Propuesta de una arquitectura para la generación de mutantes de orden superior en WS-BPEL

Emma Blanco Muñoz, Antonio García Domínguez, Juan José Domínguez Jiménez, Inmaculada Medina Bulo

Escuela Superior de Ingeniería, Universidad de Cádiz,
C/Chile, 1, CP 11002
emma.blancomu@alum.uca.es
{antonio.garciadominguez,juanjose.dominguez,inmaculada.medina}@uca.es

Resumen La prueba de mutaciones es una técnica que ha sido ampliamente utilizada en diferentes lenguajes de programación, para lo cuál es necesario disponer de un sistema generador de mutantes. Este trabajo presenta una arquitectura para la generación automática de mutantes de orden superior para composiciones WS-BPEL. Los mutantes de orden superior son creados por la aplicación de una secuencia de operadores de mutación de primer orden al programa original. La arquitectura que se presenta está basada en GAmera, un generador de mutantes de primer orden basado en algoritmos genéticos para composiciones WS-BPEL. El trabajo introduce los cambios a realizar en GAmera para transformarlo en un generador de mutantes de orden superior. Además de detallar las modificaciones que deben realizarse, se describen los nuevos operadores genéticos de cruce y mutación que deben incorporarse para tratar con la nueva estructura de los mutantes.

Keywords: Prueba de mutaciones, mutantes de orden superior, generador de mutantes, algoritmo genético, WS-BPEL

1. Introducción

La prueba de mutaciones [2,5] es una técnica de prueba que ha sido ampliamente utilizada en una gran diversidad de lenguajes de programación. Ya existen varios sistemas de generación de mutantes [7], como por ejemplo, MuJava [9] para Java, SQLMutation [11] para SQL, entre otros. De hecho, anteriormente presentamos GAmera [3], un generador de mutantes para composiciones WS-BPEL [10]. GAmera es el primer generador de mutantes basado en un algoritmo genético [4].

Sin embargo, todas estas herramientas sólo generan mutantes de primer orden. Este trabajo presenta los cambios a realizar en GAmera para que pueda generar mutantes de orden superior para composiciones WS-BPEL. Un mutante de orden superior se crea aplicando una secuencia de operadores de mutación de primer orden al programa original [7]. Además de detallar las modificaciones que deben realizarse tanto en el motor de ejecución de mutantes, como en la herramienta de transformación de mutantes a individuos, se describirán los nuevos

operadores genéticos de cruce y mutación que deben incorporarse para tratar con la nueva estructura de los mutantes.

La estructura del artículo es la siguiente: El apartado 2 introduce la prueba de mutaciones, las herramientas generadoras de mutantes y los algoritmos genéticos. Los apartados 3 y 4 describen los componentes, el funcionamiento básico y los cambios que va a experimentar GAmera para adaptarse a la nueva arquitectura. Finalmente, el apartado 5 reúne las conclusiones y líneas futuras de trabajo.

2. Antecedentes

En este apartado se presentan la prueba de mutaciones (2.1), las actuales herramientas generadoras de mutantes (2.2) y los algoritmos genéticos (2.3).

2.1. Prueba de Mutaciones

La prueba de mutaciones [2,5] es una técnica de prueba basada en fallos que consiste en introducir errores simples en el programa original mediante la aplicación de operadores de mutación. Los programas resultantes reciben el nombre de *mutantes*, que contienen pequeños cambios sintácticos con respecto al programa original. Cada operador de mutación se corresponde con una categoría de error típico que puede cometer el programador.

Si un caso de prueba es capaz de distinguir al programa original del mutante, es decir, la salida del mutante y la del programa original son diferentes, se dice que mata al mutante. Por el contrario, se dice que un mutante es vivo para el conjunto de casos de prueba original, si ninguno de los casos de prueba es capaz de diferenciar al mutante del programa original, es decir, la salida del mutante y del programa original es la misma.

El concepto de mutación simple fue ampliado por Jia y Harman [7], introduciendo la mutación de orden superior frente a la mutación tradicional, considerada de primer orden. Un mutante de orden superior se crea aplicando una secuencia de operadores de mutación de primer orden al programa original.

2.2. Herramientas generadoras de mutantes

Actualmente existen varios sistemas de generación de mutantes tales como Mothra [8] para FORTRAN, MuJava [9] para Java, Proteum [1] para C, SQL-Mutation [11] para consultas de bases de datos escritas en SQL, MiLu [6] para C y GAmera [3] para composiciones WS-BPEL.

Destacar que únicamente MiLu emplea mutación de orden superior pero para el lenguaje C. La herramienta GAmera sólo maneja mutantes de primer orden y lo que proponemos en este artículo consiste en adaptarlo para que genere mutantes de orden superior y así reducir el coste de las pruebas de mutación.

2.3. Algoritmos genéticos

Los algoritmos genéticos (AG) [4] son técnicas de búsqueda probabilísticas basadas en la teoría de la evolución y la selección natural. Trabajan con una población de soluciones, denominadas *individuos*. A través de diferentes generaciones, los AG realizan un proceso de selección para mejorar la población, y generan nuevos a través de la recombinación y mutación de los existentes.

Cada individuo lleva asociada una *aptitud* que mide la calidad de la solución a la que representa. El AG va produciendo distintas generaciones de individuos a través de las cuales va maximizando la aptitud media de la población. El esquema de codificación que se utiliza y el cálculo de la aptitud de los individuos van a depender en gran medida del problema que queremos resolver.

Los AG usan dos tipos de operadores: de selección y de reproducción. Los *operadores de selección* seleccionan los individuos de la población que se emplearán para la reproducción. La probabilidad de seleccionar a un individuo puede ser proporcional o no a su aptitud. Los *operadores de reproducción* se encargan de generar nuevos individuos en la población y existen dos tipos distintos. Por un lado, los operadores de cruce generan dos nuevos individuos, denominados hijos, de dos individuos preseleccionados o padres. Los hijos heredan parte de la información almacenada en ambos padres. Por otro lado, los operadores de mutación permiten alterar la información almacenada en un individuo. El diseño de estos operadores es altamente dependiente del esquema de codificación empleado. En este punto es importante hacer notar que los operadores de mutación de los AG son diferentes de los empleados en la prueba de mutaciones.

3. GAmera: Generador de mutantes para WS-BPEL

GAmera está constituido por tres componentes principales, éstos se muestran en la figura 1: el analizador (recibe como entrada la composición WS-BPEL a probar y determina los operadores de mutación que se pueden aplicar), el generador de mutantes (genera un subconjunto de mutantes seleccionados por el algoritmo genético a partir de la información recibida) y el sistema de ejecución (se encargará de ejecutar los mutantes generados y comparar sus salidas con las del proceso original). En [3] encontramos una descripción más detallada.

4. Adaptación a Mutantes de Orden Superior

Para la adaptación de GAmera a mutaciones de orden superior, será necesario establecer un orden máximo de mutación, N. De este modo, se pretende que la herramienta genere mutantes de orden 1 hasta un orden máximo N establecido previamente.

4.1. Extendiendo la codificación del mutante

Tal y como se describe en [3], cada individuo del AG representa a un mutante. Para la realización de mutaciones de orden superior, cada individuo codificará la secuencia de mutaciones que se realizará al programa original (figura 2).

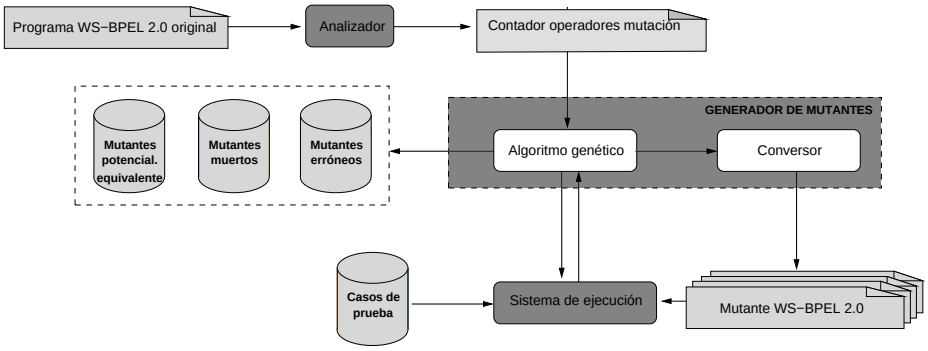

Figura 1. GAmera: sistema generador automático de mutantes para WS-BPEL

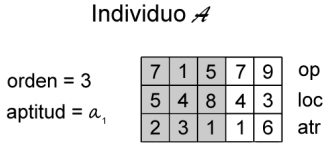

Figura 2. Representación gráfica de un individuo

De este modo, cada individuo tendrá hasta un máximo de N cambios, siendo N el orden máximo que ha sido establecido como parámetro de entrada. Cada uno de esos cambios sigue el mismo esquema de codificación original, es decir, se emplean tres campos para identificar al operador, la localidad y el atributo necesario para aplicar dicho operador. De esta forma, cada individuo llevará asociado un orden de mutación que estará comprendido entre 1 y el máximo orden de mutación N dado como parámetro de entrada.

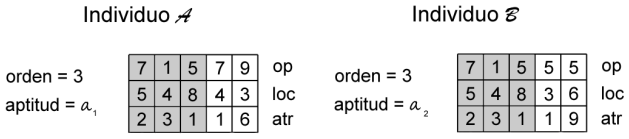

Figura 3. Individuos iguales

Hemos de tener en cuenta que esto implicará tener en la misma población individuos que representan a mutaciones de distintos órdenes. Esta codificación plantea la posibilidad de tener individuos genéticamente diferentes, pero que representan al mismo mutante, como se puede ver en la figura 3.

4.2. Nuevos Operadores Genéticos

La primera población se genera aleatoriamente. En cambio, las siguientes generaciones se basan en un AG generacional donde los individuos se crean mediante las operaciones de cruce y mutación. Estas operaciones se harán atendiendo a las probabilidades especificadas en la configuración, p_c y $p_m = 1 - p_c$.

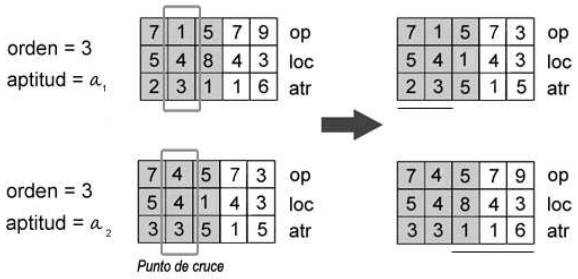

Figura 4. Operador de cruce de orden

El *operador de cruce* consiste en un intercambio de ciertos campos de los individuos. Hay dos tipos de operadores de cruce: el *operador de cruce de orden*, donde se elige aleatoriamente un punto de cruce, entre 1 y el orden del individuo. En la Figura 4 podemos ver cómo dos individuos padre generan a sus hijos a partir del punto de cruce; y el *operador de cruce individual*, donde también se elige un punto de cruce aleatorio y además, se selecciona aleatoriamente el campo a intercambiar entre el operador, la localidad y el atributo. En la Figura 5 se muestra un ejemplo. Las probabilidades de estos operadores se fijan en la configuración: p_{co} y p_{ci}, respectivamente, de modo que $p_c = p_{co} + p_{ci}$.

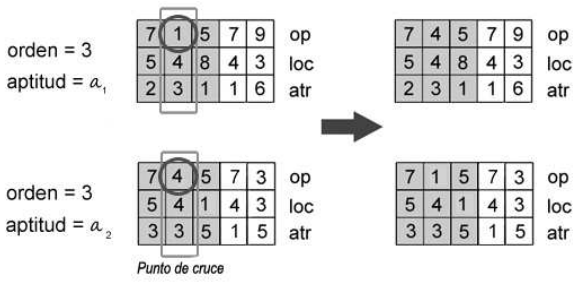

Figura 5. Operador de cruce individual

El *operador de mutación* modifica el valor de un campo de un individuo. Hay dos tipos: el *operador de mutación de orden*, donde el parámetro que será

modificado es el orden del individuo y vendrá dado por $\text{Orden}(I) = o_{actual} + rand(-1, 1) \cdot (1 - p_{om})$; y el *operador de mutación individual*, donde se elige aleatoriamente un punto de mutación y además, se selecciona aleatoriamente el campo a modificar entre el operador, la localidad y el atributo. El nuevo valor de dicho campo vendrá dado por $\text{Valor}(I) = v_{actual} + rand(-1, 1) \cdot (1 - p_{im})$. Las probabilidades de estos operadores las define el usuario: p_{mo} y p_{mi}, respectivamente, de modo que $p_m = p_{mo} + p_{mi}$.

5. Conclusiones y trabajo futuro

Hemos presentado un acercamiento hacia la implementación de un sistema automático de generación de mutantes de orden superior para composiciones WS-BPEL. Es una mejora de nuestra herramienta anterior, GAmera, que sólo maneja mutantes de primer orden. También se han descrito los cambios necesarios para adaptar los operadores de cruce y de mutación a la nueva estructura de los mutantes. Una ventaja que presenta este enfoque es que mutantes de diferentes órdenes puede estar en la misma generación.

La novedad más importante presentada es el diseño de la primera arquitectura que genere mutantes de orden superior para composiciones WS-BPEL.

Nuestra futura línea de trabajo es la implementación de este framework en Java y el diseño de una interfaz gráfica de usuario.

Referencias

1. Delamaro, M., Maldonado, J.: Proteum–a tool for the assessment of test adequacy for C programs. In: Conf. on Performability in Computing System. pp. 79–95 (1996)
2. DeMillo, R.A., Lipton, R.J., Sayward, F.G.: Hints on test data selection: Help for the practicing programmer. Computer 11(4), 34–41 (1978)
3. Domínguez-Jiménez, J.J., Estero-Botaro, A., García-Domínguez, A., Medina-Bulo, I.: GAmera: an automatic mutant generation system for WS-BPEL compositions. In: ECOWS'09: Seventh IEEE European Conference on Web Services (2009)
4. Goldberg, D.E.: Genetic algorithms in search, optimization and machine learning. Addison-Wesley, Reading (1989)
5. Hamlet, R.G.: Testing programs with the aid of a compiler. IEEE Transactions Software Engineering 3(4), 279–290 (1977)
6. Jia, Y., Harman, M.: MILU: A customizable, runtime-optimized higher order mutation testing tool for the full C language. In: TAIC-PART '08: Testing: Academic & Industrial Conference - Practice and Research Techniques. pp. 94–98. IEEE (2008)
7. Jia, Y., Harman, M.: An analysis and survey of the development of mutation testing. IEEE Transactions on Software Engineering 99(PrePrints) (2010)
8. King, K.N., Offutt, A.J.: A FORTRAN language system for mutation-based software testing. Software - Practice and Experience 21(7), 685–718 (1991)
9. Ma, Y.S., Offutt, J., Kwon, Y.R.: MuJava: an automated class mutation system. Software Testing, Verification & Reliability 15(2), 97–133 (2005)
10. OASIS: Web Services Business Process Execution Language 2.0. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html (2007)
11. Tuya, J., Cabal, M.J.S., de la Riva, C.: Mutating database queries. Information and Software Technology 49(4), 398–417 (2007)