

Operadores de mutación para WS-BPEL 2.0

Antonia Estero Botaro, Francisco Palomo Lozano e Inmaculada Medina Buló

Departamento de Lenguajes y Sistemas Informáticos, Universidad de Cádiz
C/ Chile Nº 1, Cádiz

{Antonia.estero,francisco.palomo,inmaculada.medina}@uca.es

Resumen. El gran auge que están alcanzando en los últimos años los servicios web y las composiciones de éstos realizadas con el lenguaje WS-BPEL, hace necesario prestar especial atención a la prueba de este tipo de software. El análisis de mutaciones es una técnica adecuada para medir la calidad de los casos de prueba y ha sido ampliamente aplicada a programas escritos en diversos lenguajes. Para poder utilizarla es necesario disponer de un conjunto de operadores de mutación, específicos del lenguaje, que determinen los cambios a introducir en el programa a probar. En este artículo se define un conjunto de operadores de mutación para el lenguaje WS-BPEL 2.0 que cubre un amplio abanico de características de este nuevo estándar OASIS. Se presenta una clasificación de los operadores propuestos así como su descripción.

Palabras Clave: Análisis de mutaciones, Operadores de mutación, Composición de servicios, WS-BPEL.

1 Introducción

El lenguaje WS-BPEL [10] permite crear procesos de negocio mediante la composición de Servicios Web (WS) preexistentes y ofrecerlos a su vez como WS. La importancia económica que están alcanzando las composiciones WS-BPEL [6] obliga a prestar especial atención a la prueba de este tipo de software. Se han publicado trabajos relacionados con diversos aspectos de la prueba de composiciones en WS-BPEL, en su mayoría relacionados con la generación de casos de prueba [4, 5, 16, 17, 18]. Estos trabajos, excepto [5], no estudian la calidad de los casos de prueba generados.

Una técnica apropiada para medir la calidad de conjuntos de casos de prueba es el análisis de mutaciones [11, 15] (*mutation analysis*). Esta técnica utiliza una serie de operadores de mutación para generar un conjunto de mutantes a partir del programa a probar. Aunque se han publicado diversos trabajos sobre la definición de operadores de mutación para distintos lenguajes, no hemos encontrado ninguno que defina los operadores de mutación para el lenguaje WS-BPEL.

El objetivo principal de nuestro trabajo es definir un conjunto de operadores de mutación para el lenguaje WS-BPEL 2.0 que permita modelar los fallos comunes que pueden cometer los programadores a la hora de escribir un programa en este lenguaje. Durante el proceso de definición de los operadores se ha prestado especial atención en la eliminación de aquéllos que, de manera obvia, podrían provocar la generación de mutantes equivalentes.

El resto del artículo se estructura de la siguiente forma. En la sección 2 se estudian las características fundamentales del lenguaje WS-BPEL 2.0. En la sección 3 se introducen conceptos fundamentales del análisis de mutaciones y se revisan algunos trabajos que tratan sobre la definición de operadores de mutación para diversos lenguajes. La sección 4 muestra la clasificación de los operadores propuestos y su definición. Por último, la sección 5 presenta las conclusiones y el trabajo futuro.

2 El Lenguaje WS-BPEL

WS-BPEL es un lenguaje basado en XML que permite especificar el comportamiento de un proceso de negocio basado en interacciones con WS. La estructura de un proceso WS-BPEL se divide en cuatro secciones: definición de relaciones con los socios externos, que son el cliente que utiliza el proceso de negocio y los WS a los que llama el proceso, definición de las variables que emplea el proceso, definición de los distintos tipos de manejadores que puede utilizar el proceso: manejadores de fallos y de eventos, y descripción del comportamiento del proceso de negocio; esto se logra a través de las actividades que proporciona el lenguaje.

Todos los elementos definidos anteriormente son globales si se declaran dentro del proceso. También existe la posibilidad de declararlos de forma local mediante el contenedor `scope`, que permite dividir el proceso de negocio en diferentes ámbitos.

Los principales elementos constructivos de un proceso WS-BPEL son las actividades, que pueden ser de dos tipos: básicas y estructuradas. Las básicas son las que realizan una determinada labor, mientras que las estructuradas pueden contener otras actividades y definen la lógica de negocio. A las actividades pueden asociarse un conjunto de atributos y de contenedores. Estos últimos pueden incluir diferentes elementos, que a su vez pueden tener atributos asociados.

Además, WS-BPEL permite realizar acciones en paralelo y de forma sincronizada. Por ejemplo, la actividad `flow` permite ejecutar un conjunto de actividades concurrentemente especificando las condiciones de sincronización entre ellas.

3 Antecedentes

El análisis de mutaciones es el proceso de medir la calidad de conjuntos de casos de prueba. Para ello genera un gran número de programas, denominados *mutantes*, que contienen una única diferencia con respecto al programa original. Los mutantes se generan aplicando al código fuente un conjunto de reglas definidas previamente, los *operadores de mutación*, que introducen pequeños cambios sintácticos basados en los errores que suelen cometer habitualmente los programadores, o bien pretenden forzar ciertos criterios de cobertura del código. Estos operadores introducen cambios en el programa a probar manteniendo su validez sintáctica.

Una vez generados, los mutantes se ejecutan sobre los casos de prueba; si la salida que produce el mutante es diferente de la que produce el programa original sobre un determinado caso de prueba, se dice que el mutante está muerto. En ocasiones, aparecen mutantes que siempre provocan la misma salida que el programa original, por lo que no va a existir ningún caso de prueba que permita matarlos; éstos se denominan *mutantes equivalentes*. Para medir la calidad de un conjunto de casos de prueba debemos eliminar los mutantes equivalentes, ya que ésta se va a calcular mediante la *puntuación de mutación* (*mutation score*), el cociente entre el número de mutantes muertos y el número de mutantes no equivalentes.

Se han publicado diversos trabajos que tratan sobre la definición de operadores de mutación para un determinado lenguaje de programación. En concreto, Agrawal y col. [1] definen un conjunto de 77 operadores de mutación para el lenguaje C, King y Offutt [7] definen un conjunto de 22 operadores de mutación para Fortran y los integran en la herramienta de análisis de mutaciones Mothra, Offutt y col. [12] definen un total de 65 operadores de mutación para Ada.

El lenguaje Java también ha sido objeto de estudio para definir sus operadores de mutación en diversos trabajos. Alexander y col. [2] definen un conjunto de operadores para introducir mutaciones en los objetos de Java; estos operadores son utilizados por la herramienta OME para generar mutantes. Ma y col. [8] definen 26 operadores de mutación para el lenguaje Java, estos operadores han sido

integrados en la herramienta MuJava [9]. Bradbury y col. [3] definen un conjunto de operadores específicos para el comportamiento concurrente de Java.

Tuya y col. [14] han definido un conjunto de operadores de mutación para SQL que ha sido integrado en una herramienta para la generación de mutantes.

Offutt y col. [13] han aplicado un nuevo enfoque denominado *perturbación de datos* a la prueba de WS. Este nuevo enfoque está relacionado con el análisis de mutaciones, pero en vez de mutar programas modifica valores, definiéndose nuevos operadores de mutación para la modificación de estos valores.

4 Operadores de Mutación

Esta sección describe los operadores de mutación definidos para WS-BPEL 2.0. Los operadores propuestos se han clasificado en cuatro categorías, de acuerdo con el tipo de elemento sintáctico de WS-BPEL con el que se relacionan. Las categorías se identifican con una letra mayúscula y son las siguientes: mutación de identificadores (I), mutación de expresiones (E), mutación de actividades (A) y mutación de condiciones excepcionales y eventos (X).

Dentro de cada categoría se definen varios operadores de mutación que se identifican mediante tres letras mayúsculas: la primera de ellas coincide con la que identifica la categoría a la que pertenece el operador, mientras que las dos últimas identifican al operador dentro de la categoría.

Los operadores definidos modelan fallos que podrían cometerse al generar una composición WS-BPEL 2.0, habiéndose tenido en cuenta que normalmente éstas no se suelen escribir de forma directa, sino que se utilizan herramientas gráficas. Por lo que muchos de los fallos que pueden aparecer en programas escritos en otros lenguajes, debidos a errores al teclear código, no aparecerán en WS-BPEL. Por esta razón, por ejemplo, no hemos definido un operador para insertar el operador – unario delante de una expresión, mientras que para otros lenguajes si se considera [1, 7, 12, 13].

Algunos de los operadores definidos son específicos del lenguaje WS-BPEL, mientras que otros han sido adaptados de otros lenguajes, se trata de los operadores: ISV, EAA, EEU, ERR, ELL, ECN, AIE, AWR y ASI.

Operadores de Mutación de Identificadores. Un error común que puede cometerse al escribir un programa es cambiar el identificador de una variable por el de otra; si ambas son del mismo tipo, esto no provocará un fallo que pueda ser detectado por el compilador. El operador **ISV** modela este tipo de fallos, sustituyendo el identificador de una variable por el de otra del mismo tipo; esta sustitución sólo se realizará si ambas pertenecen al mismo ámbito.

Operadores de Mutación de Expresiones. Se han definido operadores para todos los tipos de expresiones que podemos encontrar en WS-BPEL 2.0, pero sólo se han considerado aquellos operadores de mutación que sustituyen un operador por otro del mismo tipo, ya que creemos que éstos son los fallos que pueden cometerse con mayor frecuencia. Así tenemos:

EAA Sustituye un operador aritmético (+, -, *, div, mod) por otro del mismo tipo.

EEU Elimina el operador – unario de cualquier expresión en la que aparezca. No hemos considerado el añadirlo porque creemos que modelaría un fallo poco frecuente.

ERR Sustituye un operador relacional (<, >, <=, >=, =, !=) por otro del mismo tipo.

ELL Sustituye un operador lógico (and, or) por otro del mismo tipo.

ECC Sustituye un operador de camino (/, //) por otro del mismo tipo.

El operador **ECN** tiene como dominio las constantes numéricas que aparecen en el programa. Modela los errores que pueden cometerse al introducir constantes numéricas, modificándolas de varias formas: incrementa o decrementa su valor en una unidad, añade o elimina alguna cifra.

El operador **EMD** modifica una expresión de duración de dos formas: la sustituye por 0, lo que implica que la condición se cumple inmediatamente, y por la mitad del valor inicial; esto permite verificar si el margen de seguridad especificado por la duración es adecuado. Este operador es aplicable a la actividad `wait`, y al elemento `onAlarm` de la actividad `pick` y de los manejadores de eventos. Un operador similar es **EMF**, que modifica una expresión de fecha límite.

Operadores de Mutación de Actividades. Algunos de los operadores de esta categoría modelan el fallo que puede cometerse al elegir una actividad que no es la más adecuada para las acciones que se deben realizar, sustituyendo una actividad por otra. Los demás modelan la elección de un valor incorrecto para los atributos de las actividades, sustituyendo para ello su valor actual por otro valor válido. Estos operadores se han clasificado en dos tipos, los relacionados con la concurrencia y los no concurrentes.

Relacionados con la concurrencia. Las actividades `receive` y `pick` permiten recibir mensajes. Ambas pueden especificar un atributo denominado `createInstance` que, si está activado, creará una nueva instancia del proceso cuando llegue un nuevo mensaje y, si no lo está, hará que el nuevo mensaje sea consumido por una de las instancias que ya existen. El operador **ACI** cambia el atributo `createInstance` de "yes" a "no" sólo en el caso de que el proceso tenga más de una actividad con dicho atributo a "yes", ya que en caso contrario el mutante no sería válido. El cambio contrario no se contempla debido a que las restricciones que impone el estándar sobre la correlación entre el mensaje y la instancia a la que va dirigido, harían también que los mutantes generados no fueran válidos.

La actividad `forEach` permite ejecutar un conjunto de actividades un número fijo de veces. Existen dos variantes de `forEach` que se diferencian en el modo de ejecutar las iteraciones, secuencial o paralelamente. El operador **AFP** cambia el carácter secuencial de una actividad `forEach` por paralelo, modificando el valor del atributo `parallel` de "no" a "yes". No se propone el cambio contrario porque no introduciría un error que se pudiera reflejar en los resultados del proceso, generándose un mutante equivalente.

La actividad `sequence` ejecuta secuencialmente las actividades que contiene, mientras que `flow` las ejecuta en paralelo. El operador **ASF** cambia una actividad `sequence` por `flow`; el cambio contrario no se realiza porque dicha mutación generaría un mutante equivalente.

WS-BPEL proporciona un mecanismo para proteger el acceso concurrente a datos globales, se trata del atributo `isolated` de un `scope`. Cuando su valor es "yes" se protege el acceso a variables compartidas, no permitiendo que las actividades de un `scope` puedan acceder a ellas mientras las de otro, en el que se utilizan las mismas variables, no hayan terminado. El operador **AIS** pone el atributo `isolated` de un `scope` en el que se manipulan variables compartidas a "no". No se plantea el cambio contrario porque produciría un mutante equivalente.

No Concurrentes. La actividad `if` permite definir las actividades a ejecutar en función del valor verdadero o falso de un conjunto de condiciones. Permite especificar opcionalmente uno o varios elementos `elseif` y un elemento `else`. El operador **AIE** elimina un elemento `elseif` o el elemento `else` de una actividad `if`, modelando el olvido de una rama de esta actividad.

Aparte de `forEach`, WS-BPEL proporciona otras dos actividades repetitivas, `repeatUntil` y `while`. El operador **AWR** cambia una actividad `while` por otra `repeatUntil` y viceversa, modelando el error cometido al no elegir el tipo de actividad repetitiva adecuada.

A todas las actividades de WS-BPEL pueden asociarse los contenedores `sources` y `targets`, los cuales contienen elementos `source` y `target`, respectivamente. Estos elementos permiten

sincronizar actividades. Se puede especificar para el contenedor `targets` un elemento denominado `joinCondition`, cuyo valor es una expresión booleana. Cuando no se especifica este elemento, la condición que se considera es la disyunción del estado de todos los `target` de esta actividad. El operador **AJC** elimina el elemento `joinCondition`. No se considera el cambio contrario porque creemos que es un error poco probable.

El operador **ASI** intercambia el orden de dos actividades dentro de una actividad `sequence`. Si no existe dependencia de datos entre ellas se producirán mutantes equivalentes.

La actividad de recepción de mensajes `pick` puede especificar mediante los elementos `onMessage` las actividades a realizar cuando el proceso recibe un mensaje determinado. El elemento `onAlarm` indica las acciones a ejecutar si no se recibe ningún mensaje en un tiempo determinado. El operador **APM** elimina un elemento `onMessage`, siempre que haya más de uno, modelando el error que se cometería al olvidar la posible recepción de un mensaje. El operador **APA** elimina el elemento `onAlarm`, modelando el error que se produciría al olvidarlo. También se puede aplicar al elemento `onAlarm` de un manejador de eventos.

Operadores de Mutación Relacionados con las Condiciones Excepcionales y Eventos. Los manejadores de fallos permiten especificar mediante los elementos `catch` las actividades a ejecutar en caso de que se produzca un fallo determinado, y mediante el elemento `catchall` las relacionadas con cualquier otro fallo no especificado anteriormente. El operador **XMF** elimina un elemento `catch` o el elemento `catchall` de un manejador de fallos; modelando el olvido de incluir un manejador específico para un fallo determinado, o bien el de un manejador por omisión.

La actividad `reply` permite enviar mensajes de respuesta, o bien mensajes de fallo, que se especifican mediante el atributo `faultName`. El operador **XRF** elimina este atributo.

A veces, si se produce un fallo durante la ejecución de un proceso, es necesario deshacer el trabajo que ya ha sido hecho; para ello se dispone del manejador de compensación. Por otro lado, dentro de un `scope` se puede definir un manejador de terminación, que indica las actividades a realizar si se debe terminar la ejecución de dicho `scope`. El operador **XMC** elimina la definición de un manejador de compensación, y el operador **XMT**, la de un manejador de terminación. En ambos casos, WS-BPEL los sustituirá por el manejador por omisión de cada uno de ellos.

La actividad `throw` permite lanzar un fallo determinado, cuyo nombre se especifica mediante el atributo `faultName`. El operador **XTF** cambia el nombre del fallo lanzado por una actividad `throw` por otro del mismo ámbito, modelando la confusión en la especificación del fallo a lanzar. La actividad `rethrow` permite volver a lanzar un fallo previamente capturado. El operador **XER** elimina una actividad `rethrow`, modelando el olvido de su inclusión en el manejador de fallos.

Los manejadores de eventos pueden contener elementos `onEvent`, que determinan las acciones a realizar cuando se produce un evento dado, y un elemento `onAlarm`. El operador **XEE** elimina un elemento `onEvent` de un manejador de eventos, modelando el olvido a la hora de especificar un evento que puede recibir el proceso.

4 Conclusiones y Trabajo Futuro

Hemos definido un conjunto de 26 operadores de mutación que abarca múltiples características del lenguaje WS-BPEL 2.0. Es la primera vez que se define un conjunto de operadores de mutación para este lenguaje. Los operadores definidos modelan los fallos que podrían cometerse al generar una composición de servicios.

Dado que nuestro siguiente objetivo es aplicar estos operadores a la generación de mutantes, a la hora de definirlos se ha puesto especial atención en eliminar aquéllos que, de manera obvia, pueden provocar la aparición de mutantes equivalentes.

Los operadores definidos se han clasificado en cuatro categorías atendiendo al tipo de elemento sintáctico de WS-BPEL al que afectan. Así, las categorías consideradas han sido: mutación de identificadores, mutación de expresiones, mutación de actividades y mutación de condiciones excepcionales y eventos. Los operadores de mutación de actividades han sido, a su vez, clasificados en dos tipos: los relacionados con la concurrencia y los no concurrentes. Nuestra intención en un futuro próximo es completar el conjunto de operadores propuestos con otros que tengan en cuenta algunos criterios de cobertura. Actualmente se está desarrollando un sistema de análisis de mutaciones para WS-BPEL en el que van a ser integrados los operadores definidos en este trabajo, esto nos permitirá validar los operadores propuestos, es decir, evaluar si son realmente útiles o no. Asimismo, este sistema nos permitirá estudiar la calidad de conjuntos de casos de pruebas para composiciones de servicios en WS-BPEL.

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. Agrawal, H., Demillo, R., Hathaway, R., Hsu, W., Hsu, W., Krauser, E. Martin, R.J., Mathur, A., Spafford, E. Design of Mutant Operators for the C Programming Language. Technical Report SERC-TR-41-P, Software Engineering Research Center, Purdue University, Indiana (1989)
2. Alexander, R., Bieman, J., Ghosh, S., Ji, B. Mutation of Java Objects. Proceedings of ISSRE'02, pp. 341-351, IEEE CS (2002)
3. Bradbury, J.S., Cordy, J.R., Dintel, J. Mutation Operators for Concurrent Java (J2SE 5.0). Proceedings of MUTATION'06, IEEE CS (2006)
4. Dong, W.L, Yu, H., Zhang, Y.B. Testing BPEL-based Web Service Composition Using High-level Petri Nets. Proceedings of EDOC'06, pp. 441-444, IEEE CS (2006)
5. García-Fanjul, J. Tuya, J., de la Riva, C.: Generación Sistemática de pruebas para composiciones de servicios utilizando criterios de suficiencia basados en transiciones. Libro de Actas de JISBD (2007)
6. IDC: Research Reports <http://www.idc.com> (2008)
7. King, K.N., Offutt, A.J. A Fortran Language System for Mutation-Based Software Testing. *Softw. Pract. Exper.*, vol. 21(7), pp. 685-718 (1991)
8. Ma, Y.S., Kwon, Y.R., Offutt, J. Interclass Mutation Operators for Java. Proceedings of ISSRE'02, pp. 352-363, IEEE CS (2002)
9. Ma, Y.S, Offutt, J., Kwon, Y.R. MuJava: An Automated Class Mutation System. *System. Softw. Test. Verif. Reliab.*, vol. 15(2), pp. 97-133 (2005)
10. OASIS. WS-BPEL 2.0 <http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html> (2007)
11. Offutt, A.J., Untch, R.H. Mutation 2000: Uniting The Orthogonal, Mutation Testing for the New Century, pp. 34-44, Kluwer Academic Publishers, Norwell, MA, USA (2001)
12. Offutt, A.J., Voas, J., Payne, J. Mutation Operators for Ada. Technical Report ISSE-TR-96-09, Information and Software Systems Engineering, George Mason University (1996)
13. Offutt, A.J., Wuzhi X. Generating Test Cases for Web Services Using Data Perturbation. *SIGSOFT Softw. Eng. Notes*, 29(5), pp. 1-10, (2004)
14. Tuya, J., Suárez-Cabal, M.J., de la Riva, C. Mutating Database Queries. *Information and Software Technology*, vol. 49(4), pp. 398-417 (2007)
15. Woodward, M.R. Mutation Testing –its Origin and Evolution. *Information and Software Technology*, vol. 35(3), pp. 163-169 (1993)
16. Yan, J., Li, Z., Yuan, Y., Sun, W. Zhang, J. BPEL4WS Unit Testing: Test Case Generation Using a Concurrent Path Analysis Approach. Proceedings of ISSRE'06, pp. 75-84, IEEE CS (2006)
17. Yuan, Y., Li, Z., Sun, W. A Graph-Search Based Approach to BPEL4WS Test Generation. Proceedings of ICSEA'06, IEEE CS (2006)
18. Zheng, Y., Zhou, J., Krause, P. An Automatic Test Case Generation Framework for Web Services. *Journal of Software*, vol. 2(3), pp. 64-77 (2007)