

# A Framework for Mutant Genetic Generation for WS-BPEL

Juan José Domínguez Jiménez    Antonia Estero Botaro  
Inmaculada Medina Bulo

Grupo de Investigación SPI&FM  
Dpto. Lenguaje y Sistemas Informáticos  
Universidad de Cádiz



SOFSEM 2009

# Outline

- 1 Introduction
  - Web Services
  - Mutation testing
  - Genetic Algorithm
- 2 Related Works
- 3 Framework for Mutant Genetic Generation for WS-BPEL
  - Objective
  - Mutation Operators for WS-BPEL 2.0
  - Mutant generation with GA
- 4 First results
- 5 Contributions and future work

# Web Services and WS-BPEL

- One of the latest trends is marked by the emergence of so-called web services (WS).
- These allow rapid application development, characterized by a low cost and an easy distributed application composing.
- The OASIS WS-BPEL 2.0 standard is an XML-based language which allows to specify the behavior of a business process as a WS which interacts with other external WS.
- WS-BPEL is a challenge for traditional white-box testing techniques because it includes WS-specific instructions not common in other languages.

# Mutation testing

## Definition

- Mutation testing is a white box testing technique that introduces simple flaws in the original program.
- The flaws are introduced applying **mutation operators**.
- The resulting programs are called **mutants**.

## Example

*Original*

```
if (a > 5000) ...
```

*Mutant*

```
if (a < 5000) ...
```

# Mutation testing

## Steps in mutation testing

- 1 The mutant is executed on the test suite.
- 2 The mutant output and the original program output are compared.
- 3 If the two output are different for a test case, the mutant is killed.
- 4 If the two output are equal for all test case, the mutant keeps alive.

# Mutation testing

## Classification of Mutants

**Kill** A test case is able to distinguish between the original program and the mutant.

**Alive** if no test case is able to distinguish between the mutant and the original program.

**Equivalent** the mutant output and the original program output is always the same.

**Stubborn non-equivalent** the test case set is not sufficient to detect them.

# Mutation testing

## Problems

- The detection of equivalent mutant. Is the test suite sufficient? Has the test suite sufficient quality?
- The high computational cost involved for the large number of mutants.
  - **Selective mutation** Few mutations are performed without incurring a loss of information. Therefore, it is interesting the application of **optimization techniques**

# Mutation testing

## Problems

- The detection of equivalent mutant. Is the test suite sufficient? Has the test suite sufficient quality?
- The high computational cost involved for the large number of mutants.
  - **Selective mutation** Few mutations are performed without incurring a loss of information. Therefore, it is interesting the application of **optimization techniques** → Genetic Algorithms

# Genetic Algorithm

- GAs work with a population of solutions, known as **individuals**, and the set as **population**.
- Each individual will have a **fitness** that represents the quality of the solution with respect to the problem to solve.
- GAs are an iterative process. Each iteration is named **generation**.
- GAs use two types of operators: selection and reproduction.
- **Selection operators** select individuals in a population for reproduction. This selection can be proportionate or not to fitness.
- **Reproduction operators** generate the new individuals in the population: crossover and mutation.

# Related Works

## Mutation operators

- Mutation operators have been defined for several programming languages: C, Fortran, Ada, Java, SQL, . . . , but there are not for the WS-BPEL language.

## Tools for automatic mutant generation

- Mothra (Fortran), MuJava (Java), Proteum (C), SQLMutation (SQL)
- These tools generate all the possible mutants for the mutation operators supplied.

## Genetic algorithms in testing

- GAs have been limited to test case generation.
- Adamopoulos et al. using a GA to find the optimal mutant set. In this system, the individuals of the GA are mutant sets previously generated by Mothra, unlike our work, where mutants are generated

# Objective

## Our work consist in

- To define a set of mutation operators for the WS-BPEL 2.0 language which models typical errors that programmers make when developing a business process in this language.
- To design a framework with GA that
  - Generates mutants automatically,
  - Minimizes the mutants generated,
  - Detects the equivalent mutants.

# Mutation operators for WS-BPEL 2.0

- A set of 26 operators has been defined. These operators have been classified in four categories:
  - 1 Identifier replacement operators (I)
  - 2 Expression operators (E)
  - 3 Activity operators (A)
  - 4 Exception and event operators (X)

## Naming conventions

X	Y	Z
---	---	---

X: the category

YZ: the operator

# Mutation operators for WS-BPEL 2.0

## Identifier replacement operators

OPER.	DESCRIPTION
ISV	Replaces a variable identifier by another of the same type

## Expression operators

OPER.	DESCRIPTION
EAA	Replaces an arithmetic operator by another of the same type
EEU	Removes the unary minus operator from an expression
ERR	Replaces a relational operator by another of the same type
ELL	Replaces a logical operator by another of the same type
☆ ECC	Replaces a path operator by another of the same type
ECN	Modifies a numerical constant
☆ EMD	Modifies a duration expression
☆ EMF	Modifies a deadline expression

# Mutation operators for WS-BPEL: Activity operators

## Activity operators related to concurrency

OPER.	DESCRIPTION
-------	-------------

- |       |   |
|-------|---|
| ☆ ACI | Changes the <code>createInstance</code> attribute from an inbound message activity to <code>no</code> |
| ☆ AFP | Replaces a sequential <code>forEach</code> activity by a parallel one                                 |
| ☆ ASF | Replaces a <code>sequence</code> activity by a <code>flow</code> activity                             |
| ☆ AIS | Changes the <code>isolated</code> attribute of a <code>scope</code> to <code>no</code>                |

## Non-concurrent activity operators

OPER.	DESCRIPTION
-------	-------------

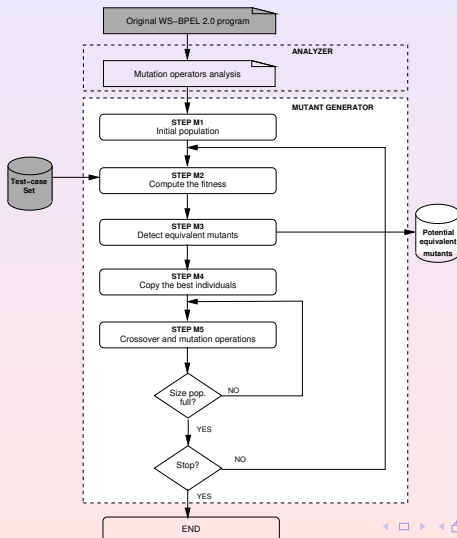
- |       |  |
|-------|--|
| AIE   | Deletes an <code>elseif</code> element or the <code>else</code> element from an <code>if</code> activity |
| AWR   | Replaces a <code>while</code> activity by <code>repeatUntil</code> and vice versa                        |
| ☆ AJC | Removes the <code>joinCondition</code> attribute from an activity  |
| ☆ ASI | Exchanges the order of two <code>sequence</code> child activities  |
| ☆ APM | Removes an <code>onMessage</code> element from a <code>pick</code> activity                              |
| ☆ APA | Removes the <code>onAlarm</code> element from a <code>pick</code> activity or from an event handler      |

# Mutation operators for WS-BPEL: Exception and event

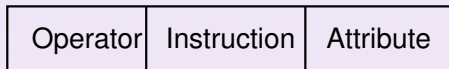
## Exception and event operators

OPER.	DESCRIPTION
☆ XMF	Removes a <code>catch</code> element or the <code>catchAll</code> element from a fault handler
☆ XRF	Removes the <code>faultName</code> attribute from a <code>reply</code> activity
☆ XMC	Removes the definition a a compensation handler
☆ XMT	Removes the definition a a termination handler
XTF	Replaces the fault thrown by a <code>throw</code> activity
☆ XER	Removes a <code>rethrow</code> activity
☆ XEE	Removes an <code>onEvent</code> element from an event handler

# Framework for Mutant Genetic Generation for WS-BPEL



# Representation of an Individual



## Description

Each individual encodes one mutation on the original program:

**Operator** identifies the mutation operator to be applied.

**Instruction** represents the instruction number of the original program where the operator will be applied.

**Attribute** represents the new value to be taken by the instruction element to be modified by the mutation operator

# Representation of an Individual

## Individual (ERR, 2, 3)

### Original Program

```
<if name="discount">
  <condition>
    $buy > 5000
  </condition>
  <invoke name="10off" ... />
<elseif>
  <condition>
    $buy > 2500
  </condition>
  <invoke name="5off" ... />
</elseif>
<else>
  <reply name="nooff" ... />
</else>
</if>
```

### Mutant

```
<if name="discount">
  <condition>
    $buy > 5000
  </condition>
  <invoke name="10off" ... />
<elseif>
  <condition>
    $buy = 2500
  </condition>
  <invoke name="5off" ... />
</elseif>
<else>
  <reply name="nooff" ... />
</else>
</if>
```

# Mutation operators and Attributes

Operator	Value	Max. Value of Attribute
ISV	1	N
EAA	2	5 +, -, *, div, mod
EEU	3	1
ERR	4	6 <, >, >=, <=, =, !=
ELL	5	2 and, or
ECC	6	2 /, //
ECN	7	4 +1, -1, adding, removing
EMD	8	2 0, half
EMF	9	2 0, half
ACI	10	1
AFP	11	1
ASF	12	1
AIS	13	1
...	...	...

# Fitness of the individuals

- Upon completion of the execution of all mutants on the test cases, we will have a matrix as follows:

$$(m_{ij})_{M \times T} = \begin{pmatrix} 0 & 0 & 1 & \dots & 0 \\ 1 & 0 & 0 & \dots & 0 \\ & \dots & \dots & \dots & \\ 0 & 0 & 0 & \dots & 0 \end{pmatrix}$$

- A mutant  $i$  can be killed by a test case:  $\sum_{j=1}^T m_{ij} = 1$
- A mutant  $i$  can continue to live:  $\sum_{j=1}^T m_{ij} = 0$
- A test case  $j$  can kill any mutant or more mutants:  $\sum_{i=1}^M m_{ij} \in [0, M], \forall j$

# Fitness of the individuals

## Fitness

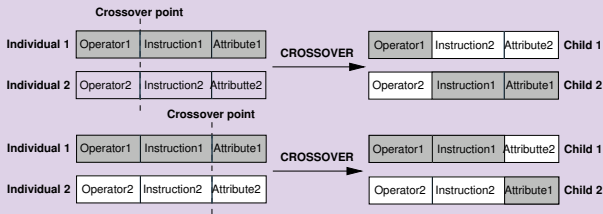
- The individual fitness function will take into account if the mutant is dead or not by test case set, and how many mutants are also killed by the same test case

$$\text{Fitness}(I) = M - \sum_{j=1}^T \left( m_{Ij} \cdot \sum_{i=1}^M m_{ij} \right)$$

- We are trying to penalize those groups of mutants that are killed by the same test case
- The function benefits the mutants alive, named potential equivalent mutants.

# Genetic operators

## Crossover



## Mutation

- It consists in adding a random number to the value of the individual selected:

$$\beta = \alpha + \text{random}(1, L - 1) \pmod{L}$$

# First results

- The system is working and freely available (<http://neptuno.uca.es/~gamera>)
- Tested with the classic Loan-Approval composition.
- Expectations confirmed:
  - Minimize the mutants generated and there is no loss of relevant information.
  - Detect the potential equivalent mutants.

## First results

- We have 22 mutants and 2 potential equivalent mutants with the initial test suite.

$P$	Generations	Equiv.	Killed	Killed (%)
<b>80 %</b>	18	<b>2</b>	16	88.9 %
<b>70 %</b>	10	<b>2</b>	14	87.5 %
<b>60 %</b>	7	<b>2</b>	11	84.6 %
50 %	5	1	10	90.9 %

- These mutants were not equivalent and adding a new test case to the initial test suite, they were killed.

# Contributions and future work

## Contributions

- We have defined a set of twenty-six mutation operators for the BPEL language. These operators model typical errors the programmers can make when developing a service composition.
- The system allows automatic mutant generation for WS-BPEL compositions and shows a novel use of GAs when applying to mutant generation.
- The system is capable of generating a subset of all mutants detecting all potentially equivalent mutants.
- Our results show there is no loss of relevant information when generating until a 60 % of the total number of possible mutants.
- The detection of potentially equivalent mutants allows to improve the quality of test suite.

# Contributions and future work

## Future work

- We will be to carry out more experiments to determine the influence of the GA configuration parameters in results.
- To complete the mutation operator set for WS-BPEL with new ones taking into account coverage criteria.
- Complement the system with a generator of test cases to improve the quality of the initial set with the potential equivalent mutants.

# Thank you for your attention. Questions?

Grupo de Investigación SPI&FM  
Universidad de Cádiz  
Proyecto SOAQSim : TIN2007-67843-C06-04

[juanjose.dominguez@uca.es](mailto:juanjose.dominguez@uca.es)