TOWARDS HIGHER-ORDER MUTANT GENERATION FOR WS-BPEL

E. Blanco-Muñoz, A. García-Domínguez, J.J. Domínguez-Jiménez, I. Medina-Bulo
University of Cádiz, C/ Chile, 1, Cádiz, Spain
{emma.blancomu,antonio.garciadominguez,juanjose.dominguez,inmaculada.medina}@uca.es

Keywords: Mutation Testing, Genetic Algorithm, Web Services, Web Service compositions, WS-BPEL

Abstract: We present an architecture for automatically generating higher-order mutants for WS-BPEL compositions based on the architecture of GAmera, a first-order mutant generation system for WS-BPEL. Higher-order mutants are created by applying a sequence of first-order mutation operators to the original program. This paper also introduces the changes that GAmera has to undergo for converting the generation of first-order mutants into a process capable of higher-order mutation, while detailing the modifications carried out for adapting the crossover and mutation genetic operators to the new structure of the mutants.

1 INTRODUCTION

The Web Services Business Process Execution Language (WS-BPEL) (OASIS, 2007) allows us to develop new Web Services (WS) modelling more complex business processes on top of preexisting WS. The economic impact of WS-BPEL service compositions is quickly increasing (IDC, 2008), and deeper insight on how to test them effectively is therefore required.

Mutation testing (DeMillo et al., 1978; Hamlet, 1977) is a testing technique that has been applied successfully to several programming languages. Several mutant generation systems already exist (Jia and Harman, 2010), such as Mothra (King and Offutt, 1991) for FORTRAN, MuJava (Ma et al., 2005) for Java or SQLMutation (Tuya et al., 2007) for SQL, among others. In fact, we have presented GAmera (Domínguez-Jiménez et al., 2009) in previous works, an automatic mutant generation system for WS-BPEL compositions, which only dealt with first-order mutants.

However, all these tools only generate first-order mutants. In this work we present the modifications of the architecture of GAmera for automatically generating higher-order mutants for WS-BPEL compositions. A higher-order mutant is created by applying a sequence of first-order mutation operators to the original program (Jia and Harman, 2009). This paper introduces the changes that GAmera has to undergo for converting the generation of first-order mutants into a process capable of higher-order mutation, while detailing the modifications carried out for adapting the crossover and mutation genetic operators to the new structure of the mutants.

The paper is divided into the following sections: Section 2 briefly summarizes the mutation testing and genetic algorithms. Section 3 describes the modifications to the architecture of GAmera, as well as the new genetic operators defined. Finally, Section 4 presents the conclusions and future work.

2 BACKGROUND

We will first introduce mutation testing and some basic concepts about genetic algorithms.

2.1 Mutation Testing

Mutation testing (DeMillo et al., 1978; Hamlet, 1977) is a fault-based testing technique that introduces simple flaws in the original program by applying mutation operators. The resulting programs are called mutants. Each mutation operator models a category of
errors that the developer could make. For instance, if a program contains the instruction \(a > 5000\) and we apply the relational mutation operator (which replaces a relational operator with another), one of the mutants produced will contain \(a < 5000\) instead. If a test case can tell apart the original program and the mutant, i.e., their outputs are shown to be different, it is said that this test case kills the mutant. Otherwise, the mutant is said to stay alive.

Equivalent mutants, which always produce the same output as the original program, are a common problem when applying mutation testing. Equivalent mutants should not be confused with stubborn non-equivalent mutants, which are produced because the test suite is not adequate to detect them.

2.2 Genetic Algorithms

Genetic Algorithms (Goldberg, 1989) (GAs) are probabilistic search techniques based on the theory of evolution and natural selection.

GAs work with a population of solutions, known as individuals, and process them in parallel. Throughout the successive generations of the population, GAs perform a selection process to improve the population, and so they are ideal for optimization. In this sense, GAs favor the best individuals and generate new ones through the recombination and mutation of information from existing ones. The strengths of GAs are their flexibility, simplicity and ability for hybridization. Among their weaknesses are their heuristic nature and their difficulties in handling restrictions.

The fitness of an individual measures its quality as a solution for the problem to be solved. The average fitness of the population will be maximized along the generations produced by the GA.

GAs use two types of genetic operators: selection and reproduction. Selection operators select individuals in a population for reproduction. The likelihood of selecting an individual may be proportional to its fitness. Reproduction operators generate the new individuals in the population by applying crossover and mutation operators. On the one hand, crossover operators generate two individuals (children) from two pre-selected individuals (parents). The children inherit part of the information stored in both parents. On the other hand, mutation operators aim to alter the information stored in an individual. The design of these operators heavily depends on the encoding scheme used. It is important to note that these mutation operators are related to the GA and are different from those for mutation testing.

3 SYSTEM ARCHITECTURE

This section describes how GAmera has changed from its previous architecture, the new genetic operators which have been adapted to the current framework and lastly, the novel genetic algorithm.

The components of GAmera are still the same, although the genetic search for mutants has been modified. Figure 1 shows the core of the system: the analyzer, the mutant generator and the execution system. The analyzer takes the original program WS-BPEL process definition and produces the information required by the mutant generator. The mutant generator is divided into the GA and a converter from the individuals of the GA to the mutants of the original process definition. Finally, the execution system runs the generated mutants against the test cases and compares their outputs with those from the original process definition.

3.1 Higher-Order Structure

Figure 2 shows the new structure for generating higher-order mutants (HOM).

3.1.1 Extending the Representation

The concept of individual has been changed. Individuals now encode \(N\) mutations on the original program, where \(N\) is the maximum order defined in the configuration. Individuals have five fields (Figure 3): the order of the individual, its fitness, the identifiers of the mutation operators to be applied, the identifiers of the locations to be mutated and the additional values which modify the behaviour of the mutation operator.

![Figure 3: Representation of an individual](image)
The three lists with the mutation operators, the locations and the additional values must have the same number of elements.

In addition, we have to redefine when two individuals are equal. Two individuals are equal if they have the same order \((n)\) and the first \(n\) operator-location-attribute triplets are equal.

The population contains a set of individuals. There might be equal individuals in the same population.

### 3.1.2 New Genetic Operators

The first population is randomly generated. The following generations are based on a generational GA, where individuals are created with crossover and mutation operations. Crossovers and mutations will be done according to the probabilities specified in the configuration, \(p_c\) and \(p_m = 1 - p_c\), respectively. These operations have undergone several changes that are detailed below.

The crossover operator consists in an exchange of certain fields of the individuals involved. There are two types of crossover operators: the order and the individual crossover operator. Their probabilities are set the configuration: \(p_{oc}\) and \(p_{ic}\), respectively, so \(p_c = p_{oc} + p_{ic}\).

In the order crossover operator, a crossover point is chosen at random, between 1 to the order of the individual. Figure 5 shows how two parents generate their children, according to the crossover point selected.

On the other hand, in the individual crossover operator, a crossover point is also chosen in the same way, but now the field which will be changed (operator, location or attribute) is randomly selected. Therefore, in this case only a value of each individual is modified. Figure 6 has an example of applying an individual crossover operator.

The mutation operator changes the value of a field
of the individual. There are two types of mutation operators: the order and the individual mutation operator. Their probabilities are set by the user: \( p_{om} \) and \( p_{im} \), respectively, so \( p_m = p_{om} + p_{im} \).

In the order mutation operator, the parameter which will be modified is the individual order and it will be given by:

\[
\text{Order}(I) = o_{\text{current}} + \text{rand}(-1,1) \cdot (1 - p_{om})
\]  

(1)

**Figure 7:** Order mutation operator

In the individual mutation operator, a mutation point is randomly chosen and the field which will be mutated (operator, location or attribute) is selected randomly. The value which will be changed is given by:

\[
\text{Value}(I) = v_{\text{current}} + \text{rand}(-1,1) \cdot (1 - p_{im})
\]  

(2)

**Figure 8:** Individual mutation operator

4 CONCLUSIONS

This paper presents an approach towards implementing an automatic higher-order mutant generation system for WS-BPEL compositions. It improves upon our previous tool, called GAmera, which only dealt with first-order mutants. We have also described the changes needed to adapt the crossover and mutation operators to the new structure of the mutants.

The most important novelty of this architecture is that has been designed the first framework which generate higher-order mutants as against other tools. This is an interesting feature because...

Our future lines of work are the implementation of this framework, as well as creating new useful genetic operators and an user-friendly graphical interface.

**REFERENCES**


