Annales UMCS Informatica AI XIII, 1 (2013) 23–35
DOI: 10.2478/v10065-012-0036-9

Annales UMCS
Informatica
Lublin-Polonia
Sectio AI
http://www.annales.umcs.lublin.pl/

How validation can help in testing business processes orchestrating web services

Damian Grela[1*], Krzysztof Sapiecha[1†], Joanna Strug[1‡]

[1] *Department of Computer Science, Cracow University of Technology, Warszawska 24, 31-155 Kraków, Poland*

Abstract – Validation and testing are important in developing correct and fault free SOA-based systems. BPEL is a high level language that makes it possible to implement business processes as an orchestration of web services. In general, the testing requires much more test scenarios than the validation. However, in the case of BPEL processes, which have very simple and well structured implementation, test scenarios limited to the validation may also be efficient. The paper describes an experiment that aims at answering a question whether or not the validation test scenarios are also adequate for testing an implementation of BPEL processes. The experiment employs a Software Fault Injector for BPEL Processes that is able to inject faults when the test scenarios are running. The results of the experiment seem very promising. Hence, it seems that validation tests might give a strong support for testing.

1 Introduction

Recently, SOA (Service Oriented Architecture) [**1**] has become the most promising architecture for IT systems. It offers a way of composing systems from loosely coupled and interoperable services. The services are independent business functions made accessible over a network by remote suppliers. A developer of a SOA-based system should only select the most appropriate services and coordinate them into business processes that cover specification requirements for the system.

BPEL (Business Process Execution Language) [**2**] is a high level language that makes it possible to implement business processes as an orchestration of web services. The

*dgrela@pk.edu.pl
†pesapiec@cyf-kr.edu.pl
‡pestrug@cyf-kr.edu.pl

orchestration consists in subsequent invoking the web services by a special element of the process, called its coordinator. It leads to a very simple and structured SOA where only the coordinator and communication links between the coordinator and the services need to be tested. A correctness of the services may be assumed, as they are provided as ready-to-use components and should be tested by their developers before being shared.

Both, validation and testing may be performed with the help of test scenarios. In [**3, 4**] a method of generation of test scenarios for validation of a BPEL process was given. Test scenarios obtained by means of the method cover all functional requirements for the process and provide high validation accuracy [**4**]. This paper presents a case study that aims at answering a question to what extent such test scenarios are adequate for testing an implementation of the process. To this end an experiment employing Software Fault Injectior for BPEL Processes (SFIBP) was carried out and fault coverage for the test scenarios was calculated.

The paper is organised as follows. In Section 2 a related work is briefly described. In section 3 the problem is formulated. Section 4 defines fault coverage for the test scenarios. Section 5 contains a description of a case study. The paper ends with conclusions.

2 Related work

The problem of testing the SOA-based systems is not new, but most researchers focused on test generation [**5, 6, 7, 8, 9, 10, 11, 12**]. Their works fall loosely into two cathegories: developing efficient algorithms for selection of adequate tests [**6, 7, 8, 9**] and automation of the selection process [**10, 11, 12**]. Y. Yuan and Y. Yan [**6, 7**] proposed the graph-based approaches to handle concurrency activities of BPEL processes, in addition to basic and structured activities. Their approach was extended, combined with other techniques and implemented by several other researchers [**8, 9**]. M. Palomo-Duarte, A. Garcia-Dominguez, and I. Medina-Bulo based their approaches on the traditional white-box testing methods [**10, 11, 12**] and used formal methods and hybrid approaches along with the ActiveBPEL [**13**] and BPELUnit [**14**] test library for generating tests. However, in the works there are not any studies concerning the adequacy of generated tests for both validation and testing of BPEL processes.

The adequacy of tests can be measured with regard to some predefined metrics or by injecting faults and observing whether they are detected or not [**15**]. Fault injection is a popular technique that has been alredy applied in the context of SOA-based systems[**16, 17, 18, 19**]. The technique was often used for test generation [**15**]. PUPPET (Pick UP Performance Evaluation Test-bed) [**16**] is a tool for automatic generation of test-beds to empirically evaluate the QoS [**17**] features of a Web Service under development. GENESIS [**18**] generates executable web services from a description provided by the user and provides an environment in which the services can be tested prior to deployment in a production system. Another fault injection tool, WSInject [**19**], is a

script-driven fault injector that is able to inject interface and communication faults. WSInject works at the SOAP level and intercepts SOAP messages.

All of these approaches concern web-services or communication between a BPEL process and web-services (i.e. a fault is injected when a Web service is invoked). In the case of business processes various types of faults (e.g. replacement of input values) may appear. Therefore, SFIBP should be easily configurable to inject a rich variety of faults appearing in the very specific operational environment.

3 Problem statement

Validation aims to determine whether a software system satisfies requirements specification or not [**20**]. Requirements specification defines, in a formal way, what the system is expected to do. Test scenarios derived from such specification may be successfully used for the validation. In [**3**] an effective method for generation of test scenarios for validation of BPEL processes against specification requirements defined in SCR [**21**] was given. However, specification requirements should not contain anything that is not of interest for a user. Thus, test scenarios derived from the specification can check all specified requirements, but not necessarily implementation details that are introduced in further stages of development of the system. Therefore, the system should be tested to detect implementation errors. As generation of tests is usually time consuming, it is of high importance to find out to what extent the validation test scenarios are useful for the testing. To this end, an experiment might be performed and the implementation error coverage for the test scenarios could be calculated.

In general, the testing requires much more test scenarios than the validation. However, in the case of BPEL processes, which have very simple and well structured implementation, test scenarios limited to the validation may also be efficient. To measure the coverage of implementation errors by the validation test set, Software Fault Injector [**22**] for BPEL Processes will be applied. Implementation errors of BPEL process will be simulated by injecting faults when the test is running.

4 Faults in the SOA-based systems

In the SOA-based systems faults may be caused by two reasons:

1. incorrect interaction between web-services, and
2. incorrect internal logic of the system components (web-services and/or coordinator).

Interaction faults affect communication between different web-services or between the coordinator and the web-services. Internal logic errors are introduced by human developers or production facilities when components of the system are implemented. Eight types of interaction faults and four types of internal logic errors were identified [**23**]. Three out of them concern the systems orchestrating web services. These are the following:

1. Misbehaving execution flow. The fault occurs when a programmer invokes improper web-service[1] (i.e. different from the specified one). Fig. 1 gives an example of an improper web-service invocation error (a) and a faulty free version of the code (b).

```
<assign name="AssignTicketRS">
    <copy>
      <from variable="inputVariable" part="payload"
            query="/client:Euro2012ProcessRequest/client:Data"/>
      <to variable="HotelBS_Invoke_getHotelBooking_InputVariable"
          part="parameters"
          query="/ns2:getHotelBookingElement/ns2:Data"/>
     /copy>
</assign>
<invoke name="TicketRS_Invoke" partnerLink="HotelBS_WS"
        portType="ns1:HotelBS_WS" operation="getHotelBooking"
       inputVariable="HotelBS_Invoke_getHotelBooking_InputVariable"
       outputVariable="HotelBS_Invoke_getHotelBooking_OutputVariable"/>
<assign name="AssignResponse">
 <copy>
   <from variable="HotelBS_Invoke_getHotelBooking_OutputVariable"
         part="parameters"
         query="/ns2:getHotelBookingResponseElement/ns2:result"/>
   <to variable="Response"/>
 </copy>
</assign>                                                                    a
-------------------------------------------------------------------------------
<assign name="AssignTicketRS">                                               b
    <copy>
        <from variable="inputVariable" part="payload"
              query="/client:Euro2012ProcessRequest/client:Data"/>
        <to variable="TicketRS_Invoke_getTicketReservation_InputVariable"
            part="parameters"
            query="/ns2:getTicketReservationElement/ns2:Data"/>
    </copy>
</assign>
<invoke name="TicketRS_Invoke" partnerLink="TicketRS_WS"
     portType="ns1:TicketRS_WS" operation="getTicketReservation"
     inputVariable="TicketRS_Invoke_getTicketReservation_InputVariable"
          outputVariable="TicketRS_Invoke_getTicketReservation_OutputVariable"/>
<assign name="AssignResponse">
 <copy>
   <from variable="TicketRS_Invoke_getTicketReservation_OutputVariable"
         part="parameters"
         query="/ns2:getTicketReservationResponseElement/ns2:result"/>
   <to variable="Response"/>
 </copy>
</assign>
```

Fig. 1. Improper (a) and correct (b) web service invocation.

2. **Incorrect response**. The fault is caused by incorrect processing, within a coordinator, of correct response of a web-service (other causes related to incorrect interanal logic of a web-service, as defined in **[23]**, are not considered due to the assumption of correctness of web-services). Incorrect processing means, that:
   - a response from a wrong output port is used (Fig. 2),
   - a response is assigned to a wrong variable (Fig. 3), or
   - a response is not assigned at all (Fig. 4).

[1]The invoked web-service should exist and the invocation should be correct with regard to the specification of the web-service (otherwise such error will be reported by the compiler).

```
<invoke name="TicketRS_Invoke" partnerLink="TicketRS_WS"
     portType="ns1:TicketRS_WS" operation="getTicketReservation"
     inputVariable="TicketRS_Invoke_getTicketReservation_InputVariable"
         outputVariable="TicketRS_Invoke_getTicketReservation_OutputVariable"/>
<assign name="AssignResponse">
 <copy>
   <from variable="HotelBS_Invoke_getHotelBooking_OutputVariable"
         part="parameters"
         query="/ns2:getHotelBookingResponseElement/ns2:result"/>
   <to variable="Response"/>
 </copy>
</assign>                                                                    a
-------------------------------------------------------------------------------
<assign name="AssignResponse">                                               b
 <copy>
   <from variable="TicketRS_Invoke_getTicketReservation_OutputVariable"
         part="parameters"
         query="/ns2:getTicketReservationResponseElement/ns2:result"/>
   <to variable="Response"/>
 </copy>
</assign>
```

Fig. 2. A response from a wrong (a) and correct (b) output port.

```
<assign name="AssignResponse">
 <copy>
   <from variable="TicketRS_Invoke_getTicketReservation_OutputVariable"
         part="parameters"
         query="/ns2:getTicketReservationResponseElement/ns2:result"/>
   <to variable="Input"/>
 </copy>
</assign>                                                                    a
-------------------------------------------------------------------------------
<assign name="AssignResponse">                                               b
 <copy>
   <from variable="TicketRS_Invoke_getTicketReservation_OutputVariable"
         part="parameters"
         query="/ns2:getTicketReservationResponseElement/ns2:result"/>
   <to variable="Response"/>
 </copy>
</assign>
```

Fig. 3. A response is assigned to a wrong (a) and correct (b) variable.

```
<assign name="AssignResponse">
</assign>                                                                    a
-------------------------------------------------------------------------------
<assign name="AssignResponse">                                               b
 <copy>
   <from variable="TicketRS_Invoke_getTicketReservation_OutputVariable"
         part="parameters"
         query="/ns2:getTicketReservationResponseElement/ns2:result"/>
   <to variable="Response"/>
 </copy>
</assign>
```

Fig. 4. A response is not assigned at all (a) and is correctly assigned (b).

3. **Parameter incompatibility**. It occurs when a web-service receives, as an input data, incorrect arguments or arguments of incorrect types. The following four errors introduced into the implementation of a coordinator cause such a fault:

- a different operation of a web-service is invoked (Fig. 5). The operation should belong to the web-service (otherwise such error will be reported by a compiler).
- a wrong input port is used (Fig. 6). The port used should be consistent with the one that should be used (otherwise such error will be reported by a compiler).
- a wrong output port is used (Fig. 6), or
- a wrong value is assigned to an input port (Fig. 7).

```
<invoke name="TicketRS_Invoke" partnerLink="TicketRS_WS"
     portType="ns1:TicketRS_WS" operation=" checkTicketReservation"
     inputVariable="TicketRS_Invoke_getTicketReservation_InputVariable"
     outputVariable="TicketRS_Invoke_getTicketReservation_OutputVariable"/>    a
-------------------------------------------------------------------------------
<invoke name="TicketRS_Invoke" partnerLink="TicketRS_WS"                       b
     portType="ns1:TicketRS_WS" operation="getTicketReservation"
     inputVariable="TicketRS_Invoke_getTicketReservation_InputVariable"
     outputVariable="TicketRS_Invoke_getTicketReservation_OutputVariable"/>
```

Fig. 5. Different (a) and proper (b) operations of a web-service are invoked.

```
</assign>
<invoke name="TicketRS_Invoke" partnerLink="TicketRS_WS"
     portType="ns1:TicketRS_WS" operation="getTicketReservation"
     inputVariable="TicketRS_Invoke_checkTicketReservation_InputVariable"
     outputVariable="TicketRS_Invoke_ckeckTicketReservation_OutputVariable"/>  a
-------------------------------------------------------------------------------
<invoke name="TicketRS_Invoke" partnerLink="TicketRS_WS"                       b
     portType="ns1:TicketRS_WS" operation="getTicketReservation"
     inputVariable="TicketRS_Invoke_getTicketReservation_InputVariable"
     outputVariable="TicketRS_Invoke_getTicketReservation_OutputVariable"/>
```

Fig. 6. Wrong (a) and correct (b) input and output ports are used.

```
<assign name="AssignTicketRS">
    <copy>
        <from variable="inputVariable" part="payload"
              query="/client:Euro2012ProcessRequest/client:PESEL"/>
        <to variable="TicketRS_Invoke_getTicketReservation_InputVariable"
            part="parameters"
            query="/ns2:getTicketReservationElement/ns2:Data"/>
    </copy>
</assign>
<invoke name="TicketRS_Invoke" partnerLink="TicketRS_WS"                       a
-------------------------------------------------------------------------------
<assign name="AssignTicketRS">                                                 b
    <copy>
        <from variable="inputVariable" part="payload"
              query="/client:Euro2012ProcessRequest/client:Data"/>
        <to variable="TicketRS_Invoke_getTicketReservation_InputVariable"
            part="parameters"
            query="/ns2:getTicketReservationElement/ns2:Data"/>
    </copy>
</assign>
<invoke name="TicketRS_Invoke" partnerLink="TicketRS_WS"
```

Fig. 7. Wrong (a) and correct (b) values are assigned to an input port.

Effects of the faults are visible because the faults make the external behaviour of the coordinator be different from the expected one. The cause-effect table is shown in Fig. 8.

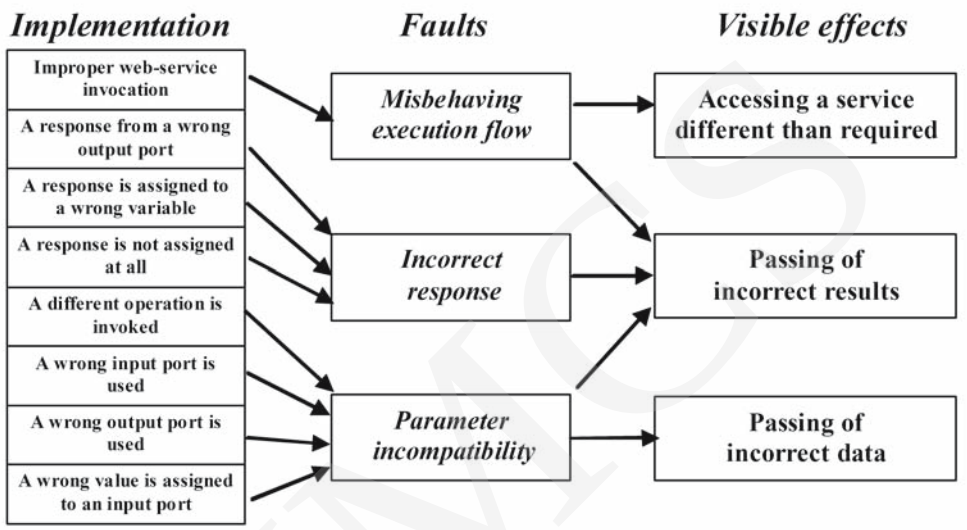

Fig. 8. Implementation errors, interaction and development faults and their effects.

All other faults defined in [**23**] are not relevant for this work. These faults are either related to a physical layer or caused by providers of web-services (incorrectness of web-services or interaction between web-services).

5 Case study

The goal of the case study is to evaluate the adequacy of validation test scenarios for testing BPEL processes. The test scenarios are evaluated based on their fault coverage calculated with respect to the faults generated by the SFIBP. The SFIBP generates the following three types of faults:

1. replacing web-service output parameters (OP),
2. replacing values of a web-service input parameters (IP),
3. replacing requested web-service with another one (WS).

The faults generated by SFIBP give the same observable effects as those described in Section 4, but their injection does not require the implementation of a coordinator to be changed.

The fault coverage for a set of test scenarios (FC) is expressed as a percentage of detected faults to all injected faults.

$$FC = \frac{F_D}{F_I} \cdot 100\%, \qquad \text{where:}$$

F_D – a number of detected faults,

F_I – a total number of injected faults.

As the faults are artificially generated and injected, their total number is known. However, it is not possible to determine the number and the types of all errors that might be the real source of the faults. Nevertheless, this is not shortcoming of the approach because only the coverage has considerable meaning.

The subsequent subsections describe briefly SFIBP that was used in the experiment to generate and inject faults (Section 5.1), an example system and test scenarios generated for the system (Section 5.2), and the experiment and its results (Section 5.3).

5.1 Software Fault Injector for the BPEL Processes

SFIBP is an execution-based injector [**15**], which is able to inject faults into the BPEL processes when test scenarios are running.

The SFIBP has been implemented as a special local service that is invoked instead of the proper web-service. Such approach helps reduce costs of the experiment, as the faults are injected without changing the implementation of a coordinator. A configuration file produced by the SFIBP defines three parameters of the proper web-services:

- identifiers of all methods provided by the web-services (ID),
- names of the methods,
- the number and names of parameters of the methods.

It also incudes predefined values of input and output parameters, values of alternative web-services IDs that are used to generate faults and the probability that a fault will be injected. Information about the injected faults is stored in a log file.

5.2 Football Reservation System

Football Reservation System (FRS) is a simple system allowing its users to book tickets for football games, hotels to stay during the games and plane or train tickets to arrive at the games.

The system was implemented as a BPEL process orchestrating five web-services. Each of the services is accessible on a different server and the whole process of reservation is coordinated through a central coordinator (Fig. 9).

Short descriptions of the web-services and their input and output parameters are given in Table 1. Types of the parameters are placed in brackets next to the parameters names.

A set of test scenarios generated for the system consists of 4 test scenarios having between two and five input/output events. The total number of the events is 16. The test scenarios were generated by means of the checking path method presented in [**3**]. Their usage provided high validation accuracy for the system.

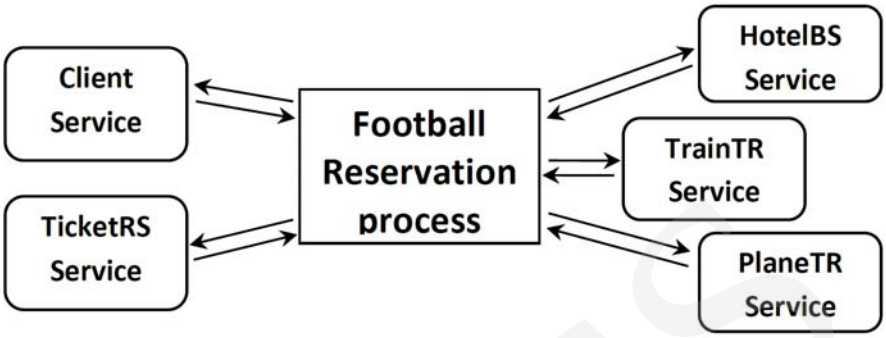

Fig. 9. Service orchestration for a Football Reservation process.

Table 1

web-service ID	*description*	*Parameters*	
		input	*output*
Client	retrieves data from the client and sends information about order	Date [String]	Result [String]
TicketRS	checks an availability of a football ticket at the given date	Date [String]	Result [String]
HotelBS	checks an availability of a hotel room at the given date	Date [String]	Result [String]
TrainTR	checks an availability of a train at the given date	Date [String]	Result [String]
PlaneTR	checks an availability of a plane at the given date	Date [String]	Result [String]

5.3 The experiment

The experiment consisted in:

1. implementing a fault free BPEL process for FRS and generating validation test scenarios,
2. configuring the SFIBP,
3. starting the SFIBP and running the BPEL process with the test scenarios,

4. comparing the outputs generated by the BPEL process with the expected ones given by test scenarios,
5. saving the results,
6. calculating the fault coverage.

Steps 3, 4 and 5 were repeated 1000 times. At each of the iteration randomly generated faults were injected into the BPEL process.

Table 2 shows the setting for all web-services of the FRS. The first row of the table shows IDs of web-service. The next two rows show the values of output and input parameters that are used to replace the proper ones when the faults are injected. IDs of web-services that are invoked instead of the proper ones are shown in the last row. The probability that a fault will occur was set to 33% for all faults.

Table 2

Web-service	**TicketRS**	**HotelBS**	**TrainTR**	**PlaneTR**
output parameter	„Yes", „No"	„OK", „No"	„Success", „Failure"	„True", False"
input parameter	„2011-07-26", „2012-01-01", „2014-04-04"	„2011-07-26", „2012-01-01", „2014-04-04"	„2011-07-26", „2012-01-01", „2014-04-04"	„2011-07-26", „2012-01-01", „2014-04-04"
alternative web-services	„HotelBS", „PlaneTR", „TrainTR"	„TicketRS", „PlaneTR", „TrainTR"	„TicketRS", „HotelBS", „PlaneTR"	„TicketRS", „HotelBS", „TrainTR"

The outputs generated by TicketRS, HotelBS, TrainTR and PlaneTR depend on an interval between a date of reservation and a date of football match. If the interval is equal or longer than it was assumed, then the respective web-service generates positive answer, otherwise the answer is negative. The intervals were set as follows: 15 days for TicketRS, 5 days for HotelBS, 1 day for TrainTR and 30 days for PlaneTR. These rules were introduced into the implementation of the web-services.

In the experiment the reservation date is an actual date (a day on which the process was invoked) and the date of the football match is the date that was specified by the user during the FRS invocation.

During the experiment the SFIBP could generate various combinations of the three types of faults (Section 5) or not introduce any fault. This gives eight different configurations of faults for each of the web-services and about 4000 for the whole system.

At the end of the experiment its results were analyzed and the fault coverage for the test scenarios was calculated. Table 3 summarises the results. It reports, for each of

the web-services, the total number of fault injected, and detected. The fault numbers were grouped based upon the type of faults.

Table 3

Faults	*TicketRS*			*HotelBS*			*TrainTR*			*PlaneTR*		
	IP	*OP*	*WS*	*IP*	*OP*	*WS*	*IP*	*OP*	*WS*	*IP*	*OP*	*WS*
injected	304	212	348	144	148	157	134	94	139	32	21	34
detected	295	208	348	140	145	157	132	92	139	31	20	34
FC	97%	98%	100%	97%	98%	100%	98%	98%	100%	97%	95%	100%

Due to the nature of the example majority of the injected faults is related to the first web-service (TicketRS) and the minority of them to the last web-service (PlaneTR). Almost all injected faults were detected by the test scenarios. The average fault coverage calculated based on the results of the experiments was 98%.

6 Conclusions

The paper describes a statistical experiment carried out to evaluate the test scenarios generated for validation of BPEL processes in context of testing the processes. Test generation is a time consuming activity, thus the possibility of having one set of tests scenarios providing accurate results for both validation and testing, was worth investigating.

The experiment was performed on a small example orchestrating five web-services. For the system, the SFIBP was able to generate three types of faults giving in total 4000 different fault configurations. For more complex systems the number of different fault configurations may be much higher than for the FRS. That is why not exhaustive but statistical testing was performed. It illustrates a general approach to the problem.

The experimental results seem very promising. The calculated fault coverage shows that almost all injected faults (98%) were detected by the test scenarios. The results confirmed the earlier assumptions that in the case of BPEL processes validation test scenarios may be adequate, also when they are used for testing. Hence, it seems that validation tests might give a strong support for testing. However, the experiment was carried out only on one simple system and focused on faults that only simulate implementation errors. More experiments are needed in order to make the conclusions more general. This will be one of the main goals of our further research.

References

[1] Weerawarana S., Curbera F., Leymann F., Storey T., Ferguson D. F., Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More, Prentice Hall (2005).

[2] Erl T., Service-Oriented Architecture: Concepts, Technology, and Design, Prentice Hall PTR (2005).

[3] Sapiecha K., Grela D., Test scenarios generation for certain class of processes defined in BPEL language, International Conference On Computer Science - Research and Applications, Annales UMCS - Informatica 8(2) (2008): 75.

[4] Sapiecha K., Grela D., Automating test case generation for requirements specification for processes orchestrating web services, 10th International Conference on Enterprise Information Systems (ICEIS), Information Systems Analysis and Specification 1 (2008): 381.

[5] Beizer B., Software testing techniques (2nd ed.), Van Nostrand Reinhold Co., New York, NY, USA (1990).

[6] Yuan Y., Li Z., Sun W., A Graph-Search Based Approach to BPEL4WS Test Generation, International Conference on Software Engineering Advances (2006).

[7] Yan J., Li Z., Yuan Y., Sun W., Zhang J., BPEL4WS Unit Testing: Test Case Generation Using a Concurrent Path Analysis Approach, 17th International Symposium on Software Reliability Engineering, ISSRE '06 (2006).

[8] Li Z. J., Tan H. F., Liu H. H., Zhu J., Mitsumori N. M., Business-process-driven gray-box SOA testing, IBM Systems Journal 47 (2008): 457.

[9] Liu C.-H., Chen S.-L., Li X.-Y., A WS-BPEL Based Structural Testing Approach for Web Service Compositions, presented at the IEEE International Symposium on Service-Oriented System Engineering (SOSE '08) (2008).

[10] Palomo-Duarte M., Garcia-Dominguez A., Medina-Bulo I., Takuan: A Dynamic Invariant Generation System for WS-BPEL Compositions, presented at the IEEE Sixth European Conference on Web Services (ECOWS '08) (2008).

[11] Palomo-Duarte M., Garcia-Dominguez A., Medina-Bulo I., Improving Takuan to Analyze a Meta-Search Engine WS-BPEL Composition, presented at the IEEE International Symposium on Service-Oriented System Engineering (SOSE '08) (2008).

[12] Palomo-Duarte M., Garcia-Dominguez A., Medina-Bulo I., An architecture for dynamic invariant generation in WS-BPEL web service compositions, presented at Proceedings of the International Conference on e-Business (2008).

[13] Contreras P., Zervas D., Murtagh F., ActiveBPEL Engine and ActiveBPEL Designer, Istanbul. Consortium meeting. June 08 (2006).

[14] Li Z. J., Sun W., BPEL-Unit: JUnit for BPEL Processes, Service-Oriented Computing (ICSOC '06), Lecture Notes in Computer Science 4294 (2006): 415.

[15] Sosnowski J., Testing and reliability in computer systems, EXIT, Warsaw (2005).

[16] Reinecke P., Wolter K., Towards a Multi-Level Fault-Injection Test-bed for Service-Oriented, 27th International Symposium on Reliable Distributed Systems, Napoli, Italy (2008).

[17] Bertolino A., Angelis G.D., Polini A., A QoS Test-Bed Generator for Web Services, ICWE, Lecture Notes in Computer Science 4607 (2007): 17.

[18] Juszczyk L., Truong H.L., Dustdar S., Genesis - a framework for automatic generation and steering of testbeds of complex web services, Proc. 13th IEEE International Conference on Engineering of Complex Computer Systems ICECCS 2008, March 31 – April 3 (2008): 131.

[19] Bessayah F., Cavalli A., Maja W., Martins E., Valenti A. W., A Fault Injection Tool for Testing Web Services Composition, TAIC PART 2010, Windsor, UK, September (2010).

[20] Tran E., Verification/Validation/Certification, Dependable Embedded Systems, Carnegie Mellon University, Spring (1999).

[21] Heitmeyer C., Kirby J., Labaw B., The SCR Method for Formally Specifying, Verifying and Validating Requirements: Tool Support , Pro c. of the International Conference on Software Engineering (1997).

[22] Voas J., McGraw G., Software Fault Injection: Inoculating programs against errors, Edit. Wiley. USA (1998).

[23] Chan K. S. M., Bishop J., Steyn J., Baresi L., Guinea S., A Fault Taxonomy for Web Service Composition, ICSOC Workshops (2007): 363.