

# Adaptive Testing, Oracle Generation, and Test Case Ranking for Web Services

Wei-Tek Tsai, Yinong Chen, Raymond Paul\*, Hai Huang, Xinyu Zhou, Xiao Wei

*Department of Computer Science and Engineering*

*Arizona State University, Tempe, AZ 85287-8809*

*\*Department of Defense, Washington DC*

## Abstract

*Web services and service-oriented architecture are emerging technologies that are changing the way we develop and use computer software. Due to the standardization of web services related description languages and protocols, as well as the open platforms, for the same web service specification, many different implementations can be offered from different service providers. This paper presents an adaptive group testing technique that can test large number web services simultaneously and effectively. Based on a case study, experiments are performed to validate the correctness and effectiveness of the technique.*

**Keywords:** Web services, service-oriented architecture, group testing, web services ranking, test case ranking

## 1. Introduction

Software development is shifting away from the product-oriented paradigm to the service-oriented paradigm. If the services are offered through Web and Internet technology, they are called Web Services (WS). Singh and Huhns described in [1] the new computing paradigm, major techniques, and their applications. Due to the open and public nature of Internet and WS, the same service specification, e.g., tax return service, stock ranking service, and equations-solving service, can be offered by many service providers, based on the same theories but different implementations.

A number of studies have been made to address the trustworthiness and dependability problem of WS through testing. In [2], WS testing technology is divided into three phases. In phase one, WS are essentially tested like ordinary software. In phase two (2003-2005), the following features should be included in testing: publishing, finding, and binding capabilities of an SOA (Service-Oriented Architecture); the asynchronous capabilities of WS; the SOAP (Simple Object Access Protocol) intermediary capability; and the quality of services. In phase three (2004 and beyond), the following features should be tested: dynamic runtime capabilities, WS versioning, and WS orchestration testing, which

invoke remote WS in a specific order to test their interoperability.

Davidson distinguished in [4] 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.

It was suggested in [5] that WS testing should include: basic WS functionality; SOAP messages; WSDL (WS Description Language) 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. Zhang discussed in [6] the differences between traditional software and WS testing, the existence of large number of WS available over the internet, and suggested to use mobile agent to automatically test remote WS by plugging the WS under test into a simulated environment in the mobile agent.

Huhns and Buell in [7] proposed peer-to-peer cooperative solutions based on the trusted autonomy among WS. Clune and Chen suggested in [8] 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 (Universal Description, Discovery, and Integration) registration, and performance considerations. Similarly, in [9], 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.

This paper proposes a technique to test large number of WS simultaneously, to determine the oracle and correctness of the WS under test by majority voting, and to provide quality ranking of WS and the test cases.

The proposed group testing technique can be used by WS service providers, brokers, and clients. A WS provider or client can use the technique to find the best WS for composing new services or applications. For example, a WS provider can compose a digital imaging using the Fast Fourier Transformation service as a component service. A WS broker can use the technique

to evaluate the quality of WS trying to be registered to make sure only WS with reasonable quality will be offered to the public.

The proposed group testing technique is not supposed to be used to rank different business logic or models. For example, several websites are available to rank products or services on the web based on price or customer satisfaction such as [www.pricingcentral.com](http://www.pricingcentral.com), [www.pricewatch.com](http://www.pricewatch.com), [www.dealtime.com](http://www.dealtime.com), [www.ebay.com](http://www.ebay.com), and [www.price.com](http://www.price.com). Instead, our techniques are used to rank different WS implementations based on the *same* specification, the *same* business logic, and the *same* input and internal states. In other words, the WS under group testing should produce the same or close results if the same inputs are applied, e.g., various Fast Fourier Transformation WS should produce the same or close results based on the same input. .

WS testing is different from traditional software testing and requires distributed and coordinated testing. Within a few years, for each particular service, an enormous number of WS will be available over the Internet. The technique proposed in this paper has the following advantages:

- It can test large number of WS rapidly and rank them according to the test results;
- It can automatically create the oracle of test cases, i.e., the expected outputs for the given inputs;
- It can rank the effectiveness of test cases and thus apply the most effective test cases first to eliminate unacceptable WS quickly; and
- Most of the steps in the process can be completely automated and this feature makes this process attractive for commercial applications.

The rest of this paper is organized as follows. Section 2 elaborates the two-phase testing algorithms of the adaptive group testing. Sections 3 and 4 outline the case study and the experiment results. Section 5 concludes the paper.

## 2. ASTRAR Group Testing

The Service-Oriented Architecture (SOA) based WS broker allows WS developers and providers, based on published WS specifications, to freely register WS and compose complex WS from other WS dynamically and at runtime. As a result, for each WS specification, many alternative implementations may be available. The question is: how can a WS broker test large number of WS efficiently at runtime? A group testing technique, originally developed for testing a large number of blood samples and later for software regression testing [13], is an attractive solution to address this problem. However, blood group testing is different from WS group testing in many ways and may not be applied directly. Blood group testing has been applied in diverse areas, primarily

to identify defectives in a homogeneous pool of items in a manner more efficient than individual testing (see [13], for example); WS group testing treats a pool of items differentiated by the service provided.

In this paper, we proposed a group testing-based technique called Adaptive Service Testing and Ranking with Automated oracle generation and test case Ranking (ASTRAR) to test a large number of WS efficiently and at runtime. ASTRAR can test a large number of WS at both the unit and integration levels. At each level, the testing process has two phases.

### 2.1 Phase 1: Training Phase

This phase is designed to test WS when the oracles (the expected output of the test input) and the effectiveness of the test cases are not available. The process assumes that a reasonably large number of test inputs or test cases are available to test the concerned WS before the start of this phase. This assumption is reasonable because the many techniques are available to generate test inputs from WS specifications [22][24]. In this phase, testing proceeds as follows:

1. Select a subset of WS randomly from the set of all WS to be tested. The size of the subset will be experimentally decided, as discussed in section 4.
2. Group testing: Apply each test case in the given set of test cases to test all the WS in the selected subset.
3. Voting: For each test input, the outputs from the WS under test are voted by a stochastic voting mechanism based on majority and deviation voting principles. The stochastic voting mechanism is studied in [23].
4. Failure detection and reliability computation: Compare the majority output with the individual output. A disagreement indicates a component failure. A dynamic reliability model is used to compute the reliability of each WS based on the failure rate and other factors [21].
5. Oracle establishment: If a clear majority output is found, the output is used to form the oracle of the test case that generates the output. A confident level is defined based on the extent of the majority. The confident level will be dynamically adjusted in the phase 2 as well.
6. Test case ranking: Test cases will be ranked according to their fault detection capacity, which is proportional to the number failures the test cases detect. In the phase 2, the higher ranked test cases will be applied first to eliminate the WS that failed to pass the test.
7. WS ranking: The stochastic voting mechanism will not only find a majority output, but also rank the WS under group testing according to their average deviation to the majority output.

By the end of training phase testing, we have tested the selected sample WS and we have the test cases ranked by their capability so far in detecting failures; the

oracle for test cases established with respect to their confidence levels; the sample WS ranked;

## 2.2 Phase 2: Volume Testing Phase

This phase continues to test the remaining WS and any newly arrived WS, based on the profiles and history (test case effectiveness, oracle, and WS ranking) obtained in the training phase. Phase 2 continues to rank the WS, rank test cases, and update the oracles. The testing process is outlined as follows:

1. Test cases have been ranked by their capabilities in detecting failures/faults in Phase 1. Now they are divided into layers, with layer one having the highest capability.
2. Select layer one test cases and apply them in the next step;
3. For each layer of test cases, group-test all the WS;
4. If an oracle with acceptable confident level (e.g., greater than 50%) exists, no voting is necessary: Use the oracle to detect failure: Determine if each WS has produced a correct answer and then compute the failure rate and possibly the reliability of each WS using the given reliability model;
5. If no oracle with acceptable confident level exists, use voting mechanism to detect failure, as described in phase 1.
6. Update the confident level of the oracles: an agreement between the oracle and the current test output increases the confident level, otherwise, decreases the confident level accordingly;
7. Update the ranking of test cases by including the new number of failures detected;
8. Update the ranking of WS and eliminate the WS that have an unacceptable level of failure rate or reliability. The elimination of unnecessary testing in this step saves testing time
9. Select next layer of test cases, and return to step 3.

By the end of Phase 2 group testing: all the WS available are tested and a short list of WS are ranked; test cases are updated and ranked; oracles and their confidence levels are updated.

The ASTRAR testing processed described above is based on unit testing of WS. However, the same processes can be applied at the integration testing level. Assume that a composite WS consists of  $n$  different units of WS. Each unit has a set of  $m_i$  functionally equivalent services. Assume that the  $n$  sets of unit WS are:  $\text{Set}_1 = \{S_{11}, S_{12}, \dots, S_{1m_1}\}$ ,  $\text{Set}_2 = \{S_{21}, S_{22}, \dots, S_{2m_2}\}$ , ...,  $\text{Set}_n = \{S_{n1}, S_{n2}, \dots, S_{nm_n}\}$ . A composite WS can be composed using one unit WS out of each set. Then, there are totally  $m_1 * m_2 * \dots * m_n$  possible composite WS. For example, a composite WS is a supply chain of a corporation and it is composed of six unit WS: ordering, retail, payment, transportation, lodging, and insurance. Assume there are 10 alternative WS for each unit WS, the total number of

possible composite WS is  $10^6$ . Please notice that we are discussing how to use find the best unit (component) WS to form a new composite WS. We are not talking about finding the best deal among different service providers.

Testing each possible combination will be expensive or even impossible. Thank to ranking in WS unit testing, one can choose the best unit WS to construct the composite services. If five unit WS are chosen as the candidates for each participating service of the composite services, the total number of possible composite services is then limited to  $5^6 = 15625$ , which is significantly smaller than  $10^6$ . Then, ASTRAR group testing technique can be applied to the composite WS by considering each composite as an individual WS in the group testing.

## 3. Case Study

This section uses a real-time stock-buy-sell WS as an example to illustrate the application of ASTRAR technique. In this example, the WS under development 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. All WS under group testing implement the same specification.

Table 1 lists the specification of the WS. The WS Server offers two functions and Client WS can access these two functions.

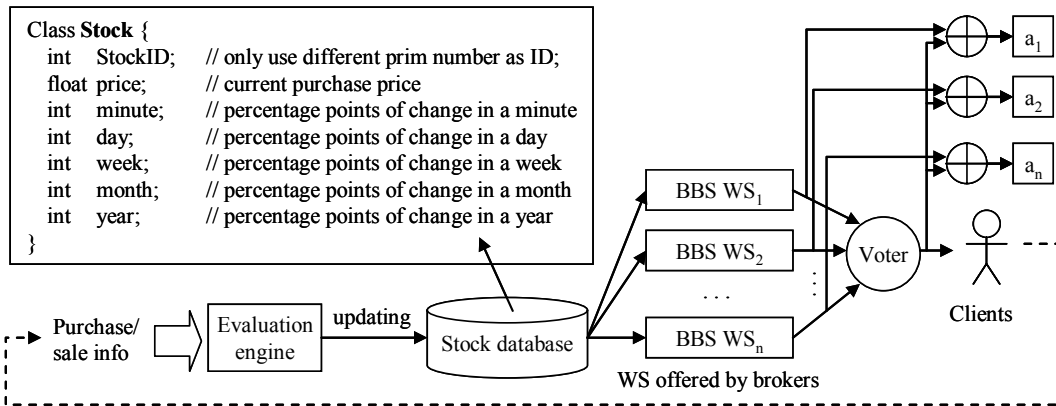
The simulation environment is shown in Figure 1. The database consists of objects of stock information, as defined in the Class **Stock**. Each stock object is set to an initial value at certain time point. The evaluation engine then uses randomly generated purchase and sale information, or uses replayed data from past stock dump, to decide the price dynamically once every minute. Once the price is changed, the other members (the percentages of changes in a minute, a day, a month, and a year) of each stock object are computed and updated.

## 4. Experiments and Results

ASTRAR technique divides testing into two phases. In phase 1 (training phase), a subset of WS is used to find the oracle (majority output) and to rank the fault detection capacity of the test cases. The size of the subset (training size) is critical. The smaller the size is, the cheaper (fewer test runs) the testing and ranking process will be. However, the smaller the size, the higher the probability that the training phase fails to find the correct oracle. An incorrect oracle will lead to incorrect ranking of the WS under test, while an incorrect ranking of test cases may result in more test runs in phase 2 of ASTRAR process. Another factor that affects the testing cost is the target size, the number of WS to be ranked. For a given large number of WS to be tested, only a short list of best WS needs to be ranked.

**Table 1.** Real-Time Stock-Buy-Sell WS

Event	Specification
(1) A client queries a stock's price	Client can query any stock's price. If queried stock name is not empty and requested stock information is available, the server WS sends the requested stock price to the requesting client.
(2) 20 minutes have past since last stock price checking	The service automatically checks stock prices every 20 minutes. If the prices of some stocks increase $\geq 5\%$ within the past 20 minutes, it will send messages to the all stock owners, reminding them to sell the stocks whose prices increase $\geq 5\%$ , or buy the stocks to sell at a higher price.
	If the prices of some stocks decreases $\geq 10\%$ within the past 20 minutes, the server WS will send messages to the stock owners, reminding them to buy the stocks whose prices decrease $\geq 10\%$ or sell them to stop further losses.
	If the advancing volume or declining volume of some stocks increases $\geq 100\%$ in the past 20 minutes compared to the same period of yesterday, it will send messages to alert the stock owners.



**Figure 1.** ASTRAR simulation environment

To explore the relationships among the training size, target size, and the test cost, we constructed 60 different WS implementations for the given specification in section 3, in which 12 WS are fault-free, 18 WS contains a single faults of different kinds, and the remaining 30 WS have multiple faults of different kinds. We have used the Swiss Cheese test case generation technique [22] to analyze the scenario specification and to generate test cases. Thirty-two test cases are generated that cover the fully functionality of the specification. Without using ASTRAR processes, the total number of test runs need to be executed is  $60 \times 32 = 1920$ . Using the ASTRAR process, only  $t$  WS will be tested by all 32 test cases, where  $t$  is the training size. Thus,  $32 \times t$  tests will be executed in the phase 1. In phase 2, the oracle (majority output for each test case) found in phase 1 will be use to determine the correctness of WS output. Those WS that can no longer enter the best  $k$  WS, based on their performance so far, will be immediately eliminated, where  $k$  is the target size. Figure 2 plots the experiment results of the 60 WS under 32 test cases, indicating the impact of the training sizes and the target sizes on the total testing cost in two phases. The cost shown in the

figure is the ratio of the test runs used to total number, 1920, of test runs if all 32 test cases are applied to test all 60 WS.

#### 4.1 Impacts of Target Size on Testing Cost

The 15 curves in Figure 2 correspond to 15 target sizes from 1 to 15. The lowest curve corresponds to size 1 and the highest curve corresponds to size 15. It can seen that

- The smaller the target size, the lower the cost. This is obvious because more WS can be eliminated sooner.
- The differences between the curves 1 to 12 are small, while a large gap exists between curves 12 and 13. The reason is that there are 12 fault-free WS under test. The number of failures detected from them is zero. If these fault-free WS are in the current target set, any WS will be eliminated if a single failure is detected.
- When the target size moves from 12 to 13 or higher, the testing cost increases sharply, because the algorithm must find a better WS among a set of imperfect WS.

#### 4.2 Impacts of Training Size on Testing Cost

In the experiments, the training size is set to be 1, 2, ..., 60. It can be seen from Figure 2:

- The smaller the training size, the lower the cost.
- When the training size is less than or equal to the target size, increasing the training size does not increase the cost (the initial part of the curves is flat). When the training size exceeds the target size, the cost increases as the training size increases. When the training size equals the total number of WS under test, it becomes exhaustive testing and no test runs can be saved.

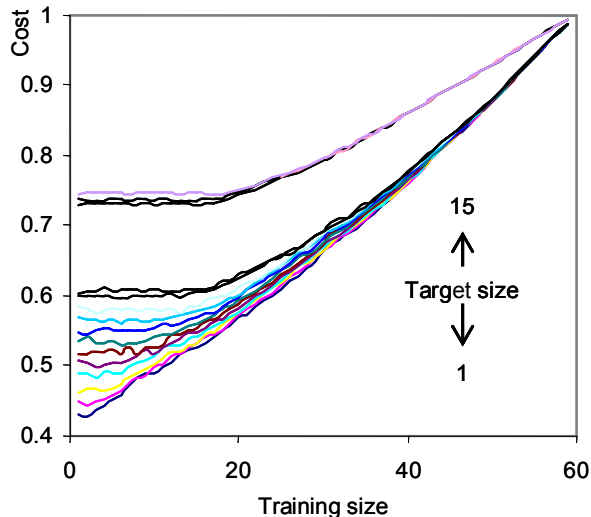


Figure 2. Impact of training sizes and target sizes

#### 4.3 The Impacts of Training Size on Oracle

Although Figure 2 suggests that a smaller training size leads to a lower cost, it does not take into account whether the correct oracle can be found in the training phase. An incorrect oracle can lead to incorrect test results or incorrect ranking of WS. Table 2 lists the experimental results on the probabilities that (1) a correct oracle is found in the training phase, (2) no oracle can be determined because insufficient data are available, and (3) an incorrect oracle is found, for training size = 1, 2, 3, 4, and 5. It can be seen that a small training size may lead to an incorrect oracle. The probabilities of "no oracle" value in the table are evaluated when the number of agreements does not exceed 50%. An incorrect oracle is established when a false majority is found. For the training size 1, an oracle is always found because there could not be a disagreement if only one WS is tested in the training phase. However, the probability of finding an incorrect oracle is high. Please notice that, although only 12 of the 60 WS are correct, the probability of finding a correct oracle for training = 1 is significantly higher than 12/60. The reason is that an incorrect WS does not always output an incorrect result.

Table 2. Probability establishing correct oracle

Training Size	Prob. of correct oracle	Prob. of no oracle	Prob. of incorrect oracle
1	76.9%	0.0%	23.1%
2	83.8%	16.3%	0.0%
3	98.1%	1.9%	0.0%
4	98.1%	0.6%	1.3%
5	98.1%	1.9%	0.0%

#### 4.4 The Impacts of Training on Test Case Ranking

Data were also collected for evaluating the impact of training size on the probabilities of finding the most powerful test case in terms of the number of faults it can detect. The most powerful test cases are used first in phase 2 to eliminate incorrect WS faster and thus saving test runs. Table 3 shows the results for training sizes 1, 2, 3, 4, and 5. It can be seen that a small training size has a higher probability of resulting incorrect test case ranking.

Table 3. Impact of training size on test case ranking

Training Size	Prob. of correct ranking	Prob. of no decision	Prob. of incorrect ranking
1	60.0%	0.0%	40.0%
2	35.6%	59.4%	5.0%
3	65.1%	21.5%	13.4%
4	71.5%	16.3%	12.2%
5	75.8%	15.2%	9.0%

### 5. Conclusions and Future Work

This paper proposed an efficient process to test a large number of web services designed based on the same specification. The process is divided in two phases. In phase 1 (training phase), a selected number of WS is tested and their results are voted. The purpose of the first phase is to establish the oracle and identify the most powerful test cases. In the phase 2, no voting is applied and the oracle created in phase 1 is used to judge the correctness of WS under testing. Furthermore, the powerful test cases are applied first, so that the incorrect WS can be eliminated in a few tests.

The experiment results reveal that the smaller the training size, the lower the cost. However, a small training size can lead to incorrect oracle, leading to incorrect WS ranking. A small training size can also lead to incorrect test case ranking, resulting a higher test cost in phase two. Therefore, it is critical to select a reasonable sized training size in WS group testing.

Further research and experiments are being performed to explore the impact of the age of the test cases. New WS may have been designed and test using the known test cases. We have defined an adaptive window to limit the age of test cases being used in group testing.

## References

- [1] M. P. Singh, M. N. Huhns, *Service-Oriented Computing*, John Wiley & Sons, 2005.
- [2] J. Bloomberg, "Web Services Testing: Beyond SOAP," ZapThink LLC, Sep 2002, <http://www.zapthink.com>.
- [3] C. J. Colbourn, Y. Chen, W. T. Tsai, "Progressive Ranking and Composition of Web Services Using Covering Arrays," Proc. of IEEE WORDS, Sedona, February 2005.
- [4] N. Davidson, "Testing Web Services", <http://www.webservices.org/>, October 2002.
- [5] B. De, "Web Services - Challenges and Solutions," WIPRO white paper, 2003, <http://www.wipro.com>.
- [6] J. Zhang, "An Approach to Facilitate Reliability Testing of Web Services Components," 15th International Symposium on Software Reliability Engineering (ISSRE) Nov. 2004, pp. 210 – 218.
- [7] M.N. Huhns, D.A. Buell, "Trusted autonomy," IEEE Internet Computing, Volume 6, Issue 3, May-June 2002, pp. 92 – 95.
- [8] J. Clune and L. Chen "Testing Web Services: Methods for Ensuring Server and Client Reliability," <http://www.sys-con.com/websphere/>
- [9] J. Myerson, "Testing for SOAP Interoperability," <http://www.webservicesarchitect.com/>, Feb 2002.
- [10] D. Beyer, A. Chlipala, T. Henzinger, R. Jhala, R. Majumdar, "Generating Tests from Counterexamples," Proceedings of the 26th International Conference on Software Engineering (ICSE'04), Scotland, UK, May 2004, pp. 326 – 335.
- [11] A. Brogi, C. Canal, E. Pimentel, A. Vallecillo, "Formalizing Web Services Choreographies," Electronic Notes in Theoretical Computer Science, [www.elsevier.nl/locate/entcs](http://www.elsevier.nl/locate/entcs).
- [12] F. Coehn, *Testing Web Services*, McGraw-Hill Osborne Media, 2003.
- [13] D. Z. Du and F. Hwang, *Combinatorial Group Testing And Its Applications*, World Scientific, 2nd edition, 2000.
- [14] M. Lyu (Ed.), *Handbook of Software Reliability Engineering*, McGraw-Hill, 1996.
- [15] 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.
- [16] C. V. Ramamoorthy, F. B. Bastani, "Software reliability — Status and Perspectives," IEEE, Trans. Soft. Eng., SE-8, No. 4, July 1982, pp. 354 – 371.
- [17] 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.
- [18] W. T. Tsai, R. Paul, Z. Cao, L. Yu, A. Saimi, B. Xiao, "Verification of Web Services Using an Enhanced UDDI Server," Proc. of IEEE WORDS, 2003, pp. 131-138.
- [19] 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.
- [20] W.T. Tsai, Y. Chen, R. Paul N. Liao, and H. Huang, "Cooperative and Group Testing in Verification of Dynamic Composite Web Services," in Workshop on Quality Assurance and Testing of Web-Based Applications, September 2004, pp. 170-173.
- [21] W. T. Tsai, D. Zhang, Y. Chen, H. Huang, R. Paul, N. 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.
- [22] W. T. Tsai, X. Wei, Y. Chen, B. Xiao, R. Paul, and H. Huang, "Developing and Assuring Trustworthy Web Services," 7th IEEE International Symposium on Autonomous Decentralized Systems (ISADS), April 2005, pp. 43-50.
- [23] W.T. Tsai, Y. Chen, D. Zhang, H. Huang., "Voting Multi-Dimensional Data with Deviations for Web Services under Group Testing," the 4th International Workshop on Assurance in Distributed Systems and Networks (ADSN), Columbus, Ohio, June 2005.
- [24] W. Xu, "Generating Test Cases for Web Services Using Data Perturbation," Workshop on Testing, Analysis and Verification of Web Services, Boston, MA, July 2004, pp. 144-149.
- [25] T. Tsai, W. Xie, I. A. Zualkernan, S. K. Musukula, "A Framework for Systematic Testing of Software Specifications," in Proc. of International Conference on Software Engineering and Knowledge Engineering, 1993, pp. 380-387.
- [26] W. T. Tsai, R. Paul, L. Yu, X. Wei, "Rapid Pattern-Oriented Scenario-Based Testing for Embedded Systems," in *Software Evolution with UML and XML*, edited by H. Yang, Idea Group Publishing, London, 2005, pp. 222-262.
- [27] W. T. Tsai, L. Yu, F. Zhu, R. Paul. "Rapid Verification of Embedded Systems Using Patterns," Proceedings of IEEE Annual International Computer Software and Applications Conference, 2003. pp. 466-471.
- [28] W. T. Tsai, F. Zhu, L. Yu, R. Paul, and C. Fan, "Verification Patterns for Rapid Embedded System Verification," in Proceedings of International Conference on Embedded Systems and Applications (ESA), 2003. pp. 310-316.
- [29] W.T. Tsai, Y. Chen, R. Paul, "Specification-Based Verification and Validation of Web Services and Service-Oriented Operating Systems," Proc. of IEEE WORDS, Sedona, February 2005.