Dynamic Content Management System for Collaborative Testing Of Web Services

Sathya P¹, Udhaya Kumar V²

¹M.TECH (Computer Science &Eng), PRIST UNIVERSITY, Pondicherry
²Assistant Professor (Computer Science &Eng), PRIST UNIVERSITY, Pondicherry

Abstract-- Software testers are confronted with great challenges in testing WS. They must meet a wide range of test requirements. They must deal with the diversity of implementation techniques used by the other services. An automated testing technique must be developed to be capable of testing on-the-fly nonintrusively and nondisruptively. The ontology can be extended and updated through an ontology management service (OMS). So that it can support a wide open range of test activities, methods, techniques, and types of software artifacts. The past few years have seen a rapid growth in the inputs and errors in invocation sequences, fault development of WS techniques and tolerance. Various Test services collaborate with each other to complete test tasks. In my project, instead of using a static system, we will use a dynamic CMS (Content Management System). This CMS allows us to use the system as dynamic which is further compatible for multiple domains. Reporting and testing can be made easy to the user when working with multiple domains. Experimental evaluation of the framework has also demonstrated its scalability.

Keywords-- Software testing, testing web service, testing tools, ontology, web services, semantic web services.

I. INTRODUCTION

The paper presents a prototype implementation of the framework in semantic WS and demonstrates the feasibility of the framework by running examples of building a testing tool as a test service, developing a service for test executions of a WS, and composing existing test services for more complicated testing tasks. Experimental evaluation of the framework has also demonstrated its scalability.

II. RELATED WORKS

First review related for works addressing the privacy and security issues in the cloud. Then we briefly discuss works which adopt similar techniques as our approach but server for different purposes.

[1] Lijun Mei WS-BPEL applications are a kind of service-oriented application. They use XPath extensively to integrate loosely-coupled workflow steps. However, XPath may extract wrong data from the XML messages received, resulting in erroneous results in the integrated process.

Surprisingly, although XPath plays a key role in workflow integration, inadequate researches have been conducted to address the important issues in software testing. This paper tackles the problem. It also demonstrates a novel transformation strategy to construct artifacts. We use the mathematical definitions of XPath constructs as rewriting rules, and propose a data structure called XPath Rewriting Graph (XRG), which not only models how an XPath is conceptually rewritten but also tracks individual rewritings progressively. We treat the mathematical variables in the applied rewriting rules as if they were program variables, and use them to analyze how information may be rewritten in an XPath conceptually

[2] Gerardo Canfora and Masimilliano DippentaTesting of Service Oriented Architectures (SOA) plays a critical role in ensuring a successful deployment in any enterprise. SOA testing must span several levels, from individual services to inter-enterprise federations of systems, and must cover functional and non-functional aspects. SOA unique combination of features, such as run-time discovery of services, ultra-late binding, QoS aware composition, and SLA automated negotiation, challenge many existing testing techniques. As an example, run-time discovery and ultra-late binding entail that the actual configuration of a system is known only during the execution, and this makes many existing integration testing techniques inadequate. Similarly, QoS aware composition and SLA automated negotiation means that a service may deliver with different performances in different contexts, thus making most existing performance testing techniques to fail. An Approach for WSDL-Based Automated Robustness Testing of Web Services

[3] Samer Hanna and Malcolm Munro Web Services are considered a new paradigm for building software applications that has many advantages over the previous paradigms; however, Web Services are still not widely used because service requesters do not trust services that are built by others. Testing can be used to solve part of this problem because it can be used to assess some of the quality attributes of Web Services. This chapter proposes a framework that can be used to test the robustness quality attribute of a Web Service.
This framework is based on analyzing the Web Service Description Language (WSDL) document of Web Services to identify what faults could affect the robustness attribute and then test cases were designed to detect those faults. A proof of concept tool has been implemented and experiments carried out that show the usefulness of this approach.

III. EXISTING SYSTEM

With respect to user viewpoint, it is essential to With respect to user viewpoint, it is essential to provide an effective method. In existing system, testing is implemented with only one kind of domain (i.e., application). Testing is done with the help of static system. Predictive based algorithm was used.

3.1 Problems on Existing System

- Only one kind of domain is implemented
- The lack of software artifacts.
- The lack of control over test executions.
- The lack of a means of observation of internal behavior.

IV. PROPOSED WORK

In order to vanquish the issues in the existing data, instead of using a static system, we will use a dynamic CMS (Content Management System). This CMS allows us to use the system as dynamic which is further compatible for multiple domains. Reporting and testing can be made easy to the user when working with multi-domain. Knowledge based algorithm is used. A WS should be accompanied by a testing service. Testing services can be either provided by the same vendor of the functional services, or by a third party, data of the user undergoes for the error correction and then data’s are sent to another place. The services are to generate test cases, to measure test adequacy, to extract various types of diagrams from source code or design and specification documents, etc. 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. STOWS is base on an ontology of software testing originally developed for agent oriented software testing such knowledge is used:

- Instances of basic relations are stored in a knowledge-base as basic facts
- Used by the testing broker to search for test services through compound relations

An ontology management mechanism is provided to enable the population of the ontology.

Figure-1: Architecture of the system

Figure 1 shows the architecture of the complete system, where normal service invocations are depicted in solid line arrows and T-service invocations are denoted by dash line arrows. These T-services are registered to the UDDI registry. A test task can be accomplished through collaborations between these T-services.

Figure-2: Collaboration process of the system

Figure 2 shows the collaboration process of the complete system. The process starts with the generation of a test task by CIB’s WS and a search request for finding a proper tester is submitted to the service registry. The search request message should contain the information about the capability of the required tester. The search result is a list of testers, from which a test broker TB capable of handling the task is selected. The test request is then sent to TB.
The test request message contains the information about the test task including the service to be tested, the test adequacy to achieve, and the criterion for checking output correctness, etc. The test broker TB decomposes the test task into a sequence of subtasks and searches for appropriate testers for each subtask by submitting search requests to the registry. It then selects one tester for each subtask. In this example, we assume two testers TG and TE are selected. The former performs the subtask of test case generation and the latter invokes test executions, checks the correctness of test output and measures the test coverage. To generate test cases, TG sends a request to insurer A’s T-service to obtain its design model. After checking the trustworthy of tester TG, the insurer A’s T-service releases its design model to TG. After successfully obtaining the design model, TG produces a set of test cases and returns a test suite to the test broker TB. The test broker then passes the test cases to TE, requests for the test invocation of the insurer A’s services using the test cases and requests it to check the output correctness and to measure the test coverage. TE performs these tasks by collaboration with the insurer A’s T-services. The test results are then returned to the test broker TB. Finally, TB assembles a test report containing information about test output correctness and test adequacy. The test report is sent to CIB, which is used to determine whether the dynamic link will take place.

Our main contributions of the paper are as follows:

- Any number of domains of different kinds can be implemented.
- Because of the usage of dynamic Content Management System performance is improved.
- Knowledge fed for applications are centralized.

### 4.1 Implementation of semantics in WS

The Web Ontology Language OWL is a semantic markup language for publishing and sharing ontologies on the World Wide Web. It is designed for applications that need to process the content of information. It is a part of the growing stack of W3C recommendations related to the Semantic Web.

#### 4.1.1 Software Testing Ontology for WS(STOWS)

Ontology of WS Testing STOWS is proposed in based on the ontology developed. It was adapted for WS testing. Concepts in STOWS are classified into three categories: elementary concepts, basic testing concepts, and compound testing concepts.

The elementary concepts are those general concepts about computer software and hardware based on which testing concepts are defined.

They include the simple objects involved in software testing, such as the types of Hardware and Software artifacts and their Format, etc. The basic testing concepts include Tester, Artifact, Activity, Context, Method, and Environment. These basic concepts are combined together to express compound testing concepts, which include Task and Capability.

![Figure-3 The structure of capability and task.](image)

#### 4.1.2 Representation of STOWS in OWL

To implement the ontology STOWS, we represent the concepts, including elementary, basic, and compound concepts, as classes in OWL data model. To use the ontology for the registration, discovery, and invocation of T-services, the compound concepts capability and task are transformed into service profiles. In OWL-S, a service profile contains the Inputs, Outputs, Preconditions, and Results (IOPR) and a classification of the service.

#### 4.1.3 T-Service registration and discovery

In this mode, the logs are automatically send to The OWL-S/UDDI Matchmaker (Matchmaker for short) extends UDDI registry with a capability-based service matching engine. It provides three levels of matching between capability and search request.

1. Exact matching: the capabilities in the registry and in the request match exactly.
2. Plug-in matching: the service provided is more general than that in the request.
3. Relaxed matching: there is a similarity between services provided and that in the request.

The Matchmaker also provides filters for users to construct more accurate service discovery: which are namespace filter, domain filter, text filter, I/O type filter, and constraint filter. A service search request is also submitted to the Matchmaker.

#### 4.1.4 Ontology management service

It is impossible to build a complete ontology of software testing given the huge volume of software testing knowledge and the rapid development of new testing technique, methods and tools. OMS provides a WS interface to read and update the ontology data model, which is open to the public. The kernel OMS is the Manager module. It provides three services to users: Add Class, Delete Class, and Update Class to add new concept, delete concept, and revise concept of the ontology.
4.2 Evolution of Proposed Algorithm

This algorithm is used in case of Content management system. That is using ontology, we fed details for processing. With the knowledge fed the algorithm processes the request. If any failure occurs, it learns from the existing requests and processes correctly. Thus according to the knowledge available, it retrieves the information. In case of the report generated, the algorithm fetches data from the ontology and displays according to the user’s request. The algorithm is efficient in the concept of dynamic retrieval of data from web services other than using predictive based algorithm. The knowledge-based algorithm solves the sequence transmission problem from one agent to a group of agents, but does not fully comply with the dialogue communication involved during a process. It ensures the liveness and safety property and also solves the sequence transmission problem from one agent to a group of agents. The knowledge-based algorithm cannot handle asynchronous communication because it uses only one index. Introducing a separate index for each sender-receiver pair solves this problem, but only for the situation where the initiator does not change.

V. MODULES DESCRIPTION

5.1 Web Content Management System

A web content management system (WCMS) is a software system that provides website authoring, collaboration, and administration tools designed to allow users with little knowledge of web programming languages or markup languages to create and manage website content with relative ease. A robust WCMS provides the foundation for collaboration, offering users the ability to manage documents and output for multiple author editing and participation. Most systems use a content repository or a database to store page content, metadata, and other information assets that might be needed by the system. A presentation layer (template engine) displays the content to website visitors based on a set of templates, which are sometimes XSLT files. Most systems use server side caching to improve performance.

5.2 Selection of Domain / Sub-Domain

In this module the selection process of the domain and sub-domain takes place. In which, the user selects the appropriate option. From the welcome page of the system, first the user has to select the domain to which he has to proceed with. Then the selection of sub-domain is done using the Post Back criteria.

5.3 Roles / Responsibilities

Once the user selects the desired option, they have to choose a role. For eg, Doctor, Patient, Pharmacy. According to the chosen role from the list, a registration form appears in which the details must be given.

For eg, If the role is chosen as Doctor, he/she must fill the details like name, designation, specialization, experience etc. Also they have to create a login id and password to access their account in future.

After the submission of form, it appears as registration successful if it has been filled correctly. Likewise patients and pharmacy users can register with different forms. To login to their account, login provision is available commonly in home page. The admin will allocate the patients to desired doctors to visit according to their visiting hours. Thus in here the different applications collaborate with each other.

5.4 Dynamic report

The dynamic report is the main objective, in which the testing is done and reported. It displays the report according to the options regarding doctors’ designation or experience or working hospital. That is the knowledge fed is retrieved to the user from the ontology, so this is where the collaboration process of the applications is performed. This is done through the crystal report in visual studio. The graph is generated based on Sum of Filter time versus Record Count. The time complexity is determined in this phase where the filtering time of a record is derived dynamically where the efficiency is high.

VI. CONCLUSION AND FUTURE WORKS

We employ the ontology of software testing STOWS to describe the capabilities of T-services and test tasks for the registration, discovery, and invocation of T-services. The knowledge intensive composition of T-services is realized by the development and employment of test brokers, which are also T-services. We implemented the architecture in Semantic WS technology. Experimental evaluation also shows the scalability of the approach.UDDI is insufficient to support collaborative testing in their framework and proposed an extension to the function of UDDI to enable collaboration. They proposed to add check-in and check-out services to UDDI servers so that a service is added to UDDI registry only if it passes a check-in test.
A check-out testing is performed every time the service is searched for. Test brokers in their framework are much more complicated than ours and aim to achieve many more functions, which include generating test cases, submitting test cases to the WS, coordinating tester and functional services, registering testers and recording the performance of testers, monitoring services, and keeping a repository of tests performed for the evaluation of services and testers, etc.

REFERENCES


