

Developing and Assuring Trustworthy Web Services

W. T. Tsai, X. Wei, Y. Chen, B. Xiao, R. Paul*, and H. Huang

Computer Science and Engineering Department

Arizona State University, Tempe, AZ 85287-8809, U.S.A

*Department of Defense, Washington DC, U.S.A

Abstract

Web services are emerging technologies that are changing the way we develop and use computer systems and software. Current Web services testing techniques are unable to assure the desired level of trustworthiness, which presents a barrier to WS applications in mission and business critical environments. This paper presents a framework that assures the trustworthiness of Web services. New assurance techniques are developed within the framework, including specification verification via completeness and consistency checking, specification refinement, distributed Web services development, test case generation, and automated Web services testing. Traditional test case generation methods only generate positive test cases that verify the functionality of software. The Swiss Cheese test case generation method proposed in this paper is designed to perform both positive and negative testing that also reveal the vulnerability of Web services. This integrated development process is implemented in a case study. The experimental evaluation demonstrates the effectiveness of this approach. It also reveals that the Swiss Cheese negative testing detects even more faults than positive testing and thus significantly reduces the vulnerability of Web services.

Keywords: Web services, Web services testing, test case generation, vulnerability, model checking.

1. Introduction

Web services (WS) have emerged in e-commerce and e-business applications in the recent years. Current WS are based on UDDI server that provides directory services similar to the telephone yellow book. A UDDI server is not responsible for the quality of the services it refers. Thus, the trustworthiness or vulnerability of WS presents a major concern for the users and a barrier to widening the application of WS. Traditional dependability techniques, such as correctness proof, fault-tolerant

computing, testing, and evaluation, could be used to improve the trustworthiness of WS. However, these techniques have to be redesigned to handle the dynamic and the open platform features of WS. This paper exploits WS-based testing and verification techniques that can lead to trustworthy WS.

A number of studies and attempts have been made to address the WS testing and verification problems.

In [5] Davidson distinguished between two kinds of WS testing: Internet-based and Intranet-based WS testing. He also listed several testing techniques for WS including: proof-of-concept testing, functional testing, regression testing, load/stress testing, and monitoring.

In [14], it was suggested that WS testing should include: basic WS functionality; SOAP messages; WSDL files; publishing, finding, binding capabilities of an SOA; asynchronous capabilities of WS; the SOAP intermediary capability; the quality of service of WS; dynamic runtime capabilities; SOAP and WS interoperability; and WS performance and load testing.

In [2], Clune and Chen suggested that both clients and service providers must be involved in WS testing, and many issues must be addressed during WS development including: security, interoperability, UDDI registration, and performance considerations. Similarly, in [8], Myerson stated that all three parties of WS including clients, providers, and brokers, need to be involved in WS testing. These discussions support the collaborative testing concept we propose for WS testing.

Commercial tools have been developed for WS testing, including e-test by Empirix at <http://www.empirix.com/>, eValid from Software Research at www.soft.com, xmlpsy at <http://webservices.smlpsy.com/>, and SOAPtest from Parasoft. These tools often debug XML files, monitoring WS testing, capture-and-replay tests, debug SOAP.

IBM's Websight is a testing and debugging tool under its WS framework WebSphere [1]. Websight traces and visualizes the execution process of WS and thus helps the

programmer find syntax and semantic errors in the WS under test.

WS-I (Web Services Interoperability Organization), an industry organization chartered to promote WS interoperability across platforms, released a WS testing tool in March 2004 <<http://www.ws-i.org>>. The tool consists of two components: WS-I monitor and WS-I analyzer. The WS-I monitor can be placed between the client and the WS. It logs all messages, requests and responses, as they go back and forth. The WS-I analyzer then goes through every message in the log and analyzes them against all the interoperability requirements.

In the past a few years, we have developed WS testing techniques that support trustworthy computing. In [13], rapid testing techniques were developed, including group testing, regression testing, pattern-based verification. In [10], an enhanced WSDL interface was developed to include dependency information, functional description, invoking, and concurrent sequence specifications so that test cases can be generated based on WSDL descriptions. In [11], a variety of test generation techniques were developed to test WS in an enhanced UDDI-based service broker. These test cases could be arranged hierarchically to test domains of related WS. An early form of WS check-in and check-out processes was also proposed to increase the confidence of WS used. In [12], WS scenarios were developed in stages. In the first stage, the individual service scenarios are developed; in the second stage, interactions among WS were modeled; and finally the overall scenarios were developed combining previously developed scenarios. These scenarios were then translated into test cases to be performed by organized distributed agents.

Based on these studies, we proposed the WebStar (Web Services Testing, Reliability Assessment, and Ranking) framework that assures the trustworthiness and reduces the vulnerability of WS by rigorous positive and negative testing, reliability assessing, and ranking. A WS-based reliability model was proposed in [15]. Ranking of WS and test cases were studied in [16]. This paper focuses on testing and test case generation techniques for WS assurance. These techniques not only perform positive testing that verify the required functionality but also perform negative testing that assures the robustness of WS under irregular inputs or attacks. Negative testing is particular important for WS, because a WS can be composed of WS from different service providers without the availability of the source codes.

The rest of the paper is structured as follows. Section 2 introduces distributed computing model based on WS. Section 3 outlines the overall WS development process including specification checking, test case generation, and

testing techniques. Section 4 presents a case study and experiments that follow the proposed development process. Section 5 summarizes the experiment results. Finally, section 6 concludes the paper.

2. Distributed Computing via Trustworthy Web Services

Typical WS are truly dynamic and distributed. A new WS can be composed at runtime based on WS reside in remote locations. The communications among the component WS are through SOAP protocol to remotely invoke WS on other sites. Current UDDI-based WS model does not provide assurance to the WS.

The WebStar framework is shown in Figure 1. At the top layer, it provides WS development, testing, reliability assessment, and ranking services. These services produce databases of test cases, reliability data, WS service directory, and assured trustworthy WS reside locally in WebStar or remotely in service provider's sites. The bottom layer, a powerful distributed system supports the computation-intensive testing, reliability assessing, and ranking tasks. The basic services that WebStar provides:

- WS development: Service developers and researchers can use the specification, verification, evaluation, and testing tool to develop trustworthy WS;
- WS publication: Service providers can publish their WS;
- WS searching: Service requesters or clients can search WebStar for WS that meet their requirements.
- WS composition: WebStar can compose new WS based on the existing WS;

WebStar can not only rank WS, nut also rank the effectiveness algorithms, models, and tools. For example, researchers can use WebStar to test and compare their new testing tool, test case generation algorithm, reliability model, or ranking algorithm, by submitting them to the WebStar for experimentation. However, these topics are beyond the scope of this paper.

Now let's consider a typical scenario that demonstrates how WebStar works.

1. A service provider attempts to register a WS to WebStar.
2. Upon receiving the request, WebStar invokes the test master to perform rigorous check-in test.
3. If the WS passes the test, WebStar will add the WS to its WS directory. The service agent can be either added to the WS repository or resides in the service provider's site.
4. A client makes an inquiry to WebStar, search for a desired service and its instructions on how to test or

use the service. Upon receiving the request, WebStar searches the WS directory.

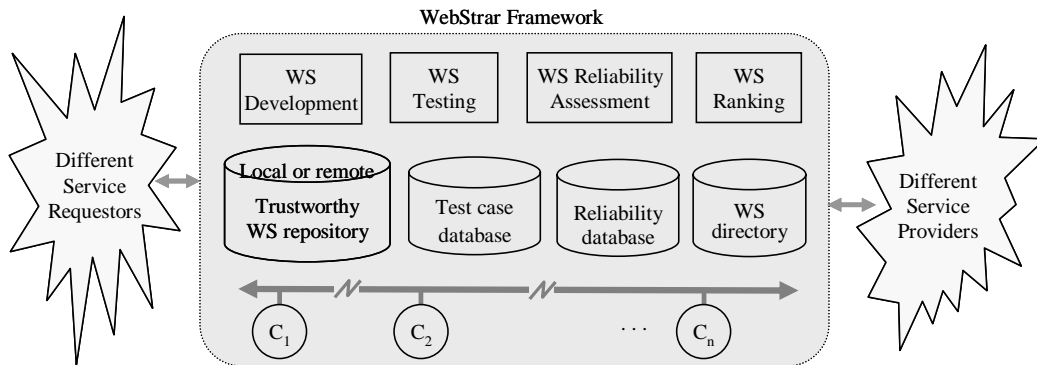


Figure 1. WebStar framework for Web Services testing, reliability assessment, and ranking

5. Once a match between the desired service and a registered WS is found, WebStar invokes the test master to perform a check-out test if the WS resides outside WebStar. The check-out test is simple and fast. It only tests if the service provider has not modified the service after its registration. If a modification is detected, the check-in test will be called to perform rigorous test.
6. WebStar returns the syntax and parameters required for a SOAP to the client.
7. The client can test the WS using its own test cases to make sure that the WS can meet its quality and dependability requirement, e.g., the reliability, timing, or accuracy requirements, etc.
8. If the test results are satisfactory, the client makes a SOAP call to WebStar to use the WS.
9. WebStar will invoke the WS, or perform a location transparent binding between the client and the service provider if the WS resides outside WebStar.

3. Development and Assurance of Web Services

OWL-S facilitates WS specification with a core set of markup language constructs for describing the properties and capabilities of WS in unambiguous and computer-interpretable form. OWL-S supports the automation of WS tasks including automated WS discovery, invocation, interoperation, composition, and execution monitoring.

As shown in Figure 2, the first step of our development process is to create WS specification written in OWL-S. With a translator, the specifications written in other specification languages such as DMAL-S, WSDL, and WS-CDL could be used too. The next step is to perform specification check. Different methods can be applied to perform the specification check. We have applied the Completeness and Consistency (C&C) analysis [13] and model checking technique based on

BLAST [1][5]. In this paper, we will focus on the C&C analysis. If the specification fails the check, it has to be modified, refined and test again.

After several iterations of test and refinement, we obtain the enhanced specification, which will be used to develop the WS and test cases. The WS development step will produce a WS, which normally consists of several component services.

The enhanced specification is used for test case generation. Boolean expression analysis method is used to extract the full scenario coverage of Boolean expressions [13], which are then applied as the input the Swiss Cheese Automated Test Case Generation [17] tool, which, in turn, generates both positive and negative test cases. Positive test cases are used to test if the WS output meets the specification for the legitimate inputs, while negative test cases are used to test the robustness, i.e., the behavior of the WS if unexpected inputs are applied. Negative test is particularly important for WS. Failing to perform extensive negative test may compromise the security and safety of the other WS that communicate with this WS.

Finally, the test cases are stored in the test case database before being applied to test the WS developed.

The key technologies applied in this development process include:

- Completeness and consistency (C&C) analysis of the WS specification: This module checks whether all conditions in OWL-S specification are consistent and whether all conditions have been covered and handled properly by the specification.
- Model checking: Model checking has been proposed recently to facilitate software testing following the idea that model checking verifies the model while testing validates the correspondence between the model and the system. One of the most promising

approaches was proposed at University of California at Berkeley using BLAST[1][5]. The BLAST model checker is capable to check safety temporal properties, predicate-bound properties (in the form to assert that at a location l , a predicate p is true or false), and identify dead code. BLAST abstracts each execution path as a set of predicates (or conditions) and then these predicates are used to generate test cases to verify programs. This approach is attractive because it deals with code directly rather than the state model in traditional model checking [2][7]. Thus, the BLAST approach is better suited for software verification than traditional model checking. However, this approach is not directly applicable to WS verification because WS providers are not required to provide the source code. Furthermore, the test cases generated in the BLAST approach are targeted mainly on the positive aspects of testing. Negative aspects such as near misses are not handled. In our approach, we extend the BLAST approach to suit WS testing in the following ways: (1)

Instead of using the source code to drive model checking, we use the OWL-S for model checking. The control flow automata used by BLAST resembles the workflow model derived by the control constructs in the process ontology of OWL-S. (2) We rely on the conditional or unconditional output, effect, and precondition of each atomic / primitive Web service to construct their essential inner control logic. (3) We enhance BLAST to handle concurrent executions of processes in OWL-S.

- Verification patterns: Testing temporal logic properties can be facilitated by the verification patterns technique [14]. This approach can generate many of test cases by recognizing patterns in system behavior and generate the corresponding test cases by composition. This approach has been used successfully to test medical devices and DoD applications in our previous projects.

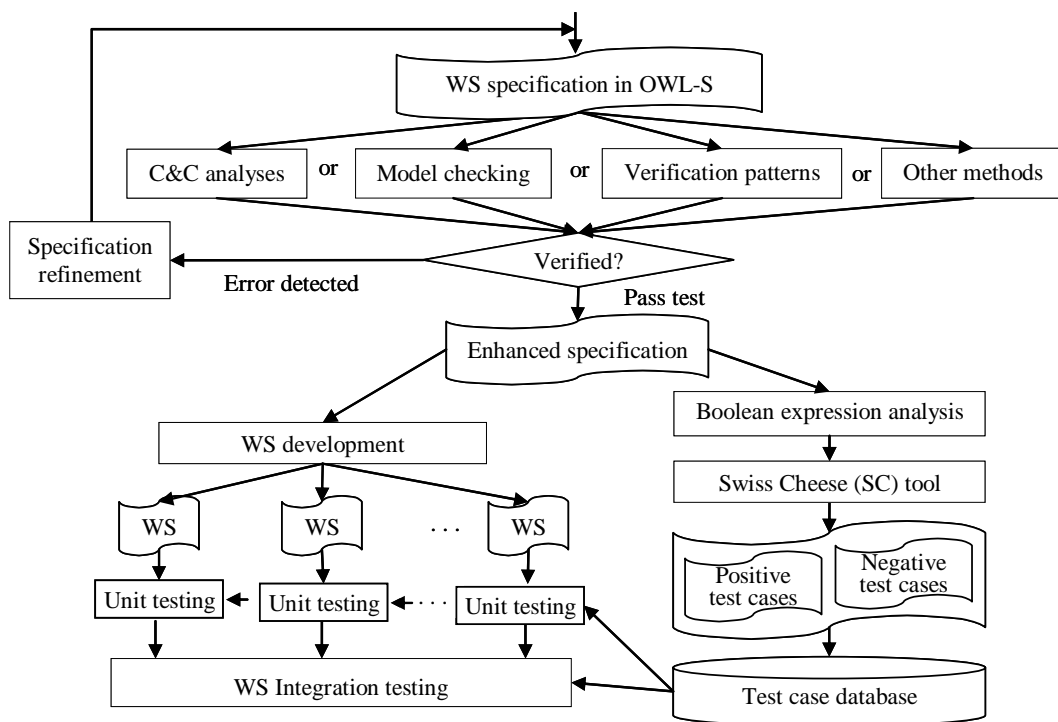


Figure 2. The integrated WS development process

- Test case generation: Test case generation techniques can be greatly enhanced by this comprehensive formal C&C checker followed by test case generation based on Boolean expressions [13]. An important distinction of this approach is that test case generation is based on quantitative Hamming distance. All previous approaches, including MC/DC and MUMCUT [3], were based on user experience and intuition. Exploring

the topological hypercube structure of Boolean expressions can easily reveal the faults not discoverable by previous approaches. Furthermore, these two mechanisms can be completely automated and thus saving significant effort and time. After the Boolean expression generation, the Swiss Cheese test case generation tool can be applied to obtain both positive and negative test cases.

- Swiss Cheese (SC) approach: An efficient iterative algorithm has been developed to identify most error-sensitive positive test cases and most critical negative test cases. Given the Boolean expressions that represent the system specification, the algorithm first maps the Boolean expressions into a multi-dimension Karnaugh map called **polyhedron**. The algorithm then iteratively identifies all boundary cells of the polyhedron and selects most error-sensitive test cases among all boundary cells. The more neighboring negative test cases (degree of vertex -- DoV) a boundary cell has, the more error-sensitive it is. The last step is post-checking, trying to identify critical negative test cases within the polyhedron. For each negative test case, the term Hamming Distance (HD) is used to define the minimum different Boolean digits between it and any boundary cells. The HD of all boundary cells is 0. The smaller the HD is, the more critical a negative test case is. It is shown in this paper that negative test cases can detect more failures. The

SC approach uses the most critical negative test cases first and then randomly chooses the remaining test cases.

4. A Case Study: Developing a Stock Query Web Service

This section uses a simple Stock Query WS as an example to explain the WS development and testing process outlined above. In this example, the WS under development and test consist of a server WS and multiple client WS, residing in different locations. A client can send requests to the server and the server responses to the requests.

4.1 The original specification

Table 1 list the original specification of the Stock Query WS. The WS Server offers two functions and Client WS can access these two function.

Table 1 Original specification of Stock Query WS

Systems	Index	Specification
WS Server specification	1	Upon receiving a request from a client WS for a ranked stock list, the server WS sends the ranked stock list to the requesting client if the client's address is available and the database (DB) is not empty.
	2	Upon receiving a request from a client to query a specific trade price, the server WS sends the specific trade price to the requesting client if the client's address is available and the specific trade price exists in the DB.
Client Side Specification	3	After a client sends a request for ranked stock list, it waits for the server WS to send back the list. The client should be able to view the ranked stock list on client side.
	4	After a client sends a request for a specified trade price, it waits for the server WS to send back the price. The client should be able to view the trade price on client side.
Integrated Specification	5	There are multiple client WS. Only one of them sends a request for a ranked stock list at the same time. If the client's address is available and the database (DB) is not empty, the server WS sends the ranked stock list to the requesting client. The client should be able to view the ranked stock list on client side.
	6	There are multiple client WS. Only one of them sends a request for a specific trade price at the same time. If the client's address is available and the specific trade price exists in the DB, the server WS sends the specific trade price to the requesting client. The client should be able to view the price on client side

4.2 C&C analysis

Following the process defined in Figure 2, the next step is to perform C&C analysis on the original Stock Query WS specification, repeatedly refines it until no uncovered conditions-event combinations exist.

The statistical results of C&C analysis on the original Stock Query WS is listed in Table 2. The number of missing condition-event (C-E) combinations represents how incomplete the specification is. The higher the number, the more incomplete the specification. The last

column explains whether current system is a whole system or only a part of it.

From Table 2 and we can see that the original specification is incomplete. For example, the Integrated Specification has as many as 248 missing C-E combinations. All those missing C-E combinations need to be handled, either being supplemented or tagged as *don't care* C-E combinations, which are supposed not to affect system behaviors.

After repeatedly C&C analysis and specification refinement, we acquire a complete Stock Query WS

specification. Perform C&C analysis on it and we can get the following results: (listed in Table 3)

Table 2 The results of C&C analysis on the original specification

Specification	# of Scenarios	# of Conditions	# of Missing C-E Combinations	# of Covered Scenarios	Whole/ Partial
WS Server specification	2	3	12	4	Whole
Client Side Specification	2	3	12	4	Whole
Integrated Specification	2	7	248	10	Whole

Table 3 The results of C&C analysis on the refined specification

Specification	# of Scenarios	# of Conditions	# of don't care C-E Combinations	# of Covered Scenarios	Whole/ Partial
WS Server specification	2	3	8	2	Whole
Client Side Specification	2	3	8	2	Whole
Integrated Specification	2	7	176	6	Whole

From Table 3 we can see that all missing C-E combinations have been supplemented based on C&C analysis results. For example, all the 72 missing C-E combinations are supplemented and only don't care C-E combinations are identified. The Stock Query WS specification is complete now.

4.3 Boolean expression analysis and Swiss Cheese vertex identification data

Following the process defined in Figure 2, the next step is to apply the Boolean expression analysis and SC tool on Stock Query WS specification and generate most error sensitive positive and most critical negative test case sets. In order to apply cross testing, we need to generate most sensitive and critical test case sets for both original specification and refined specification.

To automate the process the Boolean expression analysis and SC-based test case on the Stock Query WS, the specification needs to be converted into a formal format.

For example, the Integrated Specification is converted into the following seven conditions:

- C1: MultipleClientsSimultaneouslyQueryToStock
- C2: MultipleClientsSimultaneouslyRequestToRankStock
- C3: AllClientsSuccessfullyConnectsToServer
- C4: OneClientRequestToRankStock
- C5: DBNotEmpty
- C6: OneClientRequestToQueryStock
- C7: StockInfoIsAvailable

C&C analysis tool can automatically generate the scenario expression corresponding to the 5th refined specification item in Table 1, which is:

$$S = \overline{C1} \overline{C2} C3 C4 + C3 \overline{C4}$$

This scenario expression can be put into a Karnaugh-map styled SC diagram in Figure 3, where the items that can satisfy expression S are defined to have HD = 0 and thus the cells are marked 0 in the diagram. The items that

have HD = 1 to the items with HD = 0 are marked 1 in the diagram. The items that have HD = 2 to the items with HD = 0 are left blank.

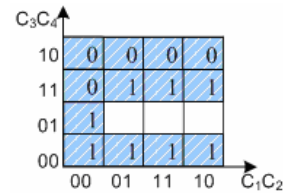


Figure 3. Swiss Cheese Diagram representing the test case set in Table 4

The SC diagram can be represented in Table 4, where values in the cells are the HD, cell corresponds to a test case that check if the scenario expression S is satisfied. Considering the topological hypercube structure of Boolean expressions degree of vertex (DoV) in the last column can be obtained [17].

Table 4 Test case set generated for the refined 5th specification in Table 1

HD	Cells (test cases) $C_1 C_2 C_3 C_4$	# of cells	DoV
HD = 0	0011	1	3
	0110, 1110, 1010	3	2
	0010	1	1
HD = 1	0111, 1011	2	2
	0001, 1111, 0000, 0100, 1100, 1000	6	1
HD = 2	0101	1	3
	1101, 1001	2	2
Total		16	

5. Analysis of Testing Results

Two WS have been developed based on original specification and the refined specification, respectively. Figure 4 illustrates the four testing combinations we performed:

1. Use the test cases generated from the original specification to test the original WS.

- Use the test cases generated from the original specification to test the refined WS based on the C&C and analysis.
- Use test cases generated from the refined specifications to test the original WS.
- Use test cases generated from the refined specifications to test the refined WS.

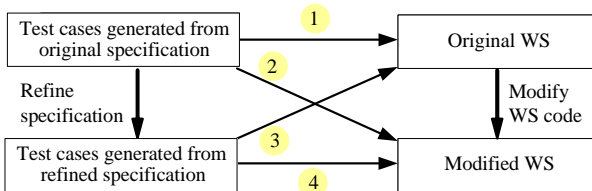


Figure 4. Four combinations of testing

Table 5 summarizes the cases used. The 1st column refers to the testing combinations. Testing combinations 1 and 2 adopt one set of test cases and testing approach 3 and 4 adopt another set of test cases. The 2nd and 4th columns are Hamming Distance (HD) and the degree of vertex (DoV). The 3rd column counts the total number of test cases with HD and DoV equal to the values in the 2nd

and 4th columns. The 5th column tags the type of the failed test cases -- negative or positive. The 6th and 7th columns count the number of positive and negative test cases used in positive and negative testing. The last column shows rate of the negative test cases used, is the number of negative test cases over the number of positive test cases.

Table 6 shows the pass/fail results of all the above four combinations of testing performed on Stock Query WS. From the table we can see that the failure rate of testing combinations 1 and 2 are the same, because we utilize the same test case set. The failure rate of testing combination 3 is much higher than that of testing combination 1 (78.6% VS 9.1%) because in testing combination 3 we used test cases generated from the refined specifications, with which we can detect more faults from the original WS. Those faults occur because the specification has been modified, but the code hasn't been updated accordingly. After modifying WS according to the refined specifications, we can see the failure rate drops to a lower level of 7.1%.

Table 5 Test cases used

Testing Combinations	HD	# of test cases	DoV	Type	# of positive test cases	# of negative test cases	Negative test case rate
1 & 2	HD = 0	6	1	Positive	6	38	86.36%
	HD = 1	18	1	Negative			
	HD = 2	20	2				
	HD = 2	2	2				
3 & 4	HD = 0	6	1	Positive	14	14	50%
		6	2				
		2	3				
	HD = 1	10	1	Negative			
		4	2				
Average							68.18%

Table 6 Test Results Statistical Data

Testing Approach	Server Side		Client Side		Integration		Total Pass	Total Fail	Failure Rate
	Pass	Fail	Pass	Fail	Pass	Fail			
1	8	0	4	4	28	0	40	4	9.1%
2	8	0	4	4	28	0	40	4	9.1%
3	2	2	0	4	4	16	6	22	78.6%
4	4	0	2	2	20	0	26	2	7.1%

Table 7 highlights the fault detection power of negative testing. The last column shows ratio between the failures detected by negative testing and the total number of failures detected by positive and negative testing. From the table 86.25% of failures are detected by negative testing!

6. Summary and Conclusion

The goal of this research is to remove the barrier that prevents WS being widely used in mission and business critical applications by improving their trustworthiness and reducing their vulnerability. We developed an integrated process with a set of automated tools to support the development of trustworthy WS. The

completeness and consistency tool can check WS specification and report inconsistent and missing conditions. The Boolean expression generation tool can extract Boolean expressions from the WS specification. The Swiss Cheese automated test case generation tool can generate both positive and negative test cases. The negative test case generation and negative testing is a major contribution of this paper that significantly reduce the vulnerability of WS. A simple Stock Query WS is

developed following this development process. For testing purpose, Two WS, based on the original specification and the refined specification, respectively, are developed and test cases are generated based both specification. The experiment results reveal that (1) the test cases generated from the refined WS specification can detect more faults that developed from the original specification; and (2) the negative test cases can detect more faults than positive test cases.

Table 7 Analysis of failed test cases

Testing combination	HD	# of failures	DoV	Type	# of failures under positive testing	# of failures under negative testing	failures detected by negative testing
1	HD = 1	2	1	Negative	0	4	100%
	HD = 2	2	2				
2	HD = 1	2	1	Negative	0	4	100%
	HD = 2	2	2				
3	HD = 0	4	1	Positive	12	10	45%
		6	2				
		2	3				
	HD = 1	6	1	Negative			
		4	2				
4	HD = 1	2	1	Negative	0	2	100%
Average							86.25%

References

- [1] D. Beyer, A. Chlipala, T. Henzinger, R. Jhala, and R. Majumdar, "Generating Tests from Counterexamples", Proceedings of the 26th International Conference on Software Engineering (ICSE'04), Scotland, UK, May 2004, pp. 326 – 335.
- [2] E. Clarke, O. Grumberg, and D. Peled, *Model Checking*, MIT Press, 2002.
- [3] T. Y. Chen and M. F. Lau, "Test Cases Selection Strategies Based on Boolean Specifications", *Software Testing, Verification and Reliability*, Vol. 11, No. 3, Sep. 2001, pp.165-180.
- [4] J. Clune and L. Chen, "Testing Web Services: Methods for Ensuring Server and Client Reliability" at <http://www.system.com/websphere/>
- [5] N. Davidson, "Testing Web Services" at <http://www.webservices.org/>, in October 2002.
- [6] T. A. Henzinger, R. Jhala, R. Majumdar, and G. Sutre, "Lazy Abstraction", In Proceedings of the 29th Annual Symposium on Principles of Programming Languages, 2002, pages 58-70.
- [7] G. Holtzman, "The Spin Model Checker," *IEEE Transactions on Software Engineering*, Vol. 23, No. 5, May 1997, pp. 279-295.
- [8] J. Myerson, "Testing for SOAP Interoperability" at <http://www.webservicesarchitect.com/>, Feb 2002.
- [9] Wim De Pauw, et al., "Websight Visualizing the Execution of Web Services", Workshop on Testing, Analysis and Verification of Web Services, Boston, MA, July 2004.
- [10] W. T. Tsai, R. Paul, Y. Wang, C. Fan, and D. Wang, "Extending WSDL to Facilitate Web Services Testing", *Proc. of IEEE HASE*, 2002, pp. 171-172.
- [11] W. T. Tsai, R. Paul, Z. Cao, L. Yu, A. Saimi, and B. Xiao, "Verification of Web Services Using an Enhanced UDDI Server", *Proc. of IEEE WORDS*, 2003, pp. 131-138.
- [12] W. T. Tsai, R. Paul, L. Yu, A. Saimi, and Z. Cao, "Scenario-Based Web Service Testing with Distributed Agents", *IEICE Transactions on Information and Systems*, 2003, Vol. E86-D, No. 10, 2003, pp. 2130-2144.
- [13] W.T. Tsai, Lian Yu, Feng Zhu and Ray J. Paul, "Rapid Verification of Embedded Systems Using Patterns", *COMPSAC 2003*: 466-471.
- [14] W. T. Tsai, R. Paul, L. Yu, X. Wei, and F. Zhu, "Rapid Pattern-Oriented Scenario-Based Testing for Embedded Systems", to appear in book *Software Evolution with UML and XML*, edited by H. Yang, 2004.
- [15] W. T. Tsai, D. Zhang, Y. Chen, H. Huang, Ray Paul, and Ning Liao, "A Software Reliability Model for Web Services", the 8th IASTED International Conference on Software Engineering and Applications, Cambridge, MA, November 2004, pp. 144-149.
- [16] W. T. Tsai, Y. Chen, Zhibin Cao, Xiaoying Bai, Hai Huang, and Ray Paul, "Testing Web Services Using Progressive Group Testing", *Advanced Workshop on Content Computing*, Zhenjiang, China, November 2004, pp. 314-322.
- [17] W. T. Tsai, X. Wei, L. Yu, R. Paul, and H. Huang, "Condition-Event Combination Covering Analysis for High-Assurance System Requirements", to appear in *Computer Systems Science & Engineering (CSSE) journal*.