Test-Suite Automatic Generation through Automatic Seeding for WS-BPEL 2.0

V. Liñeiro A. Estero A. García I. Medina

Universidad de Cádiz

University of Cádiz

First Internacional Summer School on Search-Based Software Engineering
June 29th-July 1st

Table of contents

Motivation & objectives

Research area

Test suite automatic generation for WS-BPEL web service compositions.

Previous work

- An input-based test suite automatic generation technique.
- TESTGENERATOR, a tool that implements the previous technique for WS-BPEL.

Objectives

- Definition of the Automatic Seeding, a mixed test suite generation technique based on the program input and knowledge.
- Definition of an optimization of the previous process in order to reduce the test suite size.

WS-BPEL 2.0

What is WS-BPEL?

XML-based language that allows to specify the behavior of a business process based on web services interactions.

Example

```
<flow> ← Structured activity
 <links>  ← Container
  <link name="checkFlight-bookFlight" ← Attribute/> ← Element
 </links>
 <invoke name="checkFlight" ... > ← Basic Activity
  <sources>  ← Container
   <source linkName="checkFlight-bookFlight" ← Attribute/> ← Element
  </sources>
 </invoke>
 <invoke name="checkHotel" ... />
 <invoke name="checkCarRental" ... />
 <invoke name="bookFlight" ... >
  <targets>   ← Container
   <target linkName="checkFlight-bookFlight" /> ← Element
  </targets>
 </invoke>
</flow>
```

Automatic Seeding (I)

Automatic Seeding process

1. We obtain all the constants inside the program.
2. We read all the input variables of the specification.
3. We generate the mapping between constants and variables (by data type).
4. We generate an initial random test suite.
5. For each test case, we modify the selected variable in order to add the constant value stored in the mapping for it.

Test suite size

Assuming T as basic data types collection and collections V_t (variables) and C_t (constants) $\forall t \in T$, the generated test suite size p is:

$$p = \sum_{t=1}^{|T|} |V_t| \cdot |C_t|$$

Automatic Seeding (II)

Example

```
if(state == "OPEN") {
    while(temperature > 40 && pressure > 400) {
        ...
    }
    state = "CLOSED";
}
```

Test suite						
	1	2	3	4	5	6
state	OPEN	CLOSED	...	...	...	...
temperature	...	...	40	400	...	...
pressure	...	...	...	...	40	400

Optimization (I)

Motivation

Compositions with high number of constants and variables $\longrightarrow$ high test suite size $\longrightarrow$ high computational cost.

Objectives

- Test suite size reduction to the minimum test cases needed.
- We require a relationship between the constants and the variables in order to generate a new test case.
- With this approach, we generate a mapping which discards unnecessary test cases.

Optimization (II)

Example with optimization

```
if(state == "OPEN") {
    while(temperature > 40 && pressure > 400) {
        ...
    }
    state = "CLOSED";
}
```

Test suite						
	1	2	3	4	5	6
state	OPEN	CLOSED	...	...	...	...
temperature	...	...	40	400	...	...
pressure	...	...	...	...	40	400

Optimization (II)

Example with optimization

```
if(state == "OPEN") {
    while(temperature > 40 && pressure > 400) {
        ...
    }
    state = "CLOSED";
}
```

Test suite						
	1	2	3	4	5	6
state	OPEN	CLOSED	...	...	...	...
temperature	...	...	40	~~400~~	...	...
pressure	...	...	...	...	~~40~~	400

Optimization (II)

Example with optimization

```
if(state == "OPEN") {
    while(temperature > 40 && pressure > 400) {
        ...
    }
    state = "CLOSED";
}
```

Test suite				
	1	2	3	4
state	OPEN	CLOSED	...	...
temperature	...	...	40	...
pressure	...	...	...	400

Automatic Seeding in WS-BPEL (I)

1. We obtain all the constants inside the composition

0	$assesorInput.input/ns0:amount $approverInput.input/ns0:amount
false	$processOutput.output/ns0:accept
10000	$processInput.input/ns0:amount
high	$assesorOutput.output/ns0:risk
true	$processOutput.output/ns0:accept

2. We read all the input variables of the specification

req_amount	Int
ap_reply	String
as_reply	String

Automatic Seeding in WS-BPEL (II)

[Optimization] We generate the mapping between specification and composition variables

req_amount	$assesorInput.input/ns0:amount $approverInput.input/ns0:amount $processInput.input/ns0:amount
ap_reply	$approverOutput.output/ns0:accept $processOutput.output/ns0:accept
as_reply	$assesorOutput.output/ns0:risk

3. We generate the mapping between constants and variables (by data type)

ap_reply	String	true, false
req_amount	Int	0, 10000
as_reply	String	high

Automatic Seeding in WS-BPEL (III)

4. We generate an initial random test suite

ap_reply	false	true	true	true	false
req_amount	456	4500	2345	12349	567
as_reply	high	low	low	low	low

5. We modify the test suite

ap_reply	true	false	true	true	false
req_amount	456	4500	0	10000	567
as_reply	high	low	low	low	high

6. We apply the proper format to the test suite generated

```
#set($ap_reply = ["true", "false", "true", "true", "false"])
#set($req_amount = [456, 4500, 0, 10000, 567])
#set($as_reply = ["high", "low", "low", "low", "high"])
```

Conclusions and future work

Conclusions

We have defined:

- A mixed test suite automatic generation technique which benefits from both input and composition information.
- An optimization of the technique that allows to reduce the final size of the generated test suite.

Future work

To perform an experimental study which will allow to quantify the improvement expected, comparing both conventional and optimized techniques with the basic random generation one.

Thanks for your attention!

University of Cádiz

valentin.lineiro@uca.es