2011 IEEE International Conference on Web Services

Metamorphic Testing for Web Services: Framework and a Case Study

Chang-ai Sun[1*], Guan Wang[1], Baohong Mu[1], Huai Liu[2], ZhaoShunWang[1], T.Y. Chen[2]

[1]School of Computer and Communication Engineering
University of Science and Technology Beijing
Beijing, China
casun@ustb.edu.cn

[2]Faculty of Information and Communication Technologies
Swinburne University of Technology
Melbourne, Australia
{hliu,tychen}@swin.edu.au

***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. Since the development and consumption of Web services are completely separated under SOA environment, the consumers are normally provided with limited knowledge of the services and thus have little information about test oracles. The lack of source code and the restricted control of Web services limit the testability of Web services.**

To address the prominent oracle problem when testing Web services, we propose a metamorphic testing framework for Web services taking into account the unique features of SOA. We conduct a case study where the new metamorphic testing framework is employed to test a Web service that implements the electronic payment. The results of case study show the feasibility of the framework for web services, and also the efficiency of metamorphic testing. The work presented in the paper alleviates the test oracle problem when testing Web services under SOA.

Keywords-Web services; software testing; metamorphic testing; test oracle; Service Oriented Architecture

I. Introduction

Service Oriented Architecture (SOA) has been evolving as a mainstream software development paradigm where Web services are basic elements [24]. 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 Description Language (WSDL)[17]. Under 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 [25].

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 the change of 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. However, the unique features of SOA pose new challenges for testing. For instance, white-box testing techniques become inapplicable for the service consumer to test Web services due to the lack of source code. 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 [28], is even amplified. In the example of the electronic payment service, the consumer, under 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. Thus, testing Web service under SOA calls for new testing techniques [3, 5, 16].

This paper proposes a metamorphic testing framework for Web services to address the challenges of testing Web service under SOA. Metamorphic Testing (MT) was first introduced by Chen et. al [7], and it has been shown that MT has successfully alleviated the test oracle problem [15]. We investigate how to apply MT into the testing of Web services, and report a case study. The paper makes the following contributions:

1. A MT framework which examines and answers the key issues when using MT to test Web services. The framework combines the basic principle of MT with unique features of SOA.

* Contact author

2. An efficient testing technique for Web services. As to be observed from the results of the mutation analysis, MT can detect nearly 80% faults of the subject Web service without the need of oracles.
3. A case study which describes how MT can be employed to test a representative and widely-practiced Web service and reports the effectiveness. The case study clearly shows the applicability of MT for testing Web services.

The rest of the paper is organized as follows. Section II introduces underlying concepts related to MT and mutation analysis. Section III presents a framework of MT for Web services. Section IV reports a case study where MT is employed to test a Web service implementing the electronic payment. Related work is discussed in Section V. Section VI concludes the paper and points out the future work.

II. Background

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

A. *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 code 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 et al. [4] developed a tool called TAXI that generates test cases for Web services based on WSDL specifications. Bai et al. [2] proposed an ontology-based partition testing approach for Web services. Lenz et al. [21] applied model-driven approaches to the testing of Web services. Many other testing methods for web services can be found in the literature, such as contract-based Web services testing [18], fault-based Web services testing [23], and regression Web services testing [26], etc.

B. *Metamorphic Testing*

Most testing techniques proposed so far are focused on how to effectively select test cases such that program faults can be revealed as early as possible 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 [7] 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 detecting bugs in various programs [11, 20].

C. *Mutation Analysis*

Mutation analysis [14] 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

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

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 [1]. In this paper, we will use mutation analysis technique to evaluate the effectiveness of our testing method.

III. 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 the need of 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 the need of 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*.

(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

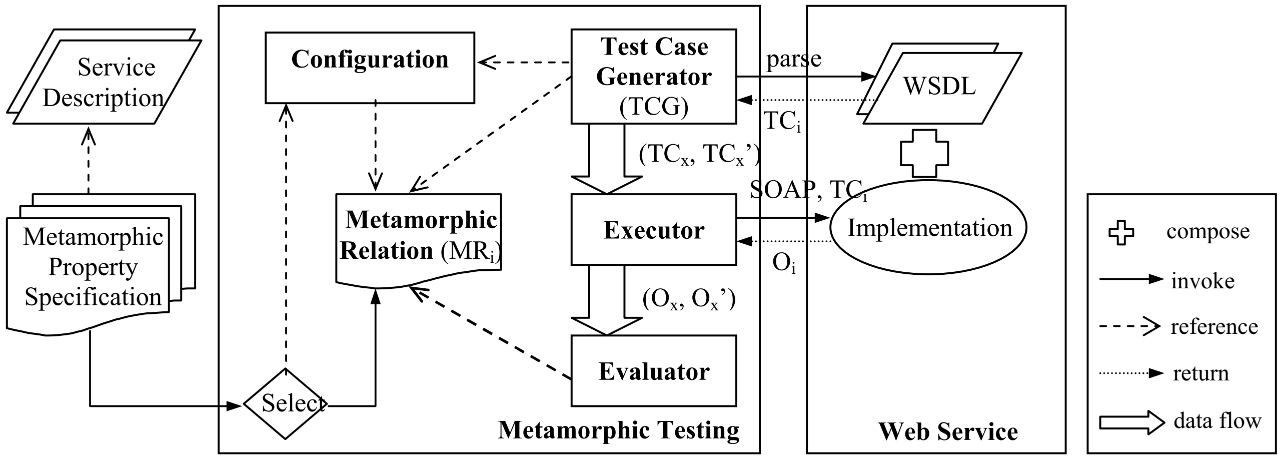


Figure 1. The framework of metamorphic testing for Web services

situation, the configuration component will specify an option whether the testing should stop or continue by trying another MR.

Currently, we have implemented a prototype which partially automates the framework. We apply the framework to a real-life Web service, validate its feasibility, and evaluate the effectiveness of MT for Web services.

IV. A Case Study

In this section, we describe a case study to validate the MT framework for testing Web services, and report the effectiveness of MT. The electronic payment service is selected as the subject program because of the test oracle as discussed before. Mutation analysis is used to evaluate the effectiveness of MT. The results of the case study show that MT can detect about 80% mutants.

A. *Subject program*

A general ATM (Automatic Teller Machine) system is implemented as Web service and deployed in the Tomcat server. 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 port. Among these features, we select the transfer feature for the case study because it is widely practised in the electronic payment and the oracle problem arises when testing such a feature.

Figure 2 shows a segment of WSDL for the transfer feature. The implementation consists of 136 lines of Java codes, and executes the connection to relevant database, SQL statements, and numerical computing on the transfer amount and commission fee. For the commission fee charging criterion, we refer to Agricultural Bank of China for the calculation rules as shown in Table I. Transfer types I-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 I, 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.

```
<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>
```

Figure 2. A segment of WSDL for the transfer interface

TABLE I. 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

B. *Applying MT to Test the Transfer Interface*

We describe how MT can be used to effectively test Web services without the need of oracles.

*(1) Inputs and Output*s. 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 I. Note that the transfer type can 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 ($\triangle A$, $\triangle B$), where

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

Note that both $\triangle A$ and $\triangle B$ 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.

(2)Deriving Metamorphic Relations (MRs). The selection of MRs is a key issue during the application of MT

[12]. Some guidelines are available for selecting MRs from the specification. In particular, Chen et al. [9-10, 12] discover 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.

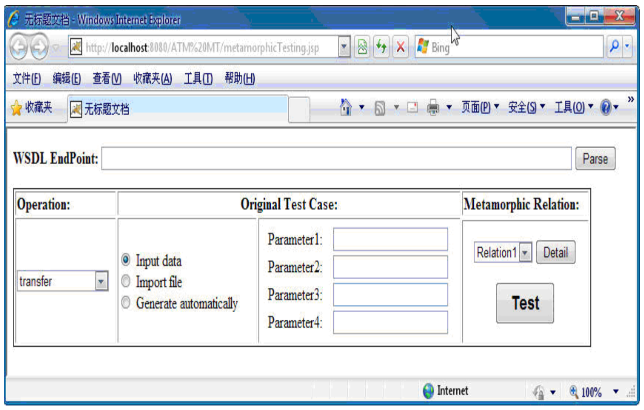
According to the guidelines, we derive a set of MRs for the transfer and they are listed in Table II. In this study, all the selected MRs can be decomposed such that each MR is a pair of R and R_f, where R denotes the relation between source and follow-up test cases (inputs) and R_f denotes the relation between their outputs. From MR1 to MR5, the follow-up test cases are derived from their *source test cases* via changing only one tuple once. Considering MR1, 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 MR6, the follow-up test cases are derived from their source test cases via exchanging the sender account with the recipient account.

TABLE II. A SET OF MRS FOR THE THE TRANSFER FEATURE

MR	R	R_f
MR1	M'=2M	△A'≤2△A and △B'=2△B
MR2	P= 1 and P'= 2	△A'- △B'=△A- △B
MR3	P= 0 and P'≠ 0	△A'- △B'>△A-△B
MR4	P= 3 and P'≠ 3	△A'- △B'≤△A-△B
MR5	M'>M	△A'> △A and △B'>△B
MR6	A'=B and B'=A	△A'=△B

Note that the derivation of these MRs is based on the specification shown in Table I, and is completely independent of the implementation. This means that MT does not need to access the coding and hence is widely applicable to SOA-based applications.

(3)Test case generation 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 [9, 29]. Thus we employ the random test value generation to generate source test cases in our case study. For the follow-up test cases, they are accordingly constructed from their source test cases using MRs defined in Table II.


无标题文档 - Windows Internet Explorer
http://localhost:8080/ATM%20MT/metamorphicTesting.jsp
Bing
文件(F) 编辑(E) 查看(V) 收藏夹(A) 工具(T) 帮助(H)
收藏夹
无标题文档
页面(P) 安全(S) 工具(O)
Parse
Detail
Test
Internet
100%

WSDL EndPoint:

Operation: transfer

Original Test Case: [✓ Input data] Import file Generate automatically

Parameter1:
Parameter2:
Parameter3:
Parameter4:

Metamorphic Relation: Relation1

Figure 3. The snapshot of MT platform prototype

(4) Test Execution. With the test cases generated above, we now can execute and test the transfer feature. In order to make the testing efficient, we develop a platform prototype supporting the MT framework for Web services discussed above. Figure 3 shows a snapshot of the platform. With the platform, one can select one or more MRs to test the transfer feature. The platform 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.

C. *Evaluation and discussions*

We here describe several experiments where mutation analysis is used to evaluate the effectiveness of MT.

Firstly, we seed faults into the implementation at the level of methods by mutation operators. This is done automatically by MuJava [22], and we hereby have a total of 139 mutants. Among them, 10 mutants are equivalent mutants and thus are excluded from experiments. Secondly, we employ the platform to generate test suites which are based on MRs in Table II. Finally, these test suites are used to test the subject program. A fault is said to be detected (that is, a mutant is killed) when an MR is violated (that is, the source and follow-up test cases satisfy R, while their outputs do not satisfy R_f).

We use mutation score (MS) and fault discovery rate (FDR) as metrics to measure the performance of MT. MS indicates the adequacy of a test suite *ts* against the program under test, while FDR indicates the detection capability of a test suite *ts* against a mutant *m*, namely

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

where N_f refers to the number of test cases that can kill the

TABLE III. A SUMMARY OF AVERAGE FDR OF 129 MUTANTS USING MT

		MR1	**MR2**	**MR3**	**MR4**	**MR5**	**MR6**
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%

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 are referred to those that do not work properly for a given MR. In the experiment, it is possible to derive some invalid follow-up test cases, because source test cases are randomly and automatically generated. Consider the MR1 in Table II, if one source test case is (1000000000, 2000000000, 3, 50000), its follow-up test case should be (1000000000, 2000000000, 3, 100000). However, such a follow-up test case violates the rules given in the Table I. These invalid test cases should not be included in our experiments.

(1) FDR evaluation results. Table III summarizes the average FDRs of all 129 distinct 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 more conclusive and stable. We observe that MR2, MR3 and MR1 are more effective compared with other MRs, and thus should have higher priority when MT is employed.

We further select ten mutants from 129 distinct 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 IV summarizes the mutation description of these mutants. Table V reports the FDRs on the ten

TABLE IV. 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

mutants when MT is used to test the transfer. Each cell of Table V shows the FDR of a test suite generated by an MR on a mutant. For example, the right-bottom cell represents that the test suite with the size of 200 test cases generated by MR6 has an FDR of 100% on M133. We can observe from Table V that

TABLE V. A SUMMARY OF FDR FOR TEN MUTANTS WHEN THE SIZE OF VALID TEST CASES IS 50, 100 AND 200

ID		MR1	MR2	MR3	MR4	MR5	MR6
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 VI. A SUMMARY OF MUTATION SCORE (MS) OF 129 MUTANTS USING MT

	MR1	MR2	MR3	MR4	MR5	MR6	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%

- Each MR has a varying sensitivity to different mutants. For instance, MR1 is sensitive to M004 while not sensitive to M057; MR3 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. Tester 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.
- Those FDRs that are neither 0% nor 100% vary a bit with the changing size of valid test cases. That is, when the size of valid test cases satisfying the same MR changes, the FDR may change even for the same mutant. This shows the dependency of source test cases in MT.

(2) MS evaluation results. Table VI 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. It can be observed from Table VI that

- The MS of each MR can be used to compare their effectiveness. Among these MRs, MR1, MR2, and MR3 are more effective, while MR4, MR5 and M6 are less effective. For instance, the MS of MR1 (45.0%) is larger than that of MR6 (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 IV, the priority order of these MRs should be MR1>MR3>MR2>MR4>MR5 >MR6.

(3) Summary. Through this case study, we have validated the feasibility of the MT framework for Web services, and evaluated the effectiveness of MT. No oracles are needed when MT is employed to test the transfer implemented in a Web service. This greatly alleviates the more prominent oracle problem when testing Web services under SOA. Furthermore, the experimental results show that up to 77.5% mutants are killed, which demonstrates a high fault detection capability of MT. In a word, the proposed MT framework is an effective and efficient testing technique without the need of oracle for Web services.

V. RELATED WORK

As pointed out in Section II.A, there are various techniques for the testing of Web services in the literature. However, most of these techniques are focused on the selection of test cases for Web services. It is often assumed that there exists an oracle. The effectiveness of these testing techniques is greatly limited when the oracle is absent. The MT technique used in this paper can conduct effective testing without the need of oracles. MT has been used to alleviate the oracle problem of fault-based testing [8] and symbolic execution [13]. 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 et al.[27] 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 [19]. 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 et al. [6] 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.

VI. CONCLUSIONS AND FUTURE WORK

We have presented a novel testing technique for Web services to address the challenge of testing SOA applications. Using metamorphic testing technique, one can effectively test Web services without the need of oracles. A framework of metamorphic testing was proposed which combines the principle of metamorphic testing with the unique features of SOA. A case study has been conducted where the proposed framework was used to test a representative Web service. The results of the case study showed the feasibility of the framework, and demonstrated the effectiveness of metamorphic testing. The work presented in the paper alleviates the test oracle problem when testing Web services under SOA.

In our future work, we would enhance the automation capability of the metamorphic testing platform prototype developed in this study. Another work is to conduct more empirical studies to further evaluate the effectiveness of metamorphic testing for Web services in practice.

ACKNOWLEDGMENT

This research is supported by 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), and Australian Research Council (Grant No.DP0771733).

REFERENCES

[1] J. H. Andrews, L. C. Briand, and Y. Labiche, "Is mutation an appropriate tool for testing experiments?", *Proceedings of the 27th International Conference on Software Engineering (ICSE2005)*, 2005, pp402-411.

[2] X. Bai, S. Lee, W.-T. Tsai, Y. Chen, "Ontology-based test modelling and partitioning testing of web services", *Proceedings of the 6th International Conference on Web Services*, 2008, pp465-472.

[3] C. Bartolini, A. Bertolino, S. Elbaum, E. Marchetti. "Whitening SOA testing", *Proceedings of ESEC-FSE'09*, ACM Press, 2009, pp161-170.

[4] C. Bartolini, A. Bertolino, E. Marchetti, A. Polini, "WS-TAXI: A WSDL-based testing tool of web services", *Proceedings of the 2nd International Conference on Software Testing Verification and Validation (ICST2009)*, 2009, pp326-335.

[5] G. Canfora, M. Di Penta. "Service Oriented Architecture testing: A survey", *LNCS 5413*, Springer, 2009, pp78–105.

[6] W. K. Chan, S. C. Cheung, K. R. P. H. Leung, "A metamorphic testing approach for online testing of service oriented software applications". *International Journal of Web Services Research*, 2007, 4(2):61-81.

[7] T.Y. Chen, S.C. Cheung, S.M. Yiu. "Metamorphic testing: A new approach for generating next test cases", Technical Report HKUST-CS98-01, Hong Kong University of Science and Technology, 1998.

[8] T. Y. Chen, T. H. Tse, Z. Q. Zhou, "Fault-based testing without the need of oracle", *Information and Software Technology*, 2003, 45(1):1-9.

[9] T.Y. Chen, F.C. Kuo, Y. Liu, A. Tang, "Metamorphic testing and testing with special values", *Proceedings of SNPD2004*, 2004, pp128-134.

[10] T.Y. Chen, D.H. Huang, T.H. Tse, Z. Q. Zhou, "Case studies on the selection of useful relations in metamorphic testing", *Proceeding of the 4th lbero-American Symposium on Software Engineering and Knowledge Engineering*(JIISIC 2004), 2004, pp569-583.

[11] T. Y. Chen, J. W. K. Ho, H. Liu, X. Xie, "An innovative approach for testing bioinformatics programs using metamorphic testing". *BMC Bioinformatics*, 2009, vol. 10, Article 14.

[12] T.Y. Chen, "Metamorphic testing: A simple approach to alleviate the oracle problem". *Proceedings of Fifth IEEE International Symposium on Service Oriented System Engineering*, 2010, pp1-2.

[13] T. Y. Chen, T. H. Tse, Z. Q. Zhou, "Semi-proving: an integrated method for program proving, testing, and debugging", *IEEE Transactions on Software Engineering*, 2011, 37(1): 109-125.

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

[15] G. W. Dong, B.W. Xu, L. Chen, C.H. Nie, L.L. Wang, "Survey of metamorphic testing", *Journal of Frontiers of Computer Science and Technology*, 2009, 3(2):130-143.

[16] A. Farooq, K. Georgieva, R. R. Dumke. "Challenges in evaluating SOA test processes", *LNCS 5338*, Springer, 2008, pp107–113.

[17] H. Haas, A. Brown, "W3C, Web Services Glossary", http://www.w3.org/TR/ws-gloss/, 2004.

[18] R. Heckel, M. Lohmann, "Towards contract-based testing of web services", *Proceedings of the 2004 International Workshop on Test and Analysis of Component Based Systems (TACoS2004)*, 2004, pp145-456.

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

[20] C. Murphy, G. Kaiser, L. Hu, L. Wu, "Properties of machine learning applications for use in metamorphic testing", *Proceedings of the 20th International Conference on Software Engineering and Knowledge Engineering (SEKE2008)*, 2008, pp867-872.

[21] C. Lenz, J. Chimiak-Opoka, R. Breu, "Model driven testing of SOA-based software", *Proceedings of the Workshop on Software Engineering Methods for Service-Oriented Architecture (SEMSOA2007)*, 2007, pp99-110.

[22] J. Offutt, Y.S. Ma, Y.R. Kwon, "*An experimental mutation system for Java*", ACM SIGSOFT Software Engineering Notes, Workshop on Empirical Research in Software Testing, 2004, 29(5): 1-4.

[23] J. Offutt, W. Xu, "Generating test cases for web services using data perturbation", *ACM SIGSOFT Software Engineering Notes*, 2004, 29(5):1-10.

[24] M. Papazoglou. P. Traverso, S. Dustdar, F. Leymann. "Service-oriented computing: a research roadmap", *International Journal on Cooperative Information Systems (IJCIS)*, 2008, 17(2): 223-255.

[25] C. Peltz. "Web services orchestration: a review of emerging technologies, tools, and standards". Technical Report, 2003, Hewlett-Packard Company, http://devresource.hp.com/drc/

[26] M. Ruth, S. Tu, "A safe regression test selection technique for web services", *Proceedings of the 2nd International Conference on Internet and Web Application and Services* (ICIW2007), 2007, pp47.

[27] W. T. Tsai, Y. Chen, R. Paul, H. Huang, X. Zhou, X. Wei, "Adaptive testing, oracle generation, and test case generation for web services", *Proceedings of the 29th International Computer Software and Applications Conference (COMPSAC2005)*, 2005, vol. 2, pp101-106.

[28] E. J. Weyuker. "On testing non-testable programs", *The Computer Journal*, 1982, 25(4):465-470.

[29] P. Wu, X.C. Shi, J.J. Tang, H.M. Lin, T.Y. Chen, "Metamorphic testing and special Case testing: a case study", *Journal of Software*, 2005. 16(7):1210-1220.