WSDLTest – A Tool for Testing Web Services

Harry M. Sneed
Anecon GmbH
Vienna, Austria
Harry.Sneed@t-online.de

Shihong Huang
Computer Science & Engineering
Florida Atlantic University
shihong@cse.fau.edu

Abstract

A significant barrier to the use of Web services is the problem of testing them. One of the solutions to deal with the problem lies in the ability to simulate the usage of the services. Requests must be generated and responses must be validated automatically in a fast and reliable manner. To accomplish this goal, we have developed a tool called WSDLTest. WSDLTest is part of a larger complex tool set – DataTest – for generating and validating system test data. The architecture and functionality of this tool, as well as the experience gained from using it, are presented.

Keywords: SOA, Web services, WSDL interfaces, XML schema, tree walking, test automation, test data generation, test result validation

1. Introduction

Web services are becoming increasingly important to the IT-Business, especially since the advent of service oriented architecture. IT users are looking for a way to increase the flexibility of their IT systems so as to be able to react quickly to changes in their environments. If a competitor comes up with a new marketing approach, they have to be able to follow that approach in a short period of time. Adaptability of the IT systems has become critical to the survival of a company. If a new law is legislated, such as the Sarbanes Oxley Act in the U.S. that protects shareholders and the general public from accounting errors and fraudulent practices in the enterprise, companies have to be able to implement it within weeks. Changes to laws and regulations cannot be postponed. They have to be implemented by a given deadline, which is often only a short time away.

Under such time pressure, it is no longer possible to plan and organize long running projects. It is necessary to design and assemble a working solution within a limited time. This requirement for immediate response presupposes the existence of reusable components, which can be glued together within a standard framework to support a customized business process. The standard framework is a service-oriented-architecture such as that offered by IBM, Oracle and SAP [10]. The components are the Web services; the overlying business process can be defined with the business process execution language (BPEL) [9]. The glue for binding the business process to the Web services as well as to link the Web services to one another is the Web service description language (WSDL) [21]. The Web service components themselves are derived from various sources. Some are bought, some are taken from the open source community, some are newly developed and others are taken from the existing software systems, i.e. they are recycled to be reused in the new environment. Normally, this entails wrapping them [24].

This paper presents a tool for testing Web services called WSDLTest. Section 2 provides an overview of the necessity for testing Web services. Section 3 briefly discusses related academic research and commercial Web service testing tools. Section 4 details the WSDLTest approach. Section 5 describes our experiences in using the WSDLTest approach in an eGovernment Project. Section 6 summarizes the paper. Section 7 outlines future work.

2. Necessity for Testing Web Services

Regardless of where they come from, no one can ensure that the web service components will work as one might expect. Even those that are bought may not fit exactly to the task at hand. The fact that they are not compatible can lead to serious interaction errors. The recycled components may be even worse. Legacy programs tend to contain many hidden errors, which in a given context counterbalance each other. However, when moved to another environment to perform a slightly different function, the errors suddenly emerge to the surface. The same can happen with open source components. Perry and Kaiser [19] have demonstrated

Eighth IEEE International Symposium on Web Site Evolution (WSE'06)
0-7695-2696-9/06 $20.00

that the correctness of a component in one environment will not hold for another environment. Therefore, components have to be retested for every environment in which they are reused.

In the case of self-developed services, the reliability problem is the same as with all new software. They have to be subjected to extensive testing at all levels – at the unit level, at the component level, and, finally, at the system level. Experience with new systems shows, that the error rate of newly developed software varies between 3 and 6 errors per 1000 statements [13]. These errors have to be located and removed before the software goes into production. A significant portion of these errors is due to false assumptions the developer has about the nature of the task and the behavior of the environment. Such errors can only be uncovered by testing in the target environment – with data produced by others with a different perspective on the requirements. This is the primary rational for independent testers.

No matter where the Web services come from, they should go through an independent testing process, not only individually, but also collectively. This process should be well defined and supported by automated tools, so that it is quick, thorough and transparent. Transparency is of particular importance in testing Web services so that test cases can be traced and intermediate results examined. Due to the volume of test data required, it is also necessary to automatically generate the inputs and to automatically validate the outputs. By generating varying combinations of representative test data, a high rate of functional coverage is attained. By comparing the test results with expected results, a high degree of correctness is ensured [3].

3. Current Research and Tools for Testing Web Services

As Web services have become increasingly important to pervasive computing, significant efforts have been spent on the testing of Web services from both academia and industry. This section briefly comments on some of the related efforts.

3.1 Current Research

Coyote [26] is an example of an earlier work on providing an XML-based object-oriented testing framework that incorporates concepts from object-oriented application framework to test Web services rapidly. In order to address the insufficient information, such as lacking of dependence information, provided by WSDL file of a Web services, four extensions of WSDL have been proposed in [25] to facilitate Web services testing. Considering applications areas for semantic Web, Narayanan [14] etc proposed a method to enable markup and automated reasoning technology to describe, simulate, compose, test, and verify compositions of Web services. They used DAML-S DAML+OIL ontology for describing the capabilities of Web services. Automatically generated a Petri Net and provide decision procedures for Web services simulation, verification and composition. Similar work of using DAML-S for service description to find the semantic match between a declarative description of the service being sought and a description of the service being offered is described in [17].

As the general presentation of Web services – a Web site, Nguyen [15] describes the involving technical challenges of Web testing and presented results of Web testing strategies at several Fortune 500 companies, and pointed out some of the aspects need to consider to improve the ROI of Web testing, and reduce risks. Related to reengineering existing Web site into Web services, Jiang and Stroulia [8] described their work on reverse the interaction between Web site servers and client browsers into XML specifications, that is syntactically and semantically close to WSDL.

3.2 Existing Tools

There is no lack of tools for testing Web services. In fact, the market is full of them. The problem is not so much with the quantity, but with the quality of the tools. Most of them are recent developments that have yet to mature. They are also difficult to adjust to the local conditions and they require users to submit data via the Web client user interface. Testing through the user interface is not the most effective means of testing Web services as has been pointed out by R. Martin in a recent contribution to the IEEE Software magazine [12]. He suggests using a test bus to bypass the user interface and to test the services directly. This is an approach that has been followed by the authors, but first to the existing tools.

Typical of the tools on the market is the Mercury tool "Mercury Quicktest Professional" [27]. It is functional test and regression test automation tool that addresses every major software application and environment. It allows the users to fill out a web page and to submit it. It then follows the request from the client workstation through the network. This is done by instrumenting the SOAP message. The message is traced

to the Web service that processes it. If that Web service invokes another Web service then the link to that service is followed. The contents of each WSDL interface is recorded and kept in a trace file. In this way, the tester is able to trace the path of the Web service request through the architecture and to examine the message contents at different stages of processing [1].

ParaSoft offers a similar solution – ParaSoft SOAtest. It is an automated Web services testing product that allows users to verify all aspects of a Web service, from WSDL validation, to unit and functional testing of the client and server, to performance testing [18]. However, unlike Mercury Quicktest Professional, rather than starting the request from a web client, it generates requests from the business process procedures written in BPEL. This creates, on the one hand, a larger volume of data while, on the other hand, simulating real world conditions. It is expected that most requests for Web services will come from the business process scripts that are driving the business processes. BPEL language has been developed for that purpose, so it is natural to test with it. What is missing in the Parasoft solution is the capability of verifying the responses. They have to be inspected visually [2].

One of the pioneers in web testing is the company Empirix. Its e-TEST Suite tool [6] allows testers to simulate the business processes using the web clients. Their requests are recorded and translated into test scripts. The testers can then alter and vary the scripts to mutate one request into several variations for making a comprehensive functional test. The scripts are in Visual Basic for applications so it is easy for any person familiar with VB to work with them. With the scripts it is further possible to verify the response results against the expected results. Unexpected results are sorted out and fed to the error reporting system.

Other testing companies such as Software Research Associates [23], Logica [11]and Compuware [5] are all working on similar approaches. So it is only a question of time until the market is flooded with Web service testing tools. After that it will take some time before the desired level of tool quality is reached. Until this is the case, there is still some potential for customized solutions such as the one described in this paper.

What is new in the approach that is presented in this paper is the generation of data from the schema definition. What is also new is the use of the same assertion language for both generating request parameters and for verifying response results. There are other tools such as the Oracle Test Suite that uses assertions to verify responses. However they do not generate requests from the schema nor do they both generate requests and verify responses using the same script. This is as far as we know original to this approach.

4. The WSDLTest Approach

The WSDLTest tool takes a slightly different approach than the other commercial Web service testing tools. It is based on the schema of the WSDL description, i.e., it starts with a static analysis of the schema. From that schema two objects are generated. One is a WSDL request with random data; the other is a test script. The test script allows the user to manipulate the arguments in the Web service request. It also allows the user to verify the results in the Web server response. The test driver is a separate tool that reads and dispatches the Web service request and that receives and writes the Web service response. The following sections detail the approach.

4.1 Generating a Random Request from the WSDL Schema

All tests are tests against something. There has to be a source of the test data and there has to be an oracle to compare the test results against [7]. In the case of WSDLTest, the oracle is the WSDL schema. That schema is either generated automatically from the interface design or the developer writes it manually. As a third and more advanced alternative, it can be created from the BPEL process description. Irrespective of how it is created, the schema defines the basic complex data types in accordance with the rules of the XML schema standard. Complex data types can include other complex data types so that the data tree is represented with single and multiple occurrences of the tree nodes. The schema then defines the base nodes, i.e. the tops of the tree and their sequence. These are the actual parameters.

Following the parameter description come the message descriptions that identify the names and the component parts of each message whether it be a request or a response. Then follows the port type definitions. Each service operation to be invoked is listed out with the names of its input and output messages. These message names are references to the messages defined before, which again are references to the parameters defined before that. After the port types come the bindings describing the SOAP prototypes composed of

service operations. At the end, the Web service interface is given a name.

A WSDL schema is a tree structure where the SOAP prototypes refer to the service operations that in turn refer to the logical messages that in turn refer to the parameters that refer to the various data types. The data types may in turn refer to one another. Parsing this tree is called tree walking [4]. The parser selects a top node and follows it down through all of its branches collecting all of the subordinate nodes on the way down. At the bottom of each branch it will find the basic data types such as integers, booleans and strings. WSDLTest goes a step further by assigning each basic data type a set of representative data values. For instance integer values are assigned a range from 0 to 10000 and string values are assigned varying character combinations. These representative data sets are stored in tables and can be edited by the user prior to generating the test data. Thus the whole structure of the WSDL structure of the WSDL interface is as follows:

```
Web service interface
→SOAP prototypes
  →service operations
    →logical messages
      →parameters
         →data types
           →elementary data types
             →representative values
```

The task of the data generator is to walk the WSDL schema tree down to the level of the basic data types and to select representative values for that type. The values are selected randomly from the set of possible values. From these values an XML data group is created such as follows:

```
<Account>
  <Account_Number>100922</Account_Number>
  <Account_Owner> Smith</Account_Owner>
  <Account_Balance>
      999.50</Account_Balance>
  <Account_Status>2</Account_Status>
</Account>
```

In this way a WSDL service request file with sample data is generated and stored for future use. Simultaneously, a test script is created that allows the tester to override the values originally generated. This script is a template with the data element names and their values.

```
Account: WSDL
  assert new.Account_Number   =  "100922"
  assert new Account_Owner    =  "Smith"
  assert new Account_Balance  =  "999.50"
  assert new Account_Status   =  "2";
end;
```

By means of altering the script, the tester can now alter the values of the service request. Since a script is also generated for the output data, the tester is given a template for verifying the service responses. (See Figure 1 below.)

```
<complexType name="getProfile">
     <sequence>
          <element name="ZbsWsRequest_1"
           type="tns:ZbsWsRequest"
           nillable="true"/>
     </sequence>
 </complexType>
  . . .
 <getProfile>
   <getProfile>
     <ZbsWsRequest>
   <applikationId>XXXXXXXX</applikationId>
       <assertionId>YYYYYYYY</assertionId>
       <kompId>ZZZZZZZZ</kompId>
       <version>ThisData</version>
     </ZbsWsRequest>
   </getProfile>
 </getProfile>
 <login>
  <ZbsWsAuthentificationAssertion>
       <assertionId>ZZZZZZZZ</assertionId>
       . . .
   </ZbsWsAuthentificationAssertion>
 </login>
```

Figure 1: Sample 1

4.2 Write Pre-condition Assertions

The test scripts for WSDLTest are sequences of Pre-condition assertions defining possible states of the Web service request. A state is a combination of given values for the data types specified in the WSDL interface definition. The values are assigned to the individual data elements, but their assignments may be mutually dependent so that a particular combination of values can be determined by the tester e.g.:

```
assert new. Account_Status =  "y"
       if  (old.Account_Balance < "0") ;
```

For assigning test data values there are six different assertion types, which are the assignment of: another existing data value from the same interface, a constant value, a set of alternate values, a value range, a concatenated value, and a computed value.

The assignment of another existing data value is done by referring to that value. The value referred to must be within the same WSDL:

```
assert new.Account_Owner  =
old.Customer_Name;
```

The assignment of a constant value is done by giving the value as a literal in that statement. All literals are enclosed in quotes:

```
assert new. Account_Balance  =  "0";
```

The assignment of a set of alternate values is made by means of an enumeration. The enumerated values are separated by an OR sign "!".

```
assert new.Account_Status="0"!"1"!"2"!"3";
```

The assignment of a value range is for the purpose of boundary analysis. It is made by giving the lower and upper bound of a numeric range:

```
assert new.Account_Status = ["1" : "5"];
```

The assert assignments can be conditional or unconditional. If they are conditional they are followed by a logical expression comparing a data variable of the WSDL interface with another data variable of the same interface or with a constant value:

```
assert new.Account_Owner =  "Smith"
  if ( old.Account_Number = "100922"&
       old. Account_Balance > "1000" ) ;
```

The tester adapts the assertion statements in the script generated from the WSDL script to provide a representative test data profile including equivalence classes and boundary analysis as well as progressive and regressive value sequences. The goal is to manipulate the input data so that a wide range of representative service requests can be tested. To achieve this, tester should be familiar with what the Web service is supposed to do and to assign the data values accordingly.

4.3 Overriding the Random Data

Once the assertion scripts are available it is possible to overwrite the random data in the Web service request by the asserted data. This is the task of the XMLGen module. It matches the WSDL file generated by the WSDLGen module with the assertion script written by the tester. The data names in the assertion script are checked against the WSDL schema and the assertions compiled into symbol tables. There are different tables for the constants, the variables, the assignments and the conditions.

After the assertion script has been compiled, the corresponding WSDL file is read and the data values are replaced by the values derived from the assertions. If the assertion refers to a constant, the constant replaces the existing value of the XML data element with the name corresponding to that in the assertion. If the assertion refers to a variable, the value of the XML data element with that variable name is moved to the target data element. Alternate values are assigned one after the other in ascending order until the last value has been reached, then it starts again with the first value. Range values are assigned as the boundary values plus and minus one. (See Figure 2 below.)

```
<ZbsWsAuthentificationAssertion>
  <assertionId>Kati</assertionId>
  <version>1</version>
  <ZbsWsConditions>
     <notAfter>notAfter</notAfter>
     <notBefore>notBefore</notBefore>
   </ZbsWsConditions>
</ZbsWsAuthentificationAssertion>
```

Figure 2: Sample 2

In the end, a sequence of Web service requests exists with varying representative states combining the original generated data with the data assigned by the assertion scripts. By altering the assertions, the tester can alter the states of the requests, thus ensuring a maximum data coverage.

4.4 Activating the Web Services

Having generated the Web service requests, it is now possible to dispatch them to the server. This is the task of the test driver. It is a simulated BPEL process with a loop construct. The loop is driven by a list of Web services ordered by the sequence in which they should be invoked. The tester can edit the list to alter the sequence in accordance with the test requirements.

From the list the test driver takes the name of the next Web service to be invoked. It then reads the generated WSDL file with the name of that Web service and dispatches a request to the specified service. When testing in synchronic mode, it will wait until a response is received before dispatching the next request. When testing in asynchronic mode, it will dispatch several requests until it comes to a wait command in the Web service list. At this point it will wait until it has received responses for all of the services dispatched before continuing with the next request.

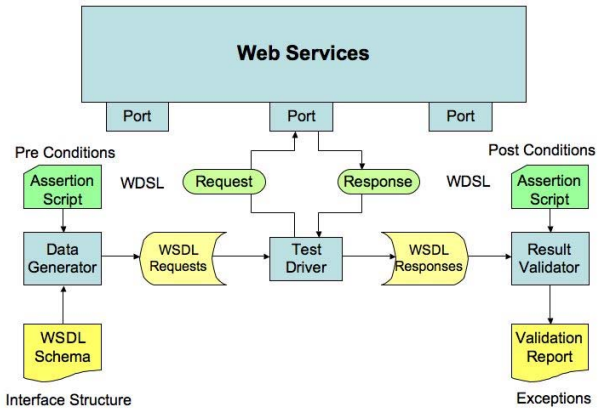

Figure 3: Web Service Test Driver

The responses are accepted and stored in separate response files to be verified later by a post processor. It is not the job of the driver to create requests or to check the responses. The requests are created by the preprocessor. The driver only dispatches them. The responses are checked by the post processor. The driver only stores them. In this way the role of the test driver is reduced to that of a simple dispatcher. BPEL procedures are very appropriate for that, since they have all of the necessary features for invoking Web services within a predefined workflow (See

.)

4.5 Writing Post-condition Assertions

The same assertion language that is used for verifying the Web service responses is used for constructing the Web service requests. Only here, the assertions have an inverse meaning. Data is not assigned from existing variables and constants, but compared with constant values:

```
assert new.Account_Owner =
old.Account_Owner;
```

implies that the value of the data element Account_Owner should match the value in the last response:

```
assert new.Account_Balance = "33.50";
```

is obviously a simple comparison.

The computed assertion, e.g.:

```
assert new.Account_Balance=
       old.Account_Balance - 50;
```

first computes a value and then compares that value with the actual value in the response. As in the case of the pre-conditions, the post-conditions can be unconditional or conditional. If they are conditional then they are qualified by a logical IF expression, comparing two variables in the Web service response or comparing a variable in the response with a constant value"

```
assert new. Account _Status = "3"
  if ( old.Account_Balance < "1000" ) ;
```

In all cases, if the assertion is not true, an error message is recorded displaying both the expected and the actual value. Provided there is an assertion check for each attribute of the WSDL response, the verification of the response will be 100%. It is not, however, required to check every attribute. The tester may decide to restrict the check to only critical variables. If so, the data coverage will be less. Data coverage is measured in terms of the number of asserted results relative to the sum of all results.

4.6 Validating the Responses

The module XMLVal fulfills the task of verifying the Web service results. For this it must first compile the post condition assertions into internal tables of variable references, constants, enumerations and ranges. In doing so, it checks the data names and types against the names and types declared in the WSDL schema to ensure consistency.

Having succeeded in compiling the assertion scripts, the tool then uses the compiled table to check the Web service response. First, the expected values are stored in a table with a key for each object occurrence. Secondly, it parses the WSDL result file matching the objects there with the objects in the assertion tables. If a match is found, the attributes of that object are extracted and their values are compared with the expected values. If they do not match the verification condition, the data names and values are written out in a list of non-matching results. It is then the task of the tester to explore why the results do not match.

In addition to listing out the assertion violations, the XMLVal tool also produces a statistic on the degree of data coverage and the degree of correctness.

5. Experience with WSDLTest in an eGovernment Project

The experience with WSDLTest in an eGovernment project has been reported at the WSE 2005 workshop

[22]. There nine different Web services were tested with an average of 22 requests per service. Altogether 47 different responses were verified. Of these, 19 contained at least erroneous result. Thus, of the more than 450 errors found within the project as a whole, some 25 were detected in the Web services.

In the mean time, the tool has been applied to a second project in the same environment. Up until this paper was written, more than 40 errors have been uncovered by comparing the results. The project is still going on.

It would appear that the tool is an appropriate instrument for testing Web services, as long as the WSDL interfaces are not overly complex. If they are too complex, the task of writing the assertions becomes difficult and errors occur there. At this point the tester cannot be sure whether an observed error is caused by the Web service or by a wrongly formulated assertion. A similar experience was reported on from the U.S. ballistic missile defense project some 30 years ago [20]. In that situation, some 40% of the errors reported were actually errors in the testing procedures. Looking at the testing technology used in that project by the RXVP test laboratory it can be seen that basic test methods – setting pre-conditions, checking post-conditions, instrumenting the software, monitoring the test paths and measuring test coverage – have hardly changed since the 1970s. Only the environment has changed.

6. Requirements for Future Work

Testing software systems is a complex task that has yet to be fully understood. There are many facets of that task, including specifying test cases, generating test data, monitoring test execution, measuring test coverage, validating test results, and tracking system errors, etc.

The tool WSDLTest only addresses two of these issues – generating test data and validating test results. Another tool Test Analysis analyses the requirement documents to extract the functional and non-functional test cases. These abstract test cases are then stored in a test case database. It would be necessary to somehow use these test cases to generate the pre- and post-condition assertions. That would entail bridging the gap between the requirement specification and the test specification. The greatest barrier to achieving this is the informality of the requirement specs. The question is one of deriving formal, detailed expressions from an abstract, informal description. It is the same problem as faced by the model driven development community.

Another direction for future work is test monitoring. It would be useful to trace the path of a Web service request through the system. This requires altering the server components to record what request they are currently working on. By treating a trace file it would be possible to monitor the execution sequence of the Web services, as there are often many services involved in the processing of one request.

There still remains the basic question as to what degree the test should be automated. It may not be so wise to try and automate the whole web testing process, but to rely instead on the skills and creativity of the human tester. Automation often tends to hide important problems. The issue of test automation versus creative testing remains a major topic in the testing literature [16].

The limitations to the approach presented in this paper are of both a technical and a theoretical nature. A technical limitation has to do with the restricted scope of the assertion scripts. Comparisons can only be made between variables defined within the WSDL schema. No data can be addressed other than that contained within the requests and responses. For instance, data base attributes are out of scope for both request generation and response validation.

A theoretical limitation is the language used to assign arguments and verify results. It is currently designed to be as simple as possible to allow ordinary testers to use it, but it does so at the cost of expressive power. It does not allow the user to nest assertions or to define complex data relationships. Also since value domains are not part of the WSDL schema, it remains up to the tester to enumerate representative values and to define value ranges in the assertion script. This can result in a significant manual effort. Assertion script templates can be generated, but the filling out of the actual test data has to be done manually. One way to alleviate this problem would be to include domain definitions in the WSDL. Another way would be to store the data values in external tables, which are referenced by the assertions. These and other issues will have to be addressed in future work.

7. Conclusions

This paper has reported on a tool called WSDLTest for supporting Web service testing. The tool generates Web service requests from the WSDL schemas and adjusts them in accordance with the pre-condition assertions written by the tester. It dispatches the requests

and captures the responses. After testing it then verifying the response contents against the post-condition assertions composed by the tester.

The tool is still under development, but it has already been deployed in an eGovernment project to expedite the testing of Web services offered by a German State Government. Future work will be in the direction of linking this tool to other test tools supporting other test activities.

References

[1] "Mercury Interactive simplifies functional testing" in Computer Weekly, Nr. 15, April, 2006, p. 22

[2] "Parasoft supports Web Service Testing", in Computer Weekly, Nr. 15, April, 2006, p. 24

[3] Berg,H.; Boebert,W; Franta,W; Moher,T. *Formal Methods of Program Verification and Specification* Prentice-Hall: Englewood Cliffs, 1982.

[4] Bradley, N.: *The XML Companion* (3rd Edition) Addison-Wesley 2001.

[5] Compuware, http://www.compuware.com/

[6] Empirix Inc.: "e-TEST Suite for integrated Web Testing", www.empirix.com

[7] Howden, *W. Functional Program Testing and Analysis* McGraw-Hill Series in Software Engineering and Technology. P. 123. McGraw-Hill College: New York, 1987.

[8] Jiang, Y.; Stroulia, E. "Towards Reengineering Web sites to Web-services Providers" *Proceedings of the eighth European Conference on Software Maintenance and Reengineering* (CSMR 2004:March 24-26, 2004; Tampere, Finland) pp. 296-305.

[9] Juric, M.; Mathew, B.; Sarang, P. *Business Process Execution Language for Web Services*, p.7. Packt Publishing: Birmingham, U.K. 2004.

[10] Kratzig, D.; Banke, K.; Slama, D. *Enterprise SOA Coad Series*, p. 6. Prentice-Hall Pub: Upper Saddle River, NJ, 2004.

[11] Logica, http://www.logicacmg.com/

[12] Martin, R. "The Test Bus Imperative: Architectures that Support Automated Acceptance Testing." *IEEE Software* 22(4): pp. 65-76, July 2005.

[13] Musa, J.; Ackerman, A. F. "Quantifying Software Validation: When to Stop Testing?" *IEEE Software*, 6(3): pp. 19-27, May 1989.

[14] Narayanan, S.; Mcllraith, S.: "Simulation, Verification and automated Composition of Web Services." *Proceedings of the 11th International Conference on World Wide Web* (WWW'02: Honolulu, HI, 2002); ACM Press.

[15] Nguyen, H. Q.: "Web Application Testing Beyond Tactics" *Proceedings of the sixth IEEE International Workshop on Web Site Evolution* (WSE 2004: September 11, 2004; Chicago, IL). Los Alamitos, CA: IEEE CS Press, 2004.

[16] Nguyen,H.Q. "Testing Web-based Applications", Software Testing & Quality Engineering, Vol. 2, No. 3, May 2000, p. 23-30

[17] Paolucci, M.; Kawamura, T.; Payne, T. R.; Sycara, K.: "Semantic Matching of Web Services Capabilities" *Proceedings of the First International Semantic Web Conference* (ISWC 2002: June 9-12, 2002; Sardinia, Italy) LNCS, Springer Berlin/Heidelberg, Volume 2342/2002.

[18] Parasoft http://www.parasoft.com/jsp/products/home.jsp?product=SOAP

[19] Perry,D.; Kaiser, G. "Adequate Testing and Object-oriented Programming." *Journal of Object-Oriented Programming*. 2(5): pp. 13-19, January/February 1990.

[20] Ramamoorthy, C.; Ho, S. "Testing Large Software with Automated Software Evaluation Systems." *IEEE Transactions of Software Engineering*. Vol. 1, pp. 46-58, March 1975.

[21] Sneed, H. "Integrating legacy Software into a Service oriented Architecture." *Proceedings of the 10th European Conference on Software Maintenance and Reengineering* (CSMR 2006: Bari, Italy; March 22-24, 2006), p.5. IEEE Computer Society Press, 2006.

[22] Sneed, H. "Testing an eGovernment Website" *Proceedings of the 7th IEEE International Symposium on Web Site Evolution* (WSE2005: Budapest, Hungary; September 26, 2005). P.3. IEEE Computer Society Press, 2005.

[23] Software Research Associates, http://www.srausa.com/

[24] Tilley, S.; Gerdes, J.; Hamilton, T.; Huang, S.; Müller, H.; Smith, D.; Wong, K. "On the Business Value and Technical Challenges of Adapting Web Services." *Journal of Software Maintenance and Evolution: Research and Practice*, 16(1-2): 31-50. John Wiley and Sons 2004.

[25] Tsai, W. T.; Paul, R.; Wang, Y.; Fan, C.; Wang, D.: "Extending WDL to Facilitate Web Services Testing." *Proceedings of the 7th IEEE International Symposium on High Assurance Systems Engineering* (HASE 2002: October 25-26, 2002; Tokyo, Japan); pp. 171; IEEE Computer Society 2002.

[26] Tsai, W. T.; Paul, R.; Weiwei Song; Zhibin Cao: "Coyote: an XML-based Framework for Web Services Testing." *Proceedings of the 7th IEEE International Symposium on High Assurance Systems Engineering* (HASE 2002: October 25-26, 2002; Tokyo, Japan); pp. 173-174; IEEE Computer Society 2002.

[27] www.mercury.com