Discovery of Invocation Model and Dynamic Test Configuration Model based on TTCN-3 Test Systems

Liu Yongpo[1, a], Chang Chuangye[1] , Liu Shuangmei[2, b] , Wu Ji[1]

[1]School of Computer Science and Engineering, Beihang University, Beijing, China

[2]Science and Information college, Qingdao Agriculture University, 266001,Qingdao, China

[a]Liuypo@sei.buaa.edu.cn, [b]Lshuangm@163.com

Keywords: TTCN-3; Reverse engineering; Invocation model; Dynamic test configuration model

Abstract. Aimed at the comprehensibility, reusability and maintainability, the thesis presents the reverse model recovery for the legacy code developed by TTCN-3. It can also help tester and maintainers to verify the test implement, etc. The thesis introduces the discovery of invocation model and dynamic test configuration model based on the reverse model discovery system framework.

Introduction

With the unceasing development of TTCN-3, it has been increasingly applied to all kinds of testing. Testing systems based on TTCN-3 have similar characteristics with software development. But as the test systems grow in size and testers change, the management and maintenance of the huge test system have become more and more difficult. Therefore the reverse engineering of the test system based on TTCN-3 can help testers grasp the test system design from higher levels and can also test the consistence between test design and test implementation, which is of great significance and important value for expansion, evaluation and maintenance of the test system.

In this paper 'Design of discovery framework of the reverse model based on TTCN-3 test systems', the system Framework of reverse model had been designed, and the TTCN-3 static analyzer had also been realized by extending TRex. On the basis, the invocation model and dynamic test configuration model are both designed and implemented.

Discovery of Invocation Model

As a programming language for the test, TTCN-3 has the similar characteristics with the ordinary programming language, it has also the call relations among functions.As the test systems become larger and the complexity increases, the number of call among functions also increases, while the structure of the whole function call becomes more complex, such as caller ring, cross call etc.. On the other hand, as the test behavior is defined by test funtions, so only if the call relations among functions are well understood, the overall test behaviors of the test system can be grasped. Therefore, in order to comprehend and restructure the function call, this paper defines the testing function call, and gives the reverse discovery algorithm.

Define Invocation Model

In TTCN-3 there are three kinds of test functions, such as testcase, function, alstep, the call relations exist among them. This paper takes testcase as the center of test cases, tests the test behaviors in the realization process of testcase, and finds function and alstep called directly or indirectly.

Test function call is an action in the process of testing, which is defined by BehavirorStatements of the syntax element FunctionStatements in TTCN-3. The integrity of its definition is as follows.

When traversing the testcase syntax tree, the testcase is as a center. If FunctionInstance or AlstepInstance is defined, it shows that the testcase calls the function or alstep. Because function can call function or alstep, it will continue to traverse the found funciton in depth, and then will find other

Published by Atlantis Press, Paris, France.

function or alstep called by this function. In TTCN-3 the calling way of function and alstep is as follows:

- Testcase or function calls function directly, here is the code segment. The code shows that myTestcase and InternetPTC both run on MTC, which needs not additional new test component.

```
testcase myTestcase () runs on MTC{
        ……
        newInternetPTC();
        ……
}
```

- Testcase or function calls function by generatiing a test component. Here we only give the code segment that testcase calls, the code indicates that myTestcase and internetUser run on different test componets, my Testcase runs on MTC, but internetUser runs on InternetType. You can find it differnet from function call of traditional programming language. Here the call to internetUser is achieved by start operation from a test component.

```
testcase myTestcase () runs on MTC{
……
var InternetType newPTC := InternetType.create;
        newPTC.start ( internetUser() );
        ……
}
```

- Testcase or function calls alstep by active. Here we also give the code segment. Usually alstep and callers both run on the same test component. The call to alstep has also different characteristics, and is implemented by activate.

```
testcase myTestcase () runs on MTC{
      ……
      var default def := activate ( StandardDefault() );
          ……
}
```

Extract invocation model

According to the call relation defined in this paper, extracting invocation model can find abstract models from higer levels on the basis of gotten test system modles. Here from the beginning of testcase, you can find function or alstep called directly or indirectly in a testcase by using depth traversal strategy. During traversing, the innercall of Behavior is mainly concerned, which shows the call relation among the functions. Through the collection of innercall used in a testcase directly or indirectly, we can get a complete call. The algorithm is as follows:

Algorithm1: The algorithm of extracting invocation model

INPUT: Behaviors := { $behavior_i$ | $behavior_i$ is the Behavior in the TestBehavior }
OUTPUT: Invocation := { <caller, callee> | caller and callee are the Behavior in the TestBehavior }
ALGORITHM BEGIN
FOR each behavior in the Behaviors
IF behavior.type == TESTCASE
WHILE behavior.innerCall != NULL
caller = behavior;
callee = behavior.getInnerCall();
Invocation.add(caller, callee);
behavior = callee;
END WHILE
END FOR
RETURN Invocation;
ALGORITHM END

In the process of traversing, you should pay attention to the circulation call among functions, otherwise the algorithm would go into an infinite loop and cannot exit.

Discovery of dynamic test configuration model

One of important characteristics of TTCN-3 is the dynamic configuration. The test components can be created and configured in the running process. The traditional reverse engineering requires the combined way of static and dynamic to display the test system. The reverse engineering of test systems is not an exception. In this paper, the corresponding test system will run on Ttworkbench provided by the German FOKUS research institutions and Testing Technologies company, the dynamic trajectory can be captured by using the hook procedure. By analyzing the execution trace log, the dynamic configuration from one implementation can be extracted. You can make a comparison with dynamic configuration and static configuarion, which can provide more accurat and comprehensive analysis of the results for users.

Introduce Ttworkbench

Ttworkbench is a graphical test development and excution enviroment of TTCN-3, it provideds the editing functions of test case text and graphics as same as the TTCN-3 test specifications. After the testers define and describe the testcases using TTCN-3 , they can be compiled into executable testcases by Ttworkbench. Ttworkbench provides a comprehensive enviroment of test management and execution, while allowing users to manage, execute and analyze their test behaviors. Because Ttworkbench supports the automatic test, in the test cycle it not only reduces the cost , but also shorten the feedback time. The series of Ttworkbench products can be applied to many test products and services of different trades. The system being tested can be hareware, software or hareware and software integrated system.

Ttworkbench fully realized the framework of TTCN-3 test system. At the same time as a commercial IDE(Integrated Development Enviroment), in order to facilitate testers to use and develop it , Ttworkbench has also made much expansion and packging on the test system framework. The architecture of Ttworkbench is shown as Fig.1.

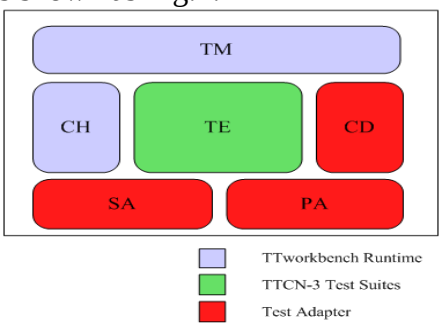

Fig1. The implementation framework of Ttworkbench

TM and CH have been provided by the platform itself, which can be used to control the test and interact with the test in the process of execution. Such that testers can concentrate on the test logic while writing test cases without considering the implementation of the test system. Here the main achievement of Ttworkbench is TCI specification. TE is mainly used to compile the test cases, this part mainly includes a compiler of TTCN-3, namely Ttthree, the testers mainly write and compile the test set; In the platform, CD, SA and PA are all defined in TestAdapter, which is mainly used to describe the test system interaction with SUT(System Under Test), including platform adapter, data exchage and timer settings, in this part Ttworkbench provides many interfaces for testers and mainly realizes TRI specification.

Capture dynamic trajectory

In TTCN-3 the dynamic generation and configuration of test components are completed through CH interacting with TE, shown as Fig.2.

In order to get the dynamic test configuration of the test system while executing the interaction interface between CH and TE is an ideal processing level. The method in this paper is to increase a hook module between CH and TE while running, and monitor the two interfaces of TciCHProvided and TciCHRequired through the hook procedure. Then the generation and configuration information of the test component will be recorded, the dynamic trajectory can be captured. At last the dynamic configuration information will be obtained.

Published by Atlantis Press, Paris, France.

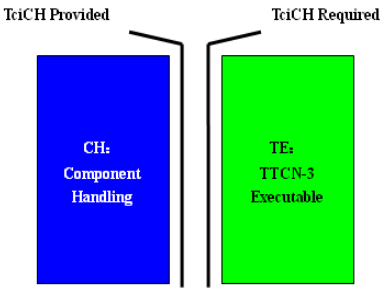

Fig.2 The ineraction interface between CH and TE

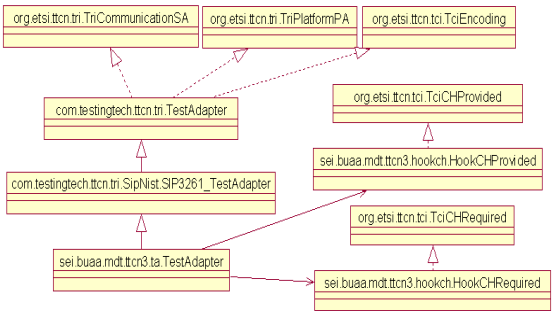

Fig.3 The Chart diagram of the hook procedure

Extract dynamic configuration information

In TTworkbench, running the test set needs to achieve the corresponding TestAdapter which must adapt to the SUT. It encapsulates the CD, SA and PA. TTworkbench provides the interface of com.testingtech.ttcn.tri.TestAdapter for testers, developers only need to inherit the testAdapter to customize the corresponding adapter, and then call org.etsi.ttcn.tck.TciCHProvided and org.etsi.ttcn.tck.TciCHRequired to complete the dynamic generation and configuration of the test component.

In this paper, in order to run the SIP conformance test suite, the adapter of its own is customized by inheriting the business adapter of SIP conformance test suite provided by Testing Technologies. In order to capture the interaction between CH and TE, the code segment is added into TciCHProvided and TciCHRequired to monitor the method calls through hook technology, so as to extract the running information of the test component. In the hook module two hook procedures have been achieved, i.e.: sei.buaa.mdt.ttcn3.hookch.HookCHProvided and sei.buaa.mdt.ttcn3.hookch.HookCHRequired, shown as Fig.3.

- sei.buaa.mdt.ttcn3.hookch.HookCHProvided: It implements org.etsi.ttcn.tci. TciCHProvided. It will use hook technology in all the methods associated with configuring and generating the test component and then serialize the captured dynamic information.
- sei.buaa.mdt.ttcn3.hookch.HookCHRequired: It implements org.etsi.ttcn.tci.TciCHRequired. It will use hook technology in all the methods associated with configuring and generating the test component and then serialize the captured dynamic information.

In the hook module, the parameters passed between method calls are mainly concerned, which record the dynamic configuration and generation information of the test component. This paper mainly analyzes three methods, shown in Fig.4.

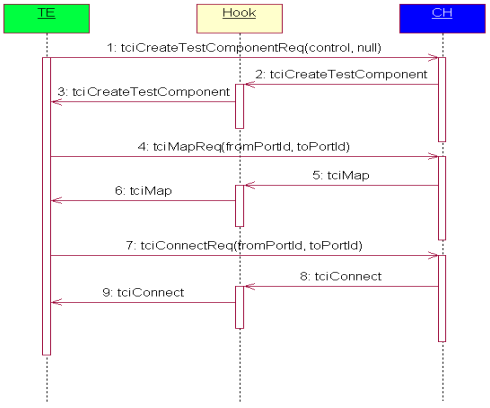

Fig.4 The sequence diagram of the hook procedure

- tciCreateTestComponent(int, Type)：It is mainly used to generate the test component dynamically. The first parameter indicates the roles of the test component, such as MTC, PTC or TSI. The second one indicates the specific types of the test component.
- tciMap(TriPortId, TriPortId)：It is mainly used to describe the mapping relation between MTC and TSI or PTC and TSI. The first parameter represents the mapping port, the second one is the mapped port.
- tciConnect(TriPortId, TriPortId)：It is mainly used to describe the connection between

MTC and TSI or PTC and TSI. The first parameter represents the mapping port, the second one is the mapped port.

By using the hook procedure, the information of dynamic generation and configuration from the test component is written into the test system meta model, which can acquire the trajectory. Then the dynamic test configuration can be gotten by analyzing the test trajectory, which shows the test configuration of the test system from two aspects of static and dynamic, and provides a more comprehensive show of the test system information for users.

Conclusions

When the test system becomes larger and the complexity increases, the function calls happen increasingly, at the same time the structural features of the overall function calls are also more complex.On the other hand, as the test behaviors are defined by test functions, only the good comprehension of the call relationships between functions can help testers grasp the overall test behaviors.

Therefore, in order to understand and restructure the call relationships between functions, this paper defines the testing function call, and gives the algorithm of call model extraction. Simultaneously by using TTworkbench, the test trajectory can be obtained, and then the dynamic test profile can be extracted.

Acknowledgement

This paper is contributed by National Key Laboratory of Software Development Environment Fund (SKLSDE-2010ZX-15) and High Technology Research and Development Program of China 863 Fund (2009AA01Z145).

References

[1] Jens Grabowski, TTCN-3-A new Test Specification Language for Black-Box Testing of Distributed Systems[C]. In: Proceedings of the 17th International Conference and Exposition on Testing Computer Software (TCS 2000), Theme: Testing Techonlogy vs. Testers' Requirements, Washington D.C., (2000)231-240

[2] PENG si-wei, ZHU qun-xiong. Inverse Modeling Based on Source Code Analysis[J]. Application Research of Computers. (2006) 52-54

[3] Chang-ai Sun, Jun Zhou, Jiannong Cao, Mao-zhong Jin, Chao Liu, YanFang Shen. ReArchJBs: a Tool for Automated Software Architecture Recovery of JavaBeans-based Applications , Proceedings of ASWEC 2005, IEEE Computer Society, Brisbane, Australia, March 29- April 1.(2005)270-280

[4] Z. R. Dai: Model-Driven Testing with UML 2.0, Second European Workshop on Model Driven Architecture (MDA) with an emphasis on Methodologies and Transformations (EWMDA'04), Canterbury, England, September 2004

[5] I. Schieferdecker, G. Din: A meta-model for TTCN-3. 1st International Workshop on Integration of Testing Methodologies, ITM 2004, Toledo, Spain, Oct. 2004

[6]TRex WebSite[OL]. http://www.trex.informatik.uni-goettingen.de, 2006

[7] Paul Baker, Dominic Evans, et al: TRex – The Refactoring and Metrics Tool for TTCN-3 Test Specifications, TAIC-PART, Proceedings of the Testing: Academic & Industrial Conference on Practice And Research Techniques.(2010) 90–94