AN AUTOMATIC TESTING FRAMEWORK APPLIED ON LIW AND IMPLEMENTATION

JIE YIN, DAN YU, SHILONG MA
State Key Lab. of Software Development Environment, Beihang University, Beijing 100191, Beijing, China

ABSTRACT

Software is an important component of Large Information Weapon (LIW). We propose a data-driven distributed automatic testing framework, which is based on the analysis of LIW testing requirements, and it supports the whole process of distributed software test. The main components of the framework consist of distributed test designing environment, execution and control environment and test results show environment. We present a set of GUI scripting language specification for automatic testing, the script language with object-oriented features. Unlike the previous GUI automatic testing scripting language, this specification records the information of the node where the test cases will be running. As an application of the framework, we develop an automatic testing system (DMATS). DMATS has been put into practices in XXX-business electronic system, and it is proved the feasibility of the framework.

Keywords: *Automatic Testing; Large Information Weapons; Test Language*

1. INTRODUCTION

Large Information Weapon (LIW) is highly information based weapon, and software plays an important role in LIW. LIW is characterized by complexity, multi-platform, distribution and collaboration. Software testing is a method to verify whether the LIW satisfies specification and performance requirement. LIW is a safety critical system, therefore, software testing is an effective means to improve the quality LIW.

The LIW consists of several subsystems, and each subsystem is developed by a unique organization. In order to guarantee the correctness of the software, it usually takes several stages of testing for LIW, including software configuration testing, subsystem level testing and system level testing. The implementation of interior function and interface is focused during the period of software configuration and subsystem level testing. The validation and correctness of multiple interface interaction between subsystem and operation flow is focused on during the period of system level testing.

At present, the testing of LIW is completed by testers manually. A large amount of functions depend on manual testing is not realistic and wasted. Only by manual test ways already cannot satisfy the existing demanding, therefore, automatic testing is an effective way to improve efficiency and quality in software testing.

The domestic and overseas scholars for software testing automation technologies usually focus on solving unique part of test automation process, such as test cases generation[1-3], test expected results validation[4], test management[5,6], test oracle[7,8], etc. Although they put forward some technology and independently developed support tools, its scope is limited. LIW testing consists of different kinds of testing, including functional test, interface test and user interface test, etc. Current test tools cannot satisfy the testing requirements of LIW. When several nodes are running their own test scripts, there is problem of synergistic action between nodes during test implementation [9,10]. Meanwhile, different kinds of tool can not be compatible, and the results produced by different tools cannot use directly. Thus, the development trend of software test method is to develop a complete set general software automatic testing framework.

The paper is organized as follows: In section 2, the foundation of automatic testing framework is described. In section 3, we state the testing requirement of LIW. In section 4 and 5, we put forward a software automatic testing framework for LIW testing, and an automatic testing system is designed which based on the framework. Finally, we apply the automatic testing system to our project and it proves the feasibility of the automatic testing framework.

2. BASIC AUTOMATIC TESTING FRAMEWORKS

An automatic testing framework is a set of assumptions, concepts, and practices that provide support for automatic software test. In the following, five basic frameworks will be described and demonstrated.

• The Test Modularity Framework

The test modularity framework requires the creation of small, independent scripts that represent modules, sections, and functions of the application-under-test. These small scripts are then used in a hierarchical fashion to construct larger tests, realizing a particular test case.

• The Test Library Architecture Framework

The test library architecture framework is very similar to the test script modularity framework and offers the same advantages, but it divides the application-under-test into procedures and functions instead of scripts. This framework requires the creation of library files that represent modules, sections, and functions of the application-under-test. These library files are then called directly from the test case script.

• The Keyword-Driven Testing Framework

Keyword-driven testing and table-driven testing are interchangeable terms that refer to an application-independent automation framework. This framework requires the development of data tables and keywords, independent of the test automation tool used to execute them and the test script code that "drives" the application-under-test and the data. Keyword-driven tests look very similar to manual test cases. In a keyword-driven test, the functionality of the application-under-test is documented in a table as well as in step-by-step instructions for each test.

• The Data-Driven Testing Framework

Data-driven testing is a framework where test input and output values are read from data files (data pools, ODBC sources, cvs files, Excel files, DAO objects, ADO objects, and such) and are loaded into variables in captured or manually coded scripts. In this framework, variables are used for both input values and output verification values. Navigation through the program, reading of the data files, and logging of test status and information are all coded in the test script.

• The Hybrid Test Automation Framework

The most commonly implemented framework is a combination of all of the above techniques, pulling from their strengths and trying to mitigate their weaknesses. This hybrid test automation framework is what most frameworks evolve into over time and multiple projects.

3. REQUIREMENTS ANALYSIS

The automatic testing system of LIW is an information- oriented and data-driven testing system. The purpose of developing the test automation system is to make a higher efficiency and lower cost, and to make a comprehensive test which will find the problems existing in LIW. Therefore, we can ensure the functions of the system meet the demand.

Test automation system task orientation electronic systems of different testing phase (including the subsystem level testing, system level testing and regression testing), in view of the different test type test automation services provided, its main features include:

1. Functional testing

Functional test should be taken on each subsystem of LIW to confirm weather the function is normal or not. Each subsystem of LIW is running in environment of each testing node which the operation platform is Windows or Linux.

2. Interface testing

The interface testing is to test the interface which is between two subsystems or in itself. While the LIW is working, Communication and Information Exchanged between two subsystems, and the correctness of internal and external interface is the assurance of normal working.

3. GUI testing

Verify the following content: the interface accuracy (characters used in the user interface, words and the correctness of the GUI, etc.), effectiveness (each function button the realization of the function of the effectiveness) and consistency (the user interface is consistent with the content which described in the software manuals). Ensure the actual operation interface is in accordance with user manual specification.

4. Validating the Test results

Verify the software automatically according to the operational results.

5. Generating the test report

With the test report automatic generating function that allows users to create and customize meet certain need formatted report, and test results shall be in a friendly way show at the end of the test

6. Managing the test resources

The tester can manage the test resource including test data, test scripts, test cases, the documentation, testing results and other test resources.

7. Distributing test

The test automation system has the following features multi-platform, distributed and network interaction. Automatic testing platform should have the support of the distributed test. Specific functions include:

- The test can be done in console which can centralized control of execution of the script on each node for remote operation;
- The test automation system can control the execution of test cases, pause, stop, and conditional execution;
- To test a distributed environment, each node send and receive messages, the message includes two types: data instructions and control instructions.

4. SYSTEM DESIGN

We propose a data-driven distributed automatic testing framework, which is based on the analysis of LIW testing requirements, as shown in fig. 1. We describe the structure of the automatic testing framework. Next, we elaborate the modules of the framework.

The automatic testing framework of LIW consists of automatic testing system and testing support environment. The automatic testing systems consist of the console side and the client-side.

Automatic testing system control side contain the following sections: distributed test design, execution and control of test, and test results show. The client-side is deployed in LIW which provide the testing environment, and the client-side realizes the following functions: generation of testing script, control test execution, test data acquisition and system communication.

The testing support environment consists of application runtime environment and basic runtime environment.

Application runtime environment is the testing object of automatic testing, for example, integration testing environment and system testing environment. The Automatic testing system and Application runtime environment are both deployed in the basic runtime environment. Testing supporting environ-ment provides the runtime and provides the necessary services.

5. THE MAIN COMPONENTS OF THE FRAMEWORK

5.1 Distributed Test Designing Environment

Test task design: Test task is a set of several test cases which are designed by the tester. The user can create, edit and modify the test tasks. When test task creation is complete, you can view the detail of the test task. Users need to select the test task before performing the test task.

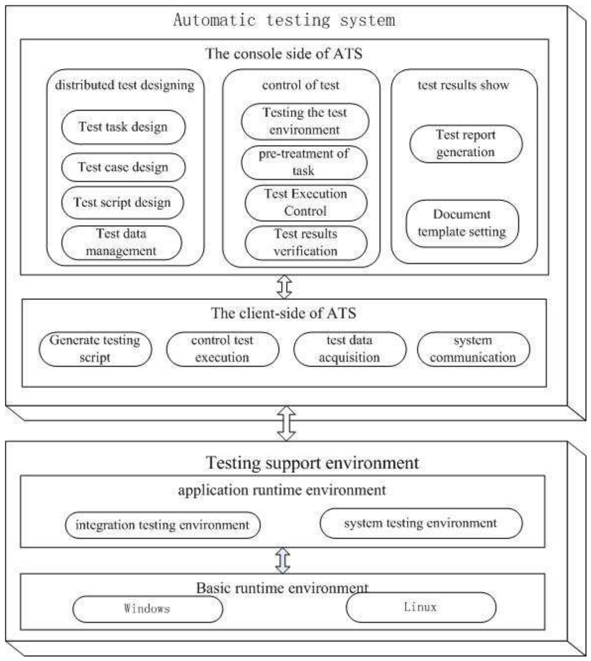


Figure 1: The Architecture Of Automatic Testing Framework.

Test case design: A test case is test unit which can be executed independently, and it contains a number of test scripts. Users can create, modify, and delete test cases on the test client of automatic test system.

Test script design: Test script is a set of test operation or instructions. Tester can manager the test script, including creation of test scripts, modify and delete. Users can edit the test script on the testing client, including modifying the script, inserting and modifying the test point, and the test script can also be generated by way of recording.

Test data management: The content of data file includes the parameters which used in the test script and the test criteria of results. The user can create data files, import and export the test data. In addition to recording, the test data which is called by a test script can also be generated by previous test data. The testing system can use the criteria to judge whether the test case is pass or not. It can be a collection of data, and it can be predefined expected output, such as the field value in the test of interface.

5.2 Execution And Control Environment

Testing the test environment: In order to ensure the normal test environment, Detection of the environment is necessary, such as the detection of network status and the detection of software configuration item. Network connectivity is a prerequisite for the normal test, such as the network is not connected or tested the software configuration item is not working, need to provide alarm information.

Pre-treatment of task: The distribution of tasks should be done before the execution of the test task. The test task will be distributed to different servers and hosts according to the scripts of test task. Means of implementation is defined according to the style of test task,

Test Execution Control: Users can perform the test tasks corresponding operations, including open test tasks, test tasks, and to stop the test suspension of the task. During the test, operator can display the running status of the test items. The input of test case consist test scripts, test data and check points, and the corresponding output is the testing process checkpoint program output, status, and specify the output. Means of implementation is to parse the script.

Test results verification: In the process of test case execution, test result of the need to be compared with the expected result. Test results verification is very important for automatic testing process. Test results verification includes functional test results verification, interface test results verification, UI verification and so on.

5.3 Test Results Show Environment

Test report generation: Automatic testing system should provide automatic generation of test reports, and it allow users to create and customize to meet their specific demands which the user can define specific report format. At the end of testing, the automatic testing system generates a test report according to the test result.

Document template setting: Testers can import the template of the test document, and choose a specific test template before test begins. At the end of testing, the result will put into the template automatically and generate test report.

6. SYSTEM DEPLOYMENT AND REALIZATION

We design an automatic testing system named DMATS on the basis of automatic testing framework, and we apply the system to the automatic test of XXX-business electronic system which is LIW. System hardware consists of the following components: test client, database server and test server, as showed in Fig. 2. Test can create new test task, control the test and view test results on the test client. All test data that include test script, test case and test task are stored in the database serve. The test server distributes the test task when it starts, and receives the test result during the period of testing.

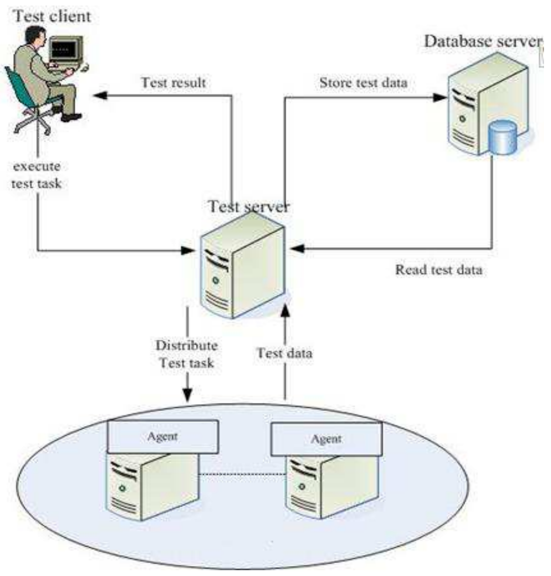
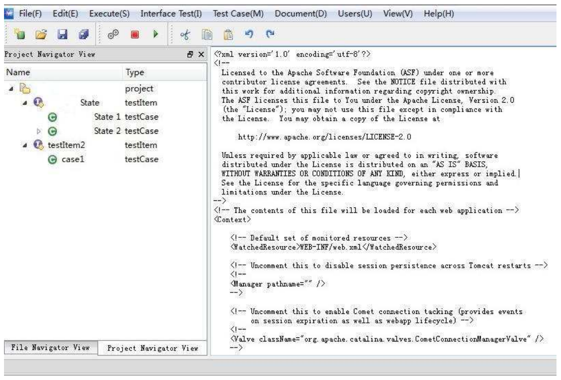


Figure 2: The component of DMATS

DMATS is implemented based on the .NET platform. The development language of it is C++, and the database is Oracle 10g. Fig.3 is one of the interfaces of DMATS. Now, DMATS has been put into practices in XXX-business electronic system.


File(F) Edit(E) Execute(S) Interface Test(I) Test Case(M) Document(D) Users(U) View(V) Help(H)
Project Navigator View
File Navigator View
Project Navigator View

Name	Type
	project
State	testItem
State 1	testCase
State 2	testCase
testItem2	testItem
case1	testCase

```
<?xml version='1.0' encoding='utf-8'?>
<!--
  Licensed to the Apache Software Foundation (ASF) under one or more
  contributor license agreements.  See the NOTICE file distributed with
  this work for additional information regarding copyright ownership.
  The ASF licenses this file to You under the Apache License, Version 2.0
  (the "License"); you may not use this file except in compliance with
  the License.  You may obtain a copy of the License at

      http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License.
-->
<!-- The contents of this file will be loaded for each web application -->
<Context>

    <!-- Default set of monitored resources -->
    <WatchedResource>WEB-INF/web.xml</WatchedResource>

    <!-- Uncomment this to disable session persistence across Tomcat restarts -->
    <!--
    <Manager pathname="" />
    -->

    <!-- Uncomment this to enable Comet connection tacking (provides events
         on session expiration as well as webapp lifecycle) -->
    <!--
    <Valve className="org.apache.catalina.valves.CometConnectionManagerValve" />
    -->
```

Figure 3: The Main Interfaces Of DMATS

In the follow part, we'll go over an example that demonstrates how to execute a test case in DMATS. For the user login interface test, the tester should finish a series of operation and verify the correctness and the time sequence of the login message. The steps of the user login test case are showed below:

Step1: input “user1” in the texbox “UserName”;

Step2: input “password1” in the texbox “PassWord”;

Step3: click the drop down box ”Authority”;

Step4: choose the “operator” on the drop down box;

Step5: click button “Login”.

Step6: capture the message between DCSA and SCM, and verify all the messages in the login process.

The messages exchange between different processes when the user login test case executes, as showed in Fig. 4. Both DCSA and SCM are different processes running in different machines. When the user login, DCSA sends the message (1): DCSA_SCM_LOGIN_REPORT to SCM; SCM verifies the message(1) and sends message(2): SCM_DCSA_LOGIN_ACK to DCSA; We should check all the fields of the messages between SCM and DCSA.

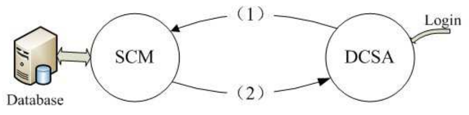


Figure 4: The Message Exchange In User Login Test Case

We define a test case script to describe the user’s operations and make it automatic to execute in the DMATS.

Next, we present a set of GUI scripting language specification for automatic testing, the script language with object-oriented features. Unlike the previous GUI automatic testing scripting language, this specification recorded the information of the node where the test will be running. Here are the EBNF definitions of test script language:

```
<scriptSentence>::=<testaction>|<cooperateClass>
<testaction>::=[<stationInfor>”.”]<windowInfor>”.”
<actionClass>::=<pushButtonClass>|<checkBoxClass
>
|<textClass>|<comboxClass>
|<tableClass>|<listClass>
<cooperateClass>::=<sendmessage>|<recvmessage >
<sendmessage>::=”sendmessage(”<stationId>”)”
<recvmessage >::=”recvmessage(”<stationId>”)”
<stationInfor>::=”station(”<stationId>”)”
<widnowInfor>::=”window(”<windowId>”)”
<pushButtonClass>::=”pushbutton(”<widgetId>”).”
<pushButtonActionInfor>
<checkBoxClass>::=”checkBox(”<widgetId >”).”
<checkBoxActionInfor>
<textClass>::=”text(”<widgetId>”).”<textActionInfor
>
<comboxClass>::=”combox(”<widgetId>”).”
<comboxActionInfor>
<tableClass>::=”table(”<widgetId>”).”<tableRowClas
s>
<listClass>::=”list(”<widgetId>”).”<listItemClass>
<tableRowClass>::=”tableRow(”<rowId>”).”
<tableRowActionInfor>
<listItemClass>::=”listItem(”<widgetId>”).”
<listItemActionInfor>
<pushButtonActionInfor>::=”click()”
<checkBoxActionInfor>::=”select()”|”unselect()”
<textActionInfor>::=”setvalue(”<value>”)”|
”assertvalue(”<value>”)”|
”waitvalue(”<value>”)”
<comboxActionInfor>::=”select(”<value>”)”
<tableRowActionInfor>::=”setvalue(”
<value>{”,”<value>}”)”|
”assertvalue(”<value>
{”,”<value>}”)”|”addvalue(”
<value>{”,”<value>}”)”
<listItemActionInfor>::=”setvalue(” <value>
{”,”<value>}”)”|”assertvalue(”
<value>{”,”<value>}”)”|
”addvalue(”<value>{”,”<value>}”)”
<windowId>::=<string>
<stationId>::=<string>
<widgetId>::=<string>
<value>::=<Alphabet >|<NaturalDigit>|<value>
<Alphabet>|<value><Digit>
<string>::=<Alphabet>|<string><Alphabet>
|<string><Digit>
<Digit>::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |9
<NaturalDigit>::=1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |9
< Alphabet>::= A | B | …| Z | a | b | …| z
```

The following are usage description.

<scriptSentence> is on behalf of a script, contains node information, the window information and test action information and test synergistic implementation information. The information of node is optional, if the node information is not recorded, it will be the same with the script statements’ above.

<cooperateClass> is on behalf of two types information of test synergistic statements: sendmessage() and recvmessage().

<actionClass> is on behalf of all types of test action. Action types are divided by type of widget, including the button test action type, check box test action type and table testing action types and so on.

Each type of test action contains the widget type identification, widget Id, and the specific action content. For example, the information contained by the button’s test action is

”pushbutton(”<widgetId>”).”<pushButtonActionInfor>.

Some information can be added into the parentheses, connecting the contents of the widget operation with “.”.

The script below is an example of main script of some test case written under the specifications above.

The content of the script:

```
station(a1).window(Dialog1).text(sendtexta).
setvalue(hello)
station(a1).window(Dialog1).pushbutton(send).click()
station(b1).window(Dialog2).text(recvtextb).waitvalue
(hello)
station(b1).window(Dialog2).text(sendtextb). setvalue
(hello)
station(b1).window(Dialog2).pushbutton(send).click()
station(a1).window(Dialog1).text(recvtexta). waitvalue
(hello)
```

This script tests the functions of sending messages of a system. Script contents are as follows:

1) Set the value of text box sendtexta to be hello in the window named Dialog1 at station named a1;
2) Click the pushbutton send in the window named Dialog1 at station named a1;
3) Wait the value of text box recvtextb to be hello in the window named Dialog2 at station named b1;
4) Set the value of text box sendtextb to be hello in the window named Dialog2 at station named b1;
5) Click the pushbutton send in the window named Dialog2 at station named b1;
6) Wait the value of text box recvtexta to be hello in the window named Dialog1 at station named a1;

7. CONCLUSION

Traditional software testing mainly accomplished by manual testing, and the domestic and overseas scholars for software testing automation technology focused on solving GUI test automation process. We have purposed an automatic testing framework which supports a whole process of software testing, and designed an automatic testing system named DMATS on the basis of the automatic testing framework. DMATS has been put into practices in XXX-business electronic system. We have presented a set of GUI scripting language specification with object-oriented features. It improves the efficiency of test, and it can help the test to find the fault of software more easily. In the future work, we will improve the performance of DMATS.

ACKNOWLEDGEMENTS

This work was supported by the National Natural Science Foundation of China (Grant No. 61003016).

REFERENCES:

[1] J. Offutt, Shaoying Liu, A. Abdurazik, “Generating Test Data From State-based Specifications”, *The Journal of Software Testing, Verification and Reliability*, Vol. 13,No. 1, 2005, pp. 25-53.

[2] A. M. Memon, M. E. Pollack, M. L. Soffa, “Hierarchical GUI Test Case Generation Using Automated Planning”, *IEEE Transactions on Software Engineering*, Vol. 27, No. 2, 2006, pp.144-155.

[3] Jeff Offutt, Shaoying Liu, “Generating Test Data from SOFL Specifications”, *The Journal of Systems and Software*, Vol. 49, No. 1, 2004, pp.49-62.

[4] Wu Hengshan and Wang Jinhong, “Automatic testing model based on the validity of GUI states”, *J. Huazhong Univ. of Sci. & Tech.*, Vol. 32, No. 12, 2004, pp.34-36.

[5] T. Parveen, S. Tilley, G. Gonzalez, "A Case Study in Test Management," *Proceedings of The 45th Annual Southeast Regional Conference*, Mar. 23-24, 2007, pp. 82-87.

[6] E. Koh, A. Kerne, S. Berry, Test Collection Management and Labeling System, *Proceedings of The 9th ACM Symposium on Document Engineering*, Sep. 15-18, 2009, pp. 39-42.

[7] A.M. Memon, M.E. Pollack, M.L. Soffa, “Automated test oracles for GUIs”, *SIGSOFT Software Engineering Notes*, Vol. 25, 2000, pp. 30–39.

[8] S.R. Shahamiri et al., “An automated framework for software test oracle”, *Information and Software Technology*, Vol. 53, 2011, pp. 774-788.

[9] R. M. Hierons, “Using status messages in the distributed test architecture”, *Information and Software Technology*, Vol. 51, Issue 7, 2009, pp. 1123-1130.

[10] R. M. Hierons, H. Ural, “Overcoming controllability problems with fewest channels between testers”, *Computer Networks*, Vol 53, Issue 5, 2009, pp. 680-690.