A Metamorphic Relation-Based Approach to Testing Web Services Without Oracles

Chang-ai Sun, University of Science and Technology Beijing and Chinese Academy of Science, China

Guan Wang, University of Science and Technology Beijing, China

Baohong Mu, University of Science and Technology Beijing, China

Huai Liu, Swinburne University of Technology, Australia

ZhaoShun Wang, University of Science and Technology Beijing, China

T. Y. Chen, Swinburne University of Technology, Australia

ABSTRACT

Service Oriented Architecture (SOA) has become a major application development paradigm. As a basic unit of SOA applications, Web services significantly affect the quality of the applications constructed from them. In the context of SOA, the specification and implementation of Web services are completely separated. The lack of source code and the restricted control of Web services limit the testability of Web services, and make the oracle problem prominent. In this context, can one alleviate the test oracle problem, or effectively and efficiently test such Web services even without oracles? It is an important issue which has not been yet adequately addressed. To address the challenge of testing Web services, the authors propose a metamorphic relation-based approach to testing Web services without oracles. The proposed approach leverages so-called metamorphic relations to generate test cases and evaluate test results. To make the proposed approach practical and effective, the authors proposed a framework taking into account the unique features of SOA, and developed a prototype which partially automates the framework. Three case studies are conducted to validate the feasibility and effectiveness of the proposed approach. The work presented in the paper not only alleviates the test oracle problem of testing Web services, but also delivers an effective and efficient test technique without oracles.

Keywords: *Metamorphic Testing, Service Oriented Architecture (SOA), Software Testing, Test Oracle, Web Services*

INTRODUCTION

Service Oriented Architecture (SOA) has been evolving as a mainstream software development paradigm where Web services are basic elements (Papazoglou, Traverso, Dustdar, & Leymann, 2008). A Web service often implements an application or part of an application, and is able to make a set of operations available to its consumers through the Web Service Descrip-

DOI: 10.4018/jwsr.2012010103

tion Language (WSDL) (Haas & Brown, 2004). In the context of SOA, Web services can be implemented and owned by one organization, and published as an independent resource that is consumed by other organizations. To implement complex applications, Web services have to be loosely orchestrated to fulfill a business goal (Peltz, 2003; Sun, Khoury, & Aiello, 2011).

Let us consider an e-bookstore system constructed by several Web services. Among them, a Web service is responsible for the electronic payment. Usually, such a Web service is developed and owned by a third-party organization, such as a software company, a bank, or an independent commercial office. Due to the fact that the consumer (i.e., the e-bookstore system) can access the Web service only through its description (namely WSDL) and cannot look into the source code of the Web service, this results in some inconsistency issues. For example, some faults may exist in the implementation regardless how many efforts are spent on testing. Also, the service owner may update the implementation due to changes of the payment policy (specification); however, the service consumer may not realize the changes happening to the implementation or specification. This may result in the situation where the consumer invokes the latest version of implementation while holds the old version of specification. All these cases bring us one question, namely "how should we assure the consistency between implementation and specification of Web services?"

Software testing provides a practical and feasible approach to the question. The basic idea of testing is to select some test cases for executing program under test, and then evaluate the output against the expected correct output (namely oracle). If the actual output equals to the oracle, such a test succeeds; otherwise the test detects a fault. In the context of SOA, new unique features pose challenges for testing Web services. For instance, the specification and implementation of Web services are completely separated, and often service consumers cannot access the source codes of Web services they invoke. The lack of source code and the restricted control of Web services not only limit the testability of Web services, but also make white-box testing techniques inapplicable. One may argue that the lack of source codes is also present in the use of any normal software library, but they are totally different in that the implementation of a Web service can be written in any programming language and its specification is written in the XML style. Moreover, the test oracle problem, which means in some situations it is impossible or practically too difficult to decide whether the program outputs on test cases are correct (Weyuker, 1982), is even amplified. In the example of the electronic payment service, the consumer, in some situations, may not know exactly how much should be charged for a given input (i.e., book price). The problem becomes even worse when the payment involves charges for transfer between accounts or currency exchange. Testing Web service under SOA calls for new techniques (Bartolini, Bertolino, Elbaum, & Marchetti, 2009; Canfora & Penta, 2009; Farooq, Georgieva, & Dumke, 2008; Sun, 2011).

Although there exist many techniques for Web service testing in the literature (Bartolini, Bertolino et al. 2009, Bai, Lee et al. 2008, Lenz, Chimiak-opoka et al. 2007, Zhang and Qiu 2006), most of these existing testing techniques always assume the presence of test oracles and thus cannot address the oracle problem. Metamorphic Testing (MT) was first proposed by Chen,Cheung et al. (1998) as a testing approach without the need of oracles. The basic idea behind MT is to leverage the metamorphic property of Web services to generate test cases and evaluate test outputs. Metamorphic property is a kind of inherent properties of software, referring to that for a pair of test cases satisfying relation R, and then their corresponding outputs should also satisfy relation R_f. During the testing, if there exist some tests where R is satisfied while R_f is violated, then a fault is detected. In this way, one can employ MT to test programs without precise oracles. It has been shown that MT has successfully alleviated the test oracle problem (Dong, Xu, Chen, Nie, & Wang, 2009). However, new features of SOA prevent the traditional MT framework from testing Web services. Recall the e-bookstore

system, the electronic payment Web services may be deployed in a different namespace with the client program, and it is invoked by a standard protocol (such as SOAP). This results in differences between testing a Web service and traditional software with libraries.

This paper proposes a metamorphic relation-based approach to testing Web services to address the challenges of testing Web services under SOA, in particular relevant to the prominent oracle problem with testing Web services. We propose a framework to examine key issues when applying MT to the testing of Web services, and developed a prototype to partially automate the framework. To validate the feasibility and effectiveness of the proposed approach, we conducted empirical studies where three real-life Web services are tested using our approach. Experimental results show that our approach can not only alleviate the test oracle problem of testing Web services, but also deliver an effective and efficient test technique without oracles. The work in this paper makes the following contributions:

1. A MT framework which examines and answers the key issues when using MT to test Web services, and a prototype which partially automates the framework.
2. An efficient testing technique for Web services. As to be observed from the results of the mutation analysis, MT can detect up to 94.1% faults of the subject Web services without oracles.
3. Empirical studies which describe how MT can be employed to test three real-life and widely-practiced Web services and report the effectiveness. Results of the empirical studies clearly show the applicability and effectiveness of MT for testing Web services.

The rest of the paper is organized as follows. Next section introduces underlying concepts related to MT and mutation analysis, and then we propose our approach including a framework of MT for Web services and a prototype. After that, we report empirical studies where the proposed approach is employed to test three real-life Web services, followed by a brief comparison of our approach with relevant methods. Finally, we conclude the paper and point out the future work.

BACKGROUND

In this section, we introduce the underlying issues or concepts related to Web service testing, MT and mutation analysis.

Testing Web Services

Web services must be trustworthy before they can be used. Testing is a major activity to assure that Web services can be trusted. However, the testing of Web services is more challenging than that of traditional software due to the unique features of SOA. In particular, the lack of source codes and the restricted control of services limit the testability of Web services.

In order to address these challenges, researchers have proposed various testing techniques for Web services. For example, Bartolini, Bertolino, Marchetti, and Polini (2009) developed a tool called TAXI that generates test cases for Web services based on WSDL specifications. Bai, Lee, Tsai, and Chen (2008) proposed an ontology-based partition testing approach for Web services. Lenz, Chimiak-Opoka, and Breu (2007) applied model-driven approaches to the testing of Web services. Zhang (2011) created an effective and efficient testing platform to facilitate Web services testing by seamlessly integrating HP LoadRunner and IBM Aglet technology. Many other testing methods for web services can be found in the literature, such as contract-based Web services testing (Heckel & Lohmann, 2004), fault-based Web services testing (Offutt & Xu, 2004), and regression Web services testing (Ruth & Tu, 2007), etc.

Metamorphic Testing

Most testing techniques proposed so far are focused on how to effectively select test cases such that faults can be revealed as early as pos-

sible or as many as possible. There is an implicit assumption behind most of these techniques, that is, there exists a test oracle that provides a systematic mechanism for verifying the test output given any possible program inputs. However, in many practical situations, the oracle does not exist, or it is very expensive, if not impossible, to verify the correctness of test outputs. Such an oracle problem hinders the applicability and effectiveness of many testing techniques.

Metamorphic testing (Chen et al., 1998) is an innovative approach to the oracle problem. In MT, testers first identify some properties from the software under test. A set of metamorphic relations (MRs) can then be constructed based on these properties. Some traditional testing techniques can be applied to generate some test cases, namely *source test cases*. MRs are used to convert source test cases into so-called *follow-up test cases*. Both source and follow-up test cases are executed. The execution results (that is, the test output) will be checked against the MRs (instead of using the oracle). If an MR is violated, a fault is said to be revealed.

One simple example for how MT works is as follows. Suppose P is a program that finds the shortest path from one node to another node in an undirected graph. For P, we can have an MR that a graph and its permutation should have the same output. In order to test P by MT, we generate a source test case (G, a, b), which G is a graph, a and b are two nodes of G, and then construct the follow-up test case (G', a', b'), where G' is the permutation of G, while a' and b' are the permutated points of a and b, respectively. We execute both (G, a, b) and (G', a', b'), and check whether $|P(G, a, b)| = |P(G', a', b')|$, where $|P(G, a, b)|$ denotes the length of the returned shortest path from node a to node b in G. If the relation does not hold, we can say that P has a fault.

Besides providing a test output verification mechanism alternative to the oracle, MT has many other advantages. For example, it can be effectively applied by end users without much knowledge of software testing. It is also very easy to automate MT. Based on MRs, a large number of follow-up test cases can be automatically generated at a low cost, and the test output verification can be easily fulfilled by writing some simple scripts. Researchers from different application areas have used MT to detect bugs in various programs (Chen, Ho, Liu, & Xie, 2009; Murphy, Kaiser, Hu, & Wu, 2008).

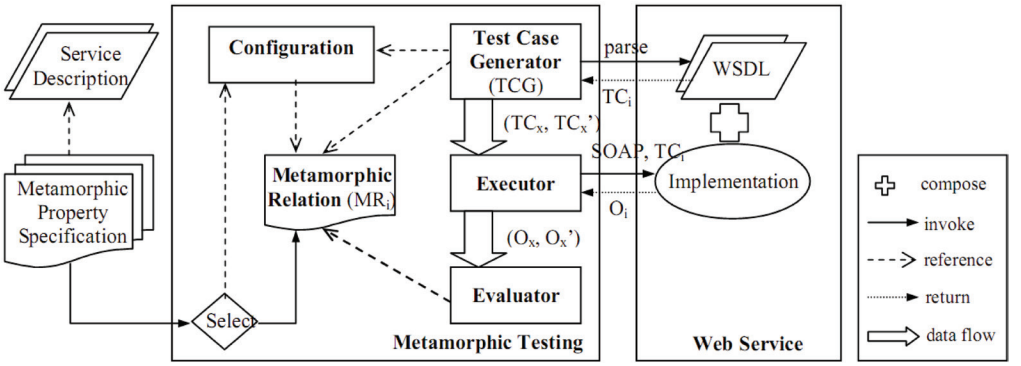
Mutation Analysis

Mutation analysis (DeMillo, Lipton, & Sayward, 1978) is widely used to assess the adequacy of a test suite and the effectiveness of testing techniques. The mutation analysis technique applies some mutation operators to seed various faults into the program under test, and thus generates a set of variants, namely mutants. If a test case causes a mutant to show a behavior different from the program under test, we say that this test case can "kill" the mutant and thus detect the fault injected into the mutant. We normally use the mutation score (MS) to measure how thoroughly a test suite can kill the mutants, which is defined as

$$\mathrm{MS}(p, ts) = \frac{N_k}{N_m - N_e} \quad (1)$$

where p refers to the program being mutated, ts refers to test suite under evaluation, N_k refers to the number of killed mutants, N_m refers to the total number of mutants, and N_e refers to the number of equivalent mutants. An equivalent mutant refers to one whose behaviors are always the same as those of p. It has been pointed out that compared with manually seeded faults, the automatically generated mutants are more similar to the real-life faults, and the mutant score is a good indicator for the effectiveness of a testing technique (Andrews, Briand, & Labiche, 2005). In this paper, we will use mutation analysis technique to evaluate the effectiveness of our approach.

Figure 1. The framework of metamorphic testing for Web services



METAMORPHIC TESTING FOR WEB SERVICES

When loosely-coupled web services are orchestrated to fulfill a business goal, the service consumer must be confident that the Web services being orchestrated should implement their expected functionalities. This assumption requires that the service owner/developer has adequately tested the Web services. However, the service owner/developer cannot cover all possible usages of Web services, and thus the executed tests are inadequate. On the other hand, the service consumers have very little documentation and cannot access source codes of Web services. In this situation, MT provides an appropriate testing technique which can help service consumers test a third-party Web services without oracles.

Figure 1 depicts a framework of MT for Web services. Within the framework, metamorphic relationships (MRs) play a key role because they determine the selection of test cases and the evaluation of test results. Note that with the framework, we assume that the service consumers can derive the metamorphic property specification from the limited documentation of Web services, and service description may record the tests already executed on the web service being tested.

When the framework is employed to test a Web service, the consumers first derive metamorphic property specifications from the description or WSDL of the Web service. Before the test starts, the consumers need to specify the options with the configuration, and select MRs to conduct tests. The consumers can employ the test case generator to construct test cases according to the selected MR. The executor is then employed to run test cases and get their outputs. Finally, the evaluator assesses the tests and judges whether the MR is satisfied or violated. Next, we examine individually how the components of the MT framework work and how they are collaborated to test Web services without oracles.

(1) *Test Case Generator (TCG)*. This component is responsible for generating test cases according to the selected MR_i. *TCG* first needs to parse the WSDL to decide the format of test cases. For generating *source test cases*, there are two ways. One is to randomly generate them from scratch; the other is to extract them from the service description that has recorded the previously executed tests. Next, the *TCG* construct the follow-up test cases TC_x' by transforming the source test case TC_x based on the MR_i. Furthermore, the *TCG* may generate test cases in either the batch mode or the one-by-one mode. If the batch mode is adopted, it needs to know where to store the generated test cases. Both the mode and the storage location are specified through the *Configuration* component.

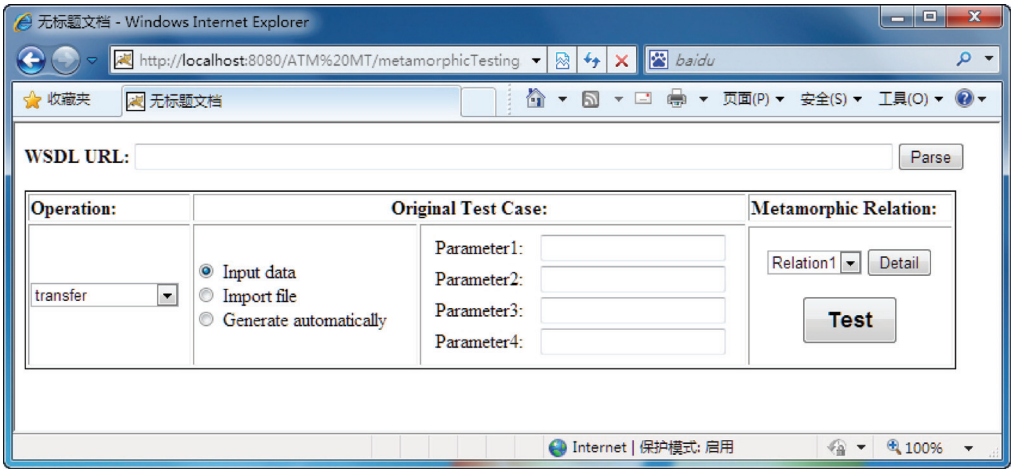
(2) *Executor*. This component executes Web services with test cases generated by the *TCG* via the SOAP message, and intercepts the output O_i from the execution. If the source test case TC_x is extracted from the service description, and the corresponding output O_x is also recorded in the previously executed tests in its service description, we can skip the execution of TC_x, and directly run the Web service with TC_x' and intercepts its output O_x'.
(3) *Evaluator*. This component compares the outputs O_x and O_x', and makes decision whether they satisfy or violate MR_i. If MR_i is violated, a fault is detected; otherwise, this test is passed.
(4) *Configuration*. This component is responsible for specifying the options during the MT process.
    - Firstly, we can derive a set of MRs from the metamorphic property specification. The configuration component must specify which MR is selected before the test.
    - Secondly, for the given MR_i, the configuration component specifies how many test cases should be selected by the *TCG*.
    - Thirdly, the configuration component must specify the mode for the *TCG* to generate test cases. For the batch mode, it also needs to further specify the file of the generated test cases.
    - Finally, if the *Evaluator* detects a fault, the testing stops. However, the testing may not detect any fault although all the generated test cases have been executed. In such a situation, the configuration component will specify an option whether the testing should stop or continue by trying another MR.

One may discover that some components in the framework can be automated, such as *Test Case Generator*, *Executor* and *Evaluator*. In order to make the approach effective and practical, we have implemented a prototype which partially automates the framework. Figure 2 shows a snapshot of the prototype. To start, one is required to input the WSDL URL of the Web service being tested, and then press the *Parse* button to automatically parse the WSDL and derive a list of operations implemented by the Web service. From the list, one can select and test operations individually. Generating a large size of source test cases is usually a trivial task and thus should be left for the prototype by selecting the *Generate automatically* option. This can be done by parsing the WSDL file to define the formats of input messages. Note that metamorphic relations (MRs) have to be defined manually; on the other hand, with the prototype one can select one or more MRs to automatically generate follow-up test cases. To execute the tests by pressing the *Test* button, the prototype converts test cases into input messages, invokes the interfaces of a Web service, and intercepts output messages. With the output messages of both source test case and follow-up test case, the prototype can then evaluate the result of tests by judging whether the selected MR is violated. If so, a fault is detected; otherwise the test is passed. Finally, the prototype produces a test summary in a file for the further analysis.

EMPIRICAL STUDIES

In this section, we report case studies which were conducted to illustrate the proposed approach and validate its effectiveness. Three real-life and commonly-used Web services are selected as the subject program of our experiments. Among them, the first one implements the electronic payment service and second one provides the query service. Testing such Web services meets with the prominent oracle problem. The third one implements a daily-practiced service related to processing financial affairs in China. To measure the effectiveness, mutation analysis is employed. The results of case studies suggest that the proposed approach is effective in testing Web services and can detect from 77.5% to 94.1% mutants.

Figure 2. A snapshot of the MT prototype



We first present two metrics for effectiveness measurement and describe the general experiment procedure. After that, we report three case studies individually. A few MRs were identified for each subject program. Mutation analysis technique was used to seed faults into subject programs and thus to generate mutants. The prototype was applied to generate test suites based on MRs and test the mutants. Finally, we summarize the proposed approach based on the experiments.

Measurement of Effectiveness

In our empirical studies, we evaluate the effectiveness of MT based on a group of mutants. When an MR is violated (that is, the source and follow-up test cases satisfies R, but their outputs do not satisfy R_f), a fault is said to be detected (that is, a mutant is killed). We use the following two metrics to measure the effectiveness of MT.

The first metric, mutation score (*MS*), as defined in the *Mutation Analysis* section, is the ratio of the number of mutants killed by a test suite over the number of all non-equivalent mutants. *MS* indicates the fault-based adequacy of test suite *ts*, that is, how thoroughly a test suite *ts* can detect possible faults that may be seeded into the program under test. Obviously, a higher MS indicates a higher effectiveness of a test suite, and hence a higher effectiveness of the MR, based on which the test suite is generated.

The second metric is the fault discovery rate (*FDR*), which is defined as follows.

$$FDR\ (m, ts) = \frac{N_f}{N_{ts} - N_i} \quad (2)$$

where N_f refers to the number of test cases that can kill the mutant m, N_{ts} refers to the total number of test cases in ts, and N_i refers to the number of invalid test cases. Invalid test cases refer to those that do not work properly for a given MR. In our experiments it is possible to derive some invalid follow-up test cases. Such a follow-up test case may violate the rules of the subject program, and thus should be excluded for experiments. FDR indicates the effectiveness of a test suite *ts* on a particular mutant *m*, that is, how likely test cases generated based on a certain MR can kill *m*. A higher FDR implies that the test suite *ts* and the related MR have higher chance to kill the mutant *m*.

Experimental Procedure

Given a subject program, we took the following steps to conduct our experiments.

- Identify MRs for the subject program

The identification of MRs is a key issue during the application of MT (Chen, 2010). Some guidelines are available for selecting MRs from the specification. In particular, Some researchres (Chen, Kuo, Liu, & Tang, 2004; Chen, Huang, Tse, & Zhou, 2004; Chen, 2010) have discovered that (1) good MRs are relations which involve the execution of the core functionality; (2) good MRs should be those that can make the multiple executions of the program as different as possible. According to the guidelines, we derived a set of MRs for each subject program, which will be reported later in this section. Note that the derivation of these MRs is based on the specification of the subject program, and is completely independent of the implementation. This means that MT does not need to access the source code and hence is widely applicable to SOA-based applications.

- Generate mutants of the subject program

All subject programs are written in Java, so we used a popular Java mutation tool, namely MuJava (Offutt, Ma, & Kwon, 2004), to automatically seed faults by various mutation operators, and thus to generate a group of mutants for each subject program. We also attempted to identify and eliminate the equivalent mutants manually. Only the non-equivalent mutants were used in our experiments.

- Generate test cases based on MRs

In order to execute MT, test cases are produced based on the MRs. For the source test cases, one can employ traditional test case generation techniques, such as the special test value generation, the random test value generation and the iterative test value generation. Among them, the random test value generation is more favorable and efficient for MT, because it can generate a large amount of test cases at a low cost, and the randomly-generated test cases can cover the test domain without any bias (Chen et al., 2004; Wu, Shi, Tang, Lin, & Chen, 2005). Thus we employed the random test value generation to generate source test cases in our experiments. For the follow-up test cases, they are accordingly constructed from their source test cases using MRs defined for each subject program.

- Execute testing

With the test cases generated, we now can execute the testing on each mutant. It should be reminded that in our experiments, no oracle is required. Instead, MRs are used to verify the test results. A fault is said to be detected if an MR is violated. In order to make the testing efficient, we employed the prototype discussed (Figure 2). With the prototype, one can select one or more MRs to test subject programs. The prototype supports both the one-by-one mode and the batch mode. Test cases can be automatically generated or manually input, or imported from a file.

- Evaluate experimental results

Based on a particular MR, we can have test suite *ts*. After running all test cases in *ts* on a mutant *m*, we can collect the value of FDR for the specific MR and *m*. We use *ts* to test all mutants of a subject program, and then we can calculate MS of the MR on the subject program.

Case Study 1

Subject Program: Balance Transfer

A general ATM (Automatic Teller Machine) system is implemented as a Web service and deployed in the Tomcat server (Sun, Wang, & Zhao, 2011). The user and business data are stored in a MySQL database. The system offers several features, such as withdrawal, deposit, transfer, query, and each of them is encapsulated as a service operation. Among these features, we select the transfer feature for the case study because it is widely practiced in the electronic

Figure 3. A segment of WSDL for the transfer interface

```
<wsdl:operation name="transfer">
   <wsdl:input message="tns:transferRequest"></wsdl:input>
   <wsdl:output message="tns:transferResponse"> </wsdl:output>
   <wsdl:fault name="fault01" message="tns:InvalidAccountID">
          </wsdl:fault>
   <wsdl:fault name="fault03" message="tns:InvalidAmount">
          </wsdl:fault>
</wsdl:operation>
…
<xsd:element name="transferRequest">
   <xsd:complexType>
     <xsd:sequence>
        <xsd:element name="from" type="xsd:string"></xsd:element>
        <xsd:element name="to" type="xsd:string"></xsd:element>
        <xsd:element name="amount" type="xsd:int"></xsd:element>
        <xsd:element name="mode" type="xsd:int"></xsd:element>
     </xsd:sequence>
   </xsd:complexType>
</xsd:element>
<xsd:element name="transferResponse" type="xsd:string">
</xsd:element>
```

payment and the oracle problem arises when testing such a feature.

Figure 3 shows a segment of WSDL for the transfer feature. The implementation consists of 136 lines of Java codes, which executes the connection to relevant database, SQL statements, and numerical computing on the transfer amount and commission fee.

As for the commission fee charging criterion, we refer to Agricultural Bank of China for the calculation rules as shown in Table 1. Transfer types I through IV refer to the transfer between two accounts in the same bank and city, in the same bank but different cities, in the same city but different banks, in different cities and different banks, respectively. From Table 1, one can see that the commission fee varies a bit with different types of transfers. When transferring the money between two accounts, the user may not know the precise amount of commission fees because details of the recipient account may not be fully known. In other word, the oracle is not always available when testing such a Web service.

By analyzing the WSDL of the Web service, we can derive the input of the transfer operation, and it is represented as a 4-tuple integer vector (*A, B, P, M*), where

- *A* and *B* denote the sender and recipient account numbers for the transfer transaction, respectively. They consist of 10 digits.
- *P* denotes the transfer type. Its value ranges from 0 to 3, corresponding to type I to IV in Table 1. Note that the transfer type can

Table 1. Commission fee calculation

	I	II	III	IV
Charge Percentage	0%	0.5%	0.5%	1%
Min(¥)	0	1	1	1
Max(¥)	0	50	50	50
Limit Per Transfer (¥)	50000	50000	50000	50000

Table 2. A set of MRs for the transfer feature

MR	R	R_f
MR1-1	M'=2M	ΔA'≤2ΔA and ΔB'=2ΔB
MR1-2	P= 1 and P'= 2	ΔA'- ΔB'=ΔA- ΔB
MR1-3	P= 0 and P'≠ 0	ΔA'- ΔB'>ΔA-ΔB
MR1-4	P= 3 and P'≠ 3	ΔA'- ΔB'≤ΔA-ΔB
MR1-5	M'>M	ΔA'> ΔA and ΔB'>ΔB
MR1-6	A'=B and B'=A	ΔA'=ΔB

be deduced from *A* and *B*. For simplicity, we explicitly specify the type by *P*.
- *M* denotes the amount of a transfer transaction, ranging from 0 to 50000, inclusive.

For example, an input (1000000000, 2000000000, 3, 5000) means that the sender account number is 1000000000, the recipient account number is 2000000000, ¥5000 is transferred from the sender account to the recipient account, and these two accounts are in different cities and different banks.

Similarly, we can derive the output of the transfer operation. For simplicity, it is represented as a 2-tuple positive real vector where

- Δ*A* denotes the difference between the balances of account *A before* transaction and *after* transaction.
- Δ*B* denotes the difference between the balances of account *B after* transaction and *before* transaction.

Note that both Δ*A* and Δ*B* must be positive after a transaction. We notice that the commission fee may be charged from either the sender account (i.e., *A*) or the recipient account (i.e., *B*). Here, we assume the former in order to follow the policy of Agricultural Bank of China.

Metamorphic Relations

We derive six MRs for the transfer feature, as listed in Table 2. In this case study, all the selected MRs can be decomposed such that each MR is a pair of *R* and R_f, where *R* denotes the relation between source and follow-up test cases (inputs) and R_f denotes the relation between their outputs. From MR1-1 to MR1-5, the follow-up test cases are derived from their source test cases via changing only one tuple once. Considering MR1-1, if one source test case is (*a, b, p, m*), its follow-up test case should be (*A'=A, B'=B, P'=P, M'=2M*). For MR1-6, the follow-up test cases are derived from their source test cases via exchanging the sender account with the recipient account.

Experimental Results

Table 3 summarizes the average FDRs of all 129 non-equivalent mutants. In the experiments reported here, we set the size of valid test cases (namely N_{ts}-N_i) to 50, 100 and 200, in order to make the experimental results conclusive and stable. We observe that MR1-2, MR1-3 and MR1-1 are more effective compared with other MRs, and thus should have higher priority when MT is employed.

We further select ten mutants from 129 non-equivalent mutants for a detailed analysis of the FDR with respect to each MR. These ten mutants are selected in order to cover all types of mutation operators supported by MuJava, and at the same time we believe the associated faults with these mutants are very typical. Table 4 summarizes the mutation description of these mutants.

Table 5 reports the FDRs on the ten mutants when MT is used to test the transfer. Each cell shows the FDR of a test suite generated by an

Table 3. A summary of average FDR of 129 mutants using MT on transfer interface

		MR1-1	MR1-2	MR1-3	MR1-4	MR1-5	MR1-6
FDR	Size=50	30.4%	31.7%	31.5%	20.3%	15.4%	13.8%
	Size=100	30.2%	32.8%	31.3%	20.3%	15.3%	13.8%
	Size=200	30.6%	31.8%	30.8%	20.3%	15.3%	13.8%

Table 4. A summary of mutation description of ten mutants

ID	Mutation Description
M004	Line 88: money => money++
M007	Line111: money => --money
M021	Line 123: commission_charge => --commission_charge
M055	Line 149: EXCEPTION_DATABASE_ERROR => -EXCEPTION_DATABASE_ERROR
M057	Line 111: money * rate2 => money / rate2
M069	Line 88: money > maxTransferAmount_Once => !(money > maxTransferAmount_Once)
M093	Line 126: commission_charge<1 && commission_charge>0 => commission_charge < 1^commission_charge > 0
M096	Line 110: same_bank == false && same_location == false => same_bank == false ^ same_location == false
M116	Line 110: same_bank == false => same_bank != false
M133	Line 88: money > maxTransferAmount_Once => money <= maxTransferAmount_Once

MR on a mutant. For example, the right-bottom cell represents that the test suite with the size of 200 test cases generated by MR1-6 has an FDR of 100% on M133. We can observe from Table 5 that

- Each MR has a varying sensitivity to different mutants. For instance, MR1-1 is sensitive to M004 while not sensitive to M057; MR1-3 is sensitive to M021 and M093, while not sensitive to M007 and M004. Such observations imply that different MRs have different effectiveness on different types of faults. It is not surprising, as MRs are just necessary conditions of the specification which reflect specific aspects of the software. Testers should identify the properties that are the most important for the consumers of the software under test, and thus identify effective MRs based on these important properties.
- Among the ten mutants, M069 and M133 can be killed by all MRs, M055 cannot be killed by any MR, and other mutants were killed by some MRs with varying FDRs but cannot be killed by other MRs. By further analysis of the implementation of M055, we found that this mutant cannot be killed because the seeded fault is related to exception processing while our MRs do not involve the feature of exception processing. This indicates that some MRs related to exception process may be able to detect such kinds of faults.

Table 5. A summary of FDR for ten mutants when the size of valid test cases is 50, 100 and 200

	ID	MR1-1	MR1-2	MR1-3	MR1-4	MR1-5	MR1-6
Size of valid test cases=50	M004	100%	0%	0%	0%	**0%**	0%
	M007	30%	0%	0%	0%	**0%**	0%
	M021	38%	0%	100%	24%	**14%**	26%
	M055	0%	0%	0%	0%	**0%**	0%
	M057	0%	100%	0%	0%	**0%**	0%
	M069	100%	100%	100%	100%	**100%**	100%
	M093	0%	0%	100%	0%	**0%**	0%
	M096	0%	0%	52%	72%	**0%**	0%
	M116	0%	18%	28%	74%	**0%**	0%
	M133	100%	100%	100%	100%	**100%**	100%
Size of valid test cases=100	M004	100%	0%	0%	0%	**0%**	0%
	M007	31%	0%	0%	0%	**0%**	0%
	M021	32%	0%	100%	35%	**24%**	26%
	M055	0%	0%	0%	0%	**0%**	0%
	M057	0%	100%	0%	0%	**0%**	0%
	M069	100%	100%	100%	100%	**100%**	100%
	M093	0%	0%	100%	0%	**0%**	0%
	M096	0%	0%	42%	65%	**0%**	0%
	M116	0%	27%	38%	69%	**0%**	0%
	M133	100%	100%	100%	100%	**100%**	100%
Size of valid test cases=200	M004	100%	0%	0%	0%	**0%**	0%
	M007	20%	0%	0%	0%	**0%**	0%
	M021	39%	0%	100%	35%	**27%**	16%
	M055	0%	0%	0%	0%	**0%**	0%
	M057	0%	100%	0%	0%	**0%**	0%
	M069	100%	100%	100%	100%	**100%**	100%
	M093	0%	0%	100%	0%	**0%**	0%
	M096	0%	0%	36%	70%	**0%**	0%
	M116	0%	19%	32%	65%	**0%**	0%
	M133	100%	100%	100%	100%	**100%**	100%

Table 6 summarizes the test adequacy of MT with respect to MS for each MR. "The "N_k" row shows the number of mutants killed by a MR, and the "*MS*" row shows the mutation score of each MR (= N_k /129). The "*Total*" column shows the performance of MT when the testing results of all six MRs are considered. We get the same MS regardless of the different sizes of test cases (50, 100 or 200). It can be observed from Table 6 that

- The MS of each MR can be used to compare their effectiveness. Among these MRs,

Table 6. A summary of MS of 129 mutants using MT on transfer interface

	MR1-1	MR1-2	MR1-3	MR1-4	MR1-5	MR1-6	Total
N_k	58	54	56	38	25	23	100
MS	45.0%	41.9%	43.4%	29.5%	19.4%	17.8%	77.5%

MR1-1, MR1-2, and MR1-3 are more effective, while MR1-4, MR1-5 and MR1-6 are less effective. For instance, the MS of MR1-1 (45.0%) is larger than that of MR1-6 (17.8%), we can say that MR1 is more effective than MR6 to some extent.

- All six MRs altogether can kill up to 77.5% of all mutants. The total value of MS is larger than MS of any single MR. That is to say, as many MRs as possible should be used to generate test suites provided that there is no concern with testing costs. Otherwise, taking into account the results in Table 4, the priority order of these MRs should be MR1-1>MR1-3>MR1-2>MR1-4>MR1-5 >MR1-6.

With Case Study 1, we have demonstrated in detail how the proposed approach was applied to test a Web service without oracles. We have also measured its effectiveness in terms of both MS and FDR, and provided a detailed analysis. In order to make the subsequent description more compact, we will discuss some key issues while skipping over the experimental details for the remaining two cases studies.

Case Study 2

Subject: Seismic Service

The Web service under test provides a seismic data query service (Sosnoski, 2009). It is based on an actual database of more than 93,000 earthquakes that occurred in the worldwide over a period of nearly 4 years. The implementation consists of 551 lines of Java codes, which are mostly of data query statements. The entire database is kept in a text file on disk. Requests to the service specify query ranges for latitude, longitude, date, or magnitude, and the service returns all matching earthquakes grouped by region and in date order. Obviously, it is impossible or difficult for one to decide for any given inputs whether a returned result is correct or not. So, the oracle problem arises when testing such a Web service.

Figure 4 shows a segment of WSDL for the seismic service.

By analyzing the WSDL of the Web Service, we derive the input of the matchQuakes operation. It is represented as a 10-tuple vector: <*minLat*, *MaxLat*, *minLng*, *MaxLng*, *minDate*, *maxDate*, *minMag*, *maxMag*, *minDepth*, *maxDepth*>. Each tuple of the input is described in Table 7.

Similarly, we can derive the output of the matchQuakes operation. It is represented as a 3-tuple vector: <*area-name*,*regions**,*quakes**>, where the "*" refers to a complex type. Each vector of the output is described in Table 8.

Metamorphic Relations

We derived 12 MRs as listed in Table 9. Note that the derivation of these MRs is quite easy and intuitive. They are achieved using the rule: for any two inputs *a* and *a'*, their corresponding outputs are *o* and *o'*, respectively, if some constraint conditions in *a'* are weaker than *a* and in the meanwhile, the other remains the same, then the relation ($o \subseteq o'$) must come into existence. Let us consider the rule in the context of MT. Taking the first two MRs which are relevant to the *minLat* and *maxLat* tuple, we expand the range of latitude in follow-up test cases. According to the expansion, the number of quakes in the output of follow-up test case should be equal to or larger than that of source test cases, and the set of quakes in the output

Figure 4. A segment of WSDL for the seismic service

```
<wsdl:operation name="matchQuakes">
  <wsdl:input message="wns:quakeQuery"/>
  <wsdl:output message="wns:quakeResponse"/>
</wsdl:operation>
<wsdl:message name="quakeQuery">
 <wsdl:part name="parameters" element="tns:matchQuakes"/>
</wsdl:message>
<element name="matchQuakes">
  <complexType>
    <sequence>
      <element name="min-date" minOccurs="0" type="xsd:dateTime"/>
      <element name="max-date" minOccurs="0" type="xsd:dateTime"/>
      <element name="min-long" minOccurs="0" type="xsd:float"/>
      <element name="max-long" minOccurs="0" type="xsd:float"/>
      <element name="min-lat" minOccurs="0" type="xsd:float"/>
      <element name="max-lat" minOccurs="0" type="xsd:float"/>
      <element name="min-mag" minOccurs="0" type="xsd:float"/>
      <element name="max-mag" minOccurs="0" type="xsd:float"/>
      <element name="min-depth" minOccurs="0" type="xsd:float"/>
      <element name="max-depth" minOccurs="0" type="xsd:float"/>
    </sequence>
  </complexType>
</element>
<wsdl:message name="quakeResponse">
    <wsdl:part name="result" element="tns:results"/>
</wsdl:message>
<element name="results">
  <complexType>
    <sequence>
      <element name="result-set" minOccurs="0" maxOccurs="unbounded" type="tns:QuakeSet"/>
    </sequence>
    <attribute name="count" use="required" type="xsd:int"/>
  </complexType>
</element>
```

of follow-up test cases should subsume that of source test cases. Similarly, we derived the other ten MRs.

Experimental Results

Table 10 summarizes the average FDRs of all 724 mutants of the seismic service. In the experiments, we set the size of valid test cases (namely N_{ts}-N_i) to 100. Based on Table 10, we observe that all MRs have similar average values of FDRs.

Table 11 summarizes the test adequacy of MT with respect to MS for each MR. The results are very exciting. It can be observed from Table 11 that (1) MRs 2-1 to 2-4 and 2-11 can kill most mutants, while the other MRs can kill more than 20% mutants, and (2) when the 12 MRs are used together, MT can kill up to 94.1% of all mutants, which is larger than the MS of any single MR.

Case Study 3

Subject: RMB Converter Service

This Web service converts a numerical form of money into Chinese character form. Such a service is very important and widely used in daily financial and business affairs in China, because Chinese people have to use Chinese characters to represent a certain amount of money under many situations (such as signing contract, getting reimbursement, and so on). Table 12 shows the Chinese characters which represent various orders of magnitude in a Chinese form

Table 7. A summary description of inputs of matchQuakes operation

Tuple	Format	Detail
maxDate	Y Y Y Y - M M - d d H H : m m : s s , the range is (2000-01-01, 2003-08-31)	Upper limit of date for querying
minDate	Y Y Y Y - M M - d d H H : m m : s s , the range is (2000-01-01, 2003-08-31)	Lower limit of date for querying
maxLat	Float type, the range is(-90.0, 90.0)	Upper limit of latitude for querying
minLat	Float type, the range is(-90.0, 90.0)	Lower limit of latitude for querying
maxLng	Float type, the range is(-180.0, 180.0)	Upper limit of longitude for querying
minLng	Float type, the range is(-180.0, 180.0)	Lower limit of longitude for querying
maxMag	Float type, the value is positive, can be null(no limitation on this condition)	Upper limit of magnitude for querying
minMag	Float type, the value is positive, can be null(no limitation on this condition)	Lower limit of magnitude for querying
maxDepth	Float type, the value is positive, can be null(no limitation on this condition)	Upper limit of quake depth for querying
minDepth	Float type, the value is positive, can be null(no limitation on this condition)	Lower limit of quake depth for querying

Table 8. Output and its tuples of matchQuakes operation

Tuple	Data type
area-name	String
regions	regionsType **Item** / **Data type** / count / Int / Number of regions Region[0…n] / Region: **Item** / **Data type** / ident / String; index / int
quakes	quakesType **Item** / **Data type** / count / int / Number of quakes Quake[0…n] / Quake: **Item** / **Data type** / time / xsd:dateTime; mills / int; latitude / Float; longitude / Float; depth / Float; magnitude / Float; method / String; region / String

Table 9. A set of MRs for the matchQuakes feature

MR	R	R_f
MR2-1	minLat'<minLat, maxLat'=maxLat	quakes.count ≤ quakes'.count, (quake$_1$, quake$_2$, …) ⊆ (quake$_1$', quake$_2$', …)
MR2-2	minLat'=minLat, maxLat'>maxLat	quakes.count ≤ quakes'.count, (quake$_1$, quake$_2$, …) ⊆ (quake$_1$', quake$_2$', …)
MR2-3	minLng'<minLng, maxLng'=maxLng	quakes.count ≤ quakes'.count, (quake$_1$, quake$_2$, …) ⊆ (quake$_1$', quake$_2$', …)
MR2-4	minLng'=minLat, maxLng'>maxLng	quakes.count ≤ quakes'.count, (quake$_1$, quake$_2$, …) ⊆ (quake$_1$', quake$_2$', …)
MR2-5	minDepth'<minDepth, maxDepth'=maxDepth	quakes.count ≤ quakes'.count, (quake$_1$, quake$_2$, …) ⊆ (quake$_1$', quake$_2$', …)
MR2-6	minDepth'=minDepth, maxDepth'>maxDepth	quakes.count ≤ quakes'.count, (quake$_1$, quake$_2$, …) ⊆ (quake$_1$', quake$_2$', …)
MR2-7	minMag'<minMag, maxMag'=maxMag	quakes.count ≤ quakes'.count, (quake$_1$, quake$_2$, …) ⊆ (quake$_1$', quake$_2$', …)
MR2-8	minMag'=minLat, maxMag'>maxMag	quakes.count ≤ quakes'.count, (quake$_1$, quake$_2$, …) ⊆ (quake$_1$', quake$_2$', …)
MR2-9	minDate'<minDate, maxDate'=maxDate	quakes.count ≤ quakes'.count, (quake$_1$, quake$_2$, …) ⊆ (quake$_1$', quake$_2$', …)
MR2-10	minDate'=minDate, maxDate'>maxDate	quakes.count ≤ quakes'.count, (quake$_1$, quake$_2$, …) ⊆ (quake$_1$', quake$_2$', …)
MR2-11	minDepth=null, maxDepth=null, minDepth'≠null, maxDepth' ≠null	quakes.count ≤ quakes'.count, (quake$_1$, quake$_2$, …) ⊆ (quake$_1$', quake$_2$', …)
MR2-12	minMag=null, maxMag=null, minMag'≠null,maxMag' ≠null	quakes.count ≤ quakes'.count, (quake$_1$, quake$_2$, …) ⊆ (quake$_1$', quake$_2$', …)

Table 10. A summary of average FDR of 724 Mutants using MT on seismic service

	MR2-1	**MR2-2**	**MR2-3**	**MR2-4**	**MR2-5**	**MR2-6**
FDR	23.9%	27.1%	23.9%	26.3%	22.7%	22.5%
	MR2-7	**MR2-8**	**MR2-9**	**MR2-10**	**MR2-11**	**MR2-12**
FDR	22.6%	22.6%	22.6%	22.6%	23.3%	22.6%

Table 11. A summary of MS of 724 mutants using MT on seismic service

	MR2-1	**MR2-2**	**MR2-3**	**MR2-4**	**MR2-5**	**MR2-6**	
N_k	665	665	664	673	193	164	
MS	91.9%	91.9%	91.7%	93.0%	26.7%	22.7%	
	MR2-7	**MR2-8**	**MR2-9**	**MR2-10**	**MR2-11**	**MR2-12**	**Total**
N_k	167	173	177	174	661	171	681
MS	23.1%	23.9%	24.4%	24.0%	91.3%	23.6%	94.1%

Table 12. Chinese characters representing various orders of magnitude in a form of money

Order of magnitude	Cent	Ten cents	Dollar	Ten	Hundred	Thousand	Ten thou-sand	Hundred thousand	Million
Chinese	分	角	☒	拾	佰	仟	万	拾万	佰万

of money. Table 13 summarizes the Chinese characters to represent all digits. For example, such a service can covert "1234567.89" into "壹佰☒拾叁万肆仟伍佰☒拾柒元捌角玖分".

Figure 5 shows a segment of WSDL for the RMB converter service.

Metamorphic Relations

We derived 4 MRs as listed in Table 14. Here, we assume that the input and output for the source test case are denoted by *N* and *O*, respectively, and the input and output for the follow-up test case are denoted by *N'* and *O'*, respectively. These MRs are derived to cover a variety of situations when transforming a number to Chinese characters, in particular those related to processing the zero and magnitude.

Experimental Results

Table 15 summarizes the average FDRs of all 195 non-equivalent mutants of the RMB converter service. In the experiments reported here, we set the size of valid test cases (namely N_{ts}-N_i) to 100. From Table 15, we observe that (1) the average FDR values of the four MRs vary noticeably, and (2) among them, the MR3-4 has the largest value of FDR, while MR3-2 has the smallest one.

Table 16 summarizes the test adequacy of MT with respect to MS for each MR. We observe that (1) All MRs have a high MS varying from 62.6% to 81.0%, which means that they can kill most of mutants, (2) when the four MRs are used together, the MS does not increase evidently.

Table 13. Chinese characters representing all digits

Digit	0	1	2	3	4	5	6	7	8	9
Chinese	零	壹	☒	叁	肆	伍	☒	柒	捌	玖

Figure 5. A segment of WSDL for the RMB converter service

```
<xsd:element name="convertInput">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="number" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="convertOutput">
   <xsd:complexType>
     <xsd:sequence>
       <xsd:element name="output" type="xsd:string"/>
     </xsd:sequence>
   </xsd:complexType>
</xsd:element>
……
<wsdl:message name="convertRequest">
  <wsdl:part element="tns:convertInput" name="parameters"/>
</wsdl:message>
<wsdl:message name="convertResponse">
   <wsdl:part element="tns:convertOutput" name="parameters"/>
</wsdl:message>
<wsdl:portType name="RmbtoGbbob">
  <wsdl:operation name="convert">
    <wsdl:input message="tns:convertRequest"/>
    <wsdl:output message="tns:convertResponse"/>
   </wsdl:operation>
```

Table 14. A set of MRs for the RMB converter service

MR	R	R_f
MR3-1	*N'* is constructed by adding a non-zero digit *d* to the left of *N*.	*O'* = C_d + C_p + *O*, where C_d and C_p are the Chinese characters of digit *d* and its corresponding order of magnitude, respectively.
MR3-2	Given that *N* does not contain ".", that is, the number is an integer. *N'* is constructed by adding "." and two digits d_j and d_f, whose corresponding Chinese characters are C_{dj} and C_{df}, respectively.	(1) *O'* = *O* + C_{dj} + "角" + C_{df} + "分", when $d_j \neq 0$ and $d_f \neq 0$; (2) *O'* = *O* + C_{dj} + "角", when $d_j \neq 0$ and $d_f = 0$; or (3) *O'* = *O* + C_{df} + "分", when $d_j = 0$ and $d_f \neq 0$.
MR3-3	*N'* is constructed by changing a non-zero digit *d* of N to another non-zero digit *d'*.	*O'* should be identical to the string replacing C_d in *O* by C'_d, where C_d and C'_d are the Chinese characters for *d* and *d'*, respectively.
MR3-4	*N'* is constructed by changing a non-zero digit *d* of *N* to "0".	*O'* should be identical to the string deleting C_d and C_p from *O*, where C_d and C_p are the Chinese characters of digit *d* and its corresponding order of magnitude, respectively.

Table 15. A summary of average FDR of 195 Mutants using MT on RMB converter service

	MR3-1	MR3-2	MR3-3	MR3-4
FDR	64.1%	58.9%	64.8%	74.2%

Table 16. A summary of MS of 195 mutants using MT on RMB converter service

	MR3-1	MR3-2	MR3-3	MR3-4	Total
N_k	141	122	158	154	161
MS	72.3%	62.6%	81.0%	79.0%	82.6%

Summary

Through the empirical studies, we have validated the feasibility of the proposed metamorphic relation-based approach and evaluated its effectiveness. The case studies have been conducted to test three real-life Web services from different domains. One advantage of the proposed approach is that no oracle is needed. This greatly alleviates the more prominent oracle problem when testing Web services under SOA. The other is that the experimental results suggest that 77.5% ~ 94.1% mutants are killed in three case studies, which demonstrates a high fault detection capability of the proposed approach. In summary, the proposed approach delivers an effective and efficient testing technique without oracles for Web services.

RELATED WORK

As pointed out in the BACKGROUD section, there are various techniques for the testing of Web services reported in the literature, such as WS-TAXI (Bartolini, Bertolino et al. 2009), ontology based partioning testing (Bai, Lee et al. 2008), fault-based web service testing (Offutt and Xu 2004), etc. However, most of these techniques are focused on the selection of test cases for Web services. There is often an assumption behind these test case selection techniques, that is, an oracle exists for test result verification. The effectiveness of these testing techniques is greatly limited when the oracle is absent. The MT technique proposed in this paper provides a test result verification mechanism alternative to oracle, and thus can conduct effective testing without the need of oracles. MT has been used to alleviate the oracle problem of fault-based testing (Chen,Tse et al. 2003) and symbolic execution (Chen, Tse et al. 2011). It is interesting to study how MT can be integrated with other Web service testing techniques, aiming at effective testing in the absence of oracles.

Several researchers have conducted studies on the oracle problem in Web services. Tsai, Chen, Paul, Huang, Zhou, and Wei (2005) proposed a technique called adaptive service testing and ranking with automated oracle generation and test case ranking (ASTRAR), where a set of Web services with the same specification are executed, and a voting algorithm is applied to the outputs of these Web services to find the majority output, which will be used to form the oracle. Such an approach is effectively N-version programming (Knight & Leveson, 1986). ASTRAR is applicable when there are a large number of Web services with the same specification. In addition, it is well known that N-version programming is not always a reliable

method to the oracle problem. Our MT method can test a single Web service, and provide a reliable test output verification mechanism alternative to the oracle. Chan, Cheung, and Leung (2007) have proposed to use MT in the online testing of service-oriented software applications. Their method takes the successful test cases for offline testing as the source test cases for online testing. However, they have assumed the existence of an oracle during the offline testing. Our method never has such an assumption.

The MT technique has been applied to solve the oracle problem in various domains, such as machine learning (Murphy, Kaiser et al., 2008) and bioinformatics (Chen et al., 2009). However, these studies only proposed MRs suitable for specific application domains, without studying the unique features of these domains and the possible impacts on applying MT technique. In this paper, we not only explore a new application domain for the MT technique, but also investigate how to integrate MT into the unique environment of Web services and thus propose a comprehensive framework for applying MT into Web service testing.

CONCLUSION AND FUTURE WORK

Web services must be trustworthy before they are integrated to construct SOA applications. Testing presents a practical approach for ensuring the trustworthiness of Web services. Testing normally consists of test case generation, test execution, and test result verification against test oracle. Most existing testing techniques assume the existence of oracles when they are employed to test Web services. However, some unique features, such as the lack of source code and the limited control on Web services, restrict the testability of Web services, and thus make the oracle problem more prominent. This indicates that existing testing techniques may not be applicable to Web Services. We have presented a novel testing technique for Web services to address the challenge of testing SOA applications. The proposed approach leverages metamorphic relations to generate test cases and verify test results. Follow-up test cases are constructed based on source test cases and according to metamorphic relations. The outputs of source and follow-up test cases are compared against metamorphic relations. Thus, our approach can verify the test results without oracles. We have also proposed a testing framework for Web services which combines the principle of metamorphic testing with the unique features of SOA. To make the framework more practical and efficient, we have implemented a prototype which automates some components in the framework. We have conducted empirical studies to validate the feasibility and effectiveness of our approach. In the experiments, three real-life and commonly-used Web services were selected as subject programs. Testing such Web services faces the oracle problem to some extent. Mutation analysis is used to seed possible faults into the implementations of Web services under test. In three case studies, 77.5% ~ 94.1% mutants are killed, which indicates exciting performance of our approach. The results of the empirical studies clearly show that our approach delivers an effective and efficient testing technique for Web services without oracles.

In our future work, we would enhance the automation capability of the prototype developed in this study. Another work is to conduct more empirical studies to further evaluate the effectiveness and identify limitations of the proposed approach. Finally, we want to explore the combination of the proposed approach with existing testing techniques for Web services to improve their applicability.

ACKNOWLEGMENTS

This research is supported by the Fundamental Research Funds for the Central Universities (New Testing techniques and Tools for SOA Applications), the National Natural Science Foundation of China (Grant No. 60903003), the

Beijing Natural Science Foundation of China (Grant No. 4112037), the Research Fund for the Doctoral Program of Higher Education of China (Grant No.2008000401051), the Open Funds of the State Key Laboratory of Computer Science of Chinese Academy of Science (Grant No. SYSKF1105) and a discovery grant of the Australian Research Council (Grant No.DP0771733). This paper is an extended version of paper titled "Metamorphic Testing for Web Services: Framework and a Case Study" in Proceedings of the 9th IEEE International Conference on Web Services (ICWS 2011).

REFERENCES

Andrews, J. H., Briand, L. C., & Labiche, Y. (2005). Is mutation an appropriate tool for testing experiments? In *Proceedings of the 27th International Conference on Software Engineering*, St. Louis, MO (pp. 402-411).

Bai, X., Lee, S., Tsai, W.-T., & Chen, Y. (2008). Ontology-based test modelling and partitioning testing of web services. In *Proceedings of the 6th International Conference on Web Services*, Beijing, China (pp. 465-472).

Bartolini, C., Bertolino, A., Elbaum, S., & Marchetti, E. (2009). Whitening SOA testing. In *Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, Amsterdam, The Netherlands (pp. 161-170).

Bartolini, C., Bertolino, A., Marchetti, E., & Polini, A. (2009). WS-TAXI: A WSDL-based testing tool of web services. In *Proceedings of the 2nd International Conference on Software Testing Verification and Validation*, Denver, CO (pp. 326-335).

Canfora, G., & Di Penta, M. (2009). Service Oriented Architecture testing: A survey. In A. De Lucia & F. Ferrucci (Eds.), *Proceedings of the International Summer Schools Lectures on Software Engineering* (LNCS 5413, pp. 78-105).

Chan, W. K., Cheung, S. C., & Leung, K. R. P. H. (2007). A metamorphic testing approach for online testing of service oriented software applications. *International Journal of Web Services Research*, *4*(2), 61–81. doi:10.4018/jwsr.2007040103

Chen, T. Y. (2010). Metamorphic testing: A simple approach to alleviate the oracle problem. In *Proceedings of the 5th IEEE International Symposium on Service Oriented System Engineering*, Nanjing, China (pp. 1-2).

Chen, T. Y., Cheung, S. C., & Yiu, S. M. (1998). *Metamorphic testing: A new approach for generating next test cases* (Tech. Rep. No. HKUST-CS98-01). Clear Water Bay, NT, Hong Kong: Hong Kong University of Science and Technology.

Chen, T. Y., Ho, J. W. K., Liu, H., & Xie, K. (2009). An innovative approach for testing bioinformatics programs using metamorphic testing. *BMC Bioinformatics*, *10*, 14. doi:10.1186/1471-2105-10-24

Chen, T. Y., Huang, D. H., Tse, T. H., & Zhou, Z. Q. (2004). Case studies on the selection of useful relations in metamorphic testing. In *Proceedings of the 4th Ibero-American Symposium on Software Engineering and Knowledge Engineering*, Madrid, Spain (pp. 569-583).

Chen, T. Y., Kuo, F. C., Liu, Y., & Tang, A. (2004). Metamorphic testing and testing with special values. In *Proceedings of the 5th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, Beijing, China (pp. 128-134).

Chen, T. Y., Tse, T. H., & Zhou, Z. Q. (2003). Fault-based testing without the need of oracle. *Information and Software Technology*, *45*(1), 1–9. doi:10.1016/S0950-5849(02)00129-5

Chen, T. Y., Tse, T. H., & Zhou, Z. Q. (2011). Semi-proving: An integrated method for program proving, testing, and debugging. *IEEE Transactions on Software Engineering*, *37*(1), 109–125. doi:10.1109/TSE.2010.23

DeMillo, R. A., Lipton, R. J., & Sayward, F. G. (1978). Hints on test data selection: Help for the practicing programmer. *IEEE Computer*, *1*(4), 31–41.

Dong, G. W., Xu, B. W., Chen, L., Nie, C. H., & Wang, L. L. (2009). Survey of metamorphic testing. *Journal of Frontiers of Computer Science and Technology*, *3*(2), 130–143.

Farooq, A., Georgieva, K., & Dumke, R. R. (2008). Challenges in evaluating SOA test processes. In R. R. Dumke, R. Braungarten, G. Büren, A. Abran, & J. J. Cuadrado-Gallego (Eds.), *Proceedings of the International Conferences IWSM, Metrikon and Mensura on Software Process and Product Measurement* (LNCS 5338, pp. 107-113).

Haas, H., & Brown, A. (2004). *Web services glossary*. Retrieved from http://www.w3.org/TR/ws-gloss/

Heckel, R., & Lohmann, M. (2004). Towards contract-based testing of web services. In *Proceedings of the International Workshop on Test and Analysis of Component Based Systems*, Barcelona, Spain (pp. 145-456).

Knight, J. C., & Leveson, N. G. (1986). An experimental evaluation of the assumption of independence in multi-version programmings. *IEEE Transactions on Software Engineering*, *12*(1), 96–109.

Lenz, C., Chimiak-Opoka, J., & Breu, R. (2007). Model driven testing of SOA-based software. In *Proceedings of the Workshop on Software Engineering Methods for Service-Oriented Architecture*, Hannover, Germany (pp. 99-110).

Murphy, C., Kaiser, G., Hu, L., & Wu, L. (2008). Properties of machine learning applications for use in metamorphic testing. In *Proceedings of the 20th International Conference on Software Engineering and Knowledge Engineering*, San Francisco Bay, CA (pp. 867-872).

Offutt, J., Ma, Y. S., & Kwon, Y. R. (2004). An experimental mutation system for Java. *ACM SIGSOFT Software Engineering Notes*, *29*(5), 1–4. doi:10.1145/1022494.1022537

Offutt, J., & Xu, W. (2004). Generating test cases for web services using data perturbation. *ACM SIGSOFT Software Engineering Notes*, *29*(5), 1–10. doi:10.1145/1022494.1022529

Papazoglou, M., Traverso, P., Dustdar, S., & Leymann, F. (2008). Service-oriented computing: A research roadmap. *International Journal on Cooperative Information Systems*, *17*(2), 223–255. doi:10.1142/S0218843008001816

Peltz, C. (2003). *Web services orchestration: A review of emerging technologies, tools, and standards* (Tech. Rep.). Palo Alto, CA: Hewlett-Packard Company. Retrieved from http://devresource.hp.com/drc/

Ruth, M., & Tu, S. (2007). A safe regression test selection technique for web services. In *Proceedings of the 2nd International Conference on Internet and Web Application and Services*, Mauritius, République de Maurice (p. 47).

Sosnoski, D. (2009). *Electronic source with authors and publication time*. Retrieved from http://www.ibm.com/developerworks/java/library/j-jws6/index.html

Sun, C. (2011). On open issues on SOA-based software development. *Journal of China Science Paper Online*. Retrieved from http://www.paper.edu.cn/index.php/default/releasepaper/content/201107-461

Sun, C., Khoury, E., & Aiello, M. (2011). Transaction management in service-oriented systems: Requirements and a proposal. *IEEE Transactions on Services Computing*, 4(2), 167-180.

Sun, C., Wang, G., & Zhao, Y. (2011). Web service development: A process framework and case study. *Journal of China Science Paper Online*. Retrieved from http://www.paper.edu.cn/index.php/default/releasepaper/content/201104-18

Tsai, W. T., Chen, Y., Paul, R., Huang, H., Zhou, X., & Wei, X. (2005). Adaptive testing, oracle generation, and test case generation for web services. In *Proceedings of the 29th International Computer Software and Applications Conference*, Edinburgh, UK (Vol. 2, pp. 101-106).

Weyuker, E. J. (1982). On testing non-testable programs. *The Computer Journal*, *25*(4), 465–470.

Wu, P., Shi, X. C., Tang, J. J., Lin, H. M., & Chen, T. Y. (2005). Metamorphic testing and special case testing: A case study. *Journal of Software*, *16*(7), 1210–1220. doi:10.1360/jos161210

Zhang, J. (2011). A Mobile Agent-Based Tool Supporting Web Services Testing. *Wireless Personal-Communications*, *56*(1), 147-172.

Zhang, J. and Qiu, R.G. (2006). Fault Injection-based Test Case Generation for SOA-oriented Software. In *Proceedings 2006 IEEE International Conference on Service Operations and Logistics, and Informatics*, Shanghai, China (pp. 1070-1078).

Chang-ai Sun is an Associate Professor in the School of Computer and Communication Engineering, University of Science and Technology Beijing. Before that, he was an Assistant Professor at Beijing Jiaotong University, China, a postdoctoral fellow at the Swinburne University of Technology, Australia, and a postdoctoral fellow at the University of Groningen, The Netherlands. He received the bachelor's degree in Computer Science from the University of Science and Technology Beijing, China, and the PhD degree in Computer Science from the Beijing University of Aeronautics and Astronautics, China. His research interests include software testing, software architecture, and Service-Oriented Computing.

Guan Wang is a master student at the School of Computer and Communication Engineering, University of Science and Technology Beijing. He received a bachelor degree in Computer Science from University of Science and Technology Beijing. His current research interests include software testing and Service-Oriented Computing.

Baohong Mu is a PhD student at the School of Computer and Communication Engineering, University of Science and Technology Beijing. He received a master degree in Computer Science from Taiyuan University of Technology. His current research interests include software testing and Service-Oriented Computing.

Huai Liu is a Research Associate at the Faculty of Information and Communication Technologies in Swinburne University of Technology. He received his PhD degree in Software Engineering from Swinburne University of Technology, Australia, and M.Eng. in Communications and Information Systems and B.Eng. in Physioelectronic Technology from Nankai University, China. His current research interests include software testing, web services, telecommunications, and end-user software engineering.

ZhaoShun Wang is a Professor in the School of Computer and Communication Engineering, University of Science and Technology Beijing, China. He obtained the PhD degree in Computer Science from University of Science and Technology Beijing, and BSc and MPhil. in Mathematics from Beijing Normal University. His research interests include software testing and information security.

T. Y. Chen is a Chair Professor of Software Engineering at the Faculty of Information and Communication Technologies in Swinburne University of Technology. He received his PhD degree in Computer Science from the University of Melbourne; MSc, and DIC in Computer Science from Imperial College of Science and Technology; and BSc, and MPhil. from The University of Hong Kong. His current research interests include software testing and debugging, software maintenance, and software design.