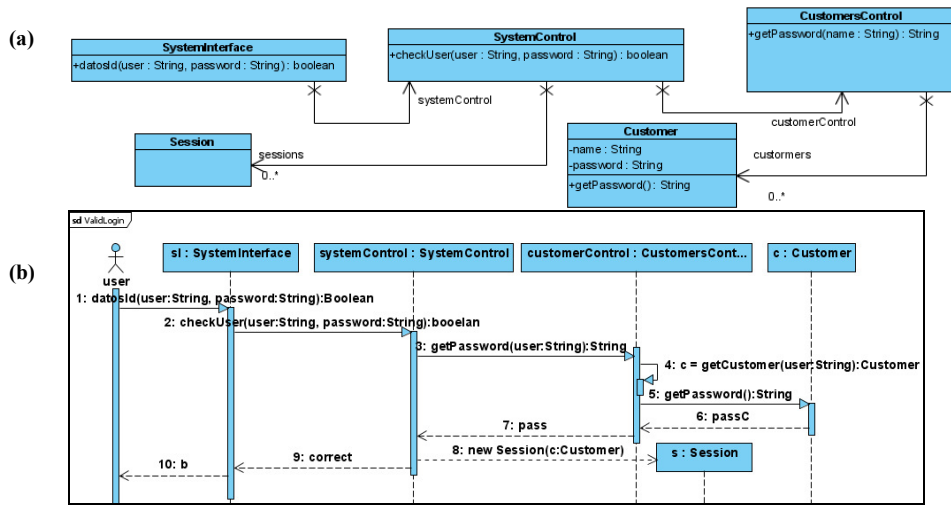


Figura 4. Modelos de diseño del sistema en UML

Tabla 1. Pre y postcondición en OCL para el ejemplo de la figura 6

context ValidLogin::SystemInterface::datosId(user:String, password:String):Boolean pre: self.systemControl.customers->exists(c: Customer c.name = user and c.password = password) post: result = true and self.systemControl.Sessions ->exists(s: Session s.oclIsNew())

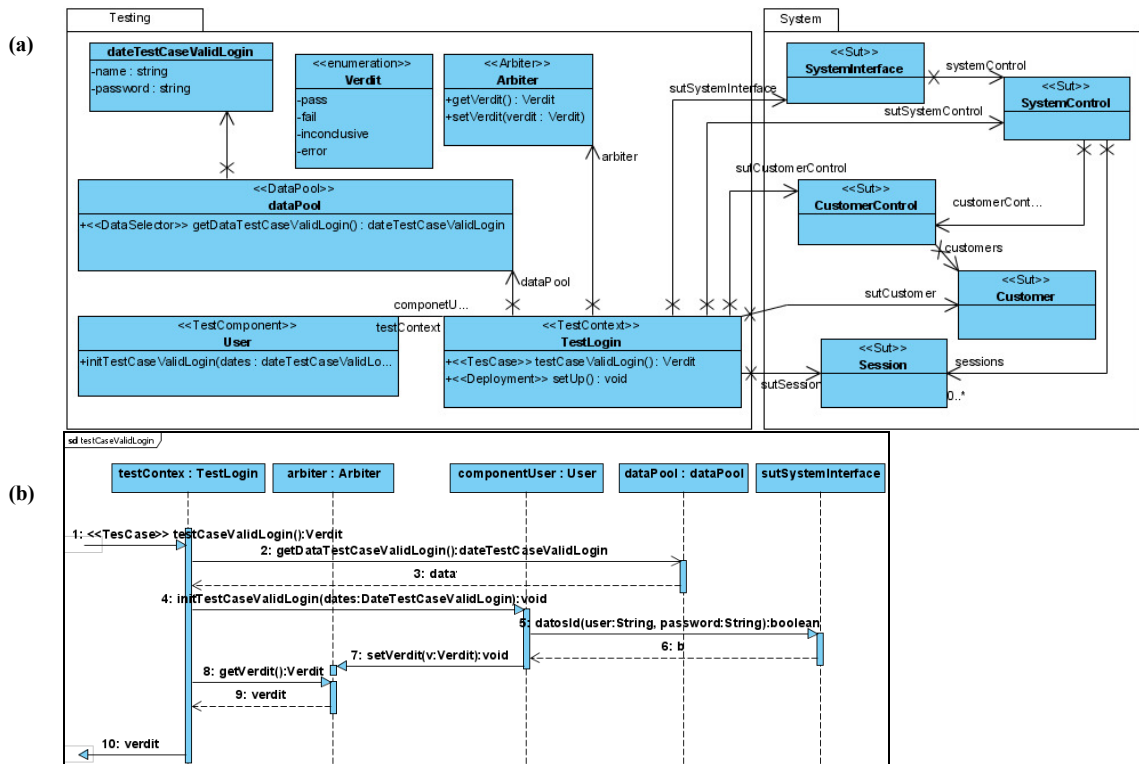


Figura 5. Modelo de clases del sistema de pruebas y diagrama de secuencia del caso de prueba

La segunda parte contiene la ejecución de la funcionalidad a probar, por lo que se inician los testComponents y se ejecutan en el mismo orden los mensajes que aparecían en el diagrama original que partían o llegaban a un actor, solo que ahora parten o llegan a los testComponents que representan

a esos actores. Y la última parte es la de actualización del veredicto, en la cual los testComponents tendrán que tener en cuenta las postcondiciones anotadas en OCL del diagrama origen (ver Tabla 1).

5 Conclusiones y trabajo a futuro

Esta investigación se centra principalmente en la generación automática de pruebas siguiendo un enfoque MDA. La idea central es que los modelos de diseño del sistema (diagrama de secuencia y diagrama de clases, que corresponden al PIM) sean transformados mediante QVT en un modelo de pruebas, que luego puede ser refinado según la plataforma final, también mediante transformaciones QVT (a un modelo JUnit, por ejemplo).

Para lograr una automatización completa del proceso, se requiere contar con una especificación del oráculo de las pruebas, del cual se puedan derivar los datos de prueba en forma automática. Para esto se ha definido una extensión del metamodelo de UML donde se expresan las pre y pos condiciones de cada diagrama de secuencia mediante el lenguaje OCL.

Para llevar a cabo la propuesta, resta la realización de las transformaciones QVT definidas en el trabajo. Las transformaciones a implementar se llevarán a cabo en dos pasos: las transformaciones para obtener el modelo de prueba; y las transformaciones para generar el modelo de pruebas dependiente de la plataforma (ej. JUnit) desde el modelo de pruebas anterior.

Dentro del trabajo a futuro se encuentra llevar los resultados de esta investigación al ámbito de las pruebas en líneas de producto de software. La importancia de la trazabilidad entre los distintos modelos que definen la línea hace pensar que un enfoque combinado entre MDA y líneas de producto es factible. La propuesta de este artículo puede ser extendida para tratar la variabilidad a nivel de los modelos de diseño del sistema y que esta variabilidad sea transformada a los modelos de prueba. De esta forma se obtendrán los casos de prueba de cada producto cuando la variabilidad sea resuelta.

Agradecimientos

Este trabajo ha sido parcialmente soportado por el proyecto PRALIN (Junta de Comunidades de Castilla-La Mancha, PAC08-0121-1374)

Referencias

1. Schmidt, D., Model-Driven Engineering. IEEE Computer, 2006. 39(2): p. 25-31.
2. Miller, J. and J. Mukerji, MDA Guide Version 1.0. 1, in Object Management Group. 2003.
3. Moreno N., R.J., Romero R., Vallecillo A., Desarrollo de Software dirigido por modelos, in Fábricas de Software: experiencias, tecnologías y organización, Ra-Ma, Editor. 2007, Ra-Ma: Madrid.
4. Meta Object Facility (MOF) 2.0 Query/View/Transformation, v1.0, OMG, Editor. 2008.
5. OMG, UML testing profile Version 1.0, O.M. Group, Editor. 2005.
6. Bertolino, A. Software Testing Research: Achievements, Challenges, Dreams. in International Conference on Software Engineering. 2007: IEEE Computer Society.
7. Dai, Z. Model-Driven Testing with UML 2.0. in Second European Workshop on Model Driven Architecture (MDA) with an emphasis on Methodologies and Transformations. 2004. Canterbury, England.
8. García-Rodríguez de Guzmán, I., M. Polo, and M. Piattini. Un Método para la creación de Servicios Web a partir de Bases de Datos Relacionales. in Desarrollo del Software Dirigido por Modelos y Aplicaciones 2006 (DSDM'06). 2006. Sitges, Barcelona (España).
9. Rodríguez, A., et al. Implementación de Heurísticas en QVT para la obtención de Clases de Análisis a partir de Modelos de Proceso de Negocio Seguros. in IV Taller sobre Desarrollo de Software Dirigido por Modelos, MDA y Aplicaciones (DSDM'07). 2007. Zaragoza, España.