A preliminary study on BPEL process testability

Sébastien Salva, Issam Rabhi
Outline

- Introduction
- Testability
- ABPEL to STS?
- Testability Issues
- Testability Propositions & Enhancement
- Conclusion & Perspectives
Introduction

• **Web Services**: independent object instances called by operations

• **BPEL**: an OASIS standard language used for describing interactions in Service Oriented Architectures (SOA)
Testability gathers several criteria which evaluate the system capacity

- To reveal its faults
- The accessibility of its components
- Its testing cost

Testability can be used to model and to implement testable systems
Testability

Testability in software life cycle
Testability

- **Observability**
  - ”a system is observable if for each input given to the system, a different output is observed”

- **Controlability**
  - ”a system is controllable if for each observed output, it exists an input which forces the observation of this output”.
Testability

Architecture 1
Testability

Architecture 2
ABPEL to STS?

- To flatten the nested BPEL activities
- To spread fault handlers into sub-activities
- To retrieve irrelevant properties

- STS offers a large formal background
  - definitions of implementation relations
  - test case generation algorithms
ABPEL Example
STS Example
Testability issues

- **Observability issues:**
  - (?amount, amount = 1000) and (?risk, risk = high; amount = 1000) give the same reaction (!request1, amount = 1000)
  - (?risk, risk = low) and (?approval2, app2 = yes) are followed by the same reaction (!approval, app2 = yes)
Testability issues

- Controlability issues:
  - A3 (invoke assessor), A4 (B.Invoke approval1), A6 (B. Invoke Approval2), A9(B. Reply Customer) on account of partner roles not initialized
  - A15: two conditions [amount >= 1000] and [amount <= 1000] are not exclusive.
Propositions

- Observability propositions:
  - An ABPEL specification not terminated by a “reply” (one-way “invoke”) activity is not observable.
  - An ABPEL specification composed of a couple of non identical “catch” (”catchall”) activities, followed by two ”invoke” activities using the same operation and parameter values, is not observable.
Propositions

• Controlability propositions:
  ▫ ”invoke” activities, depending on partners whose the role is not initialized, involve to uncontrollable ABPEL processes
  ▫ An ABPEL process, composed of a ”faulthandler” activity gathering two identical ”catch” activities, is not controllable.
Testability Enhancement

- Testability Enhancement Tool:
Testability Enhancement

• Observability Enhancement:

  ▫ ”reply” activity addiction:

```
Algorithm 1: ”reply” activity addiction

input : ABPEL specification bpel
1 Compute $sts = < L, l_0, Var, var_0, I, S, \rightarrow >$ from bpel;
2 if $\exists l_i \xrightarrow{e_i, \tau, e} l_f$ with $e \in S_I \cup \{\tau\}$ and $l_f$ a final location
   then
3 Add a reply activity $\text{reply}(\text{param}, \text{partner}, \text{op})$ in
   bpel with partner=client, op= client operation used
   for calling the ABPEL process, param=”last message
   from branch$_i$”;
```
Testability Enhancement

- Controlability Enhancement:
  - partner role addiction:

```
Algorithm 3: PartnerRole addiction

input : ABPEL specification bel

1 foreach "invoke" activity
   invoke(mess, resp, partner, op) do
2      if partner has not a role in the BPEL "partnerLink" section then
3         add <partnerLink name= partner_name
3         partnerRole="partner_nameProvider"
3         partnerLinkType="ns:partner_name"/>
4 , with "ns" a new variable equals to the web service
4 WSDL URL (xmlns:ns="http://...");
```
Testability Enhancement

- Controlability Enhancement:
  - fault distinction in fault handlers:

```
Algorithm 4: Fault distinction
1 foreach "faulthandler" activity composed of the catch activities catch_1(fault_1, act_1), ..., catch_n(fault_n, act_n) do
2   // fault_k = (faultName_k, faultElement_k, fault Message Type_k);
3   if it exists catch_i(fault_i, act_i), catch_j(fault_j, act_j) with fault_i == fault_j then
4     if MessageType_i == null then
5       MessageType_i = type in (string, integer, etc.) such as ;
6       \forall (1 \leq k \neq i \leq n), faultMessageType_k \neq type;
7     else
8       Add a random integer value at the end of faultName_i;
```
The modified STS
Conclusion & Perspectives

- We suggest some propositions to:
  - directly write more testable ABPEL specifications
  - evaluate observability and controllability criteria.

- We also propose some testability enhancement methods, which have been implemented in an academic tool.
Conclusion & Perspectives

- The execution time
- The completeness
- The accessibility of BPEL parts