http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 12 (4): 829-834, 2013
ISSN 1812-5638 / DOI: 10.3923/itj.2013.829.834

Towards an Interface-based Automation Testing Framework for Silverlight Applications

Jingfan Tang, Qin Zhu and Ming Jiang
Institute of Software Intelligent Technology, Xiasha Campus,
Hangzhou Dianzi University, Hangzhou, Zhejiang, China

Abstract: Nowadays software applications are increasingly becoming large in scale and complexity, thus, Graphical User Interface (GUI) testing plays a formal important role in ensuring the correctness and reliability of software applications. A variety of approaches in the area of GUI testing have emerged in recent years. One notable trend is Model-Based Testing (MBT) which creates an abstract test model that simulates the anticipated behavior of the System Under Testing (SUT) by using some software generated tools to generate model tests. This study highlights the designing as well as the presentation of a new automation testing framework for silverlight applications with particular focuses upon the integration of the Spec Explorer based MBT with a free web framework called WebAii. Both of the tools are available as open source. Spec Explorer can be used to generate the test cases automatically, while WebAii is used to simulate human action and operation processes to complete the test execution in an automated way.

Key words: Automation test, Spec Explorer, model-based testing, WebAii, GUI testing

INTRODUCTION

Software testing is a process of verifying software before delivering it to market. It focuses on determining if the developed software meets the specified requirements as well as detects any differences between the actual result and expected result. Test automation is now becoming the fundamental and crucial test method in the life-cycle of software development, particularly after using tools with GUI that help programmers quickly create applications (Dustin *et al*., 1999).

GUI testing is now also increasingly becoming an essential aspect or tool to check the quality of software because it is performed based on the position of end user for the applications. Various functions of the application can be reflected through the GUI and therefore GUI tests can cover the entire application (Li and Wu, 2004). Presently, nearly all GUI applications constructed and assembled by means of tool kits and interface builders. This type of construction facilitates swift execution of repetitive tasks, as well as other functions like prototyping, usability testing (Myers *et al*., 2000). As a result of the iterative development process, the requirements, design and implementation of GUI software ultimately and inevitably changes thereby increasing the time and resources required for testing. Currently, many capture/replay tools are used to test GUIs in the market, such as WinRunner, IBM Rational Robot, Astra quick test and so on. But there is still much problem for applying these automated testing tools (Singhera *et al*., 2008; Grechanik *et al*., 2009), especially for Silverlight applications which allow a user to record and play back UI interactions as test cases. Microsoft Silverlight is a cross-browser and cross-platform technology, Silverlight application has rich graphics and user interaction (Villa *et al*., 2007). Any minor changes in the GUI of Silverlight application will influence the corresponding test cases. The changes of user interface elements (UI-elements) may occur frequently to make the software applications more comfortable which may need to re-implement all test cases defined by means of capture/replay so that it makes maintenance of test cases very time-consuming and expensive.

Having considered all the arguments stated, this study presents a novel framework for executing automation testing on Silverlight applications. This framework focuses on integration of Model-Based Testing (MBT) (Swain *et al*., 2010; Mlynarski *et al*., 2009) and Spec Explorer (Campbell *et al*., 2005) to automate the software testing on GUI of Silverlight applications. Generally in MBT, the System Under Test (SUT) is represented by a model describing its expected behavior at a higher abstraction level, and a set of chosen algorithms are used to generate test cases from this model

Corresponding Author: Jingfan Tang, Institute of Software Intelligent Technology, Xiasha Campus, Hangzhou Dianzi University, Hangzhou, Zhejiang, China Tel: +86-131-8422-0530 Fax: +86-571-8691-9178

(Kanstren, 2010). A number of research results have shown model-based testing as a promising solution to overcome the maintenance weakness of capture and replay tools (Utting and Legeard, 2007). The Spec Explorer can be used to generate the test cases automatically from requirements of the system's models rather than writing test cases manually. A free Web test framework called WebAii is used to simulate human action and operation processes to complete the test execution in an automated way.

The main advantages of this framework are as follows: (1) Automatically generate and validate test cases by executing the system under test and keep a high test coverage, (2) Having effective functional and regression testing as well as keeping them in a highly maintainable state with low effort for test process, the change of a user interface element doesn't need to redefine the whole test case and (3) A clear separation of abstract test case description through the usage of models from the execution of the user operation will make the tests more maintainable. The main goal of this study is to reduce the work load required to create and maintain test models and increase the efficiency of automated testing of Silverlight applications.

INTERFACE-BASED AUTOMATION TEST FRAMEWORK

Based on the special requirements of Silverlight applications, the automation test framework is composed of four components: GUI modelling, test cases generation and execution, Adapter and Spy layer. The framework architecture is shown in Fig. 1.

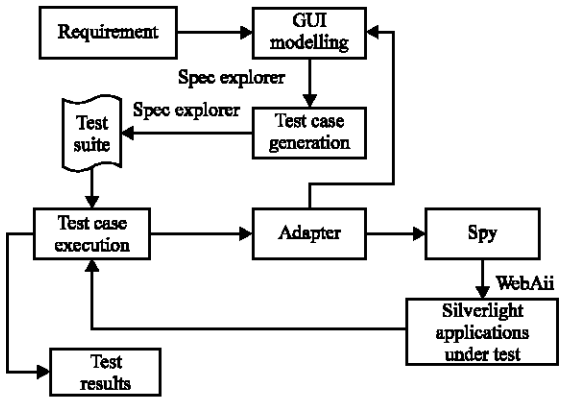

Fig. 1: Testing framework for silverlight applications

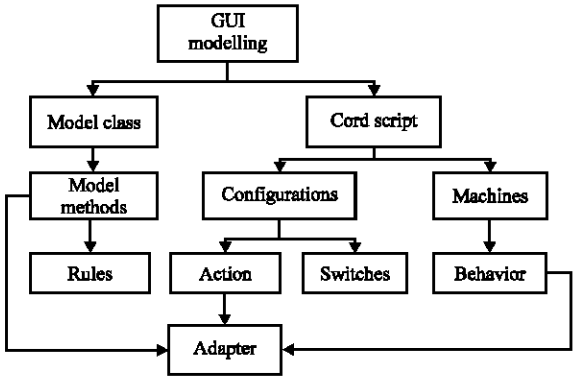

Fig. 2: The structure of the GUI modelling

GUI modelling with Spec# and Spec explorer: Modelling represents an abstraction of a system from the test system's perspective. An effective model describes the possible user-interactions with the graphical user interface, which helps users to analyze, describe, explore and build the system. The modelling with Spec Explorer is stimulated by Abstract State Machines (ASMs) (Grieskamp *et al*., 2002) which provides a way to model system behavior of abstraction at any level. FSM (Finite State Machine) (Swain *et al*., 2010; Ye *et al*., 2007) is another important model testing used in object-oriented software. Figure 2 illustrates the structure of the GUI modelling which used by Spec Explorer.

A formal model program can be written in a high level specification language such as AsmL (Gurevich *et al*., 2005), Spec# (Barnett *et al*., 2004), or in a programming language such as C# or Visual Basic. A model program class will be defined in the model with the specific methods to simulate user basic activities in the System Under Test (SUT) and these defined methods often correspond to SUT implementation. Model methods are annotated as rules that represent the possible state changes of GUI (Yuejin and Jianyu, 2009). To enable conformance testing of the outputs displayed to users, rules are also used to observe the state of the actions. These actions can have pre-conditions, written as "condition" clauses () that define the states, and can be seen as the symbol for updating of ASM rules.

In order to enable the model program to generate the effective test cases, a Cord script will be defined with the configurations and action machines to express the behaviors. It contains at least one configuration that declares the actions of the Model and defines the interface between Model and Adapter. For example, the action of all Silverlight Automation. Adapter is written for calling methods in Adapter layer. The Cord script can also have some exploration switches.

Test cases generation and execution: One of the biggest advantages of this framework is to use Spec Explorer

automatically to generate test suites from a Spec# or AsmL specification, which will reduce plenty of time and costs. As soon as the Model program has finished the constructed process, Spec Explorer will begin to explore the machine and generates a project file with the test suites. A test suite is a set of test cases derived from one action machine created by translating the exploration results into test cases according to the selected algorithm. And this action machine will drive the generation of the corresponding test suite in turn which describes the detail executed steps of the SUT. For example, the step of SilverlightAutomation.OneAccount.Adapter.Initialize() in a test case will call the method in Adapter.

After test cases have been constructed, Spec explorer will begin to execute them. The Spec Explorer supports both a unit test format and a customizable format, hence, every test case is represented by an executable unit test. When the button "Run all tests in solution" in Visual Studio is clicked, all test cases will be executed automatically.

Adapter layer: A separated Adapter layer needs to be implemented to support the effective communication between Spec Explorer and WebAii framework which likes an adapter in Design Pattern. The Adapter layer is a modification of the Spec Explorer adaptation which is made for Silverlight applications. It creates methods that make product abstract behaviors to the actual implementation steps and responses with spy layer execution results. Figure 3 illustrates the structure of the Adapter layer and the connections to the WebAii.

An interface method is associated with one UI operation and the rule can be treated as the behavior constraint for corresponding interface method. The methods of Adapter layer should keep consistent with those in GUI modelling layer. "Using all adapter" in cord file will link up with the Model and Adapter. Test cases method generated by model will contain Adapter methods, so that it will not call the method of adapter if the method of adapter and model are not consistent.

Spy layer: The Spy is the lowest layer of the four layers that interacts directly with Silverlight applications. It expresses the actual operations such as clicking a button, input data in a textbox and so on. As mentioned in the overview of the framework, the WebAii framework is implemented to provide relevant APIs to support the operation of LINQ on the browser to control the Silverlight, which controlls the mouse movement and keyboard to replace user's repetitive task and regression testing, bring more reliable results and fewer mistakes. WebAii is a free Web testing framework developed by

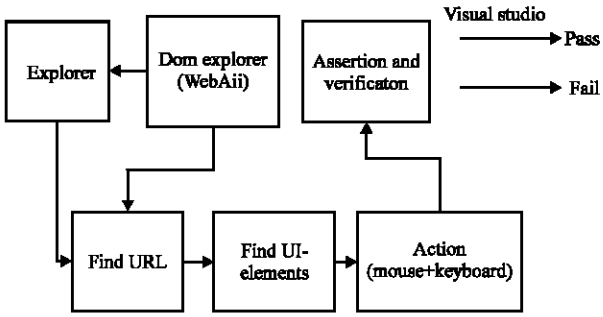

Fig. 3: An overview process of Spy layer

Telerik (Paz, 2010) which can also be integrated with a unit test framework into Visual Studio with two types of test automated oracle (Ye *et al*., 2007) verification and assertion.

WebAii framework is based on Telerik testing framework that provides all the base functionalities used by WebAii. The main functionality of them is that it owns the abstraction functionality of Browser/Dom, which can support all of the mainstream of the browser, such as IE, firefox and so on. Figure 3 shows an overview process of Spy layer.

In Spy layer, browser first navigates to the corresponding URL, then enter the page of applications. Spy inside has all kinds of controls interation between Silverlight and Telerik, such as click, double click, input text, drag, etc. The precondition of its interaction is that the locations of the controls in SilverlightApp can be found in some way. WebAii provides powerful element finding strategies to find these controls, such as directly find through the name, assigning the control AutomationId and so on. In the following, WebAii will operate on the DOM (Document Object Model) tree file structure to simulate people's mouse operation. When the controls have been found, WebAii interacts with controls through the User Attribute e.g., it uses the method of Control.User.Click() to present "click". After finishing these operations, the verification will occur during the execution of test cases. If one of the commands fails, the execution continues till the completion of the test cases. And, the test results will show the failed test cases. In the other hand, Assertion mode occurs during the execution of commands, many function of Assertions are defined in Spy methods. if one of commands fails, the execution ends at once and the test results are displayed.

In order to reduce the code changes, it should consider the conditions as many as possible in the interface design of model layer and the designing of the method parameters to address the good extensibility. The spy layer code need to follow the specific criteria through

the configuration files for configuration of control, so that it only needs to change the configuration file when there are some changes on requirements.

APPLICATION AND EXPERIMENTAL RESULTS

In order to verify this automation testing framework, it was applied to an application called OneAccount, which is a financial information release system based on Silverlight. This application is written for financial companies and can support the financial managers in making investment decisions after analyzing the condition of various bond's risks from difference aspects. In here we will choose a SetDataPrecision button in OneAccount for an example, the functionality is to set the precision of data in DataGrid.

Creating the usage model: Figure 4 shows the Spec# model of setting data precision with a class that represents the abstract state and operations. The class will contain many variables and pre-conditions. As the model shown in Fig. 4, Spec Explorer will extracts representative business behavior according to user-defined parameters for scenario control and the specific state exploration algorithm. The corresponding cord script is shown in Fig. 5.

Test cases construction: Figure 6 shows a scenario extracted from the SetDataPrecision model program and visualized by Spec Explorer through an application of model programing and cord script file. The result has 6 transitions and 7 states where S0 is the initial state and S18 is the final state. Spec Explorer will explore the state space which starts from the initial state S0. After the procedure of configuration initialization and some operations such as OpenBrowser and OpenReport, it will navigate to the page of data precision setting. Since the scope of data precision is from one to six, there are six paths in the exploration to the operation of CloseBrowser. ClearUp will finish all the operations. In order to generate the test suits of SetDataPrecision, it needs to select this machine and click "Generate Test Code" on the Exploration Manager toolbar in Spec Explorer and a new project will be automatically generated with the test suites.

Construct method of adapter layer: As mentioned in previous sections, the Adapter layer handles the communication between the Spec Explorer and the WebAii framework. The Adapter method of

```
namespace SilverlightAutomation.OneAccount
{
static class OneAccountModel
{
private static bool isInit = false;
private static bool isBrowserOpened = false;
 private static string openedReport = string.Empty;
 [Rules]
 public static void DataPrecision(int pre)
 {
 Condition.IsTrue(isInit);
 Condition.IsTrue(isBrowserOpened);
 Condition.IsTrue(openedReport ! = string.Empty);
 Condition.IsTrue(pre ≥ 0 && pre ≤ 6);
 }
}
}
```

Fig. 4: Model program written in Spec#

```
machine OneAccountModelProgram (): Main where
ForExploration = false
{
   Construct model program from
Main
  where Namespace = "SilverlightAutomation.OneAccount"
}
//Determent state machine
machine OneAccData precision (): Main where ForExploration =
true
{
 (
Initialize (); OpenBrowser (); OpenReport
(SearchConditions.Private,
"private Report 1_yuntian");
  Data precision(_);Close Browser (); Cleanup();
)
 || One AccountModelProgram
}
machine One AccDataPrecision_TestCase (): Main where
For Exploration = True, Test Enabled = true
{
Construct test cases where strategy =
"Short Tests", Allow Undeterm inedCoverage = true
 for OneAccDataPrecision ()
} // Construct test cases
```

Fig. 5: Piece code from cord script file

SetDataPrecision is shown in Fig. 7. This method mainly provides the interface of actual implementation steps in Spy layer.

Define Spy layer method: Next, Spy layer is created. WebAii will firstly find the location of precision control and then do the related operations such as click "precision icon", open its dowpdown list, set the value of the textbox "Value" and click on the "Save" button (Table 1).

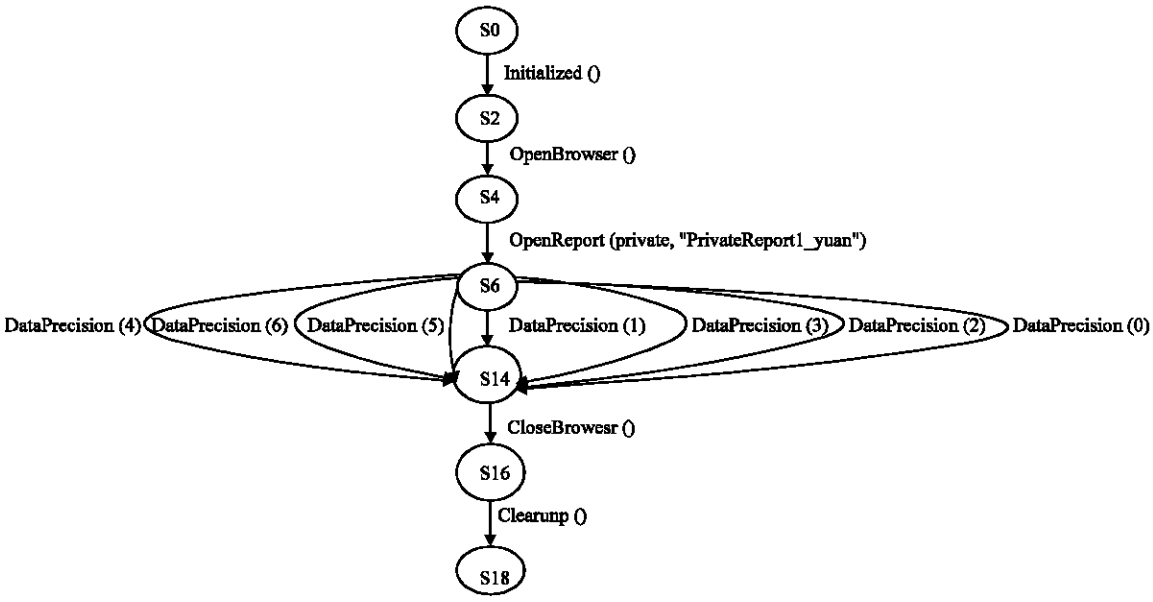

Fig. 6: An FSM model visualized by Spec explorer

```
public class Adapter
{
public static void DataPrecision (int pre)
 {
  DataZone.SwitchToMatrix Tab();
  PivotConfig.SwitchMode(DataPresentationMode.Grid);
  RibbonBarZone.Precision.OpenPrecison();
  RibbonBarZone.Precision.Setprecision(pre);
  pivotTotalValue = DataZone.GetGridTotalValue();
 }
}
```

Fig. 7: A defined adapter method

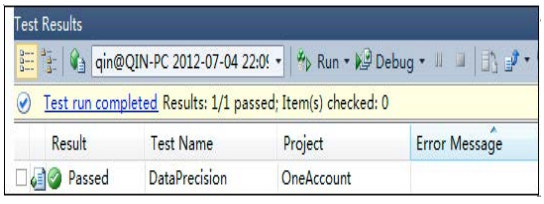

Fig. 8: Test execution result

Table 1: Find and Operate a control result

Operation name	Find control strategy	Action In	Element type
Click precision icon	Name and type	Click	Button
Set precision value	Name, AutomationId	Save value	Textbox
Click save button	Type	Click	Button

Test results: As soon as the spy layer is constructed, the related reference of WebAii will be added to the project. The test cases will be automatically executed without user intervention after the button "Run Tests in Current Context" in Visual Studio is clicked. Figure 8 shows the result after test execution. This presented result shows the functionality of SetDataPrecision works fine.

CONCLUSION

In this study, it basically highlights a proposed automation testing framework for Silverlight applications, which allows to combine Spec Explorer based model-based testing and WebAii testing framework during functional and regression testing on graphical user interfaces. The combination makes it possible to seamlessly execute test cases on sliverlight applications. Another important aspect that the presented approach addresses is the reduction of maintainability costs of test cases in such highly dynamic software development environment. Through the application on a financial information release system, it verified the efficiency of the framework on the automation testing of silverlight applications.

ACKNOWLEDGMENT

This study was granted by the Natural Science Foundation of Zhejiang Province, China (No. Y1111192).

REFERENCES

Barnett, M., R. Leino and W. Schulte, 2004. The Spec programming system: An overview. Proceedings of the International Workshop on Construction and Analysis of Safe, Secure and Interoperable Smart Devices, Volume 3362, March 10-14, 2004, Springer, Marseille, France, pp: 49-69.

Campbell, C., W. Grieskamp, L. Nachmanson, W. Schulte, N. Tillmann and M. Veanes, 2005. Model-based testing of object-oriented reactive systems with Spec Explorer. Technical Report MSR-TR-2005-59, Microsoft Research, May 2005.

Dustin, E., J. Rashka and J. Paul, 1999. Automated Software Testing: Introduction, management and performance. Addison-Wesley, New York, USA., ISBN: 9780201432879, Pages: 575.

Grechanik, M., Q. Xie and C. Fu, 2009. Maintaining and Evolving GUI-Directed Test Scripts, Proceedings of the IEEE 31st International Conference on Software Engineering, May 16-24, 2009, Vancouver, BC., Canada, pp: 408-418.

Grieskamp, W., Y. Gurevich, W. Schulte and M. Veanes, 2002. Generating finite state machines from abstract state machines. Proceedings of the 2002 ACM SIGSOFT International Symposium on Software Testing and Analysis, July 22-24, 2002, Rome, Italy, pp: 112-122.

Gurevich, Y., B. Rossman and W. Schulte, 2005. Semantic essence of AsmL. Theor. Comput. Sci., 343: 370-412.

Kanstren, T., 2010. A Framework for Observation-Based Modelling in Model-Based Testing. VTT Technical Research Centre of Finland, Espoo, Finland, ISBN: 9789513873769, Pages: 727.

Li, K. and M. Wu, 2004. Effective GUI Test Automation: Developing an Automated GUI Testing Tool. John Wiley and Sons, New York, USA., ISBN: 9780782143515, Pages: 445.

Mlynarski, M., B. Guldali, M. Spath and G. Engels, 2009. From design models to test models by means of test ideas. Proceedings of the 6th International Workshop on Model-Driven Engineering, Verification and Validation, October 4-9, 2009, Denver, CO., USA.

Myers, B., S.E. Hudson and R. Pausch, 2000. Past, present and future of user interface software tools. ACM Trans. Comput. Human Interact., 7: 3-28.

Paz, J.R.G., 2010. Pro Telerik ASP. Net and Silverlight Controls: Master Telerik Controls for Advanced ASP.Net and Silverlight Projects. Apress, Berkeley, CA., USA., ISBN: 9781430229407, Pages: 696.

Singhera, Z., E. Horowitz and A. Shah, 2008. A Graphical User Interface (GUI) testing methodology. Int. J. Inform. Technol. Web Eng., 3: 1-17.

Swain, S.K., S.K. Pani and D.P. Mohapatra, 2010. Model based object-oriented software testing. J. Theor. Applied Inform. Technol., 14: 30-36.

Utting, M. and B. Legeard, 2007. Practical Model-Based Testing: A Tools Approach. Morgan Kaufmann Publishers Inc., San Francisco, CA., USA., ISBN: 9780123725011, Pages: 433.

Villa, A., I. Cassarino and D. Antonelli, 2007. Extending group technology to the identification and the analysis of enterprises networks. Int. J. Prod. Res., 45: 3881-3892.

Ye, M., B. Feng and L. Zhu, 2007. Automated oracle based on multi-weighted neural networks for GUI testing. Inform. Technol. J., 6: 370-375.

Yuejin, L. and X. Jianyu, 2009. Conditioning for state space reduction in program model checking. Inform. Technol. J., 8: 990-997.