A SOFTWARE RELIABILITY MODEL FOR WEB SERVICES

W. T. Tsai, D. Zhang, Y. Chen, H. Huang, R. Paul*, N. Liao
Department of Computer Science and Engineering, Arizona State University
Tempe, AZ 85287-8809, U.S.A.
*Department of Defense, Washington, DC

ABSTRACT

This paper proposes a service-oriented software reliability model that dynamically evaluates the reliability of Web services. There are two kinds of Web services: atomic services without the structural information and the composite services consisting of atomic services. The model first evaluates the reliability of atomic services based on group testing and majority voting. Group testing is the key technique proposed in this paper to support the service-oriented reliability model. Then, the reliability model evaluates the overall reliability of composite services using an architecture-based model and based on reliabilities of the atomic services, execution scenarios, and operational profiles. The reliability model is dynamic and the reliabilities of the services are evaluated in the actual operational environment. A case study is designed, implemented, and analyzed using the design of experiment technique. The results show the significances of the model and its components.

KEY WORDS

Service-oriented architecture, Web services, reliability model, and group testing

1. Introduction

Software reliability has been defined as the probability that no failure occurs in a specified environment during a specified (continued) exposure period. For some programs the appropriate time unit of exposure period is the calendar or CPU time, and for some other programs the appropriate time unit of exposure period is an application run corresponding to a selection of an input case (a vector of input values needed to make an application run) from the input case domain (ICD) of the programs. For the evaluation of software reliability there exist different kinds of reliability models. [1] classifies software reliability models according to the development phases of software life-cycle. Since fault corrections are necessary in the testing and debugging phase of the life-cycle, reliability growth models which take fault corrections into account are mainly used in this phase. The models without dealing with fault corrections, say input domain models, can only be applied in this phase by treating the program after each correction as a new program.

Goel [2] categorized software reliability models according to the nature of failure process into four types: times-between-failures models, failure-count models, fault seeding models, and input domain based models. The times-between-failures models and failure-count models are usually applied in the iterations of the testing and debugging phase in the software development process based on the fault corrections and reliability history. It is normally assumed that the reliability of software increases with the removals of faults in the software and thus these types of models are also called reliability growth models. On the other hands, fault seeding models and input domain based models do not consider the fault correcting activities. They are usually used at the end of the development cycle to assess the final reliability of software.

All existing software reliability models are developed for the software products that are statically constructed, normally by a company or institution that has the full control of the development process.

The evolutional shift from the product-oriented software architecture to the service-oriented architecture (SOA) and Web services (WS) invalids many techniques developed for traditional software, including the software reliability models.

Under SOA and WS, a system consists of a collection of loosely coupled services [3]. These services can make use of each other's services to achieve their own desired goals and end results. Simple services can cooperate in this way to form a complex or composite service dynamically and at runtime.

A few practical attempts have been made to address the testing problems of WS. In [4], De suggested that WS testing should include basic WS functionality, SOAP messages, WSDL files, the publishing, finding, binding capabilities of a SOA, the 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 [5], Bloomberg suggested to develop WS testing technologies in three phases. In phase one, WS are mainly tested like the ordinary software. In phase two (2003-2005), the following features should be included in testing: SOA, the publishing, finding, and binding capabilities of a SOA, the asynchronous capabilities of WS, the SOAP intermediary capability, and the Quality of services. In phase three (2004 and beyond) following features should be tested: dynamic runtime capabilities, WS orchestration testing, and WS versioning.

IBM's Websight is a test and debugging tool under its WS framework [15], which 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 cooperation with industry and government agencies, we developed in the past few years techniques related to testing and verification of WS. In [6], real-time scheduling algorithms were proposed in a service-oriented trustworthy environment. In [7], a variety of test generation techniques are proposed to test WS in an enhanced UDDI-based service broker. These test scripts can be arranged hierarchically to test domains of related WS. An early form of WS check-in and check-out processes is also proposed to increase the confidence of WS used.

In [8],[9],[10], rapid testing techniques were developed, including group testing, regression testing, and pattern-based verification.

In [11], WS scenarios are developed in stages, the first stage the individual service scenarios are developed, in the second stage, interaction among WS are modeled, and finally the overall scenarios are developed combining previously developed scenarios. These scenarios are then translated into test scripts to be performed by organized distributed agents.

In [12], an enhanced WSDL interface was developed to include dependency information, functional description, invoking, and concurrent sequence specifications so that test cases/scripts can be generated based on WSDL descriptions.

This paper proposes a Service-Oriented software Reliability Model (SORM). This model evaluates the reliability of WS in two steps: (1) Use highly efficient group testing to evaluate the reliability of atomic services. An atomic service is a service agent submitted by a service provider that does not call other WS and thus should be treated as a unit that is not to be broken, thus atomic. (2) Evaluate the reliability of a composite service based on the reliabilities of the component services (they can be atomic services or composite services) and the structure (relationships) among the component services. Both evaluation steps are dynamic and at runtime.

The rest of the paper is organized as follows. Section 2 outlines the overall architecture, the group testing scheme, and the reliability model for atomic services. Based on the reliability model for atomic services and the structure and relationship among the component services, section 3 derives the reliability model for composite services. Section 4 applies the reliability models a WS example and show the experiment results. Section 5 concludes the paper.

2. Group Testing For Reliability

The SORM is a part of our project to establish an infrastructure for Web Services Testing, Reliability Assessment, and Ranking (WebStrar) supporting the development and application of WS. The WebStrar and its user groups are shown in Figure 1.

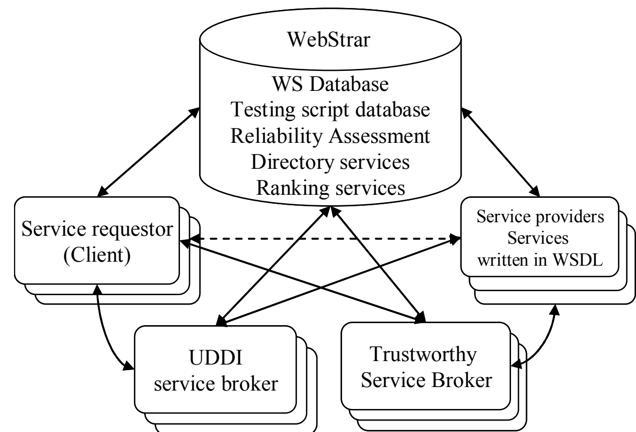

Figure 1. WebStrar and its user groups

The WebStrar can take registration from service providers and various kinds of service brokers. It considers the services registered as atomic services and uses them to compose composite services. Both atomic and composite services can be provided by the WebStrar directly to the clients.

To ensure the quality and trustworthiness of services, WebStrar will perform rigorous check-in test on all services registered to WebStrar and take the responsibility for all services it offers. In 0, the solid lines indicate explicit calls of services while the dotted line indicates that, although data and results can transfer back and forth between a client and a service provider, the operations can be transparent to the client.

Figure 2 illustrates the part of the system in WebStrar that performs group testing, component reliability evaluation, and composite reliability evaluation, which is the core topic of this paper.

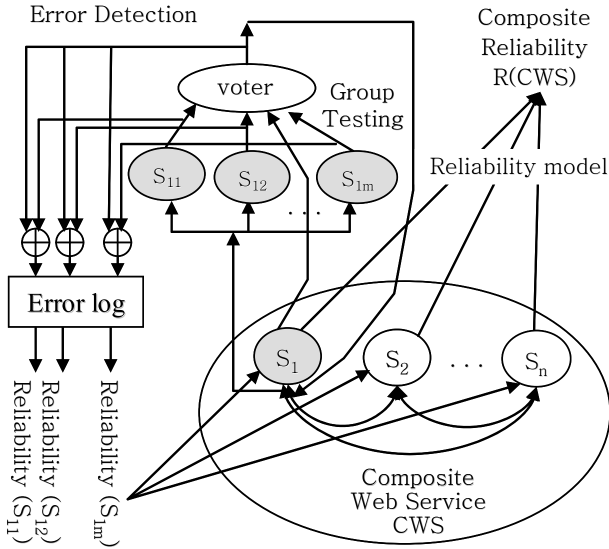

Figure 2. Group testing for reliability

Group testing technique [13] was originally developed for testing large samples of blood and this paper uses it to test complex composite WS at runtime. It tests the contamination of an entire group of services by applying one test. Assume CS_n is a composite service consisting of n services S_1, S_2, ..., S_n, where S_i can be an atomic service or a composite service. While CS_n is performing services, many atomic services could be registered. Assume services S_{11}, S_{12}, ..., S_{1m} are functionally equivalent to the service S_1 in CS_n. We can forward (broadcast) the input to S_1 to S_{11}, S_{12}, ..., S_{1m}, as shown in figure 2. The results from all services, including that from S_1, are voted by a voting service. The voting is weighted based on the current reliabilities of the services under test. The voting service can set the initial weight of each incoming service to zero while the exiting service S_1's weight to the reliability $R(S_1)$. The voting service detects faults by comparing the output of each service with the weighted majority output. A disagreement indicates a fault. Assume the reliability of service S at time point t is $R(S, \Delta t)$. In the next Δt time, k runs are executed and f disagreements have been detected, then the reliability $R(S, t + \Delta t)$, at time $t + \Delta$t is calculated by

$$R(S,t+\Delta t)=\frac{M-k}{M}R(S,t)+\frac{k}{M}R(S,\Delta t) \qquad (1)$$

where, M is the total number of tests that the service has ever been tested. Since $R(S, \Delta t)$ can be estimated by $1 - f/k$, the above formula is converted to

$$R(S,t+\Delta t)=\frac{M-k}{M}R(S,t)+\frac{k-f}{M} \qquad (2)$$

The formula (2) is the reliability model for atomic services under group testing. The model takes four parameters. The first two parameters $R(S, t)$, the current reliability, and M, the total number tests performed on S so far, represent the state at time point t. The next two parameters, k and f, represent the reliability behavior in the next Δt time, and are used to adjust the reliability at time point $t + \Delta t$.

The novelty and advantages of the model include:

- One of the toughest problems in software testing is to construct an oracle that can determine if a fault has occurred. In this model, the voting service serves as the oracle according to the majority principle.
- The model estimates the reliability of each incoming service while performing the normal operation. In other words, the incoming services are tested in the real operational environment at no extra time if sufficient computing power is available. A remote test agent can take spare computers to perform distributed testing was proposed to address this problem [7].
- The model is dynamic, i.e., the data are collected and computed at runtime in real time. The reliability of each service involved in group testing is updated after each run or after a given period of time.

One situation in which SORM would not work well is when there are no alternative services available. In this case, the SOA is basically degraded to the traditional software architecture: The service is only tested by the service provider in its development cycle. However, this is an unlikely situation because SOA is an open platform that allows and encourages cooperation and competition among service providers to create increasingly improved services.

3. Architecture-Based Reliability Model

Section 2 proposed a scheme to determine the reliability of atomic services. This section proposes techniques to evaluate the reliability of a composite WS consisting of atomic WS and other composite WS. The model can be generalized to evaluate any system, or a sub system in a system, with the knowledge of the reliability of its components which could be atomic or a subsystem itself, operational profile, and the architecture of the system. The architecture determines the contribution of the reliability of each atomic component to that of the system. Hence the approach is named as architecture-based reliability model.

First we give the definition of the component's reliability and present our assumptions; then we discuss on how the architecture affects the propagation of reliability; Finally, we derive the mathematics formulas that compute the reliability.

We base our reliability model on ACDATE/Scenario model which is generic specification language and can be used to express complex computing process.

ACDATE/Scenario model describes the components of a system with *actors*, *conditions*, *data*, *actions*, *timing attributes*, and *events*; and the behavior of the system with *scenarios*.

The *reliability* definitions of ACDATE entities and scenarios are summarized in Table 1.

The assumptions of our reliability are

1. Assignment assumption: The assignment operation introduces no new failure.
2. Condition assumption: The condition fails when any data that constitutes the condition fails.
3. Acyclic dependency assumption: There is no cyclic dependency among ACDATE entities.

ACDATE	Reliability of the ACDATE entity
Actor	The *probability*, in the period from [0, t], that the: - actor presents the expected behavior
Condition	- condition presents the expected Boolean value
Data	- data presents the expected value
Action	- action presents the expected behavior
Timing	- action completes in given time frame
Event	- event is sent or received successfully
Scenario	- scenario presents the expected behavior
System	- system presents the expected behavior

Table 1. Reliability definition of ACDATE entities

The system behaviors are specified by a few system level scenarios and the reliabilities of those system level scenarios will contribute to that of the system. Different scenarios may have different execution rates (the operational profile), which determine the weight of the contributions.

A scenario is a sequence of activities connected by the four operators: *sequence*, *choice*, *loop* and *concurrency*. Each *activity* is a data assignment, exchanging an event, doing an action, or executing a sub-scenario. Hence the reliability of data, events, actions and sub-scenarios will contribute to that of a scenario. The choice and loop operator are associated with one or more conditions, which determine the branches to take. Hence the reliability of conditions also contributes to that of a scenario. Moreover, the *true* / *false* rate of each condition will affect the reliability of the scenario through the choice and loop operators (mathematics formulas will be presented later).

Each top level scenario would be invoked by an *external* event. Hence the occurrence rates of external events determine the operational profile of top level scenarios. A scenario may emit *internal* event, whose sole function is to resume or invoke the execution of a sub-scenario. Hence the occurrence rates of internal events will affect the operational profile of sub-scenarios (in addition to direct calling from other scenarios). The occurrence rates of internal events can be determined by that of external events invoking top level scenarios, and the control flow of those scenarios that emit internal events.

To summarize, assume we know the reliability of each scenario and the occurrence rate of each external events (hence internal events), we can evaluate the reliability of the system following the formula: $Rel_{system} = (\sum(w_i * Rel_{scenario_i}))/\sum(w_i)$, where w_i is the execution rate of the corresponding scenario. In the following we present the calculation of the reliability of a scenario.

The reliability of data is determined by that of the storage, which is modeled as atomic components (memory, external database, file system, etc), and hence is known. The reliability of actions is known if it is atomic (e.g., a system call) or is that of the sub-scenario that implements it. The reliability of events is determined by that of the communication link (atomic component) and hence is known.

Following Assumption 1, the reliability of assignment is that of the right hand side data. Hence we know the reliability of each activity in a scenario.

If several activities are connected by sequence operator, then the overall reliability follows the formula:

$$Rel_{sequence} = \prod Rel_{activity_i}$$

where $Rel_{activity_i}$ is the reliability of each activity that participates the sequence. If several activities are connected by concurrency operator and all of them are replicas, then the overall reliability follows the formula:

$$Rel_{concurrency} = 1-\prod(1-Rel_{activity_i})$$

Otherwise, it is the same with sequence since any failure fails the overall concurrency. For loop operator, the formula is:

$$Rel_{loop} = (Rel_{cond_set} * Rel_{block})^{Pt}$$

where Rel_{cond_set} is the reliability of the condition set associated to the loop operator, Rel_{block} is the reliability of the block of activities enclosed in the loop operator, and the Pt is the expected times of loop. We will discuss the Rel_{cond_set} later. For choice operator, the formula is

$$Rel_{choice} =$$

$$Rel_{cond_set} * (Pt * Rel_{true_block} + (1 - Pt) * Rel_{false_block})$$

where Pt is the probability of that the condition set evaluates to be *true*. Since a scenario consists of only the four types of operators, we can calculate its reliability following above formulas.

The reliability of conditions is determined by the data that constitute the condition or is known if the condition is atomic (e.g., a system call). Following Assumption 2, if a condition is composed of several data, its reliability is the product of that of all the data. We omit the deduction

process due to the space restriction and only present the final formulas here:

Rel_{cond_set} = 1-$\sum ProTop_{c(m,o)}$, where each $ProTop_{c(m,o)}$ = $\sum ProTop_{\{c\}o}$

Each $ProTop_{\{c\}o}$ is calculated by the following formulas:

$ProTop_{\{c\}o} = Rel_{\{c\}o} * (ProTF_{\{c\}o} + ProFT_{\{c\}o}) * Prob_{remv}$

$Rel_{\{c\}o} = \prod(1-Rel_{Ck}) * \prod(Rel_{Cl})$

where Rel_{Ci} is the reliability of the *i*-th condition in the condition set. Each condition set is evaluated in Disjunction Normal Form (DNF), whose Boolean value may be dominated by a true disjunct. $Prob_{remv}$ is used to compensate the possible domination and is defined as the probability that the disjunct evaluates to be false. $ProTF_{\{c\}o}$ and $ProFT_{\{c\}o}$ are probabilities of a condition falsely turns from *true* to *false* and turns from *false* to *true*, respectively, which are determined by the condition's reliability.

4. Application of Reliability Models

This section uses examples to illustrate the applications of the proposed service-oriented reliability model. Assume a space agency plans to launch a satellite on a specific date and from a specific location. The launch is heavily depends on the weather conditions in the launch location, including rain, wind, and temperature. We designed 10 independent weather services, each of which offers three component services, RainForecast TempForecast, and WindForecast. The forecasts are given their probabilities.

4.1 Evaluation of component Services

To build a trust on the reliability of the component services, the space agency puts them in a group testing framework, as shown in figure 2, and sets their initial reliability to zero. After a period of group testing, the space agency has the reliability estimation of each service. Table 2 shows a set of sample results obtained in our experiments.

Component Service	Reli-ability	Forecast Probability	Adjusted Prob.
RainForecast	0.764	18% heavy rain	33.1%
TempForecast	0.98	31% extreme temp	31.76%
WindForecast	0.90	23% strong wind	28.4%

Table 2. The most reliable services and their forecast

The first column of the table lists the component services under test. The second column shows the highest reliability of the service in the given test period. Column 3 shows the forecasted probabilities of heavy rain, extreme temperature, and the strength of the wind, respectively, of the component services. Taken the reliability of the service into account, the adjusted forecast probabilities from the service are given in column 4, which are the final evaluation values for the component services.

4.2 Evaluation of Composite Services

To base the decision whether to change the launch date on the most accurate whether forecasting information, the space agency then constructed a composite service, as shown in Figure 3. The numbers in the diamond boxes are the reliability of the best component services. The numbers on the branches are the probabilities forecasted by the best service. The decision is based on these two factors.

Note that the reliabilities and probabilities given in the composite service in 0 is an instance in a sequence of dynamic evaluations. Assume that the plan of launch is made a year before the launch date. The composite service in 0 is up and running from day one. At the beginning, the space agency would have little data about the reliability of each service and the weather forecast a year before the launch date won't be accurate too. However, by the time, say a month or a week before the launch, we already have sufficient data about the reliability of the services. These reliability data will be used in the future applications too. When the agency plans its next launch, or another event that needs weather forecast, it already has the reliability data.

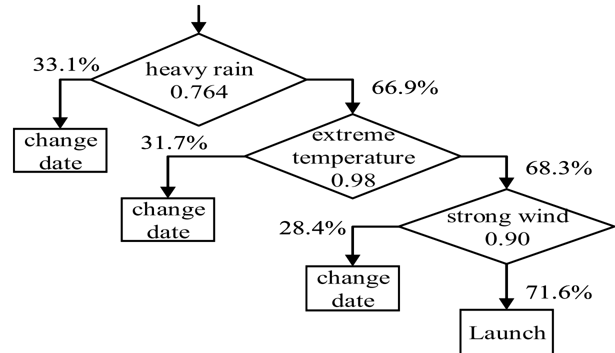

Figure 3. Decision making service

4.3 Design of Experiment Analysis

Design Of Experiment (DOE) is an engineering technique [14] that can be used to determine the extend of the impact of the parameters (factors) of a model on the final results.

This section applies DOE to analyze the impact of the reliability of the component services on the reliability of the composite service.

There are three factors in our example, the reliabilities of (A) RainForecast, (B) TempForecast, and (C) WindForecast. We use 2 level DOE techniques, i.e., use high and low values of each factor: RainForecast (70%, 90%), TempForecast (90%, 99%) and WindForecast (85%, 95%). In our experiment, the 3-factor and 2-level design generated an ANOVA (ANalysis Of VAriance) table in Table 3.

The F-Value represents the significance of the impact of a model and its components. In general, if a componet generates a significance value “Prob>F-Value” of less than 0.05, the impact of the componet is significant. For example, the F-Value 1.421E+5 for the overall Model in the table implies the Model is significant. There is only 0.01% of chance that the Model's F-Value of this size could occur due to noise.

Source	F-Value	Prob>F-Value
Model	1.421E+5	< 0.0001
A (RainForecast)	*3.898E+5*	*< 0.0001*
B (TempForecast)	*2.2943E+4*	*< 0.0001*
C (WindForecast)	*1.3558E+4*	*< 0.0001*

Table 3. ANOVA Significance analysis

The experiment results in Table 3 also show that the F-Values and significances of RainForecast, TempForecast, and WindForecast are all less than 0.0001, and thus they are all significant model components.

DOE can be used to compare the significances among the components. Figure 4 shows the impacts of the three component services on the overall reliability in our example. As can be seen that the higher the component reliability, the higher the overall reliability. However, the imapact of the RainForecast service is much more significant than that of the others. This suggests that the space agency should pay more attention to the quality of rain-forecast service-provider.

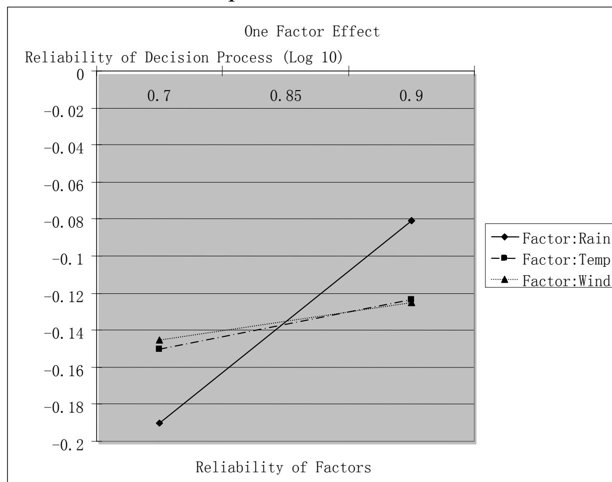

Figure 4. Impact of components

5. Conclusion

We proposed the first software reliability model that can be used to evaluate the reliability of both atomic and composite WS. The evaluation process is dynamic and at runtime. The vastly available WS on-line make it necessary to perform group testing, which, in turn, makes it possible to identify the correct service output without having to design an oracle. Initial implementation and experiments on the model have been performed to prove the concept. The model is being integrated into our project on building an infrastructure that supports the development and applications of WS.

References:

[1] C. V. Ramamoorthy, F. B. Bastani, Software Reliability — Status and Perspectives, *IEEE, Trans. Soft. Eng.*, *SE-8*, No. 4, July 1982, 354 - 371.

[2] A. L. Goel, Software Reliability Models: Assumptions, Limitations, and Applicability, *IEEE, Trans. Soft. Eng.*, *SE-11*, No. 12, Dec. 1985, 1411 – 1423.

[3] Andrew D. Gordon, Riccardo Pucella, Validating a Web Service Security Abstraction by Typing, *Microsoft MSR-TR-2002-108*, December 2002.

[4] B. De, Web Services - Challenges and Solutions, *WIPRO white paper*, 2003, http://www.wipro.com.

[5] J. Bloomberg, Web Services Testing: Beyond SOAP, *ZapThink LLC*, Sep 2002, http://www.zapthink.com

[6] Y. Chen, A Service Scheduler in a Trustworthy System, *Proc. of the 37th Annual Simulation Symposium*, Arlington VA, April 2004, 89-96.

[7] 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, 131-138.

[8] A. K. Onoma, W. T. Tsai, M. Poonawala, and H. Suganuma, Regression Testing in an Industrial Environment, *Communications of ACM, Vol. 41*, No. 5, 1998, 81-86.

[9] W.T. Tsai, Lian Yu, Feng Zhu, R. Paul, Rapid Verification of Embedded Systems Using Patterns, *Proc. of IEEE COMPSAC 2003*, 466-471.

[10] W.T. Tsai, Y. Chen, R. Paul, N. Liao, H. Huang, Cooperative and Group Testing in Verification of Dynamic Composite Web Services, *Workshop on Quality Assurance and Testing of Web-Based Applications, in conjunction with COMPSAC, September 2004*.

[11] 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, Vol. E86-D*, No. 10, 2003, 2130-2144.

[12] W. T. Tsai, R. Paul, Y. Wang, C. Fan, and D. Wang, Extending WSDL to Facilitate Web Services Testing, *Proc. of IEEE HASE*, 2002, 171-172.

[13] D.Z. Du and F. Hwang, *Combinatorial group testing and its Applications* (World Scientific, 2nd edition, 2000).

[14] D. C. Montgomery, *Design and Analysis of Experiments* (Hardcover, 5th Edition, 2000).

[15] Wim De Pauw, et al., Visualizing the Execution of Web Services, *Workshop on Testing, Analysis and Verification of Web Services*, Boston, MA, July 2004.