A Framework for Service-Oriented Testing of Web Services

Hong Zhu
Department of Computing, Oxford Brookes University, Oxford OX33 1HX, UK
Email: hzhu@brookes.ac.uk

Abstract

Testing Web Services (WS) application systems is difficult and expensive. It imposes great challenges to existing testing methods, techniques and tools. This paper analyses the problems in testing WS applications and proposes a service oriented framework to solve the problems. It enables collaborations between various parties involved in the development of WS applications via service request and service providing. It also enables special testing services to be provided as WS to perform testing tasks on behalf of their customers. The key technical issues of the approach are discussed.

1 Introduction

The recent years has seen a rapid growth of the development of Web Services (WS) technology. In comparison with other distributed computing techniques such as CORBA, Java RMI and DCOM, WS offers more flexibility and looser coupling so that it is more suitable for internet computing [1]. It is characterised by the dominant of program-to-program interactions [2]. In view of the infrastructure of WS becoming pervasive, a new paradigm of service-oriented computing is emerging. As Stal pointed out [3], it is fundamentally different from the others. The components of WS applications, such as service providers, are autonomous, active and persistent computational entities that control their own resources and their own behaviours. They have social ability and collaborate with each other through dynamic discovery and invocation of services. It is widely recognised that WS technologies will profoundly change the ways that computer systems and software are developed and used [4]. However, the current infrastructure and standards of WS do not support adequate testing of WS applications.

One of the key features of service-oriented computing is that a requester's search for service providers, and an invocation and delivery of a service can all be determined at run-time. To enable such dynamic composition of services, standards have been developed for service registration, service enquiry and retrieval, service request and delivery. The stack of standards of WS includes WSDL for service description and publication [5], UDDI for service registration and retrieval [6], and SOAP for service invocation and delivery [7]. More advanced standards were also being developed to enable collaborations between service providers and requesters. For example, BPEL4WS uses notions of business process and workflow models. OWL-S is based on ontology for the description of semantics of services. Methodologies for developing WS have also been proposed; e.g. [8].

Despite of the active research on WS technology, few works have also been reported in the literature for testing WS and quality assurance of WS applications. For example, in [9], metamorphic testing methods are applied to testing WS applications. In [10, 11, 12], multiple WS applications that provide the same services are regarded as a multiple version system to enable the comparison of testing results. These methods test services as black-boxes. Other testing methods have also proposed to test specific aspects of WS applications, such as testing the XML schema [13, 14], and by modifying the data passing between the services [15], etc. While these works are important and make positive contributions to the quality assurance of WS applications, the main difficulties in testing WS applications are yet to be addressed.

In this paper, we first analyse the impact of the novel features of service-orientation on software testing and identify the requirements on infrastructural support to testing WS applications. We then propose an approach to the solution, which also takes the advantages of service-oriented architectures by regarding software testing as services. Finally, based on our previous work on software testing ontology, we discuss the key technical issues that must be addressed to facilitate the solution.

2 The Challenges

To analyze the challenges of WS technology to software testing, let's consider the following scenario of a typical e-commence application.

2.1 A typical scenario

Suppose that a fictitious car insurance broker CIB is developing a web-based system that provides a complete service of car insurance. In particular, the end users should be able to submit car insurance requirements and get quotes from various insurers that the car insurance broker CIB is connected to, and then select one insurer to insure the car. To do so, the broker will take information from the user of the car, its usages,

Proceedings of the 30th Annual International Computer Software and Applications Conference (COMPSAC'06)
0-7695-2655-1/06 $20.00 © 2006 IEEE

and the payment. It will also check the validity of user's payment information, pass the payment to the selected insurer as well as take commissions from the insurer or the user. Although the car insurance broker's software system may have a user interface to enable the end users' interactive uses, the system also has a WS interface to enable other programs to connect to it as its service requesters. Assume that CIB uses the WS of its bank B to check its customer's credit, validate their payments, and to perform financial transactions. Its binding to the bank's WS can be static and stable, if the company does not change its bank so frequently. As insurance is an active business domain, new insurance providers may emerge and existing ones may leave the market from time to time. It is desirable to bind the broker's software dynamically to multiple insurance providers to ensure that the business is competitive on the market. The structure of the system is

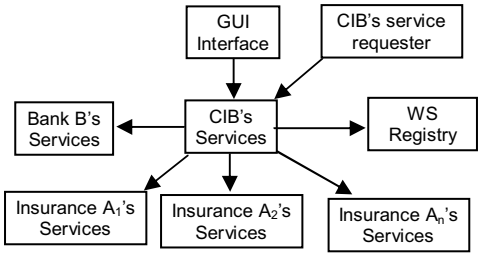

Figure 1. Structure of CIB Application

illustrated by the following diagram.

The developer of the broker CIB's services must test not only its own code, but also the integration of its own code with the WS systems of the insurers, and the bank. Both of these two testing tasks have its new features that challenge the current software testing techniques and methods of their capabilities and effectiveness. The following discusses the similarities and differences between such a testing and the corresponding testing tasks in the development of traditional software systems.

2.2 Testing Own Side Services

The testing of a service by its own developers has similarity with the testing of software components. Many existing work on software component testing can be applied or adapted to take special consideration of the WS standard into consideration. Such WS special issues include the following.

- The stateless feature of HTTP protocol;
- XML encoding of the data passing between services as in SOAP standard;
- Confirmation of implementation to the descriptions of the WS as published in WSDL for the syntax of the services, or any other standards such as workflow specification in BPEL4WS and semantic specification in an ontology description language such as OWL-S.

In addition to these issues, the testing must take the following situations into consideration.

A. Dealing with abnormal behaviours

Because of the stateless feature of HTTP protocol, the system must keep track of the progresses of all its requesters' transactions, especially when it consists of complicated workflows. Because the requesters of the service are autonomous, a requester may stop cooperation in the middle of a transaction for many reasons, such as intentional quit, network failure, or failures of requester's software system due to fault. Such abnormal behaviours of service requesters cannot be ruled out by the design and implementation of the services. Thus, burdens are on the testers to ensure that the system handles such behaviours properly.

B. Dealing with unexpected usages

As all web-based applications, load balance is essential. Therefore, load testing is necessary for testing WS. Such testing must take various usages of the system into consideration in order to obtain realistic test results. However, the knowledge of the usage of a WS may not be available during the design and implementation of the system.

C. Dealing with incomplete systems

Services may be significantly different from software components because components are mostly self-contained with well-defined interfaces. However, a service may have to rely on other services to perform its functionality properly. For example, the insurance broker CIB may provide a service to its customers that compares the prices of a number of insurance providers and makes suggestions based on the cheapest quote. This service cannot be in function without requesting the services provided by insurers. Therefore, it can be hard to separate the testing of the own services from the integration testing, especially when it involves complicated workflows. For example, for the developers of the CIB WS, it could be too costly and time consuming to build a test harness and stubs of the other services to enable the adequate testing of CIB's own services, especially when they involve a large number of different service providers using different techniques in their implementation of the services. In the worst case, because the insurers' WS are dynamically bound to the broker's services, the knowledge of their format and semantics can only be based on assumptions and standards. Adequate testing has to be postponed to integration testing when the binding actually happens.

2.3 Testing of Service Composition

While the integration testing in service composition has similarity to component integration, the difference between them is dominant and causes significant difficulties to apply existing testing techniques. It has become the technology bottleneck that hampers the wide spread of WS.

A number of software integration testing methods and techniques have been developed in the past decades of research and practices of software testing. In particular, various strategies of integration has been developed and investigated, such as top-down, bottom-up and hybrid integration strategies. These strategies and corresponding techniques aim at effective observations of the interfaces between parts of software systems through the development and uses of test drivers, component stubs and test harnesses as well as special purpose software instrumentation. A condition for applying such techniques is that the tester has access to the source code of the parts to be integrated. This condition is no longer valid in testing WS. A similar problem has been investigated in testing component-based systems, where source code is not always available, too [16, 17, 18]. However, testing service composition is even more difficult for the following reasons.

A. Lack of software artifacts

Testers not only have no access to the source code of the services provided by the other parties, but also have no control over the executable code, which typically runs on the service providers' computers over the Internet. For statically bound services, it is possible for the testers to write test harnesses based on the standards and the published description of the provided services. This could be costly and error-prone because the correctness of the stubs that represents the services could not be ensured. Techniques that automatically derive stubs from source code are not applicable. Automatic instrumentation of original source code or executable code to enable observation of the correctness of the data passing between the interfaces is also not applicable. For dynamic bound services, human involvement in the integration becomes completely impossible. Two possible solutions to this problem are: (a) automatic derive test harness from WS descriptions; (b) the service provider not only provides the functionality of the service, but also provide the service of testing and make the test stubs and drivers available for dynamic integration.

B. Lack of control over test executions

As mentioned earlier, services are typically located on a computer on the Internet that testers have no control over its execution. An invocation of the service as a test must be distinguished from a real request of the service. It is not imaginable if the tester of CIB WS actually purchases hundreds and thousands of car insurances to cover the combinations of various conditions and makers of cars with various conditions of the owners of the cars in testing his software. In many situations the results of executing a software system on a test case must be removed to set the system back to a normal state in order to carry out further test. Such controls of the software systems under test are necessary. The situation could become much more complicated when a WS is simultaneously tested by many service requesters. The only solution to this problem is that the service provider must provide a mechanism and the service that enable service requesters control the testing executions of the service. However, currently there is no support to such mechanisms in W3C standards of WS.

C. Lack of means of observation on system behaviour

It has been recognized for a long time that black-box testing alone is not adequate to ensure the correctness of software systems; see e.g. [19]. The observation of internal behaviour of a software system is necessary to achieve adequate testing. A consequence of unavailability of documentation and source and executable code of a service is that a tester cannot observe the internal behaviours of the services. There are also two possible solutions to this problem. One is that the service provider also provides a mechanism and the services to the outside tester to observe its software's internal behaviour in order to achieve the test adequacy that a service requester requires. However, for many reasons, service providers are unwilling to provide the internal information of their software to the public. The second solution is that instead of provide such a mechanism and service to the public, it can open its document, source code as well as other software artifacts that are necessary for testing to some trusted test service providers. These test service providers access the internal information of the service on behalf of the service request while keep the internal information confident. The service requesters only know the test results, which is actually what they want.

The analysis above naturally leads to a service-oriented framework to solve the problems in testing WS. The next sections further discuss the framework.

3 Architecture

In the proposed approach, a WS should be accompanied by a testing service. In the sequel, the services of the original functionality are called *functional services*, while the services to enable test the functional services are called *testing services*. Such testing ser-

vices can be either provided by the same vendor of the functional services, or by a third party. In addition to such accompany testing services, testing tool vendors and companies of specialized in software testing can also have independent testing services to perform various kinds of test tasks, such as to generate test cases, to measure test adequacy, to extract various types of diagrams from source code or design and specification documents, etc. Figure 2 illustrates the structure of service oriented testing by the CIB example.

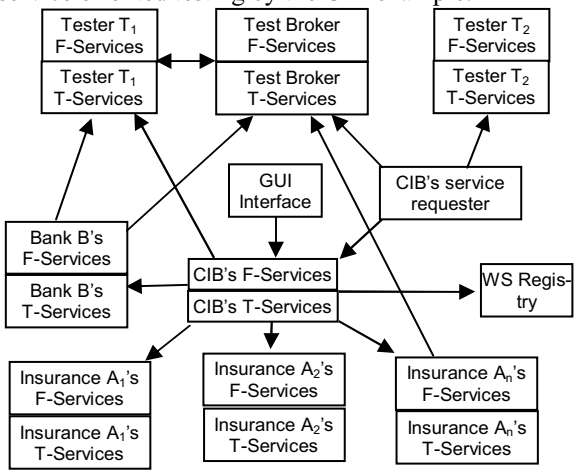

Figure 2. Illustration of Service Oriented Testing

In Figure 2, the F-Services are the functional services provided by the WS. T-Services are the corresponding testing services. They provide testing services through a control mechanism, an observation mechanism, and necessary documents with certain access control. The T_1 and T_2 Tester F-Services provide services to perform various testing tasks on behalf of its customers. Its T-Services provides the facility to test its own services as all other T-Services. They may be invoked by other F-Services when dynamic service binding requires a testing of an F-Service, or invoked by a developer during integration testing of statically bound services. A special tester service is test broker, which searches for testers who are capable of performing certain testing tasks when requested by a customer according to tester services registered to the registry.

4 Automating Test Services

The key technique issues that the proposed approach must address in order to enable automated online test of WS include the following.

(a) How a testing service should be described, published and registered at WS registry with machine understandable encoding;
(b) How a testing service can be retrieved automatically so that testing dynamically bound services can be performed automatically;
(c) How a testing service can be invoked by both a human tester and a program to dynamically discover a service and then test it before bind to it.
(d) How testing results can be summarized and reported in the forms that are suitable for both human beings to read and machine to understand.

These issues can be resolved by utilization of a software testing ontology [20, 21].

4.1 Ontology of software testing

Generally speaking, ontology defines the basic terms and relations comprising the vocabulary of a topic area as well as the rules for combining them to define extensions to the vocabulary [22]. It is widely recognised that ontology can be used where domain knowledge specification is useful. It is one of the main approaches to provide machine understandable descriptions of the semantics of WS.

Our ontology of software testing is called *STOWS,* which stands for Software Testing Ontology for WS. It is based on taxonomy of test concepts.

The concepts related to software testing are divided into two groups: the basic concepts and compound concepts. The basic concepts include *context*, *activity*, *method*, *artefact*, and *environment*. Based on these basics concepts, compound concepts were defined, which include *tester*, *capability* and *test task*.

For each basic concept, there may be a number of sub-concepts. For example, a testing activity can be the generation of test cases, the verification of test results, the measurement of test adequacy, etc. A basic concept may also be characterized by a number of properties, which are the parameters of the concept. For example, a software artefact is determined by (a) its format, such as HTML file, JavaScript, (b) its type, such as a program, or a test suite, (c) its creation and revision history, such as who and when created the artefact, and who and when revised it, and the version number of the artefact, etc. (d) the location that the artefact is stored, and (e) the data, i.e. the contents, of the artefact.

Relationships between concepts play a significant role in the management of testing activities. They are a very important part of the knowledge of software testing. They must be stored in a knowledge-base as basic facts. This type of knowledge includes the following.

- *Subsumption relation between testing methods*
- *Compatibility between artefacts' formats*
- *Enhancement relation between environments*
- *Inclusion relation between test activities*
- *Temporal ordering between test activities*

Based on these basic concepts and relations, more complicated entities, concepts and relations can be defined to provide direct support to service oriented software testing.

4.2 Registration of testing services

To register a WS that provides services of software testing, it is necessary to specify the semantics of the services in detail in addition to the syntax format in WSDL. Such a registration must provide the identity of the service provider and its capability of performing certain test tasks.

A tester refers to a particular party who carries out a testing activity. A tester can be a *human being*, a *software system*, which includes WS and software agents, etc., or a *team*, which consists of one or more testers. Its structure in UML is given in Figure 3.

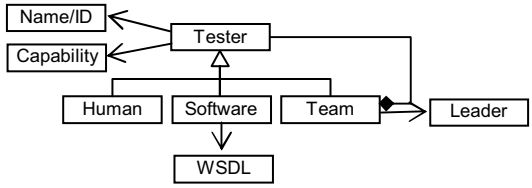

Figure 3. The Concept of Testers

The following is an example of a tester which is a software system that provides a WS. Its capability and description in WSDL are given in files at the URLs.

```
<TESTER TESTER_TYPE="SOFTWARE"
   TESTER_NAME="TestWS"
   TESTER_CAPABILITY=
    URL:"//cms.Brookes.ac.uk/STONEWS/TWSC.txt"
   TESTER_WSDL=
   URL:"//cms.Brookes.ac.uk/STONEWS/WSDL.txt"
</TESTER>
```

An important attribute of tester is capability that describes what a tester can do. It is defined in Figure 4.

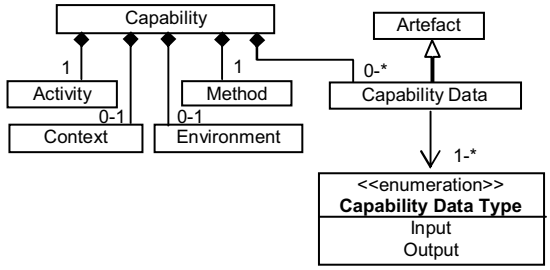

Figure 4. The Compound Concept of Capability

The capability of a provider of test services, or more generally a tester, is determined by the activities that a tester can perform together with the context to perform the activity, the testing method used, the environment to perform the testing, the required resources (i.e. the input) and the output that the tester can generate.

4.3 Request of a testing service

A request of a testing service contains a specific specification of a testing task, which is defined as a compound concept in the STOWS ontology. It specifies a testing activity and related information about how the activity is required to be performed, such as the context, the testing method to be used, the environment in which the activity must be carried out, the available resources and the expected outcomes. It can be represented by the following UML class diagram.

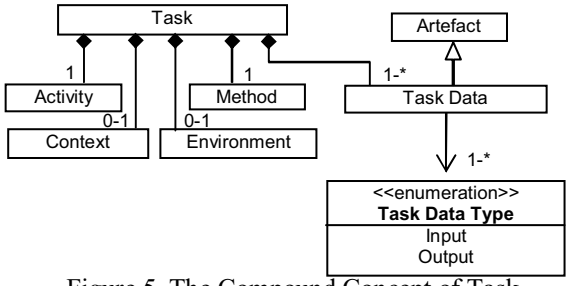

Figure 5. The Compound Concept of Task

The following is an example of testing task that requires generating test cases according to the node coverage criterion for the HTML pages at the URL http://www.brookes.ac.uk.

```
<TASK>
   <CONTEXT CONTEXT_TYPE="SYSTEM_TEST" />
   <ACTIVITY
      ACTIVITY_TYPE="TEST_CASE_GENERATION" />
   <METHOD
      METHOD_NAME="NODE_COVERAGE_TESTING" />
   <TASK_DATA TASK_DATA_TYPE="INPUT">
      <ARTEFACT
         ARTEFACT_TYPE="OBJECT_UNDER_TEST"
         ARTEFACT_FORMAT="HTML">
         <ARTEFACT_LOCATION>
            http://www.brookes.ac.uk
         </ARTEFACT_LOCATION>
      </ARTEFACT>
   </TASK_DATA>
</TASK>
```

4.4 Query and retrieval of testing services

A provided testing service can be more powerful than a required task. The query and retrieval of suitable service providers cannot be just simple syntax matching. It is crucial to the success of the proposed approach to facilitate the reasoning about the matching between test tasks and service provider's capabilities. The STOWS ontology defines three relations to support this. Figure 6 shows the structures of these compound relations.

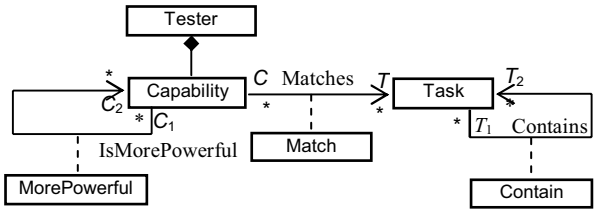

Figure 6. Compound Relations

The relation *MorePowerful* is defined between two capabilities. Informally, *MorePowerful*(c_1, c_2) means that a tester has capability c_1 implies that the tester can do all the tasks that can be done by a tester who has capability c_2. In UML, the *MorePowerful* relation is an

association class; see Figure 6 for its structure. The *MorePowerful* relation is also a partial ordering.

The relation *Contain* is defined between two tasks. Informally, *Contain*(t_1, t_2) means that accomplishing task t_1 implies accomplishing task t_2. Similar to the relation *MorePowerful* on capabilities, the *Contains* relation is also an association class and can be similarly represented in UML; see Figure 6. It is also a partial ordering.

In the search for a testing service provider that is capable of performing a testing task, a broker agent must answer the question whether the task matches the capability of the service provider. For example, assume that a service is registered as capable of generating statement coverage test cases for Java Applets and a test task is requested for structural testing a Java Applet. The broker agent needs to infer that the agent is capable of fulfilling the task. Therefore, we define the *Matches* relation between a capability and a task. *Match*(c, t) means that a tester with capability c can fulfil the task t. The following properties of the relations form the foundation of the inferences that the broker agent requires in the assignment of testing tasks.

Proposition.

(1) $\forall c_1, c_2 \in Capability, \forall t \in Task,$

$MorePowerful(c_1, c_2) \wedge Match(c_2, t) \Rightarrow Match(c_1, t).$

(2) $\forall c \in Capability, \forall t_1, t_2 \in Task,$

$Contain(t_1, t_2) \wedge Match(c, t_1) \Rightarrow Match(c, t_2).$

5 Conclusion

This paper proposed a service oriented framework to testing WS applications. The utilization of software testing ontology STOWS is discussed to register testing services with semantics about the capability of the services, to request testing service with semantic specification of the testing task, and to query and retrieve testing services to match the capability of services to the requested test tasks. There are many more technical issues yet to be addressed. We are investigating the encoding of the ontology in OWL-S and case studies.

We recognize that the proposed approach to testing WS applications not only imposes technical challenges, but also social challenges for the approach to be adopted by IT industry and software testing tool vendors. We are seeking for collaborations from the industry and academia for taking the challenges and push the approach forward.

References

[1] Lau, C. and Ryman, A. Developing XML Web services with WebSphere studio application developer, *IBM Systems Journal*, Vol. 41, No.2, 2002, pp178-197.

[2] Gottschalk, K. *et al.* Introduction to web services architecture, *IBM Systems Journal*, Vol.41. No.2, pp170-177.

[3] Stal, M. Web Services: beyond component-based computing, *C. ACM*, Vol.45, No.10, 2002, pp71-76.

[4] Singh, M. and Huhns, M., *Service-Oriented Computing: Semantics, Process, Agents*, Wiley, 2005.

[5] W3C, *Web Services Description Language (WSDL) Version 2.0 Specifications*: *Part 0: Primer, Part 1: Core Language, Part 2: Adjuncts*.

[6] UDDI.org, *UDDI Version 3 Specification*.

[7] W3C, *SOAP Version 1.2, Part 0: Primer, Part 1: Messaging Framework, Part 2: Adjuncts*, June 2003.

[8] Zhu, H. and Shan, L., Agent-Oriented Modelling and Specification of Web Services, *Proc. of WORDS'05*, IEEE CS, 2005, pp152-159.

[9] Chan, W. K., Cheung, S. C., and Leung, K. R. P. H. Towards a Metamorphic Testing Methodology for Service-Oriented Software Applications, *Proc. of the 5th Annual International Conference on Quality Software (QSIC 2005)*, IEEE CS, 2005.

[10] Tsai, W. T., Chen, Y., Cao, Z. Bai, X., Hung, H., and Paul, R., Testing Web services using progressive group testing, *Proceedings of Advanced Workshop on Content Computing (AWCC 2004)*, LNCS 3309, Springer-Verlag, Berlin, Heideberg, 2004, pp314–322.

[11] Tsai, W. T., Chen, Y., Paul, R., Huang, H., Zhou, X., and Wei, X., Adaptive Testing, Oracle Generation, and Test Case Ranking for Web Services, *Proc. of COMPSAC 2005*, IEEE Computer Society, 2005, pp101–106.

[12] N. Looker, M. Munro, and J. Xu, WS-FIT: A Tool for Dependability Analysis of Web Services, *1st Workshop on Quality Assurance and Testing of Web-Based Applications, in Proc. of COMPSAC 2004*, Hong Kong, 2004.

[13] Emer, M.P., Vergilio, S. R., Jino, M., A Testing Approach for XML Schemas, *Proc. of COMPSAC 2005 - QATWBA 2005*, July, 2005.

[14] Li, J.B., Miller, J., Testing the Semantics of W3C XML Schema.*Proc. of COMPSAC-QATWBA 2005*, July, 2005.

[15] Offutt, J., Xu, W., Generating test cases for Web services using data perturbation, *Workshop on Testing, Analysis and Verification of Web Services, SIGSOFT Software Engineering Notes*, Vol. 29, No.5, 2004.

[16] Sami Beydeda and Volker Gruhn (ed.), *Testing Commercial-Off-The-Shelf Components and Systems*, Springer, 2005.

[17] Zhu, H. and He, X., An Observational Theory of Integration Testing for Component-Based Software Development, *Proc. of IEEE 25th International Conference on Computer Software and applications (COMPSAC'2001)*, 8-12 October 2001, Chicago, Illinois.

[18] Zhu, H. and He, X., A Methodology of Component Integration Testing, in [16] , 2005, pp239-269.

[19] Goodenough, J.B. & Gerhart, S.L., Toward a theory of test data selection, *IEEE TSE*, Vol.SE_3, June 1975.

[20] Zhu, H., Huo, Q.,, Greenwood, S., A Multi-Agent Software Environment for Testing Web-based Applications, *Proc. of IEEE COMPSAC'03*, Nov. 2003. pp210-215.

[21] Zhu, H. and Huo, Q., Developing A Software Testing Ontology in UML for A Software Growth Environment of Web-Based Applications, Chapter IX of *Software Evolution with UML and XML*, Hongji Yang (ed.), IDEA Group Inc. 2005, pp263-295.

[22] Uschold, M. & Gruninger M., Ontologies: Principles, Methods, and Applications. *Knowledge Engineering Review*, vol. *11*, No.*2*, 1996, pp93-155.