

Study on Application of .NET Reflection in Automated Testing

RuiZhen ZHOU*, ZuKuan WEI*, ShuGuang TANG**, Jae-Hong KIM***

**School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 610054, P.R.China*

***Sichuan WHAYER Information Industry CO., LTD., Chengdu, P.R. China*

****Dept. of Computer & Information Engineering, YoungDong University, Yongdong-Gun, Chungbuk, 370701 KOREA
Email: zhourz99@163.com, anlexwee@uestc.edu.cn, tsgtx@163.com, jhong@youngdong.ac.kr*

Abstract—To solve the problems in testing third-party components with Quick Test Professional, this paper designed and implemented a tool named DLL Analyzer on the basis of .NET reflection mechanism to analyze and obtain the internal information of .NET component. The experimental results indicate that this analyzer shall improve the efficiency in development of automated testing scripts.

Keywords—automated testing; reflection; Quick Test Professional

I. INTRODUCTION

Software testing occupies a fairly important position in software development. In fact, manual software testing is very time-consuming, labor-intensive, and monotonous. It may also introduce a number of human errors. Therefore, in the current computer technology, the use of automated testing methods and the corresponding test system has become an essential supporting means in the process of software development.

1) Automated testing is conducive to rapid regression testing and the speed and quality of releasing a new version. It is difficult to make a rapid assessment of the new version through manual testing owing to the high frequency and heavy workload of regression testing, particularly in the currently prevailing iterative development.

2) Some common mistakes in manual testing can be avoided by automated testing: wrong testing, repetitive testing and omitted testing.

3) Typical applications such as multi-user concurrent operation, concurrent transaction requests and concurrent responses, are almost impossible in manual testing, while the automated testing can realize them easily.

4) In the commonly used functional boundary testing, manual testing is time-consuming, and automated testing can be quick as well as accurate.

To sum up, the use of automated testing tools helps reduce testing costs, improve efficiency, rapidly position the

functionality, performance deficiencies of the tested versions and strengthen system reliability.

At present, some automated testing tools are quite popular including Winrunner, Loadrunner, Test Director, QuickTest Professional from HP-Mercury, Robot, Rational Function Tester from IBM and Apache's Jmeter. They are applied to black-box testing, white-box testing, performance testing and test management.

II. WHAT IS REFLECTION

A. The Concept of Reflection

The concept of reflection is first proposed by Smith B C in 1982, and mainly refers to the program's ability to access, test and modify its own state and behaviour [1]. A reflection system provides the presentation of its own behaviour. The presentation can be inspected and adjusted and it's associated to the behaviour it describes. Through the association, the change of the presentation will instantly be shown in the practical behaviour and status of the system. This concept soon led to the research on reflection's application in the field of computer science. It was first used in the design field of programming language, and then made accomplishments in Lisp and object-oriented technology.

Both .NET and Java have offered support in reflection mechanism. The reflection technology of .NET enables us to get the type information of component and create object instance during run-time. Type information is stored in the component's metadata. Metadata is binary information describing the types of the assembly. Every type and member defined or referenced in the assembly is describe in metadata, which gives components the capacity of self-describing. The essence of .NET reflection is to read and parse the binary metadata of assembly with metadata reader. To create an instance of type dynamically is the process of addressing the component's binary code according to type information and implementing the type's constructor.

B. The Application of Reflection in Automated Testing

Reflection has been an important mechanism of .NET platform. We can get the members of every type(class,

structure, delegate, interface and enum) in .NET, including method, attribute, event, qualifier and parameters, as well as constructor. Reflection can also dynamically create instance of types, bind it to an existing object, and then call the method to access their fields and attributes, or invoke some behaviours. Reflection mechanism can be used in testing. We can load the tested object through reflection, obtain its various properties, invoke specific events, so as to achieve the purpose of automated testing.

III. LIMITATIONS OF QTP'S OBJECT OPERATING

Nowadays, the automation of software testing is implemented primarily by the following methods: static and dynamic code analysis by specific tools, recording and replaying the process of testing, test scripting technology, virtual user technology, etc [2].

HP's Quick Test Professional (QTP) is an automated testing tool based on recording and playback. It records every step of the user's operation at the first stage of testing. There are two ways to record: the pixel coordinates of the user interface or the positions of the objects displayed by program (windows, buttons, scroll bars, etc.) [3], as well as the corresponding operations, properties changes or status changes. All the record are converted to the corresponding Visual Basic Script to simulate the operation of the user. In the latter part of regression testing, through the playback of the VBS script, you can achieve the same test. GUI object recognition technology is the basis for recording and playback.

QTP's identification of GUI objects is based on its plug-ins, because applications of different programming languages and development tools are slightly different in interface performance and response to events. The latest QTP 10.0 consists of more than 10 plug-ins for testing different software development platforms, including .NET, Java, Delphi and so on.

In practical applications, QTP object identification technology has certain restrictions: Software development uses a large number of third-party components and custom controls, but QTP's capacity to identify and operate these controls is weak, which may pose many problems to the development and operation of test cases.

Take a power management system as example ,the interfaces of the system universally use a third-party toolbar control named DevComponents.DotNetBar. It is identified by the QTP as SwfObject, a general object in QTP. clicking a button will generate the VB Script statement as follows:

```
SwfObject("bar1").Click 27,12
```

It can be seen that QTP records a click event on some coordinates of the toolbar instead of the click event of the button embedded. In this case, it may lead to errors in operation if the window size changes when this statement is played. In addition, the entire tool is identified as an object, which can not operate every single button separately. For example, it is unable to judge if some button is enabled or not, or set its enabled state through the direct operation of SwfObject.

To solve such problems, we can turn to the interfaces of the third party component itself. QTP provides users with two kinds of interfaces to operate objects: interfaces encapsulated by QTP, and those of the object itself in the development environment. QTP implements the encapsulated interfaces by calling the objects' own interfaces and methods. For example, SwfObject("bar1").Click was an interface encapsulated by QTP for the general object SwfObject. Object's own interfaces are included in the tested program, which are defined and implemented by the programming language and development platform.

After some study, the button object can be operated by the following methods:

```
set bar= SwfObject("bar1").Object // get the object
instance of SwfObject("bar1")
```

```
set but=bar.Items // obtain the items of the toolbar
object, which is an array of buttons
```

```
but.item(1).raiseclick() // call the raiseclick() method of a
certain button
```

raiseclick() is a method defined within DevComponents.DotNetBar to raise a click event. However, this method's name and parameters will not be shown in the prompt of QTP script code. In the current test script development, testers have to read the help document of the third-party component, analyze how to use it in a .NET project in order to obtain relevant information. And the uncertainty of third-party components' source code makes the analysis more complicated, reducing the efficiency of the script development.

To address these issues, this paper centers on mechanism of reflection provided by .NET platform, and proposes a method to analyze the standard .NET component automatically, implement a component analyzer to extract the internal information of components, display it on graphical interface and store it into XML document.

IV. PRINCIPLE AND IMPLEMENTATION OF DLL ANYLYZER

DLL Analyzer is designed to be a DLL, which is run by a statement "DotNetFactory.CreateInstance()" in QTP.

The path of the tested software is passed to it as a parameter. Testing script developers can get access to the internal information of the components used in the software through the analysis result.

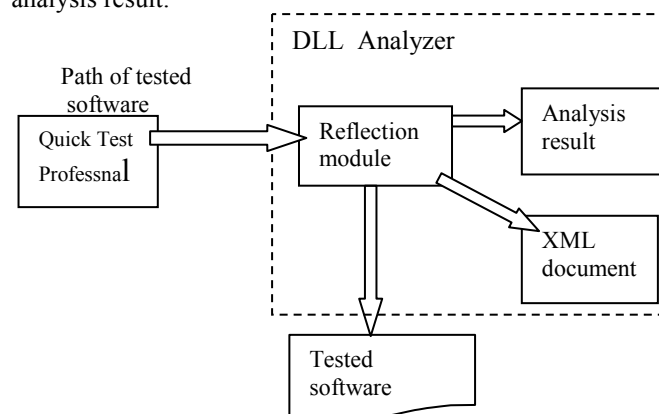


Figure 1. The interaction between DLL Analyzer and QTP

A. The Detailed Implementation of Reflection Module

step 1:search the directory of the tested software to find all the DLL files, and display them on the list. The main code is as follows:

```
// Load the appointed third-party components such as
DevComponents.DotNetBar. location is a variant storing the
name of the DLL.
```

```
Assembly assembly = Assembly.Load(location);
```

Step 2: obtain the internal information of the components according to logic level, and then enumerate its members(including the methods, events, interface),display it in the graphical interface.

```
// get all the types from the assembly
```

```
foreach (Type type in assembly.GetTypes())
```

```
{
```

```
// create a dynamic object according to the type
```

```
Object obj = Activator.CreateInstance(type);
```

```
// obtain the name, data type, statement type, reflection
type and read-only or writable status
```

```
PropertyInfo[] memberInfo = type.GetProperties();
```

```
//Obtain the name, return type, parameters and
implementation information(such as abstract or virtual) and
so on. Since QTP can call the public methods only, set the
flag "BindingFlags.Public" to get all the public methods
```

```
MethodInfo[] methodpri =
type.GetMethods(BindingFlags.Instance
|
BindingFlags.Public);
```

```
// obtain the name, event handler data types, custom
attributes, statement type and reflection type
```

```
EventInfo[] eventinfo = type.GetEvents();
```

```
// Divide the information into categories according to their
namespaces ,then present it respectively with nodes on the
graphical interface
```

```
.....
```

```
}
```

B. The Structure of The Output XML Document

Stored the component information and testing elements reflected from assembly into XML with defined DTD structure. For example, DevComponents.DotNetBar was reflected to be as follows:

```
<assmebly>
  <assname>DevComponents.DotNetBar2</assname>
  <fullname>DevComponents.DotNetBar2,
Version=6.6.0.4, Culture=neutral,
PublicKeyToken=bd7258ab2241255c</fullname>
  <namespace name="DevComponents.DotNetBar">
    <class name="Bar">
      <property>
        <pname>Items</pname>

        <datatype>DevComponents.DotNetBar.SubItemsCollection</
datatype>
```

```
</property>
```

```
...
```

```
<method name="GetItem">
```

```
  <paras>System.String</paras>
```

```
  <return>DevComponents.DotNetBar.BaseItem</return>
```

```
</method>
```

```
...
```

```
<event> ItemClick</event>
```

```
...
```

```
</class>
```

```
</namespace>
```

```
</assmebly>
```

V. EVALUATION OF DLL ANALYZER

Comparison between the QTP's own object analysis tool .NET Windows Form Spy and DLL Analyzer:

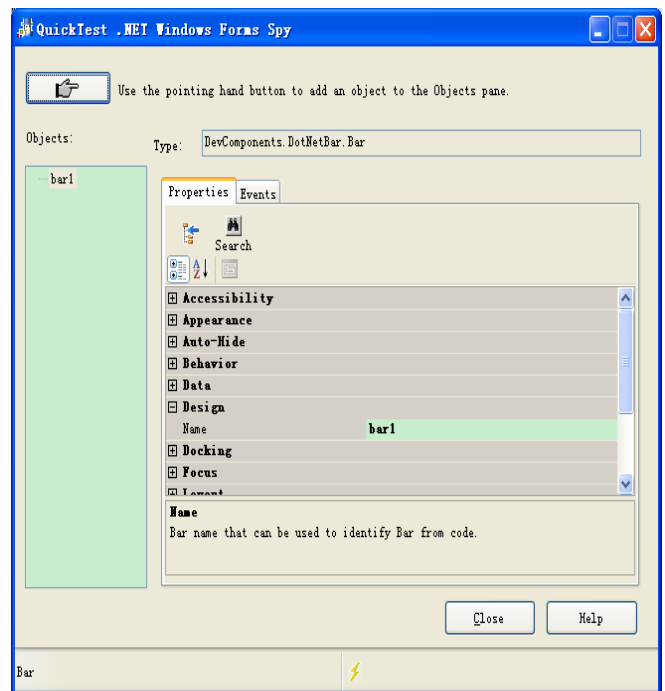


Figure 2. .Net Windows Form Spy of QTP

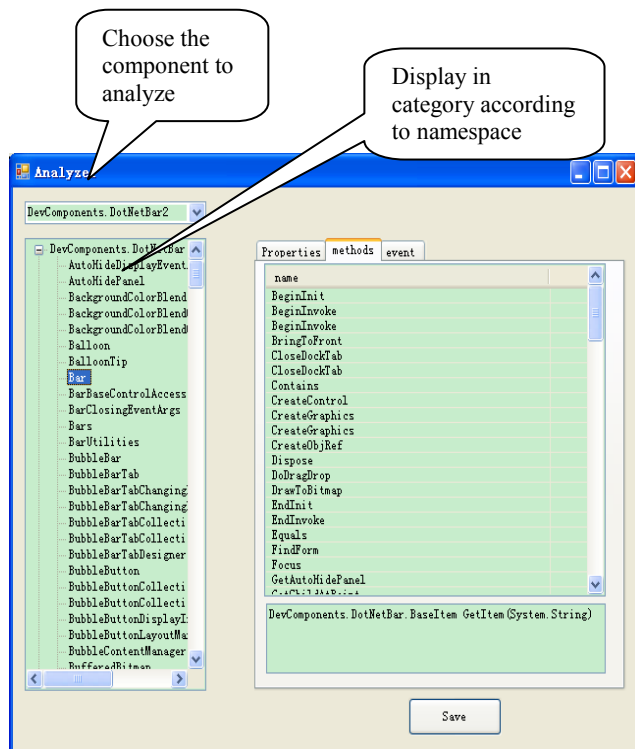


Figure 3. DLL Analyzer

Table 1. DETAILED COMPARISON OF FUNCTION

	DLL Analyzer	.Net Windows Form Spy
Objects analyzed	all the types defined and used in the DLL of the tested software	The entire third-party component will be identified as a whole object. It is difficult to view the information of sub-objects contained within the component
Information reflected	Properties, methods, and events of all types	Properties and methods of the object identified

Operational conditions	Analyze statically, and not need to run the tested software	Analyze dynamically and the tested software need to be running
Value of attribute reflected	Default value because of the static analysis	current attribute value of the running object
Savable	The result can be saved as an xml document	The result can't be saved
How to use	Call by the statement "dotnetfactory.C reateInstance"	Call by the menu of QTP

VI. CONCLUSION

This paper applies .NET reflection technology in automated testing, designs and implements the DLL Analyzer to obtain internal information of third-party components to give testers a quick access to the structure and interfaces information of components. Experiments show that it can improve the efficiency of testing in QTP.

REFERENCES

- [1] Smith B C. Massachusetts Institute of Technology [D]. Phd thesis, 1982
- [2] Kanglin Li, Mengqi Wu. Effective Software Test Automation. Translated by CAO-WENJING. Beijing: Electronic Industry Press, 2004
- [3] J A M Memon, M L Soffa, M E Pollack. Coverage Criteria for GUI Testing[C]. Proceedings of the 8th European Software Engineering Conference (ESEC) and the 9th ACM SIGSOFT International Symposium on the Foundations of Software Engineering, 2001, 256-267.
- [4] Glenford J. Myers. The Art of Software Testing. New Jersey: John Wiley & Sons, Inc., 2004
- [5] Elfriede Dustin, Jeff Rashka, John Paul . Automated Software Testing. Beijing: TsingHua University Press, 2003
- [6] James D. McCaffrey. Software Testing: Fundamental Principles and Essential Knowledge. Translated by LIU-XIAO WEI. Beijing: Electronic Industry Press, 2007.
- [7] Robinson, Christian Nagel. Professional C # Simon. Translated by LI-MIN BO. Beijing: TsingHua University Press, 2005
- [8] QTP Automation Object Model Reference[EB/OL]. Mercury Corporation. <http://quicktestprofessional.wordpress.com/category/automation-object-model-for-qtp/>