Técnicas de prueba basadas en modelos para Procesos de Negocio

Federico Leonardo Toledo[1], Beatriz Pérez Lamancha[2], Macario Polo Usaola[3]
[1]Abstracta, Montevideo, Uruguay, ftoledo@abstracta.com.uy
[2]Centro de Ensayos de Software, Montevideo, Uruguay, bperez@ces.com.uy
[3]Alarcos Research Group, University of Castilla-La Mancha, Spain, macario.polo@uclm.es

Abstract. La importancia de los Procesos de Negocio ha crecido en el último tiempo, desde el área de negocios hasta el área informática, donde se apunta a automatizar toda su gestión y ejecución, permitiendo adaptarlo fácilmente y mejorarlo de manera continua. En este trabajo se presenta una propuesta para la generación automática de casos de prueba en entornos de procesos de negocio, aplicado para BPMN, dado que es el estándar indicado por OMG. Los procesos de negocio se representan en la etapa de modelado como un tipo especial de grafo. En la literatura sobre testing hay propuestas para generar casos de prueba a partir de diferentes tipos de modelos, como por ejemplo diagramas de secuencia, diagramas de actividad, máquinas de estados, expresiones regulares, etc., siguiendo lo que se le conoce como un enfoque MBT (*model-based testing*). Por tanto, las técnicas para generación automática de casos de prueba a partir de máquinas de estado y expresiones regulares podrían adaptarse para procesos de negocio. Esto cobra mayor relevancia al considerar que la versión 2.0 de BPMN contiene información para su ejecución en un motor adecuado.

Keywords: procesos de negocio, calidad de software, model based testing.

1 Introducción

La definición, modelado y análisis de los BP (del inglés *business processes* o Procesos de Negocio [1]) han adquirido gran importancia en la industria [7], como mecanismo para la obtención de objetivos de negocio. En general el ciclo de vida de un BP consta de cuatro etapas bien diferenciadas: **Modelado**, en etapa de análisis, se modela el BP utilizando alguna notación como puede ser BPMN (*Business process modelling notation* [2]); **Implementación**, el modelo representado gráficamente se traduce a un lenguaje intermedio, como para que pueda ser luego interpretado por un motor de ejecución. Por ejemplo puede ser WS-BPEL [3] o XPDL [4]; **Ejecución**, la representación del BP en el lenguaje intermedio se procesa por el motor de ejecución; **Control y Gestión**, a partir de los datos de la ejecución se pueden llevar a cabo actividades de control, monitorización y mejora del BP definido. La última versión de BPMN (versión 2.0) propone un modelo más rico, con información suficiente como para que se desarrollen motores de ejecución que lo interpreten directamente, sin necesidad de pasar a un modelo intermedio de ejecución como BPEL.

La mayoría de los trabajos relacionados con las pruebas de BP apuntan más a la etapa de construcción [12] [19] que a la etapa de diseño, o sea, las pruebas están pensadas a más bajo nivel.
La propuesta es trabajar en la generación automática de pruebas en entornos de BP a partir de modelos (por ejemplo BPMN dado que es el estándar indicado por OMG), aplicando técnicas de MDT (*model driven testing*) y MBT existentes para otros tipos de sistemas, analizando su practicidad y beneficios. Los BP se representan en la etapa de modelado como un tipo especial de grafo, que pueden asimilarse a máquinas de estado, las cuales pueden a su vez asimilarse con expresiones regulares, por lo que se podría pensar que hasta los BP podrían asimilarse con expresiones regulares. Por tanto, las técnicas para generación automática de casos de prueba a partir de máquinas de estado y expresiones regulares podrían adaptarse para BP. Luego, para ver la calidad que se logra con las pruebas generadas se analizan los criterios de cobertura, para lo que ya hay estudios sobre cobertura de expresiones regulares [21] y autómatas finitos y máquinas de estado [20]. Por otra parte, para analizar también la calidad de los casos de prueba, se suelen utilizar técnicas de mutación, donde se realizan modificaciones al programa bajo pruebas, para analizar si el conjunto de casos de pruebas es capaz de detectar el cambio [10]. Al generar casos de prueba para BP se definirán también criterios de cobertura a utilizar sobre los modelos BPMN, así como técnicas para mutar estos modelos y evaluar así la calidad de las pruebas generadas.

2 Trabajos Relacionados

Existen trabajos en el área que pueden ser profundizados o extendidos [11][12], así como también trabajos realizados en otras áreas pueden ser adaptados para tomar los beneficios en este campo en concreto. Como ya se mencionó, se podrían adaptar y aplicar técnicas de generación de casos de prueba tales como las existentes para:

- Expresiones regulares y máquinas de estado [13][21]
- WS-BPEL [12] [19]

En nuestro grupo de investigación se ha estado trabajando en varias de las temáticas que se propone trabajar, por ejemplo:

- MDE (*model driven engineering*) aplicado a Business Processes [8]
- MBT: a partir de UML-TP (*UML - testing profile*) generar pruebas unitarias xUnit [9]
- Técnicas de Mutación [10]

3 Testing de Procesos de Negocios

Existen también metodologías que abordan la temática de cómo probar BP [6]. Un BP orquesta una serie de pasos, o actividades a realizar (generalmente los sistemas de BP coordinan tareas implementadas con *Web Services* [5]). A modo de “test unitario”, aquí se deberían probar esos pasos o actividades en forma aislada. Esto generalmente es algo bastante atómico, tal como cuando se prueba una funcionalidad o método

concreto en cualquier tipo de sistema. Generalmente esto se logra invocando directamente un Web Service, considerando sus entradas y salidas. Luego de tener los componentes probados, se podría probar el proceso a más alto nivel. Para esto es que teniendo en cuenta el diseño del proceso a nivel de BPEL o de BMPN, se podría pensar en construir un test que invoque los distintos servicios con secuencias interesantes. Luego de esto, se pasaría a una etapa "end-to-end" testing, en donde la idea es probar el sistema completo. Esta tarea es la del test funcional de sistema tradicional, así como también el testing de regresión y la automatización de estas pruebas. Se puede aportar a estas últimas tareas, aplicando el enfoque de MBT, logrando mejorar y simplificar el diseño de pruebas de sistema, e incluso lograr que sea más fácil su automatización.

4 Adaptación de técnicas clásicas al contexto de BP

Las técnicas para generación automática de casos de prueba a partir de máquinas de estado y expresiones regulares podrían adaptarse para BP, siguiendo el enfoque MBT. Lo mismo sucede con las técnicas de generación de mutantes.

4.1 Model based testing: Máquinas de estado y Expresiones regulares

En particular planteamos dos formas de modelar el BP, para que a partir de ese modelo se puedan aplicar técnicas conocidas para genera casos de prueba. Estas son máquinas de estado y expresiones regulares.
Una **máquina de estados** es un grafo dirigido donde los nodos representan los estados y las aristas representan las transiciones entre estos estados. De esta forma se puede representar el comportamiento de un sistema. Al momento de querer validar este comportamiento se buscará cumplir con ciertos criterios de cobertura sobre la máquina de estados, como para garantizar cierta calidad del conjunto de pruebas. Esto puede ser por ejemplo, diseñar pruebas que visiten todos los estados, que pasen por todas las transiciones, o que recorran todas las combinaciones de pares entrada/salida de cada estado. Será necesario para esto definir un correcto mapeo entre la representación del BP y la máquina de estados.
Si el comportamiento del sistema bajo pruebas se puede describir mediante máquinas de estado, estas se pueden entender como un artefacto similar a un autómata finito, de los que pueden derivarse **expresiones regulares**. En ese sentido pueden aprovecharse las técnicas existentes sobre expresiones regulares, útiles para generar casos de prueba que recorran la máquina de estados asociada a la expresión. Las "palabras" generadas a partir de la expresión regular serán equivalentes a casos de pruebas, donde se especifican la secuencia de pasos a seguir y el estado esperado al que debe llegar tras la ejecución de cada paso. Para esto será importante analizar la aplicabilidad a sistemas de mediano y gran porte, como para asegurar que la cantidad de pruebas generadas no sea excesiva, o que se pueda seleccionar el conjunto reducido más relevante.
Considerando que BPMN 2.0 tiene información suficiente para ejecución, los casos de prueba generados a partir de este modelo podrían llegar a tener información

suficiente para su ejecución automática. Para esto se deberá definir un metamodelo de pruebas el cual debería ser implementado por el motor de ejecución del BP, como una extensión para soportar la ejecución de pruebas automáticas.

4.2 Mutación en testing tradicional

Las técnicas de testing basadas en mutantes implican introducir pequeños cambios en un programa para producir un conjunto de mutantes (variaciones del programa original). Estos son generalmente utilizados para medir la calidad de las pruebas generadas [17]. Mientras más mutantes "mate" el conjunto de pruebas, más calidad tendrá. Por otra parte, la idea es generar mutantes de tal forma que sean difíciles de matar. De esa forma, si los mutantes son más resistentes a las pruebas, serán mutantes de mejor calidad, brindando la posibilidad de evaluar mejor la calidad de las pruebas.
Si bien la mutación generalmente se aplica directamente a código fuente, también puede ser aplicada a distintos modelos [16]. El principal desafío es que para generar mutantes es necesario definir operadores de mutación de forma eficiente [17] sobre el elemento que se esté aplicando la mutación. Por ejemplo, si se trata de mutar código fuente, se pensará en cambiar operadores aritméticos (suma, resta, etc.); si se trata de una máquina de estados se pensará en remover estados, cambiar condiciones, orientación de las aristas, cambiando el destino de una transición, etc. [18]. Un mutante es bueno si logra sobrevivir a los casos de prueba. Si un operador genera mutantes que son eliminados por cualquier caso de prueba, entonces el operador no es bueno. De esta forma los operadores de mutación deben ser suficientemente buenos como para generar buenos mutantes, lo cual se logra verificando empíricamente, probando contra distintos conjuntos de casos de prueba.

4.3 Aplicación de la metodología de mutantes al testing de BP

Para poder luego medir la calidad de las pruebas que se generen, sería deseable mutar los modelos BPMN, definiendo algún operador de mutación sobre ellos.
Existen técnicas de mutación de modelos [16] [18], pero estas no son aplicables para modelos de negocios de forma directa. El problema es que si por ejemplo, pensando en pruebas a nivel de sistema, se cambia una transición, se quita un estado, o se cambia el estado final, seguramente se generen mutantes muy fáciles de matar. En este sentido es interesante profundizar la investigación para generar mutantes interesantes a nivel de BPMN, que permitan de esa manera evaluar la calidad de las pruebas que se puedan generar a partir de él (ya sean para ejecución manual o automática). Como ya se mencionó, ya hay trabajos similares para BPEL [14][15] y otros que podrían tomarse como base, analizando su aplicabilidad y calidad, ya que es necesario evaluar si con esos operadores se generarían mutantes de calidad. Se podría pensar en situaciones típicas de error, analizar qué errores se cometen habitualmente, y de esa forma generar mutantes basados en estos errores comunes, como para asegurar que los casos de prueba contemplados verifican que no se den esas situaciones de error típicas.

Trabajo futuro

Se plantea como trabajo futuro continuar con la investigación en el área, contribuyendo en aspectos en los que el grupo ha adquirido experiencia. Entre estos puntos se podrían mencionar los siguientes:

- Generación automática de escenarios interesantes de prueba, basados en patrones y distintas técnicas de MBT, como máquinas de estado y expresiones regulares, aplicados a BPMN.
- Estudiar operadores de mutación a modelos BPMN que sean efectivos, de modo que generen mutantes de buena calidad.

Agradecimientos

Este trabajo ha sido parcialmente financiado por la Agencia Nacional de Investigación e Innovación (ANII, Uruguay) y por el proyecto DIMITRI (Desarrollo e Implantación de Metodologías y Tecnologías de Testing, TRA2009_0131, Ministerio de Ciencia e Innovación).

Referencias

1. Weske, M., BPM Concepts, Languages, Architectures, Springer, 2007,ISBN 978-3-540-73521-2
2. Business Process Modeling Notation (BPMN), v.2.0, OMG, http://www.omg.org/spec/BPMN/2.0
3. WS - Business Process Execution Language (WS-BPEL), v.2.0, OASIS, http://docs.oasisopen.org/wsbpel/2.0
4. XML Process Definition Language (XPDL), v. 2.1, WfMC, http://www.wfmc.org/xpdl.html
5. Web Services Architecture (WSA), W3C, http://www.w3.org/TR/ws-arch
6. Business Process Testing - A New Approach - by Verónica Puymalie, Diego Marín, Marcelo Arispe, Core Magazine (http://www.coremag.eu)
7. Mutschler, B., Reichert, M., Bumiller, J.: Unleashing the effectiveness of process-oriented information systems: Problem analysis, critical success factors and implications. IEEE Transactions on Systems, Man, and Cybernetics (Part C) 38 (2008) 280–291
8. A. Delgado, F. Ruiz, I. García-Rodríguez, and M. Piattini. 2009. MINERVA: model driven and service oriented framework for the continuous business process improvement and related tools. In Proceedings of the 2009 international conference on Service-oriented computing (ICSOC/ServiceWave'09).
9. Beatriz Pérez Lamancha, Pedro Reales Mateo, Ignacio Rodríguez, Macario Polo Usaola, and Mario Piattini Velthius. 2009. Automated model-based testing using the UML testing profile and QVT. In Proceedings of the 6th International

Workshop on Model-Driven Engineering, Verification and Validation (MoDeVVa '09)
10. Mateo, P.R., Usaola, M.P., Offutt, J. Mutation at system and functional levels (2010) ICSTW 2010 - 3rd International Conference on Software Testing, Verification, and Validation Workshops, art. no. 5463638, pp. 110-119.
11. Alin Stefanescu, Sebastian Wieczorek, and Andrei Kirshin. 2009. MBT4Chor: A Model-Based Testing Approach for Service Choreographies. In Proceedings of the 5th European Conference on Model Driven Architecture - Foundations and Applications (ECMDA-FA '09)
12. Zhongjie Li, Wei Sun, Zhong Bo Jiang, Xin Zhang, "BPEL4WS Unit Testing: Framework and Implementation," Web Services, IEEE International Conference on, pp. 103-110, IEEE International Conference on Web Services (ICWS'05), 2005
13. Offutt J, Liu S, Abdurazik A, Ammann P. Generating test data from state-based specifications. Software Testing, Verification and Reliability 2003; 13(1):25–53.
14. Benatallah, Boualem; Casati, Fabio; Kappel, Gerti; Rossi, Gustavo; Domínguez-Jiménez, Juan-José; Estero-Botaro, Antonia; García-Domínguez, Antonio; Medina-Bulo, Inmaculada; GAmera: A Tool for WS-BPEL Composition Testing Using Mutation Analysis, Web Engineering, Lecture Notes in Computer Science, 2010, Springer Berlin / Heidelberg
15. Estero-Botaro, A.; Palomo-Lozano, F.; Medina-Bulo, I.; Quantitative Evaluation of Mutation Operators for WS-BPEL Compositions, 2010, Software Testing, Verification, and Validation Workshops (ICSTW)
16. S. C. P. F. Fabbri, J. C. Maldonado, T. Sugeta, and P. C. Masiero. Mutation testing applied to validate specifications based on statecharts. In 10th IEEE International Symposium on Software Reliability Engineering (ISSRE 1999), pages 210.219, 1999.
17. L. Bottaci and E. Mresa. Efficiency of mutation operators and selective mutation strategies: An empirical study. Software Testing, Verification and Reliability, 9(4):205.232, 1999.
18. Robert M. Hierons and Mercedes G. Merayo. 2007. Mutation Testing from Probabilistic Finite State Machines. In Proceedings of the Testing: Academic and Industrial Conference Practice and Research Techniques - MUTATION (TAICPART-MUTATION '07)
19. García-Fanjul, J. Tuya, C. de la Riva, “Generating test cases specifications for BPEL compositions of web services using SPIN”. In Proceedings of the Int. Workshop on Web Services – Modelling and Testing. Palermo (2006), pp. 83-94.
20. A. Jefferson Offutt, Shaoying Liu, Aynur Abdurazik, Paul Ammann. Generating test data from state-based specifications. Softw. Test., Verif. Reliab., 2003: 25~53
21. Flores, A. P. (2009). Proceso de evaluación basado en pruebas para la sustitución de componentes software. Tesis doctoral, Dpto. de Tecnologías y Sistemas de Información, Universidad de Castilla-La Mancha.