Evaluación Automática de Modelos aplicando Técnicas de MBE

Javier Criado, Joaquín Cañadas, y Luis Iribarne

Departamento de Informática, Universidad de Almería
{javi.criado, jjcanada, luis.iribarne}@ual.es

Resumen. La construcción de modelos, como proceso de abstracción para definir una solución software, es una tarea que requiere ingenieros con cierta experiencia. Por un lado, modelos diferentes pueden ser igual de válidos para describir una misma solución y, por otro lado, disponer de una guía durante el aprendizaje de tareas de modelado puede ayudar a optimizar el proceso de desarrollo. Este artículo describe una propuesta para dar soporte a la evaluación de modelos utilizados durante las fases de análisis y diseño de un desarrollo de software. En particular, nuestro trabajo se aplica en la evaluación de modelos de casos de uso, clases y secuencias, como artefactos principales en la captura de requisitos, la descomposición modular y la descripción de comportamientos, respectivamente. Para evaluar dichos modelos, se ejecuta un conjunto de pruebas unitarias que son creadas automáticamente a partir de modelos de pruebas definidos conforme a un lenguaje específico de dominio.

Palabras clave: Ingeniería Basada en Modelos (MBE), Transformación Modelo-a-Texto (M2T), Generación automática de pruebas, Evaluación.

1. Introducción

Los ingenieros de software recorren un largo camino de aprendizaje hasta adquirir las aptitudes necesarias para crear buenas representaciones abstractas de los sistemas y aplicaciones que desarrollan. Uno de los lenguajes más utilizados para crear dichas abstracciones es el estándar UML (*Unified Modeling Language*) [1], que nos permite construir diferentes tipos de modelos, tanto estructurales como de comportamiento. No obstante, las amplias posibilidades que ofrece UML hacen que el dominio de este lenguaje se vaya afianzando no sólo con el estudio del estándar, sino también a partir de la experiencia que se adquiere al resolver problemas de distinta índole y al modelar aplicaciones de diferentes dominios.

En muchos casos, algunas de las dificultades para dominar este lenguaje de modelado vienen provocadas por errores comunes en los que se puede incurrir según la interpretación de los elementos que pueden usarse [2]. En otras ocasiones, es posible que distintos modelos sean válidos para llevar a cabo la representación de una misma solución y, además, cada compañía o equipo de desarrollo pueden preferir ciertas alternativas frente a otras equivalentes.

Por los motivos mencionados, creemos que una aproximación que sirva a los ingenieros para evaluar los modelos que construyen, teniendo como referencia las pautas establecidas por el equipo, puede ser beneficiosa y contribuir a la mejora de los desarrollos. Una aproximación de este tipo estará compuesta por los siguientes pasos. A partir de una especificación del problema a resolver, una persona con el rol de supervisor generará un modelo con las pruebas que deben ser ejecutadas. Este modelo lo construye teniendo en cuenta las posibles soluciones que él establece como ideales (puesto que soluciones distintas pueden ser igual de válidas) y conforme a un lenguaje específico de dominio (DSL, *Domain-Specific Language*). Como paso posterior, este modelo de pruebas se utilizará para generar automáticamente el código JUnit de las pruebas unitarias que serán ejecutadas sobre la solución propuesta por un ingeniero de software. Finalmente, el resultado de la ejecución de las pruebas será analizado por el ingeniero para poder evaluar el grado de adecuación de la solución que ha propuesto para llevar a cabo las medidas oportunas, por ejemplo, acometer los cambios necesarios para que todas las pruebas se ejecuten correctamente (ver Figura 1).

Además, este proceso de evaluación trata los modelos como un artefacto software más que someter a la gestión del ciclo de vida del desarrollo; en particular, es el primer artefacto que es sometido a pruebas para verificar así que las etapas de análisis y diseño han sido realizadas correctamente (asumiendo que en dichas etapas se generan los modelos suficientes para realizar esta verificación).

2. Propuesta

Uno de los elementos principales de la propuesta es el DSL utilizado para la descripción de las pruebas que deben ser ejecutadas sobre la solución ideal, junto con el proceso de transformación modelo-a-texto (M2T, *Model-to-Text*) que genera de forma automática las pruebas unitarias. En el estado actual de la propuesta, el lenguaje desarrollado se encuentra fuertemente ligado al dominio que suscitó el presente trabajo, la evaluación de los modelos construidos por estudiantes de segundo curso del Grado en Ingeniería Informática en la asignatura de Ingeniería

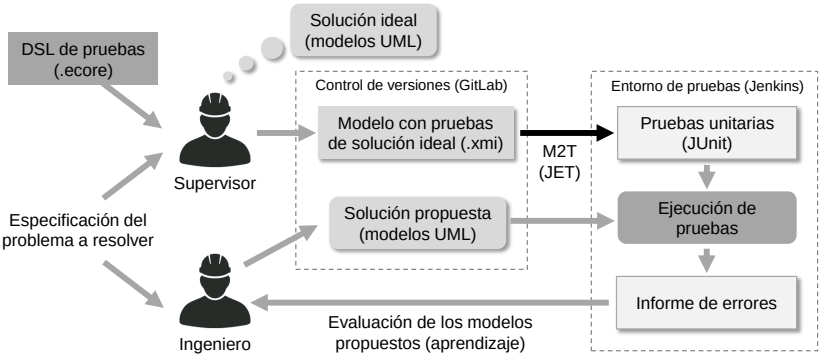

Figura 1. Propuesta para la evaluación de modelos

del Software. Por este motivo, tal y como muestra la Figura 2, cada *Curso* está compuesto por *Estudiantes* y *Profesores*, entendiendo que los primeros tienen el rol del `ingeniero` que está aprendiendo y que los segundos actúan con el rol de `supervisor`. Los estudiantes suben sus modelos a un *Repositorio* remoto y tienen asociado un *Proyecto* para la ejecución de las pruebas (en este caso, de tipo Jenkins). Todos los profesores tienen acceso a este proyecto para poder revisar y evaluar el trabajo de los alumnos.

Con estos elementos y a partir de un listado de alumnos, en nuestra aproximación utilizamos un repositorio de tipo GitLab para cada alumno y generamos un proyecto (*job*) de Jenkins relacionado con dicho repositorio. Cada vez que se suben nuevos modelos al repositorio de un alumno, se produce la construcción del proyecto asociado, lo que conlleva la ejecución de las pruebas unitarias que se encuentran en otro repositorio gestionado por los supervisores.

Para cada *Prueba*, es posible especificar cierta información descriptiva como su directorio, o el mensaje mostrado en caso de fallo. El *tipo* de prueba determina si se trata de una comprobación básica (como la existencia de un archivo) o de una aserción. En cualquier caso, las pruebas se traducen a código JUnit mediante una transformación M2T escrita en lenguaje JET.

Las aserciones comprobadas con las pruebas unitarias pueden realizarse sobre los modelos directamente, evaluando un conjunto de restricciones [3], o también sobre informes de métricas generados a partir de los modelos [4,5]. En nuestro caso, hemos usado como ejemplo la herramienta SDMetrics [6]. Por lo mencionado, una prueba puede estar asociada a un conjunto de *Archivos* que contendrán la información a evaluar, *i.e.*, archivos con los modelos o con las métricas.

Con estos elementos del lenguaje, la propuesta ha sido aplicada en la evaluación de modelos de casos de uso, clases y secuencias, ejecutando un conjunto de pruebas para cada solución software que se modela. De este conjunto, algunas

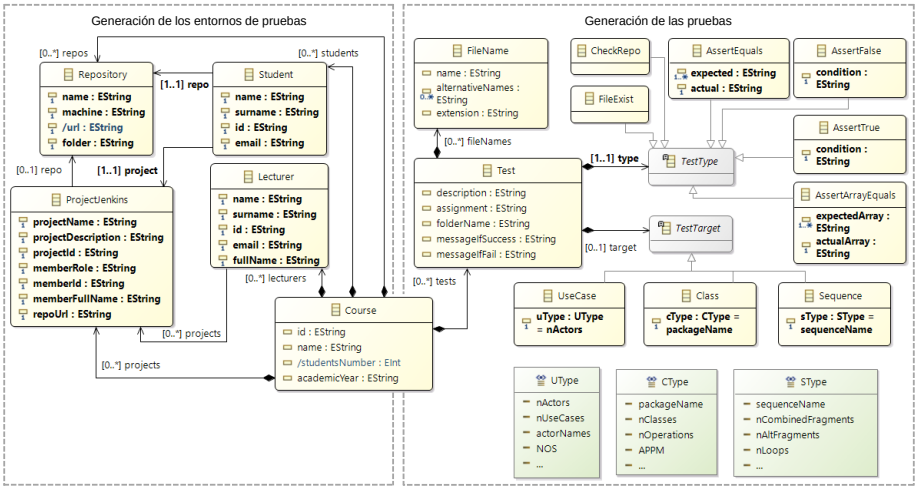

Figura 2. DSL para la evaluación automática de modelos

son específicas de cada solución, como por ejemplo: (a) que el número de casos de uso se encuentre en un intervalo recomendado, (b) que el paquete del modelo de clases tenga un nombre determinado, (c) que exista un número concreto de clases abstractas, (d) que estén presentes un número recomendado de relaciones de composición, o (e) que existan dos bucles anidados en un diagrama de secuencias particular.

Otras pruebas pueden ser de carácter genérico y aplicables a todas las soluciones software que se estén desarrollando, como por ejemplo: (a) que el número de pasos (NOS) en los casos de uso se encuentre en un intervalo recomendado, (b) que la media de parámetros de los métodos de las clases (APPM) no supere cierto valor, o (c) que el nivel máximo de anidación de los fragmentos combinados de los diagramas de secuencias esté por debajo de un valor límite.

3. Conclusiones y Trabajo Futuro

Este artículo describe una propuesta para facilitar la evaluación de los modelos construidos en las primeras etapas de desarrollo de soluciones software. Para ello, haciendo uso de un DSL, se generan automáticamente los repositorios y entornos de ejecución de pruebas de manera que, cuando un ingeniero sube nuevos modelos al repositorio, éste obtiene información acerca de la adecuación de su solución propuesta. Esta información se genera mediante la ejecución de pruebas construidas automáticamente a partir las posibles soluciones ideales establecidas por un supervisor. La propuesta ha sido validada por alumnos de segundo curso del Grado en Ingeniería Informática, tras su uso para la construcción de modelos de casos de uso, clases y secuencias en la asignatura de Ingeniería del Software.

Algunos de los objetivos futuros de la propuesta están relacionados con la mejora del DSL para poder describir de forma explícita la estructura interna de las pruebas, así como poder generarlas de forma automática a partir de las soluciones ideales construidas por los supervisores.

Agradecimientos. Este trabajo ha sido financiado con fondos del MINECO (proyectos TIN2013-41576-R y TIN2017-83964-R), de la UE (proyecto IoF2020 - H2020-IOT-2016) y del Grupo de Innovación Docente de Ingeniería y Tecnologías del Software de la UAL.

Referencias

1. Rumbaugh, J., Jacobson, I., and Booch, G.: Unified modeling language reference manual, the. Pearson Higher Education (2004)
2. Seidl, M., Scholz, M., Huemer, C., and Kappel, G.: UML@Classroom: An introduction to object-oriented modeling. Springer (2015)
3. Cabot, J., Clarisó, R., and Riera, D.: On the verification of UML/OCL class diagrams using constraint programming. J. Syst. Softw, 93, 1-23 (2014)
4. Genero, M., Piattini, M., and Calero, C.: A survey of metrics for UML class diagrams. Journal of object technology, 4(9), 59-92 (2005)
5. Lavazza, L., and Agostini, A.: Automated Measurement of UML Models: an open toolset approach. Journal of Object Technology, 4(4), 115-134 (2005)
6. Wüst, J.: SDMetrics (2012), `http://www.sdmetrics.com/`