

Framework para la Generación Dinámica de Invariantes en Composiciones de Servicios Web con WS-BPEL

Antonio García Domínguez, Manuel Palomo Duarte e Inmaculada Medina Buló

Departamento de Lenguajes y Sistemas Informáticos, Universidad de Cádiz
C/ Chile nº 1, CP 11002 Cádiz

antonio.garciadominguez@alum.uca.es, {manuel.palomo,inmaculada.medina}@uca.es

Resumen. Los lenguajes para la composición de servicios web, como el estándar OASIS WS-BPEL 2.0, permiten programar a gran escala. Sin embargo, este lenguaje presenta un reto para la realización de pruebas de caja blanca, debido a la inclusión de instrucciones específicas para compensación o descubrimiento e invocación dinámicos de servicios. Por otro lado, la generación automática de invariantes ha demostrado ser una técnica eficaz para ayudar en la prueba y mejora de programas escritos en lenguajes imperativos tradicionales y creemos que también lo sería para WS-BPEL. En este artículo proponemos un framework para generar invariantes potenciales dinámicamente a partir de registros de ejecución de una composición de servicios web en WS-BPEL.

Palabras Clave: Servicios web, Composición de servicios, WS-BPEL, Prueba de caja blanca, Generación dinámica de invariantes.

1 Introducción

Los servicios web (WS) y las arquitecturas orientadas a servicios (SOA) parecen ser una de las claves para entender el futuro de las TIC a corto y medio plazo [6]. El estándar OASIS WS-BPEL 2.0 [8] permite ofrecer servicios más potentes basados en otros disponibles mediante orquestación. Además hay otras tecnologías, como la pila de WS (*WS-Stack*) [9], que pueden extenderlo.

Pero WS-BPEL representa un reto para las técnicas tradicionales de caja blanca, debido a la inclusión de instrucciones específicas para el manejo de WS no presentes en lenguajes tradicionales, como compensación o descubrimiento e invocación dinámicos de servicios [3].

La generación automática de invariantes potenciales [4] ha demostrado ser una técnica eficaz para ayudar en la prueba y mejora de programas escritos en lenguajes imperativos tradicionales. Conviene aclarar en este punto que en este artículo se utilizan los términos “invariante” e “invariante potencial” en el mismo sentido en que se usa en la bibliografía de la materia [4], refiriéndose “invariante” a cualquier propiedad que es cierta en un determinado punto del programa (como un aserto, pre-condición, invariante de bucle, etc), e “invariante potencial” a cualquier propiedad que se mantiene en los distintos casos de prueba ejecutados.

Este artículo propone una arquitectura basada en sistemas disponibles actualmente para implementar un framework que genere dinámicamente invariantes potenciales a partir de información recopilada en diversas ejecuciones de una composición WS-BPEL.

El resto de este artículo se organiza del siguiente modo: el segundo apartado comenta la utilidad de la generación dinámica de invariantes de composiciones WS-BPEL. La sección tercera presenta la arquitectura del framework, indicando sistemas apropiados para su implementación. Por último, el artículo finaliza comparando nuestra propuesta con otras alternativas y presentando conclusiones, así como unas líneas generales de nuestro trabajo próximo.

2 Invariantes y Composiciones WS-BPEL

Hasta la fecha se ha investigado poco sobre la aplicación de técnicas de prueba de caja blanca directamente sobre el código de composiciones WS-BPEL ejecutado en un entorno real. Las principales propuestas [3] crean un modelo de simulación en un entorno especializado para pruebas.

Pero la simulación de un motor WS-BPEL es algo complejo, dado que hay una gran cantidad de características nada triviales que implementar. En caso de que alguna de estas características no se implementara correctamente, la composición no se estaría probando adecuadamente. Por ello, entendemos que es un proceso propenso a errores, dado que no se basa en la ejecución del código WS-BPEL en un entorno real (es decir, un motor WS-BPEL que invoque a servicios reales).

Por contra, el framework que proponemos genera dinámicamente invariantes potenciales a partir de una serie de ejecuciones de la composición WS-BPEL en un motor real. Por lo tanto, consideramos que se adapta mejor a la naturaleza del lenguaje y puede ser una ayuda interesante para la prueba de programas WS-BPEL.

2.1 Uso de invariantes

La generación automática de invariantes ha demostrado ser una técnica eficaz para ayudar en la prueba de caja blanca y la mejora de programas [4]. Los invariantes generados a partir de un programa pueden usarse de diversas formas para mejorarlo:

Depuración de errores Un invariante inesperado puede hacernos ver un fallo en el código que de otra forma podría haber pasado desapercibido. Esto incluye, por ejemplo, llamadas a funciones con valores no válidos en algún parámetro o fallos en las condiciones de bucles.

Asistencia al ampliar un programa Tras comprobar qué invariantes deben mantenerse y cuáles no entre dos versiones de un programa, podrían compararse los resultados esperados con los realmente obtenidos. Cualquier diferencia indicaría un error en el nuevo código.

Documentación Los invariantes más destacados pueden incluirse en la documentación del código del programa, para que los desarrolladores que tengan acceso a él puedan consultarlos.

Verificación Se puede comparar la especificación del programa con los invariantes obtenidos para ver si esta se cumple.

Mejora de los casos de prueba Un invariante potencial erróneo que se haya generado dinámicamente, como se observará en el siguiente apartado, puede indicar una deficiencia en el conjunto de casos de prueba usado para inferirlo.

2.2 Generación Automática de Invariantes

Existen dos tipos de generadores automáticos de invariantes: estáticos y dinámicos. Los generadores estáticos [2] son los más usados: deducen los invariantes de un programa analizando su código fuente. Los invariantes generados de esta forma son siempre ciertos. Sin embargo, su número y alcance es reducido, debido a las limitaciones inherentes al mecanismo formal que analiza el código, sobre todo en lenguajes poco convencionales como WS-BPEL.

Por el contrario, los generadores dinámicos de invariantes potenciales [4] informan de posibles invariantes de un programa observados en la información recopilada en varias ejecuciones de él. Para ello incluyen un mecanismo formal que analiza dicha información e infiere los invariantes observados. Por lo tanto, esta técnica se basa en la ejecución del código sobre un conjunto de casos de prueba y es independiente del lenguaje concreto.

De este modo, la obtención de invariantes erróneos no implica necesariamente fallos en el programa, sino que puede venir originada por el empleo de un conjunto incompleto de casos de prueba. Por ejemplo, si un programa recibe como entrada un entero, x , pero sólo le proporcionamos casos de prueba en los que reciba valores naturales, probablemente obtengamos el invariante falso $x \geq$

0. Dicho invariante indicaría que es necesaria una mejora del conjunto de casos de prueba, incluyendo casos en los que x reciba un valor negativo, para que esto no ocurra.

2.3 Generación Dinámica de Invariantes en Composiciones WS-BPEL

Consideramos que la generación dinámica de invariantes puede ser una técnica adecuada para ayudar en la prueba y mejora de composiciones de WS en WS-BPEL. Si se dispone de un buen conjunto de casos de prueba, los diferentes registros de ejecución serán una buena muestra de la lógica interna de la composición, incluyendo sus aspectos más específicos de la composición (como compensación o descubrimiento e invocación dinámica de servicios), y, por lo tanto, el generador inferirá invariantes correctos y significativos que podrán ayudar a su mejora.

Una ventaja a destacar es que toda la información de los registros se recopila directamente de ejecuciones del código de la composición, sin usar ningún tipo de lenguaje intermedio. De este modo, se evitan errores que podrían producirse en la traducción del código WS-BPEL o el modelado del motor WS-BPEL y los servicios invocados en un entorno de su simulación.

Hay que tener en cuenta que, por lo general, no podemos suponer que todos los servicios externos vayan a estar disponibles a la hora de realizar las ejecuciones. Esto puede deberse a limitaciones de recursos, costes de uso, etc. Incluso puede ser que se prefiera probar el comportamiento de una composición con respuestas predeterminadas de algunos servicios que definan un escenario del tipo “¿qué pasaría si el servicio x respondiera el valor y ?”. Por ello el framework debe permitir que, en el momento de la ejecución de los casos de prueba, las llamadas a ciertos servicios se sustituyan por otras que respondan otros con sus mismas interfaces.

3 Arquitectura propuesta

En este apartado comentamos la arquitectura del framework que proponemos para la generación dinámica de invariantes potenciales para composiciones WS-BPEL (figura 1). Sus tres etapas se corresponden con las tres etapas de la generación dinámica de invariantes potenciales [4].

3.1 Etapa de Instrumentalización

En esta primera etapa recibimos el código del programa original y le realizamos los cambios oportunos para que durante su posterior ejecución produzca la información que necesitará el generador de invariantes.

Para ello podemos incluir en el motor WS-BPEL funciones que extiendan XPath ofreciendo posibilidades de registro, y ejecutar en dicho motor una versión del programa *instrumentado*, es decir, que llame a dichas funciones cuando sea necesario.

3.2 Etapa de ejecución

Según los servicios externos que deseemos suplantar (ninguno, algunos o todos) tendremos la posibilidad de obtener posteriormente invariantes que reflejen el comportamiento de distintos elementos de la composición: lógica interna de la composición llamando a servicios reales, sólo de la lógica interna del proceso WS-BPEL o, incluso, de su lógica interna llamando a algunos servicios reales y a otros cuyo comportamiento controlemos. Para ello necesitamos principalmente dos componentes:

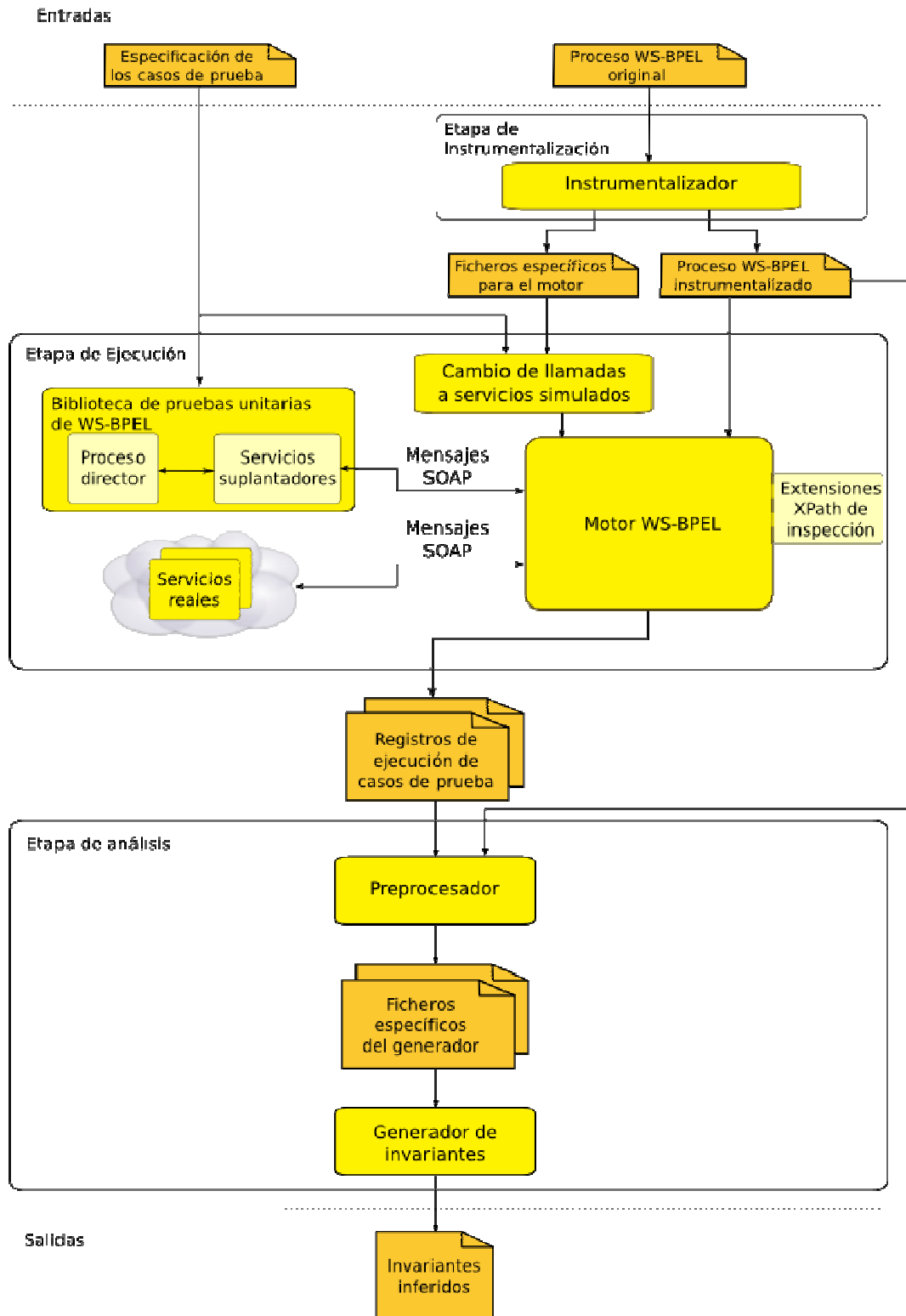


Fig. 1 Arquitectura general del framework

- Un motor WS-BPEL que ejecute el proceso. Tras analizar distintos sistemas nos decantamos por ActiveBPEL [1], que incluye soporte completo del estándar WS-BPEL 2.0, tiene un consumo relativamente bajo de recursos, se distribuye bajo licencia libre, y está mantenido y actualizado por la empresa ActiveVOS. Además cada composición que despliega tiene un fichero con las direcciones de los servicios a invocar, por lo que modificando dicho fichero podemos controlar si se invocan WS reales o suplantadores.
- Una biblioteca de pruebas unitarias de WS-BPEL. Esta debe incluir un *proceso director* que prepare y monitorice la ejecución de todos los casos de prueba. Igualmente debe gestionar un servidor de *servicios suplantadores*, que atienda a las indicaciones del director. BPELUnit [7] puede cubrir perfectamente nuestras necesidades. Este entorno disponible bajo licencia libre automatiza la ejecución de un conjunto de casos de prueba en un motor WS-BPEL 2.0 como ActiveBPEL.

3.3 Etapa de análisis

Una vez finalizada la etapa de ejecución disponemos de los registros correspondientes a todos los casos de prueba, así que es el momento de pasarle dichos registros al generador de invariantes.

Consideramos que se podría utilizar el generador dinámico de invariantes Daikon [4]: es altamente configurable, permite conectar su salida con el simplificador formal *Simplify* y lleva publicado con licencia libre varios años, durante los cuales ha sido mantenido y documentado correctamente. Además, se ha usado con éxito en diversos lenguajes, lo que nos da ciertas garantías de su correcto funcionamiento y de la utilidad de los invariantes que genera.

Dado que el formato de los ficheros de entrada de Daikon presenta ciertas restricciones, es necesario que nuestro framework incorpore un preprocesador que se traduzca a dicho formato los registros de ejecución producidos por la etapa anterior. Para ello necesita información incluida en los ficheros WS-BPEL instrumentalizados y sus ficheros WSDL y XML Schema asociados.

4 Trabajos relacionados

En este apartado comentamos algunos trabajos relacionados con nuestra propuesta. En [10] se presenta una propuesta para la generación dinámica de invariantes potenciales con Daikon como método para evaluar el cumplimiento del contrato de calidad de servicio (en inglés *Service Level Agreement*) de un WS. Por lo tanto, este enfoque está orientado a la prueba de caja negra de WS. Mientras que, por el contrario, nuestra propuesta puede ayudar a la prueba de caja blanca de una composición WS-BPEL mediante la generación de invariantes potenciales referentes a su lógica interna.

La relación entre los casos de prueba usados para la generación dinámica de invariantes potenciales y la calidad de los invariantes deducidos se estudia en [5]. Aumentar el conjunto de casos de prueba con ejemplos adecuados puede mejorar la precisión de los invariantes generados.

En [11] se comenta la generación automática de casos de prueba para composiciones WS-BPEL de acuerdo a criterios de cobertura de estados y de transiciones. Usando dichos casos como parte de la entrada de nuestro framework se podría mejorar la confianza en los invariantes generados.

5 Conclusiones y Trabajo Futuro

En este artículo hemos mostrado cómo la generación dinámica de invariantes potenciales, respaldada por un buen conjunto de casos de prueba, puede ser una solución adecuada para ayudar a la prueba y mejora de composiciones WS-BPEL. Esto es, en gran medida, gracias a que está basada en la

información de registros obtenidos a partir de ejecuciones reales de la composición, lo que permite tratar sus peculiaridades (como compensaciones, concurrencia, etc).

Hemos propuesto también un framework para generar dinámicamente invariantes potenciales de la lógica interna de una composición WS-BPEL. Se han identificado los requisitos de cada uno de sus componentes y hemos propuesto sistemas libres disponibles actualmente para su implementación: BPELUnit como biblioteca de pruebas unitarias, ActiveBPEL como motor WS-BPEL y Daikon como generador de invariantes.

También hay que destacar que nuestro framework permite que, al ejecutar los casos de prueba, bien se llamen WS reales, bien se reemplacen ciertas llamadas por otras a servicios con igual interfaz que se respondan con un mensaje predeterminado. De este modo, se pueden realizar pruebas aunque existan restricciones en el uso de algún WS y, además, se posibilita la prueba de escenarios en los que se controle el comportamiento de algunos de ellos.

Nuestro siguiente trabajo será implementar el framework. Una vez implementado, podremos realizar pruebas con diversos ejemplos al objeto de comprobar su funcionamiento. Para evaluar el comportamiento del framework se pueden aplicar métricas como la fiabilidad de los invariantes generados o el tiempo necesario para inferirlos.

El siguiente paso será estudiar la relación entre la calidad de los invariantes inferidos por el framework y la del conjunto de casos de prueba empleado para generarlos. Para ello nos podemos ayudar de diversas composiciones WS-BPEL con sus especificaciones y conjuntos de casos de prueba generados bajo ciertos criterios de cobertura, como cobertura de ramas o de transiciones. Por último, podríamos utilizar los invariantes generados como apoyo a la prueba de caja blanca para WS-BPEL y comprobar si mejora sus resultados.

Agradecimientos

Este trabajo ha sido financiado por el Programa Nacional de I+D+I del Ministerio de Educación y Ciencia y fondos FEDER mediante el proyecto SOAQSim (TIN2007-67843-C06-04)

Referencias

1. ActiveVOS: ActiveBPEL. <http://sourceforge.net/projects/activebpel> (2008)
2. Nikolaj Bjørner, Anca Browne, Zohar Manna: Automatic Generation of Invariants and Intermediate Assertions. *Theoretical Computer Science*, vol. 173(1), pp. 49–87 (1997)
3. Antonio Bucchiarone, Hernán Melgratti, Francesco Severoni: Testing Service Composition. En: ASSE: Proceedings of the 8th Argentine Symposium on Software Engineering (2007)
4. Michael D. Ernst, Jake Cockrell, William G. Griswold, David Notkin: Dynamically Discovering Likely Program Invariants to Support Program Evolution. *IEEE Transactions on Software Engineering*, vol. 27(2), pp. 99–123 (2001)
5. Neelam Gupta: Generating Test Data for Dynamically Discovering Likely Program Invariants. En: ICSE, Workshop on Dynamic Analysis (2003)
6. Randy Heffner, Larry Fulton: Topic Overview: Service-Oriented Architecture. Forrester Research, Inc. (2007)
7. Philip Mayer, Daniel Lübke: Towards a BPEL Unit Testing Framework. En: TAV-WEB: Proceedings of the 2006 Workshop on Testing, Analysis, and Verification of Web Services and Applications, pp. 33–42, ACM (2006)
8. OASIS: WS-BPEL 2.0 Standard. <http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html> (2007)
9. M. Papazoglou: Web Services Technologies and Standards. *Computing Surveys* (enviado para revisión) (2007)
10. SeCSE: A1.D3.3: Testing Method Definition V3. <http://secse.eng.it/wp-content/uploads/2007/08/a1d33-testing-method-definition-v3.pdf> (2007)
11. Yongyan Zheng, Jiong Zhou, Paul Krause: An Automatic Test Case Generation Framework for Web Services. *Journal of Software*, vol. 2(3), pp. 64–77 (2007)