2012 International Conference on Computer Technology and Science (ICCTS 2012)
IPCSIT vol. 47 (2012) © (2012) IACSIT Press, Singapore
DOI: 10.7763/IPCSIT.2012.V47.71

Remote testing service model of TTCN-3

Ying Zhao, Shaoyin Cheng, Xueqin Sun, Feng Wang, Fan Jiang

School of Computer Science and Technology, University of Science and Technology of China, Hefei, 230027,P.R.China

Abstract. Testing is an essential part of software development. Building a test platform is expensive and time-consuming, which always holds up the progress of testing. In order to facilitate testing, we suggest a Web Service method, i.e. testing is provided in the form of Web Service. But existing local testing model of TTCN-3 could not meet this requirement. So we propose a remote testing service model based on TTCN-3 in this paper. It extends current local testing model and offers a more flexible way to do testing. We have built a remote testing service prototype based on our testing platform and it has already been used in practice.

Keywords: TTCN-3, remote testing, testing service

1. Introduction

It's widely known that testing has become more and more expensive and complex nowadays. How to control the resource spending in testing and find a cost-efficient way of testing has become the focus of many software developers.

It is a hard job to build testing infrastructure from the very beginning. Small and medium-sized companies may lack the investment and the expertise. This becomes a burden and prevents them from focusing on their core business. As a matter of fact, most companies do not need to have their own testing infrastructure, what they really need is testing service. And the service will be much cheaper for it is provided in a pay-per-use way. However, doing test remotely means that a test system and its user control procedure should be respectively deployed in the location of service provider and service consumer's organization. Existing TTCN-3's (Testing and Test Control Notation Version 3) local testing model could not meet the requirement. This paper provides a remote testing service model of TTCN-3 to solve the problem.

The paper is organized as follows: Section 2 describes the local testing architecture of TTCN-3. Section 3 presents the implementation of remote testing service model, followed by related work in section 4. And finally we conclude this study and discuss future works in Section 5.

2. Testing architecture of TTCN-3

TTCN-3 [1] is the only standardized language for testing and there have been several mature TTCN-3 testing tools like TTWorkbench[2], OpenTTCN[3], and LoongTesting[4][13]. TTCN-3 is a test specification and implementation language used for black-box testing. It provides a notation which is independent of test methods, layers and protocols. It is widely used for conformance, robustness and regression testing. Its syntax is similar to other advanced programming languages, but it contains some language elements which are particularly useful for the black-box testing.

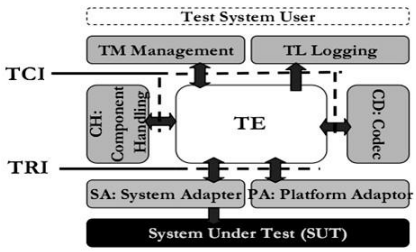

Fig. 1. Components of TTCN-3 test system

As shown in Fig. 1[5], a TTCN-3 test system includes the following functional parts [6]:

- TE (TTCN-3 Executable) is the most important part in charge of running executable test suites.
- TM (Test Management) is the entity providing testers with test control interfaces.
- TL (Test Logging) is responsible for keeping records of testing events which testers are interested in.
- CD (Codec) encodes TTCN-3 test system TTCN-3 value to messages which the SUT (System Under Test) could recognize and decodes messages to TTCN-3 value when receiving message from SUT.
- SA (SUT Adaptor) provides communication methods e.g. TCP, UDP or SSL between SUT and Test System.
- PA (Platform Adaptor) implements external functions and timers.

And there are two groups of interfaces: TCI (TTCN-3 Control Interface)[5]and TRI (TTCN-3 Runtime Interface)[7] : TCI defines the interactions among the TE, CH, TM, CD, TL entities. TRI defines the interactions among the TE, SA, and PA entities.

Fig. 2 illustrates the local testing architecture of TTCN-3. The dotted ellipse contains two components: one is TTCN-3 test system and the other is user control procedure. The two components are deployed in the same machine and communicate with each other using TCI. Most of the time, SUT is at a different machine and communicates with TTCN-3 test system through network.

In order to do testing in different places, the local testing architecture of TTCN-3 ought to be modified. SaaS [8] (Software as a Service) might be a good idea, i.e. turning TTCN-3 testing tools into service. Test system and user control procedure are made apart in the local testing architecture. And TCI is exposed to testers in the form of web service so testers are able to control test progress in their place and do not have to go to test system's renter's to do test.

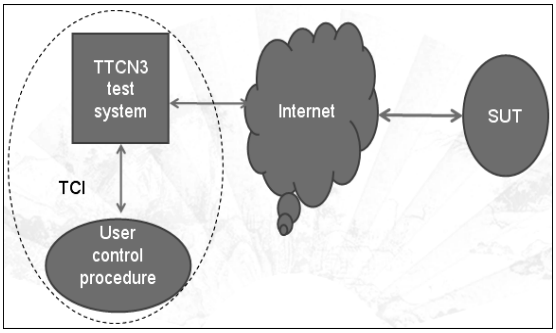

Fig. 2. Local testing architecture of TTCN-3

3. Remote testing model of TTCN-3

3.1. Foundation

Before we go any further, firstly we have to find out what SOAP and WSDL are.

SOAP (Simple Object Access Protocol) is a lightweight protocol for exchange of information in a decentralized, distributed environment. It is an XML based protocol that consists of three parts: an envelope that defines a framework for describing what is in a message and how to process it; a set of encoding rules for expressing instances of application-defined data types; and a convention for representing remote procedure calls and responses. SOAP can potentially be used in combination with a variety of other protocols [9].

WSDL (Web Service Description Language) is an XML-based language for describing Web services and how to access them. WSDL describes four critical pieces of data: interface information describing all publicly available functions; data type information for all message requests and message responses; binding information about the transport protocol to be used; address information for locating the specified service. In a nutshell, WSDL represents a contract between the service requestor and the service provider.

And secondly we need to know some basic knowledge of TTCN-3. The top-level unit of TTCN-3 is a module. A module can import definitions from other modules. Modules can have module parameters to allow test suite parameterization. A module consists of a definition part and a control part. The definition part of a module defines test components, data types, templates, functions, test cases, etc. Test case expresses

dynamic test behavior and it may have parameters. The control part of a module calls the test cases and controls their execution [6]. A test suite is allowed to contain several modules but only one of them could be configured as the main module, which is the entry module of a test suite.

3.2. Implementation

In our remote testing model, testers configure testing parameters on the web site which provides testing service. The web page notifies its background test system of the connection method to SUT according to these parameters. Once the connection is established, testers start a test case on the web page, and the web page asks its test system to execute the test case required. Then the test system sends stimulus to SUT and sets test verdict according to the response from SUT. And the verdict is passed to the web page and shown to tester finally.

In order to realize our idea, the test system has to support remote test control and test result feedback. We could depart test system from user control procedure. Test system and user control are not at the same place any more. Test system still communicates with user control procedure using TCI. However, in remote testing model the TCI messages are exchanged in SOAP package through Internet. In this way, test system changes into test server, and user control part becomes client. The remote testing model of TTCN-3 is shown in Fig. 3.

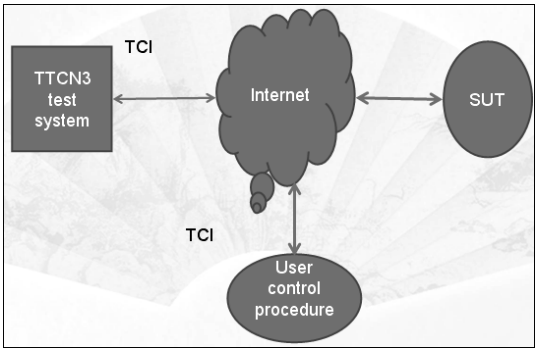

Fig. 3. Remote testing model of TTCN-3

The TCI of test system should be exposed to clients in the form of web service which is described in WSDL. The set of core interfaces that a remote test server needs to implement is listed in table 1.

Table 1. Interfaces exposed to client

Interface Name	Comment
StartTestCase	Start a test case identified by a qualified name.
StopTestCase	Stop a test case which is running.
GetModule	Get a module's information, e.g. module's name, types and constant defined in the module, etc.
GetImportedModules	Get imported module's information.
GetModuleParameters	Get a module's parameters.
GetTestExecutableState	Get a test suite executing state which could be ready, running or stopped.
GetTestCaseParameters	Get a test case's parameters.
StartControl	Start the control part of a module. The control part calls the test cases and controls their execution.
StopControl	Stop the control part of a module.

Besides, remote test server needs to provide extra interfaces in order to meet various demands of test. For instance, there should be interfaces for testers to upload their test cases because test cases are compiled on

server. And SA and CD are supposed to be reconfigurable for they are always different according to test suites. Therefore the testing server needs to provide interfaces of configuring SA and CD as well.

The interaction between client and remote test server is demonstrated in Fig. 4. The client is on the left side of the dashed line and the test server is on the right side. The client forms a SOAP message according to the WSDL document of the server and encapsulates a test request in it. The service interface which handles this request calls the corresponding test method of TTCN-3 test system and waits for the response from test system. Once the service interface gets the result, it will send it back to the client.

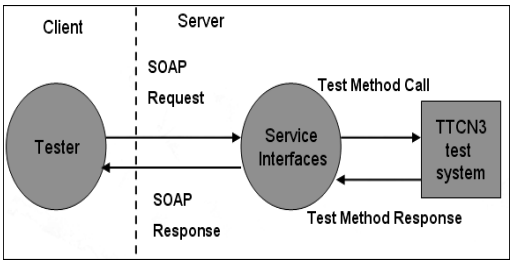

Fig. 4. Testing interaction

3.3. Preliminary example using remote testing model

As described above, the client and server interact with each other with SOAP message. Before doing test, the two parties have to make a contract which clearly formulate the definition of types. Later on they must sending message according to the contract.

For example, starting a test case using remote testing service model involves:

1) Defining types in WSDL (server-side job):

```
<!-- Input type definition for starting a testcase-->
<element name="StartTestCase">
  <complexType>
    <sequence>
      <element name = "testcaseId" type="ns:QualifiedName"/>
      <element name = "parameterList" type=" ns:ParameterList"/>
    </sequence>
  </complexType>
 </element>
<!--Response type definition for starting a testcase. -->
 <element name = "StartTestCaseResponse">
  <complexType>
    <sequence> <element name = "result" type=" ns:Result "/>  </sequence>
  </complexType>
</element>
```

There are two types definition for starting a test case: one is “StartTestCase” and the other is “StartTestCaseResponse”. The element “StartTestCase” includes two elements, “testcaseId” and “ParameterList”. “testcaseId” is a qualified name which identifies the test case to be executed and “parameterList” is a list of parameters that will be passed to the test case. “StartTestCaseResponse”, just as its name implies, is the type of response message of starting a test case. It contains one element named “result” which indicates whether the test case has started successfully.

In addition there are three other types, i.e. “QualifiedName”, “ParameterList” and “Result”, which are referred in the definition of “StartTestCase” and “StartTestCaseResponse”. These three types are defined in other parts of the WSDL document and their definitions are not listed here for the sake of brevity.

2) Defining operations in WSDL (server-side job):

```
<!--Operation definition for starting a testcase -->
<operation name="StartTestCase">
        <input message="StartTestCase"/>
         <output message = "StartTestCaseResponse"/>
</operation>
```

The description of "StartTestCase" interface (see first row of table I) is given by "StartTestCase" operation. The operation takes a message of type "StartTestCase" (defined in part 1) as input and a message of type "StartTestCaseResponse" (defined in part 1) as output.

3) Sending a SOAP message to server to start the testcase (client-side job):

```
<SOAP-ENV: Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/" …>
<SOAP-ENV:Body>
<m:StartTestCase xmlns:m = "SoapDef.xsd">
<testcaseId>
        <moduleName>M</moduleName>
        <objectName>T</objectName>
</testcaseId>
<parameterList>
    <param>
        <parameterName>P</parameterName>
        <parameterValue>V</parameterValue>
        <passMode>inout</passMode>
    </param>
</parameterList>
</m:StartTestCase>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The client sends the SOAP message above to invoke "StartTestCase" interface starting a test case which is in module M and named T. The test case has a parameter named P and the client passes value V to it when calling testcase T. And the pass mode "inout" is used to denote passing parameter by reference.

4) Returning a SOAP message to client to indicate whether the testcase has been started successfully(server-side job)

```
<SOAP-ENV:Envelopexmlns: SOAP-ENV = " http: // schemas. xmlsoap. Org /soap /
envelope / " … >
  <SOAP-ENV: Body>
      <m: StartTestCaseResponse xmlns:m = "SoapDef.xsd">
          <result>true</result>
       </m: StartTestCaseResponse>
  </SOAP-ENV: Body>
</SOAP-ENV: Envelope>
```

The server response to the client's starting test case request by sending a message of type "StartTestCaseResponse". The result "true" means the test case has been started successfully.

4. Related work

There has been some work concentrating on how to do testing more efficiently by web service. The versatile merits of cloud computing paradigm are revealed and a cloud computing model for software testing

is developed in [10]. However, the model is abstract and it's more theoretical than practical. An approach to enable the online change of test organization, test scheduling, test deployment, and service binding is presented in [11]. It puts more emphasis on configuration management in SOA (Service Oriented Architecture) testing. In [12], the author developed a software test service which used the Internet to receive test requests and return test results. But it's mainly constructed for computation of special functions in mathematics and physics which is not scalable.

5. Conclusion and future work

TTCN-3 is playing more and more important role in software testing. The remote testing service model of TTCN-3 extends current local testing architecture. It provides a more flexible way for testers to do testing using TTCN-3. Its working principle is easy to understand and the workload of transforming an existing local testing architecture of TTCN-3 to remote service model is small. We've turned our own local testing platform LoongTesting [4] [13] into remote model and it has already provided testing service to clients.

Furthermore we could introduce Cloud computing resource (provided by Cloud platform vendors such as Google, Amazon.com, etc.) into our model. When the test system receives high computing cost requests, e.g. compiling and linking TTCN-3 test cases to executable test suites, it could turn to Cloud for help. Another advantage of utilize Cloud Computing is that it could simulate extremely large real-world user traffic. So it could be applied easily in stress tests of Web sites.

When remote testing is used in practice we found there is still some work to do. For example, when test control part is deprived from test system, the latency is becoming larger though. This side effect is not negligible when doing performance test. How to minimize the impact still needs further study.

6. Acknowledgements

This research was supported in part by the National High Technology Research and Development Program of China (No. 2009AA01Z145).

7. References

[1] SIO/IEC 9646-3(1998). Information technology –Open systems interconnection – Conformance testing methodology and framework –Part3:The Tree and Tabular combined Notation(TTCN).

[2] http://www.testingtech.de/products/ttwb_intro.php.

[3] http://www.openttcn.com/.

[4] http://ttcn.ustc.edu.cn/.

[5] ETSI ES 201 873-6 V4.1.1. Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 6: TTCN-3 Control Interface (TCI). 2009-06.

[6] ETSI ES 201 873-1 V4.1.1. Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language. 2009-06.

[7] ETSI ES 201 873-5 V4.1.1. Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 5: TTCN-3 Runtime Interface (TRI). 2009-06.

[8] Turner, Mark, Budgen, David, Brereton and Pearl. Turning software into a service . Computer 2003;38-44.

[9] http://www.w3.org/TR/2000/NOTE-SOAP-20000508/.

[10] T.Vengattaraman, P.Dhavachelvan, R.Baskaran. A Model of Cloud Based Application Environment. International Journal of Computer Science and Information Security(IJCSIS) 2010; 257-260.

[11] Xiaoying Bai, Dezheng Xu, Guilan Dai. Dynamic Reconfigurable Testing of Service-Oriented Architecture. 31st Annual International Computer Software and Applications Conference 2007; 368-378.

[12] D. W. Lozier. A Proposed Software Test Service for Special Functions. Quality of Numerical Software: Assessment and Enhancement 1997; 167-178.

[13] Feng Wang , XueQin Sun ,Ying Zhao, YanWu Tang, Fan Jiang. GFT and its application in web test. International Conference on Networking and Information Technology 2010; 393-396.