A Study of Test Oracle for Application Interface Testing of Distributed System

Cui Jingyan
School of Computer Science and Engineering
Beihang University
Beijing, China
jingyan1949@126.com

Ye Gang
School of Computer Science and Engineering
Beihang University
Beijing, China
gang.ye@cs2c.com.cn

Li Xianjun
School of Computer Science and Engineering
Beihang University
Beijing, China
lixianjun@nlsde.buaa.edu.cn

Ma Shilong
School of Computer Science and Engineering
Beihang University
Beijing, China
slma@nlsde.buaa.edu.cn

***Abstract*—The application interface testing is an important part of automated software testing of distributed system. It is based on network data which is produced when the distributed system is running. It is used to verify whether the network data is correctly exchanged. In this paper firstly we design test oracle for application interface testing of distributed system. Test oracle includes two parts: oracle information and oracle procedure. Oracle information represents expected output while oracle procedure compares the expected output with the actual output. Test cases need to be decided whether their operating results are correct by test oracle. And then, four types of test oracle are designed and divided according to the range of oracle information and the strategy of oracle procedure. Through the experiment, we can see that the choice of test oracle will seriously affect the efficiency and cost of software testing. At last, some suggestion is given about how to design test oracle in the software test.**

Keywords: test oracle; application interface testing; distributed system

I. INTRODUCTION

The distributed system is a multiprocessor system whose structure and processing functions are distributed. These processors interact with each other via LAN or WAN connection. Normally, one or more application software are installed in a processor. Each software can complete part of the function in the system. Sometimes each application software is developed by different institutes; they have a high degree of independence, only through the interaction of the interface messages to accomplish a specific function. Therefore, the automated application interface testing of the distributed system is an important means to ensure the quality of the distributed system.

Automated testing includes capabilities to generate test cases and expected outputs. Generating test cases automatically is relatively easy while generating expected results is a relatively difficult assignment. In the automated testing, test oracle is a trusted source of expected outputs. It has been applied to verify test case results that are produced during the testing. Many testers believe that automated test efficiency mainly depends on the number of test cases and the design of each test case. In fact, how to design a test oracle also has a significant impact on test efficiency. For that matter, Atif Memon has illustrated the important role of the test oracle in the GUI testing through experiments [1][2].

In the rest sections of the paper, we first define the application interface testing model of distributed system, and then the test oracle for the specially testing is designed. At last, through an experiment, we tell you the importance to design a suitable test oracle for the regression testing of the application interface testing and give some suggestions.

II. TNE APPLICATION INTERFACE TESTING MODEL OF DISTRIBUTED SYSTEM

Before explain the application interface testing of distributed system, we first describe the structure of the distributed system, which is shown in Fig. 1.

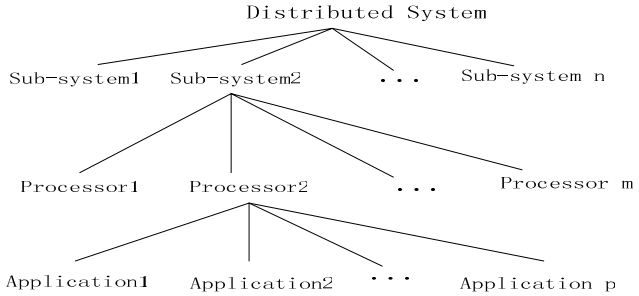

Figure 1. The structure of distributed system under test

The application interface testing can verify the correctness of network communication and the interaction between applications. At the point of view of the application interface testing, we describe distributed system as follows.

© 2012. The authors - Published by Atlantis Press

A distributed system has several subsystems:

$P = \{p_1, p_2, ..., p_i\}$

One or more applications on each subsystem:

$W = \{w_1, w_2, ..., w_i\}$

Each application may send or receive a variety of network messages:

$M = \{m_1, m_2, \ldots, m_i\}$

Definition 1: The state of a distributed system under application interface testing at a particular time *t* is the set of triples $\{(p_i, w_j, m_k)\}$, where $p_i \in P$, $w_j \in W$, $m_k \in M$.

Definition 2: A set of events $E = \{e_1, e_2, ..., e_i\}$ includes the events which will occur during the testing, where e_i is a man-made or system-triggered event. Each event will trigger several messages transfer which changes the state of the distributed system.

Definition 3: A set of states $S_I = \{S_1, S_2, ..., S_i\}$ is called the valid initial state set, where S_i is the state of the distributed system when it is beginning to run some test case of application interface testing.

Definition 4: A test case T of application interface testing of distributed system is a pair $<S_0, E>$, where $S_0 \in S_I$, $E = <e_1, e_2, ..., e_n>$, $<e_1, e_2, ..., e_n>$ is a legal event sequence, $n \geq 1$. The relationship between the event e_i and the messages it may trigger is expressed as: $M(e_i) = \{m_1, m_2, ..., m_n\}$, $n \geq 0$. And that, any message is related to some application, n = 0 means there is no message.

III. TEST ORACLE FOR APPLICATION INTERFACE TESTING OF DISTRIBUTED SYSTEM

A. *Test Oracle*

Intuitively, the oracle information is a description of the distributed system's expected state for a test case of application interface testing.

Definition: For a given test case $T = <S_0, E>$, $E = <e_1, e_2, ..., e_n>$, $M(e_i) = \{m_1, m_2, ..., m_m\}$, $n \geq 1$, $m \geq 0$, the test oracle information is a sequence$<S_1, S_2, ..., S_n>$, such that S_i is the expected state of the distributed system immediately after event e_i has been executed on it and messages $M(e_i)$ has been transferred.

The oracle procedure is the process used to compare the executing distributed system's actual state with the oracle information. If the actual equal the expected it will return true, otherwise false.

In later experiment, we design a comparator used to perform the verification process for oracle procedure.

We can see that the oracle information for a test case is a sequence of the sets of triples. According to the size of the sets, there may be the least descriptive oracle information set which just contains a single triple, describing a message that is sent or received by an application in a processor. On the contrary, the most descriptive oracle information would contain all the messages that are sent or received by all the applications of the distributed system. So on the basis of the messages of the oracle information are in the level of event, application, subsystem, or the entire system, we correspondingly divide oracle information into four levels G1, G2, G3 and G4. For a test case, it is true that $G1 \subseteq G2 \subseteq G3 \subseteq G4$. We may choose different levels of the oracle information for the same test case under different situations.

According to the partition of the test oracle information and the comparison strategy in the verification process, four kinds of test oracle are designed (LTO: level of test oracle).

LTO1: The oracle information is a set of triples $G1 = \{(p_i, w_j, m_k)\}$, which m_k is the message associated with the event e_i executed in the test case.

LTO2: The oracle information is a set of triples $G2 = \{(p_i, w_j, m_k)\}$, which w_j is the application associated with the event e_i executed in the test case, and all the messages associated with w_j are in the set G2.

LTO3: The oracle information is a set of triples $G3 = \{(p_i, w_j, m_k)\}$, which p_i is the subsystem associated with the event e_i executed in the test case, and all the messages associated with any application in the p_i are in the set G3.

LTO4: The oracle information is a set of triples $G4 = \{(p_i, w_j, m_k)\}$, which G4 is the most descriptive oracle information, and all the messages sent or received during the test are in the set G4.

For all kinds of test oracle, during the execution of the test case, only the actual results corresponding to their triples set are collected. After the last event of the test case, we compare the corresponding set and judge whether the results of the test case is true.

There is an example of the test oracle information shown in Fig. 2. After the event e_i is executed during this test case, a set of messages $M(e_i)$ will be triggered. There is only one message *m* named heart message in the $M(e_i)$. *m* is sent from application *w2* in the radar subsystem *p2* to application *w1* in the total control subsystem. At this time, the maximum set of triples of the distributed system is shown as *A* in the Figure 2. For the test oracle information in the level of LTO1, we just need to save triples $\{(p_1,w_1,m), (p_2,w_2,m)\}$ associated with the heart message *m* shown as *B* in the Fig. 2.

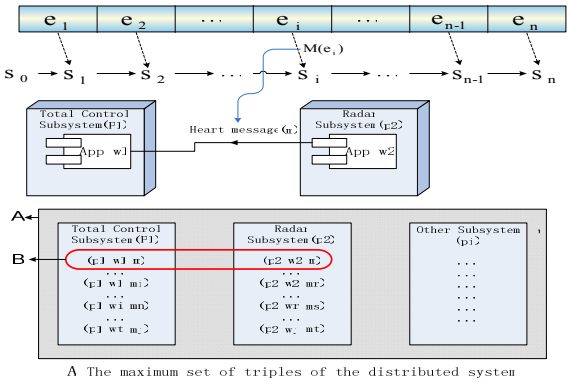

Figure 2. Test oracle information of the heart message

Many scholars have proposed many ways to generate oracle information: Memon et al. applied AI planning as automated GUI test oracle [3], Vanmali and his colleges proposed an approach to apply ANNs as test oracle [4], Schroeder and Korel used I/O relationship analysis to generate a reduced set of expected outputs with adequate cost [5], and so on. However, these methods are either suitable for GUI testing, but not suitable for the application interface testing, or difficult to be applied to the actual test items. Consequently, we use the method named execution extraction to generate the oracle information.

B. *Test Oracle Process*

Possible in the entire software testing activities, test oracle process can be described as follows [6]:

1) Generate expected outputs
2) Saving the generated expected outputs
3) Execute the test cases
4) Compare expected with actual outputs
5) Generate report of the result

See that test case is part of the oracle process, but it is not part of test oracle. Fig. 3 shows the oracle process and its activities.

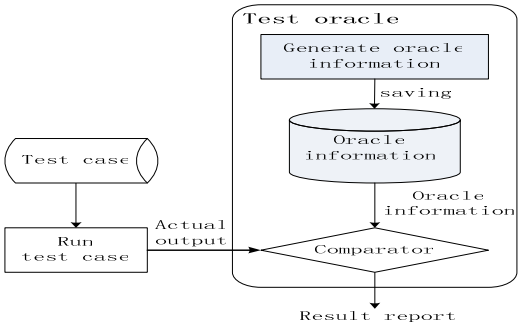

Figure 3. Test oracle process

IV. EXPERIMENT

Through this experiment, for a given system under test and the test case library, we do the regression testing with the automation test tool LoadRunner. Let's observe the performance of different oracles and how the difference of oracles affects test efficiency and test cost. Some parameters as bellow will be inspected.

Number of faults detected: We record the total number of faults detected by each oracle type.

Time used by oracle: The time includes time *t1* all the message sniffers spend capturing all the messages and time *t2* the comparator spends comparing expected outputs with actual outputs(in order to capture all the messages sent between applications, it is necessary to install the message sniffer on all the computers of the distributed system).

Space used by oracle: Because of the level of detail of the oracle information, each oracle type has different space requirements. We measure the space required to store expected outputs and actual outputs for different levels.

A. *Experimental Preparetion*

1) SYSTEM UNDER TEST

Our laboratory developed a simplified system according to some institute's large-scale distributed software system. We simplify the original system, while maintaining its main functions. Fig. 4 shows the structure of the system. The main parameters of the system can be seen in TABLE I.

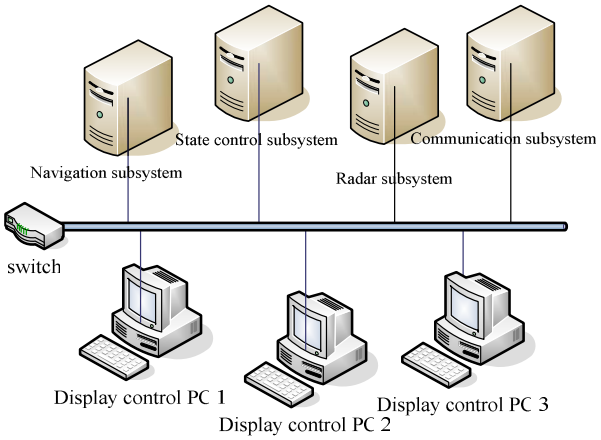

Figure 4. The network structure of the distributed system

We can learn from TABLE I that there are totally 163 kinds of messages. Each message consists of message header and message body. In addition, a message may have multiple uses by setting the value of the fields in the message body.

TABLE I. THE MAIN PARAMETERS OF THE DISTRIBUTED SYSTEM

Subsystem	Hardware	Number of applications	Number of kinds of messages sent or received by applications	
State control subsystem	State control PC	3	Receive: 58	send: 60
Navigation subsystem	Navigation PC	2	Receive: 20	send: 22
Radar subsystem	Radar PC	2	Receive: 33	send: 25
Communication subsystem	Communication PC	3	Receive: 37	send: 31
Display control subsystem	Display control PC 1	2	Receive: 15	send: 25
	Display control PC 2	2	Receive: 15	send: 25
	Display control PC 3	2	Receive: 15	send: 25

2) FAULT SEEDING

After several rounds of testing and modification, we have got a stable and correct version, which is called the standard version. Then we invite some students who didn't participate in develop of the system to modify some logic in the code and change filed values of some messages randomly. Last, more than 200 errors are injected.

For example, there is a field *state_request_e* of the message *SCM_RADAR_STATECONV_COMAND* which is sent from the state control subsystem (SCM) to the radar subsystem (RADAR), the students may change the value of the field randomly while the radar has several states such as standby, maintenance, no-response and work.

3) TEST CASES

Select 100 and 300 test cases from test cases library as group 1 and group 2. All the 510 test cases of the library are considered to be group 3. Each group of test cases should to be run 4 times with each type of test oracle.

4) DETAIL OF TEST ORACLE

1. Oracle information

Before the regression test, there is a correct version called standard version. We get the information of all the messages as the oracle information for some oracle by running all the test cases in the library with the oracle. The oracle information can tell us the expected results after running each test case.

The oracle information is saved with xml file. For a test case, the structure diagram of the oracle information's xml file is shown as fig. 5. The *messageName* is the child node of the *testCase*, and its number is changed according to the type of the test case. The child nodes of the *data* represent data fields, and the number of the child nodes is determined according to the type of the message.

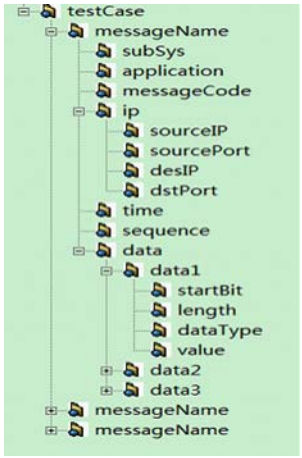

Figure 5. Structure diagram of the oracle information's xml file

2. Oracle procedure

The application interface message of the distributed system can be divided into two types: periodic message and aperiodic message. Each message is required to test their existence, accuracy and time sequence. TABLE II shows the comparator algorithm.

TABLE II. THE COMPARATOR ALGORITHM

BEGIN
1. Get the first message from the expected message queue
2. Whether it is periodic; if true then goto 5
3. Whether the message exists in the actual message queue; if false then goto 10
4. Check the values of assigned fields, record the results; goto 11
5. Whether the message has expected number in the actual message queue; if true then goto 8
6. Whether the message does not exist even one; if true then goto 10
7. Record that the number of the message is not enough, and cancel the test of time sequence
8. Check the assigned fields of each actual message, record the results
9. According to the arrival time of each message, compare the time interval, and record the results; goto 11
10. Record that the message is non-existent, and cancel the test of time sequence
11. Whether it arrives in the tail of the expected message queue; if true then goto 13
12. Get the next message from the expected message queue; goto 2
13. Determine whether do the test of time sequence according to the previous information; if false then goto 15
14. Test the time sequence according to the arrival time of each message in the actual queue, record the results
15. Write all the results to xml file

END

B. *Results*

There are 4 kinds of test oracle and 3 groups of test cases, so we need run the system for a total of 12 times. At last, the number of faults detected by the test cases, the time and space costs are counted for each software test. The results are as follows.

1. Oracle fault-detection ability

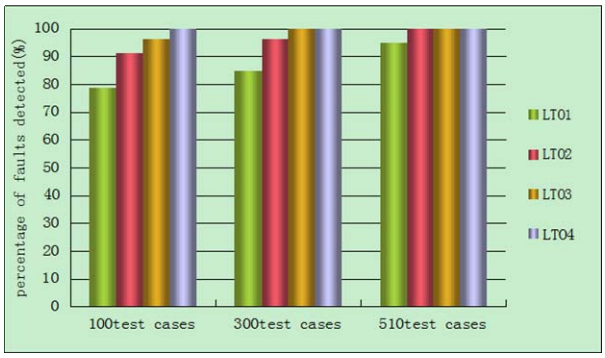

Figure 6. The fault-detection ability of 4 kinds of test oracle

As shown in fig. 6, with the oracle level increases, the fault detection ability is constantly enhanced. Moreover, with the number of test cases increases, it is possible to compensate the fault detection capability of the low level oracle.

In the other hand, if all the faults need to be detected, the higher the level of oracle, the less number of test cases it needs. However, even with all of the 510 test cases, LTO1 still cannot detect all the faults. In this case, in order to detect all the faults, we need to design some other test cases, or describe part of oracle information of LTO1 with a high level of oracle.

2. Time costs

The time costs of test oracle include time *t1* all the message sniffers spend capturing all the messages and time *t2*

the comparator spends comparing expected outputs with actual outputs. During the test, we record *t1* and *t2* of all the test oracle and count all of the time costs.

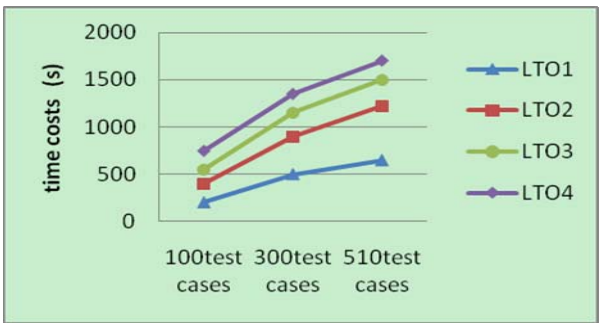

Figure 7. Total time for test oracle

As shown is Fig.7, with the increase of the oracle level and the number of test cases, the time spent is also increasing. That's because the sniffers need more time to capture the messages and the comparator needs check the messages captured with more times of comparisons.

3. Space costs

Since the computers now normally have a huge storage capacity, it is not very important that how much space test oracle costs to store the information. But if the regression testing will be executed frequently, this indicator has certain reference value. We measure the space required to store expected outputs and actual outputs for different oracle levels.

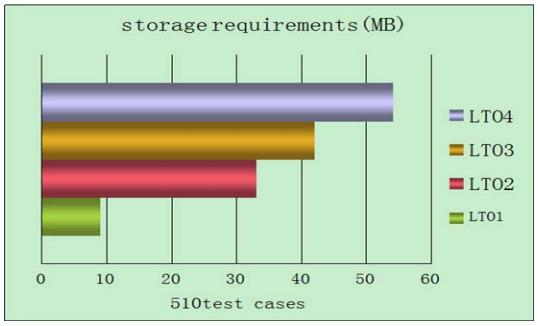

Figure 8. The storage requirements for the group 3

As can be seen from Fig. 8, the higher the level of oracle, the larger storage space is required. Besides, the difference is very obvious from LTO1 to LTO2 which shows that more detailed information need to be recorded from LTO2.

C. *Analysis of Results*

The above results show that the test oracle does play an important role in the application interface testing of distributed system. According to the test oracle we defined, from LTO1 to LTO4, the fault detection ability is stronger but more expenses of time and space are also needed. In the regression testing, if the limit of the cost of time and space is not high, then we should try to choose a high-level oracle so that all the software errors can be detected. But in Fig. 6, we can also see that if there are enough test cases, low-level oracle can fully meet the requirements. So in the actual test, we can select a suitable oracle according to the number of regression testing, the number of test cases, test time requirements and the restrictions on the storage space. When necessary, different test oracle can be combined. In order to reach an optimal balance point of fault detection efficiency and reducing testing expenses, we can run part of test cases with high-level oracle while other part of test cases with relatively low-level oracle.

V. FUTURE WORK AND CONCLUSION

This paper first formally describes the test oracle for application interface testing of distributed system, then we design several different test oracle. After the experiment that the distributed system of our lab is tested automated, we can see that the choice of test oracle will seriously affect the efficiency and cost of application interface testing. At last, some suggestions are given to help you choose the right test oracle for automated testing.

In the future, we will study the new ways to generate oracle information except the way *execution extraction* we used. Some methods used in the GUI testing such as IFN Regression Tester and ANN Based Test Oracle will be considered how to work on the application interface testing of distributed system.

REFERENCES

[1] A. M. Memon, I. Baneree and A. Nagarajan, "What Test Oracle Should I use for Effective GUI Testing?," Proc.IEEE International Conference on Automated Software Engineering (ASE'03), Montreal, Quebec, Canada, pp. 164-173, October, 2003.

[2] Xie, Q., and Memon, A.M.: 'Designing and comparing automated test oracles for GUI-based software applications', ACM Transactions on Software Engineering and Methodology, 2007, 16, (1), pp. 4

[3] Memon, A.M., Pollack, M.E., and Soffa, M.L.: 'Automated test oracles for GUIs', SIGSOFT Softw. Eng. Notes, 2000, 25, (6), pp. 30-39

[4] Vanmali, M., Last, M., and Kandel, A.: 'Using a neural network in the software testing process', International Journal of Intelligent Systems, 2002, 17, (1), pp. 45-62

[5] Schroeder, P.J., Faherty, P., and Korel, B.: 'Generating expected results for automated black-box testing', in Editor (Ed.)^(Eds.): 'Book Generating expected results for automated black-box testing' (2002, edn.), pp. 139-148

[6] S.R.Shahamiri, M.N.W.K.Wan and Z.M.H.Siti. "A Comparative Study on Automated Software Test Oracle Methods," Proc. International Conference on Software Engineering Advances (ICSEA'09), IEEE Press, Sep 2009.

[7] Last, M., Friendman, M., and Kandel, A.: 'Using data mining for automated software testing', International Journal of Software Engineering and Knowledge Engineering, 2004, 14, (4), pp. 369-393

[8] Aggarwal , K.K., Singh, Y., Kaur , A., and Sangwan , O.P.: 'A Neural Net based Approach To Test Oracle', ACM Software Engineering Notes, 2004

[9] D.K. Peters and D.L. Parnas, "Using Test Oracles Generated from Program Documentation," IEEE Trans. Software Eng., Vol. 24, No. 3, Mar . 1998, pp.161–173.

[10] Hu, J., Yi, W., Nian-Wei, C., Zhi-Jian, G., and Shuo, W.: 'Artificial Neural Network for Automatic Test Oracles Generation'. Proc. Proceedings of the 2008 International Conference on Computer Science and Software Engineering - Volume 022008.