Event-Driven Modeling and Testing of Web Services

Fevzi Belli, Michael Linschulte
Univ. of Paderborn,
Faculty for Computer Science, Electrical Engineering and Mathematics, Germany
{belli, linschu}@upb.de

Abstract

A service-oriented architecture (SOA) for web applications is often implemented using web service (WS) standards and consists of different functions the executions of which are perceived as events. The order and time-appropriateness of occurrences of these events play a vital role for the proper working of a real-time SOA. This paper presents an event-based approach for modeling and testing of functional behavior of WS in SOA by event sequence graphs (ESG). Nodes of ESG represent events, e.g., "request" or "response", and arcs give the sequence of these events. For representing parameter values, e.g., for time-out of function calls, ESG are augmented by decision tables (DT). A case study carried out on a commercial web system with SOA validates the approach and analyzes its characteristic issues.

The novelty of the approach stems from (i) its simplicity and lucidity in representing complex real-time web applications based on WS in SOA, and (ii) its modeling that considers a comfortable fault management.

1. Introduction and related work

Service-oriented architecture (SOA) is one of the current buzzwords in information technology. SOA [9] allows enterprises to centralize computer-based services and to offer those services over a network. On the basis of a published interface, the service can be used platform-independent inside and outside of the enterprise. The concept of a SOA is often realized by *web services (WS)* [7]. The interface of a WS is usually described by *WSDL* (*Web Service Description Language*) in accordance with *SOAP* (*Simple Object Access Protocol*). Both standards, SOAP and WSDL, are based on XML.

Interaction with WS consists of requests and responses which are perceived by the environment (human users or other systems) as *events*. *Requests* are function calls to a WS; *responses* are corresponding answers. In spite of its functional correctness, a request can be useless if it is not delivered "in time", e.g., a request for a stock price is of no interest if it is not delivered before the next change. Taking such time constraints into account requires time monitoring from request to response; in other words, we need a *real-time* service-oriented architecture (*RTSOA*). Time can be measured either server-sided or client-sided. The time difference between client and server time primarily depends on SOAP calls which entail intense XML parsing. Also the size of a response plays an important role: A search function, triggered by a single query, can return huge amounts of entries.

This paper presents an event-based approach for modeling and testing the functional behavior of WS. Functions of WS are modeled by *event sequence graphs (ESG)* where each node represents an event, e.g., "request" or "response". A model is usually used for analyzing the specified functional behavior of the *system under consideration (SUC)*, thus for testing whether or not SUC is doing the right things (testing the *desirable* behavior, *or positive testing*). Our ESG view will allow us also the other way around, thus to check whether or not SUC is *not* doing the incorrect, illegal things (testing the *undesirable* behavior, *or negative testing*). This is what we meant "a comfortable fault management" in the abstract.

Contracts [5] describing the structure of input data are useful for WS, but the WSDL-specification of a WS describes only the syntax of an interface. Thus, there is a lack of semantic information, e.g., a hotel reservation system will need an arrival date and a departure date. It is obvious that the arrival date should lie before departure date. Those additional constraints are not defined in a WSDL-Specification. This is the reason why we model the parameters of each function and their interrelationships, especially the ones concerning time constraints, by *decision tables (DT)*.

The simple and lucid, nevertheless powerful representation capability of DT-augmented ESG, together with their *holistic* testing concept that integrates positive and negative testing of WS-based web applications in RTSOA is the novelty of the approach introduced in this paper.

0730-3157/08 $25.00 © 2008 IEEE
DOI 10.1109/COMPSAC.2008.144

The next chapter discusses the related work. Section 3 summarizes the notional background using a non-trivial example borrowed from the case study of Section 4 which does not only validate the approach, but also exploits its characteristic issues. Section 5 concludes the paper, summarizing the approach introduced and future work planned.

2. Comparison with related work

A great deal of research work is dedicated to SOA [11] and some of them to analysis and testing of WS, as summarized by Baresi et al. [6]. Most of enclosed work deals with composition of WS and communication among WS. The reason is the unlimited number of WS that can be integrated into one application in which a WS itself can even integrate other WS. An extension to the WS interaction model is the web services Business Process Execution language (BPEL/WSBPEL). Its goal is to support orchestration of WS which support business transactions [12]. In this context, Marconi et al. presented a semi-automated development process to support the composition task that is able to generate the appropriate WSDL- and BPEL-Files [14].

We already mentioned in Section 1 that there exists a lack of semantic information in WSDL-Files that complicates black-box testing of WS. Tsai et al. [13] presented an extension to WSDL to support testing of WS that incorporates sequence specifications as well as input-output dependencies. Unfortunately it does not consider dependencies among input parameters. There exist also few approaches in literature dealing with systematic model-based testing of a single WS. Close to our approach, but not event-based, is "stateful" WS Testing [1]. This approach is based on *Stream X-machines*, a state-based model that is able to generate and execute test cases. However, real time aspects are not considered. Also negative testing is not considered by this approach. As far as we know, it was not subject of similar approaches, too.

Event-based modeling was paid some attention during the last years. In 2003, Gartner Analyst Roy Schulte declared *event driven architecture* (*EDA*) as successor of SOA [15]. EDA promises a faster reaction to events "as they happen": Whereas a SOA is a "pull" architecture, in EDA applications register for certain business events and are informed immediately when an event occurs ("push" architecture). Hence, EDA is a form of publish and subscribe. David Luckham [8] distinguishes *complex event processing* (*CEP*) as a more advanced idea from EDA. CEP concentrates on ordering and making use of event information whereas EDA concentrates on distribution and management of event information.

In this paper, we are not primarily interested in processing of event's and their information, but in their variety and "desirability" for the user of the SUC that is to be designed, modeled and tested. We assume that the user is mostly interested in reaching a final, *desirable* event; the modeling process has to anticipate *undesirable* ones that can prevent him, or her, from reaching of this goal. Accordingly, alternative ways have to be created to reach the goal. An example is given by the WS for credit card check that a bank integrates into different applications and is also publicly available for integration into diverse shopping systems. The desirable final event for both the owner of the credit card and credit card institute is to conclude the business. Nevertheless, undesirable events, as to constraints given by credit limit, that can be extended, or, more critical, fraud attempts must be taken into account.

3. Event-driven modeling and testing of web service behavior

The idea of WS and SOA in general entails providing central functions for recurrent tasks. Now, imagine the likely implementation of a shopping cart: Before transactions to the shopping cart are allowed, a valid login is required. If the login is successful, a function "addItem(itemID)" can be called to put an item into the shopping cart. Finally, a "checkout()"-function can be activated for ordering the selected items. But what happens if the addItem()-function is called first? This should result in a fault message. The motivation is to test such situations systematically. As stated in Section 2, response time of functions may also play a vital role for WS and can depend on selected parameter values as well as the response itself depends on them. Therefore, a further motivation is to enable systematic modeling and selection of input parameters for testing.

3.1 Holistic view of valid and invalid behavior

This work uses ESG notion [10] for modeling WS and web applications. Due to the lack of space, we will give only a brief, informal introduction. An *event* is specified as an externally observable phenomenon, e.g., a user stimulus or a system response. An event according to WS is a *request* (call of a function) to the SUC or a *response* (server response message to a "request").

ESG are directed graphs; their nodes represent events and edges valid, correct sequences of events. Two pseudo vertices, '[' and ']', symbolize *entry* and *exit* where any node can be reached by entry, and any node can reach the exit. Any sequence of vertices connected by an edge is called a *legal event sequence* (*ES*), building an *event pair (EP)*. In Figure 1, EP are represented by solid lines. An ES starting at the pseudo vertex '[' and ending with the pseudo vertex ']' is called *complete event sequence (CES*). CES are considered to perform *successful* runs

through the ESG, i.e., they are expected to arrive at the exit of the ESG that models SUC, in other words they deliver *desirable events*. For (positive) testing of SUC, CES are used as test inputs.

Now, view the ESG depicted in Figure 1 and ignore its dashed lines at the first glance; they will be introduced after the explanation of SUC. This ESG exemplifies a WS implementing a hotel reservation process; it will be used also in the case study of Section 4. In Figure 1, requests and responses that belong together are circled. Note that the function "bookHotel()" returns only then a result list if an hotel was found that fulfills the search criteria. Furthermore, a successful login is required before a hotel can be searched.

A node of the ESG representing a function call with input parameter is refined by a DT, i.e., input parameters and their structure are modeled by DT. Thus, refined nodes are augmented by DT and in the corresponding ESG they are double-circled. DT allow the modeling of valid system behavior depending on input parameters the variety of which is to be analyzed.

Table 1 shows a rudimentary DT for WS. The top left cell identifies the function represented by the DT. The *constraint part* specifies (a) valid domains to input parameters and (b) additional constraints. These constraints are the pre-conditions of a contract and have to be resolvable to true or false. The action part specifies valid system responses with respect to constraint sets, called *rules* (see column "R1", "R2", …). The action part represents the post-conditions of a contract. Thus, every rule of the DT defines a *contract*. System responses can be extended by time constraints where a response is expected within a certain time range. Table 1 illustrates an example of a DT that represents the function "LoginRequest" of Figure 1. System responses are extended by time constraints, e.g., selecting data according to **R1**, a response is expected within 200ms. If a response takes longer, the corresponding test fails.

It can be expected that third parties have access to the WSDL-file which describes the interface of the WS. Unfortunately, the WSDL does not describe valid or invalid orders in which WS's functions are supposed to be called. Therefore, testing of invalid sequences of events is also of substantial interest. To test the undesirable behavior, the ESG is to be *completed* by additional edges, represented as follows (dashed lines in Figure 1):

- Add edges from response vertices to request vertices where no edges exist ("Response" → "Request").
- Add edges from pseudo vertex "[" to request vertices where no edges exist ("[" → "Request").
- Add edges from request vertices to request vertices, where no edges exists ("Request" → "Request").

The additional edges symbolize *faulty event pairs (FEP)* as they should lead to a negative system response, i.e., executing such EP the SUC is expected to react with a warning. For enabling testing the system reaction upon an unexpected, undesirable event, FEP are to be extended to *faulty complete event sequences* (*FCES*). A FCES starts with the pseudo vertex '[' and ends with the FEP under consideration. FEP containing refined "Request"-nodes have to be thoroughly resolved with data. Every rule of the DT should carefully be checked as the result could depend on selected values. The ESG given in Figure 1 is extended by dashed edges which are *FEP*. Executing those FEP should result in a SOAP fault (symbolized by vertex "Fault"). Wherever another system response arises (perhaps one of the regular ones), the corresponding tests are failed ones (negative testing).

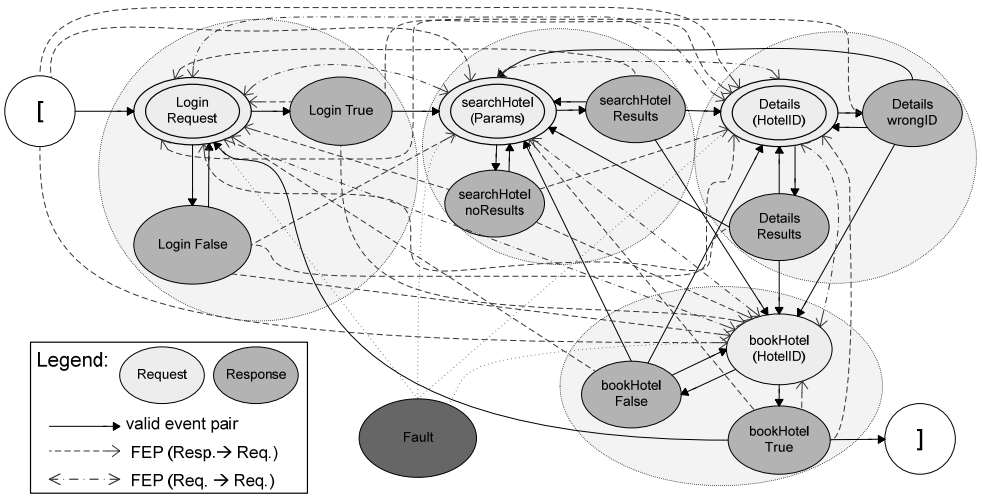

Figure 1. ESG for the web service "hotel reservation"

Table 1. DT modeling "LoginRequest"

		Rules				
	LoginRequest(usr, pwd)	**R1**	**R2**	**R3**	**R4**	**R5**
Constr.	usr ∈ Σ*	T	T	T	F	F
	pwd ∈ Σ*	T	T	F	T	F
	(usr, pwd) ∈ Login	*T*	*F*	-	-	-
Act.	LoginResponse(True)<200ms	X				
	LoginResponse(False)<200ms		X	X	X	X

The described way of testing both desirable and undesirable behavior by means of positive and negative testing using the same model enables a *holistic testing*.

3.2 Fault model and test process

The model developed in Section 3.1 defines by means of its completion a set of fault models and enables a meaningful coverage criterion: All EP of the ESG given are to be covered by CESs of minimal total length, and/or of minimal number. This problem is a derivation of the *Chinese Postman Problem (CPP)* [16] that has $O(n^3)$ runtime complexity in our case [10].

The generation of data out of decision tables leads to the *Constraint Satisfaction Problem* (*CSP)* that has a polynomial runtime complexity. Russell, Norvig [2] describe CSP as follows: "… A constraint satisfaction problem (or CSP) is defined by a set of variables, X_1, X_2, …, X_n, and a set of constraints, C_1, C_2, …, C_m. Each variable X_i has a nonempty domain D_i of possible values. Each constraint C_i involves some subset of the variables and specifies the allowable combinations of values for that subset." Each rule of the DT under consideration represents a CSP.

Algorithm 1 sketches the overall test process. Lines 1 to 10 deal with generation of test sequences, lines 11 to 21 describe their execution. As given in Algorithm 1, faults are identified if execution of a CES does not reach the final event. This is the case if another response as the expected one occurs or time constraints are violated. Faults with respect to FCES are identified if the sequence does not lead to a fault response; maybe because a valid response under regular circumstances has occurred.

The detected faults are classified into *positive* and *negative sequencing faults* according to faults detected by CES or FCES, respectively. Both categories can be refined into *structural sequencing faults* and *data-dependent sequencing faults*. Faults are characterized as structural sequencing faults if parameters have not caused this undesirable event; otherwise we have data-dependent sequencing faults. Where time limits of system responses are exceeded, faults are categorized as *time-out faults*.

It should be announced that the explosion of state space is a general problem of generating test cases on the basis of graphs. We addressed this problem with (a) considering events instead of states, (b) a hierarchical collection of *structured ESG (sESG)*, i.e., a node can represent another ESG that is to be tested all alone and (c) the restrictions contained in DT.

Algorithm 1: Test Process

```
01 cover all EPs by means of CESs (CPP)
02 cover all FEP by means of FCES (DFS)
03 FOREACH CES and FCES with decision tables DO
04   generate data-expanded CES/FEP on the basis of DT (CSP)
05 FOREACH CES and FCES with sESG DO
06   generate multiple CES/FCES by replacing the structured nodes
       with the CES of the sESG [10]
07 FOREACH CES DO
08   execute CES
09   IF final event is not reachable OR time constraint was hurt
10     remark test as failed
11   ELSE
12     remark test as passed
13 FOREACH FCES DO
14   execute FCES
15   IF FCES leads to a Fault-Response
16     remark test as passed
17   ELSE
18     remark test as failed
```

4. Case study

System under consideration. The SUC is a large commercial web portal with 53.000 LOC (lines of code): ISELTA (*Isik's System for Enterprise-Level Web-Centric Tourist Applications*; visit *http://www.iselta.com*). ISELTA enables travel and tourist enterprises, e.g., hotel owners, recreation/fitness centers, etc., to create their individual search & service offering masks (Figure 2). These masks can be embedded in the existing homepage of the hotels as an interface between customers and system. Potential customers can then use those masks to select and book services, e.g., hotel rooms.

A special service is given by the implementation of hotel reservation as a WS. This WS enables enterprises to embed ISELTA into own applications. Amongst others, following functions are available:

- ***login(username, password)***: function for login, returns true or false
- ***search(params)***: function for searching an offer, returns a result list
- ***details(HotelID)***: returns detailed information
- ***bookHotel (HotelID)***: enables reservation of an hotel, returns true or false

Searching and booking of services are carried out in four successive steps: First, a user has to enter his or her login data. Second, search criteria as to arrival date and departure date, service level (three stars, four stars, etc.), or type of catering has to be submitted. As a third step, de-

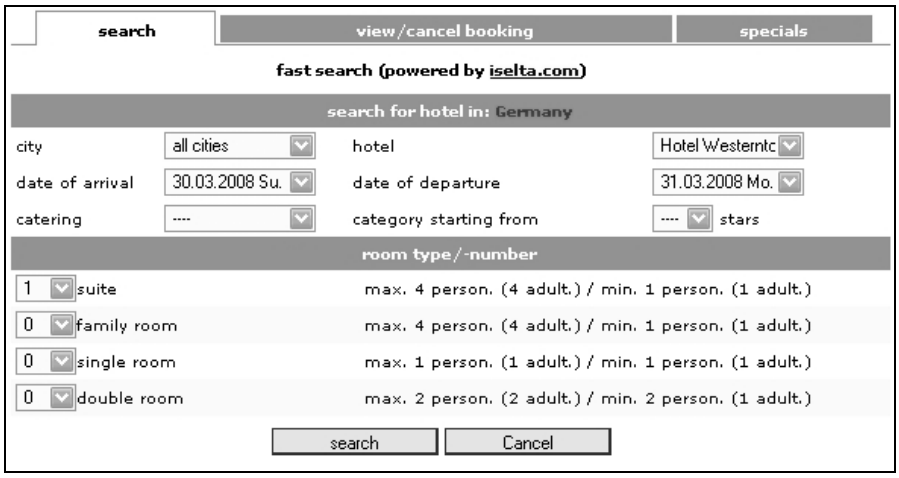

Figure 2. Screenshot of ISELTA-WA

tailed information to an offer can be demanded. Finally, the user can book an offer. Moreover, the user can alter relevant factors to achieve further results.

Architecture of ISELTA. Figure 3 sketches the architecture of ISELTA and the shared resources of the ISELTA web service (ISELTA-WS) and the web application (ISELTA-WA). ISELTA-WA includes Logic Files for checking data correctness before passing them to the classes. Furthermore, those Logic Files select and include HTML-templates on the basis of constitution of data. ISELTA-WS is built up on the same classes and database. Thus, ISELTA-WA and ISELTA-WS have the same basis.

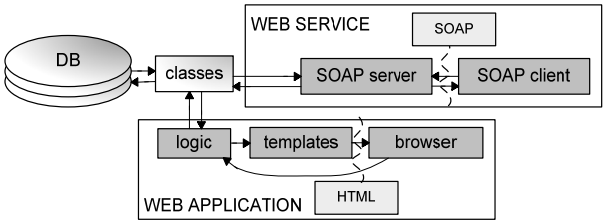

Figure 3. Architecture of ISELTA

Experiments & analysis. Figure 3 shows the corresponding ESG for ISELTA-WS in a very simplified way for ease of understanding. Refinement of this ESG and corresponding DT, test cases are generated and tests are run by methods described in Section 3.1 and 3.2. Table 2 contains an example for a test sequence with input data added. Those test sequences have been implemented as web services client.

For our experiments we measured time in two different ways: The *server-sided time* reflects the computational effort for generating a result, the *client-sided time* represents the time between request and response. To be independent of network bandwidth, we tested locally on one computer. Figure 4 depicts the calls of the method "searchHotel" where a result list was generated. The difference between server time and client time was nearly constant and reflects the circumstance that we tested locally. The experiment results show that the limit of 1000 ms was never exceeded, neither server-sided nor client-sided.

Table 2. Test sequence example

1. login("false", "user")
2. login → false < 200ms
3. login("max", "pwd")
4. login → true < 200ms
5. searchHotel("08-08-2008","12-08-2008", "Paderborn", "SR=1", "all", "all")
6. searchHotel → ResultList < 1000ms

…

Testing ISELTA-WS revealed four faults. For security reasons, function "bookHotel" should have logged out the user, but did not, e.g., it was possible to call "searchHotel"-function und get a result list. We detected also some SOAP faults where a response contained the German characters 'ö', 'ü', 'ä'. Hence, those characters should be replaced by their HTML pendants. We further detected a fault that calling function "details" requires storage of search parameters in session cache. Function "searchHotel" should reset parameters in cache before executing the search query. The fourth fault occurred by calling "searchHotel". A call of "details" led to wrong details information.

Note that ISELTA-WA was already tested in prior case studies. As ISELTA-WS and ISELTA-WA have the same basis, faults that have its origin in classes should

also be revealed by testing ISELTA-WS, e.g., testing ISELTA-WA revealed a fault that a password check had not differentiated between upper and lower cases. This fault could be confirmed.

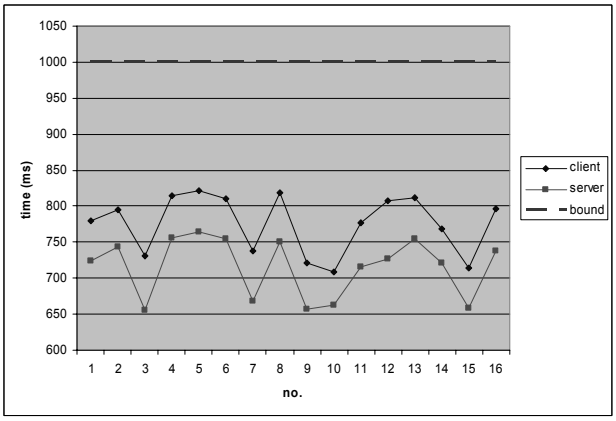

Figure 4. Time from request to response for function "searchHotel"

Summing up the results. The case study shows that especially negative testing using FCES has the power to reveal substantial faults and thus must not be left out. In our case study time constraints have not been violated, however they are always to be considered.

Worthwhile to be mentioned is also the fact that measuring the time only once means not much and is thus insufficient; functions should be executed several times to check several factors of them.

5. Conclusions and future work

This paper introduced an approach for modeling and testing of WS in real-time SOA. Events as to requests and responses are modeled by ESG, whereby corresponding input parameters are modeled by DT. This allows a simple, nevertheless powerful modeling and analysis of the system under consideration. The option to test also whether it reacts correctly in unexpected situation completes the approach, which we call negative testing. A case study briefly validates the approach and points out its characteristic features.

Test cases can automatically be generated from ESG; generation of data to consider real-time aspects from DT is the subject of our ongoing work. Future work aims at further reduction of the manual test effort by a better self-adaptability of the model due to changes in applications. Furthermore, dealing with data leads to the necessity of checking results, handling the problems as to, referring to a case study's typical question: "Were all of the hotels found correctly or is still some missing?" Thus, one of the future works planned concerns the extension of data checking.

References

[1] D. Dranidis, D. Kourtesis, and E. Ramollari, "Formal Verification of Web Service Behavioural Conformance through Testing", *Proceedings of the 3rd South-East European Workshop on Formal Methods (SEEFM07),* South-East European Research Centre (SEERC), Nov. 2007, pp. 112-125.

[2] Russell, S., and P. Norvig, *Artificial Intelligence*, Prentice Hall Series in Artificial Intelligence, Englewood Cliffs, New Jersey, 2003, pp. 137—160.

[3] L. Perrochon, W. Mann, S. Kasriel, and D. Luckham. "Event Mining with Event Processing Networks", *Methodologies for Knowledge Discovery and Data Mining*, Third Pacific-Asia Conference, PAKDD-99, Beijing, China; 1999; pp. 474-478 (in [8]).

[4] J. Vera, L. Perrochon, and D. Luckham. "Event-Based Execution Architectures for Dynamic Software Systems", *TC2 First Working IFIP Conference*, 1999; pp. 303-318 (in [8]).

[5] R. Heckel, and M. Lohmann, "Towards Contract-based Testing of Web Services", *International Workshop on Test and Analysis of Component Based Systems (TACoS)*, 2004, pp. 145-156.

[6] Baresi, L., and E. Di Nitto, *Test and Analysis of Web Services*, Springer, 2007.

[7] World Wide Web Consortium (W3C), Web Services Architecture, http://www.w3.org/TR/ws-arch/.

[8] Luckham D., *The Power of Events*, Addison Wesley, 2002.

[9] D. Nickull, F. McCabe, and J.B. Clark, OASIS SOA Reference Model, http://www.oasis-open.org/committees/tc_home.php ?wg_abbrev=soa-rm

[10] F. Belli, C.J. Budnik, and L. White, "Event-based Modeling, Analysis and Testing of User Interactions: Approach and Case Study", *The Journal of Software Testing, Verification and Reliability,* Vol. 16(3), 2006, pp. 3-32.

[11] P. Bianco, R. Kotermanski, and P. Merson, "Evaluating a Service-Oriented Architecture", *Technical Report of the Software Engineering Institute of Carnegie Mellon University,* http://www.sei.cmu.edu/publications/documents/*07.reports /07tr015.html*

[12] D. Jordan, and J. Evdemon, OASIS Web Services Business Process Execution Language, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel

[13] W.T. Tsai, R. Paul, Y. Wang, and C. Fan, "Wang D. Extending WSDL to Facilitate Web Services Testing", *7th IEEE International Symposium on High Assurance Systems Engineering (HASE'02)*, IEEE, 2002, pp. 171-172.

[14] A. Marconi, M. Pistore, and P. Traverso, "Process-level Composition of Web Services: a Semi-Automated Iterative Approach", *Annals of mathematics, computing & teleinformatics*, 2007, pp. 11-25.

[15] M. Michelson, "Event-Driven Architecture Overview", 2006, http://dx.doi.org/10.1571/bda2-2-06cc

[16] A.H. Aho, A.T. Dahbura, D. Lee, and M. Uyar, An Optimization Technique for Protocol Conformance Test Generation Based on UIO Sequences and Rural Chinese Postman Tours, *IEEE Trans. Commun. 39*, 1991, pp. 1604-1615.