Study of Methodology of Testing Mobile Games based on TTCN-3*

Huiqun Zhao, Jing Sun
Department of Computer Science
North China University of Technology
Beijing, China
zhaohq6625@sina.com

Gongzhu Hu
Department of Computer Science
Central Michigan University
Mount Pleasant, Michigan, USA
hu1g@cmich.edu

Abstract

Systematic software testing plays a critical role in software development. Various approaches have been proposed and developed, but software testing is still far from maturity. A very promising strategy is model-based testing that is aimed to automatically generate test cases and evaluate test results based on the software models that specify the behavior of the software under test. In this paper, we propose a general rule model for mobile games by combining the features of the game software and the Wireless Application Protocol. We also build an adjusted model for generating test cases with the characteristics of mobile game software. A series of patterns are created to facilitate the generation of TTCN-3 test suite and test case. A case study on the Fruit Machine game is discussed to illustrate the use of the proposed model for software testing.

Keywords: Mobile game software, software testing, game rule model, test case, TTCN-3

1. Introduction

The computer game market has exploded in the last decade and will continue to grow much faster than many other industries. According to the 2006 PricewaterhouseCoopers media outlook report [1], the global video game market (including console, hand-held, PC, online, and wireless) was estimated to grow at the annual rate of 11.4% to reach $46.5 billion by 2010, with the Asia Pacific region leading the growth at 12.3% from 2005 to 2010, and the USA at a bit slower pace of 8.9%. Among the game categories, wireless is expected to grow fastest at about 23-28% during the 2005–2010 period. The wireless game market in China grew close to 100% in 2004 and over 50% in 2005, and number of game subscribers increased 66.7% in 2006 from the previous year [10]. We believe that the most recent data would show a even higher grow prediction of the game market. For example, Verdict Research predicted [8] that that the video games market would grow 42% in UK in 2008.

All the data show that the game software industry is moving towards the wireless direction to provide ubiquitous access to games from hand-held devices, particularly from mobile phones, via wireless communication networks. There are several technical platforms for mobile phone games, notably Short Message Service (SMS), Wireless Application Protocol (WAP) and JAVA Micro Edition. SMS has a much larger market size but it appears to have a very limited growing potential, while WAP and JAVA games are expected to keep a rapid growth.

Many aspects of today's society depend more and more on software, and the quality of software directly affects the quality of everyone's life. To this extent, software testing is considered an integral part of the core requirements for development of quality software. Testing of mobile phone games is a new area of study in software testing, mainly because mobile phone game software present some new challenges that may not be very critical in other types of games. such as limited computing power and distribution of software modules between the phone and the back-end server.

Automation has been a primary goal for software testing for many years. Software testing was often done manually until recently when *model-based testing* [3] was introduced. It automates the validation testing process based on a theoretical model on which the software product was design and implemented. Based on the model, the testing process directly links to the specification and requirements of the software.

To meet the need for automating software testing, a testing language called *Test and Testing Control No-*

*This work is partially supported by the Natural Science Foundation of China (Grant No. 4062012) and Beijing Government and Education Committee (Grant No. KM200710009009).

978-0-7695-3642-2/09 $25.00 © 2009 IEEE
DOI 10.1109/SNPD.2009.26

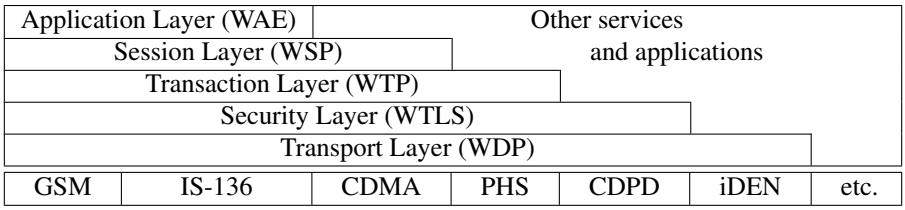

Figure 1. WAP architecture (source: [5]).

tation (TTCN-3) [9] was developed during 1999–2002 at at the European Telecommunications Standards Institute (ETSI). It is adopted as a standardized language for software testing applicable to many types of software specifications and communication interfaces, such as the Internet and mobile protocols. More details of TTCN-3 can be found in [6, 7, 14].

In this paper, we propose a general rule model for mobile games and build an adjusted model for generating test cases. A series of patterns are created to facilitate the generation of TTCN-3 test suits and test cases.

The paper is structured as follows. After a short introduction about WAP in Section 2, the methodology for testing mobile game that design based on WAP is provided in Section 3, where different test aspects of mobile games testing are discussed. Section 4 examines a game test example. A brief discussion on related work is given in Section 5, and Section 6 summarizes and concludes this paper.

2. Modeling of WAP Mobile Games

Among mobile games, WAP games are now in a dominating position [4]. To demonstrate the idea of WAP game software testing, we briefly review the WAP game software design framework and discusses how to model WAP game software that may be generalized for other types of model-based testing.

The WAP architecture is structured as a layered protocol as shown in Figure 1. Each of the layers of the architecture is accessible by the layers above, as well as by other services and applications. When the mobile game user enters WAP portal, only the Application Layer (WAE) is needed to communicate with the mobile phone. It employs the HTTP/XML protocol with POST format request/response, and the request and response commands are encoded in "Entity Body" and its XML description as follow:

```
<?xml version="1.0" ?>
<misc_command version="1.5">
  <command_name>command</command_name>
  <command_data_block>
    <data_tag1>data1</data_tag1>
    <data_tag2>data2</data_tag2>
    ......
  </command_data_block>
</misc_command>
```

The `misc` in `misc_command` is an identification of the command, the `command_name` is a name of command and it may take different representations such as provision, provision_response, sso and sso_response. The `command_data_block` and `data_tag` are data parameters of the command.

The general mobile communication process between the user, the WAP application, and the WAE can be illustrated in the diagram in Figure 2.

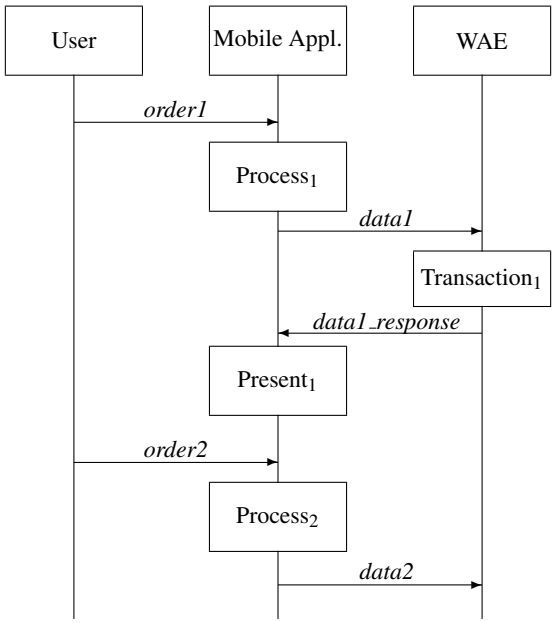

Figure 2. Massage Sequence Chart of mobile application.

This figure shows a Massage Sequence Chart (MSC) that is a formal presentation framework employed by TTCN-3 and UML Testing Profile. Each message (or `misc_command`), such as *order1*, *order2*, *data1*, *data1_response* and *data2*, is send from the User

and Mobile Appl to WAE and vice versa. The Mobile application performs processing and presentation, as Process_i and Present_i indicated in the figure.

Although this MSC can clearly specify the interaction among the participating modules in a mobile application, the model has to be adjusted for mobile game software development for the following reasons:

- The computing resources and power of mobile devices are very limited. This limitation constrains the way games are developed simply because most games require high computing power, from data loading speed to frame refreshing rate. One of the commonly used approach to solve this dilemma is to rely on the servers to bear the heavy computing load. J2ME, for example, a popular platform for developing games for wireless devices, only forwards user's `misc_command` to the servlet engine on a server to perform the task of a game. This spreading of modules of a game software among the mobile device and its servers make testing much more difficult than the case where all game components are on the same machine.
- In order to avoid memory overloading, some development platforms have to combine classes into one if they vary only slightly in behavior. For example, the packages and classes in J2ME are structured much more compact and smaller than standard Java API. This practice also makes it harder for testing because the "units" to be tested are no longer as clearly cut as in the standard case.

To reflect these constraints on wireless games, an adjusted model for game software is developed as shown in Figure 3.

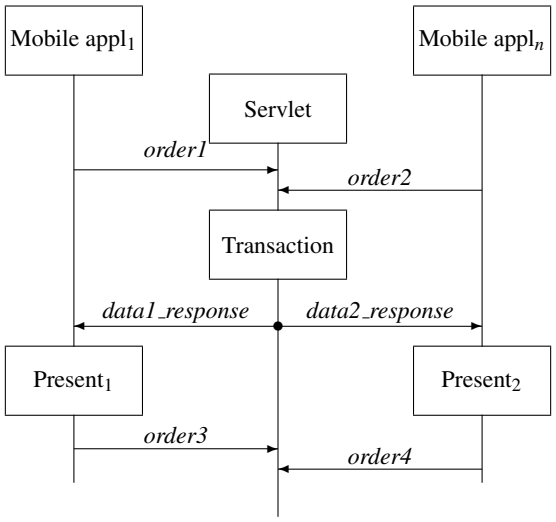

Figure 3. An adjusted game model.

In this figure, Mobile appl_1 and Mobile appl_n are game terminals, Servlet is the server that provides computing ability to terminals, and Transaction means that the Servlet handles all requests from different terminals of same game within one transaction. In the adjusted model a game terminal handles only the presentation task of the game such as animating pictures and note promotions. The servlet takes on all of process of a transaction including data-loading, logic-computing, game-verdict and so on.

3. Game Model and Test Case Generation

The methodology of generating test case based on the adjusted game model is discussed in this section. We first outline the general ideal to generate test case based on MSC, and then propose a specific method for game software testing based on the adjusted model.

The TTCN-3 graphic presentation format GFT [2], inspired by MSC, defines the relation between the GFT diagrams and the TTCN-3 Core Language. It is quite easy to map GFT diagrams to TTCN-3 core language framework based on that relation. For example, an instance of GFT diagrams shown in Figure 2 can be mapped to following core language framework.

```
Module Myinstance () {
   User.send(order1)
   Mobile_Appl.receive(order1)
   Mobile_Appl.Execute(process1)
   Mobile_Appl.send(data1)
   WAE.receive(data1)
   WAE.Execute(transaction)
   WAE.send(data1_resp)
   ......
}
```

According to the GFT standard, each GFT diagram has its corresponding core language framework. Therefore, generating test case from the GFT diagram is quite straightforward. For example, a sample of the test case framework in the TTCN-3 core language for testing the WAE is given below:

```
testcase example1()
         runs on Mobile Appl
{
   User.send(order1)
   WAE.receive (data1)
   WAE.send(data1_resp)
   User.send(order2)
   WAE.receive(data2)
   ......
}
```

It appears that the test case can be automatically created from MSC with ease. For some kinds of software that are based on network protocols, yes indeed, this observation is almost right because the MSC or GFT can describe those software specifications correctly except data structure definitions. However, it is not easy to generate test case for game software. As mentioned above the software model has to be adjusted for memory and speed limitations on wireless devices. In order to facilitate test case generation from adjusted model, we propose a new test case generating pattern that relies on the servlet on the server rather than running on WAE. The previous test case can be adjusted as follows:

```
testcase example2() runs on Servlet
{
   Mobile_Appl1.send(order1)
   Mobile_Appl2.send (order2)
   Mobile_Appl1.receive(data1_resp)
   Mobile_Appl2.receive(data2_resp)
   Mobile_Appl1.send(order3)
   ......
}
```

In this pattern, each of the mobile applications is an *Up Test* or *Low Test*, and the `Servlet` is a *System Under Test* (SUT). It is obvious that this pattern takes advantages of GFT map rule. Unfortunately, this pattern loses the messages, which are indispensable for generating test case from model adjustment. To solve this problem the above pattern is modified to include an invocation of the transaction function on the servlet. That is, in contrary to the above generated test case, a function execution `Servlet.Execute` is invoked between the `send` and the `receive` commands that take `data1` and `data2` as the reference parameters:

```
testcase example2() runs on Servlet
{
  Mobile_Appl1.send(order1)
  Mobile_Appl2.send (order2)
  Servlet.Execute Tran1(data1,data2)
  Mobile_Appl1.receive(data1_resp)
  Mobile_Appl2.receive(data2_resp)
  Mobile_Appl1.send(order3)
  ......
}
```

With the models and test case patterns discussed above, a general algorithm for generating test case from game model with several steps can be outlined in Algorithm 1.

Algorithm 1: *Test Case Generation*

```
1  begin
2      Design a game rule model based on MSC;
3      Adjust the rule model as following:
4      begin
5          foreach message m do
6              Map m transmitted/received into
               parameter of send/received
               command;
7          end
8          Insert the function invocation commands
           into the proper positions using the "lost"
           messages as parameters;
9      end
10     Build the TTCN-3 Core Language TestCase
       Module;
11 end
```

4. Case Study

In this section, we use the mobile game *Fruit Machine* by Nokia as a case study to illustrate our approach for test case generation based on the model discussed in the previous sections. We shall also discuss the implementing method of the test case pattern.

4.1. Fruit Machine and Test Case

The Fruit Machine mobile game that Nokia released is a typical client-server application consisting of a Mobile Information Device application framework (MIDlet) client and a Java servlet server. The MIDlet for the Fruit Machine mobile game, *FruitMachineMIDlet*, runs on a MIDP-enabled phone. It provides a user interface for logging in to the server and playing the game. The server-side service module of the game, *FruitMachineServlet*, runs in an HTTP servlet engine, such as Apache's Jakarta Tomcat. The servlet maintains a database of user accounts, and generates random results during the game. A Web browser can be used to administer the *FruitMachineServlet*, using an HTML form-based interface to create and delete user accounts. The game protocol is a custom application protocol implemented over HTTP. Figure 4 shows the game rules.

4.2. Test Case Implementation

Based on the game rule model in Figure 4, an adjusted game rule model for the Fruit Machine mobile game is derived and shown in Figure 5. A TTCN-3 test case pattern can be easily mapped from this adjusted

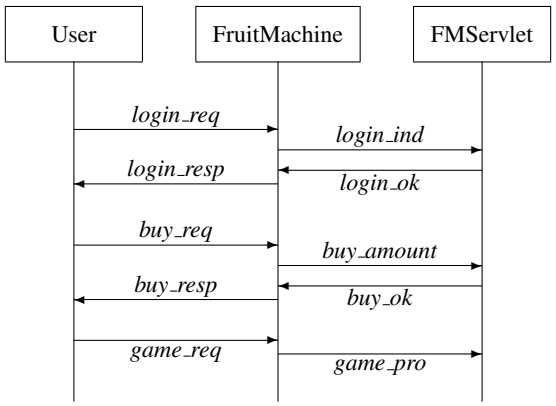

Figure 4. Game rule for Fruit Machine.

game rule model, as given below:

```
testcase Fruit_Machine()
         runs on FMServlet
{
   FruitMachine.send(login_req)
   FMServlet.Execute Tran1(login_ind)
   FruitMachine.receive(data1_resp)
   FruitMachine.Execute Pres1()
   FruitMachine.send(login_req)
   FMServlet.Execute Tran2(buy_req)
   FruitMachine.receive(data1_resp)
   FruitMachine.Execute Pres2()
  ......
}
```

To implement this test case, we use the open source tools *TTthree* (a TTCN-3 compiler) and *TTman* (an IDE for TTCN-3 run-time) provided by TestTech. Unlike other TTCN test tools, *TTman* only defines a framework, and the Test Runtime Interface (TRI) needs to be complemented by the user.

The TRI defines the interaction between several modules in the TTCN system. It consists of two sub-interfaces, a *TriCommunication* interface and a *TriPlatform* interface.

According to the TRI definition, we introduced several sub-interfaces to complement *TriCommunication* and *TriPlatform* for testing the Fruit Machine game. To simplify the discussion, we describe only two sub-interfaces in this section.

```
1. public TriStatus triMap(
      TriPortId FruitMachine,
      TriPortId FMServlet)
```

A `triMap` operation returns a flag of `TriStatus` as a success or failure of the operation:

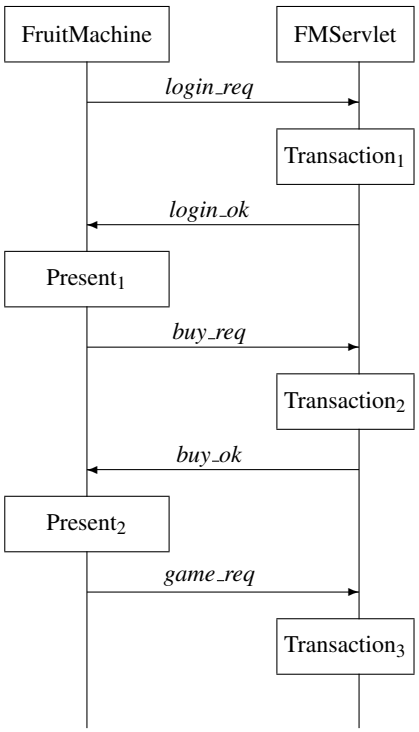

Figure 5. An adjusted game model.

- `TRI_OK` — if connection is established successfully or no dynamic connection needs to be established by the test system.
- `TRI_Error` — if a connection could not be successfully established.

This operation is invoked by the TTCN-3 Executable when it executes a TTCN-3 map operation. In this interface, `FruitMachine` is an identifier of the test component port, and `FMServlet` is an identifier of the system interface.

```
2. public TriStatus triSend(
      TriComponentId FruitMachine,
      TriPortId tsiPortId,
      TriAddress address,
      TriMessage message)
```

The interface `triSend` defines the components for sending the test case to the System Under Test (SUT). An operation of `triSend` returns `TRI_OK` if the operation is completed successfully, `TRI_Error` otherwise.

In the interface, `FruitMachine` is the identifier of the sending test component, `tsiPortId` is the identifier of the test system interface port, through which the message is sent to the SUT Adapter, `address` (optional) is the destination address within the SUT, and

message is the encoded message to be sent such as *Login_Req*, *Buy_Req*, *Game_Req*, and so on.

5. Related Work

Model-based testing, the principle we used in this paper, has been researched and studied extensively [11, 12, 13, 15]. As the great promise of model-based testing shown in the last decade, increasing number of model-based testing tools started to emerge to be used commercially (e.g. T-Vec and Reactis) and for academic research (e.g. SpecTest, Korat, and TGV).

Since the TTCN-3 is a new international standard for protocol testing, more and more corporations and academic institutions are conducting research on software testing using TTCN-3. Michael Ebner [2] provided a method for translating MSC elements to TTCN-3 statement. It uses MSC to define test cases and test case execution order in TTCN-3. Xiong, Probert and Stepien [16] proposed a formal testing approach for web services with TTCN-3, which distributes the test activities to both server and client sides. However, both of those studies did not discuss how to adjust the MSC model, that this paper is to address.

6. Conclusion

In this paper, we briefly reviewed the issue of mobile game development and testing, and the WAP protocol that WAP mobile games are usually based on. We proposed a general mobile games rule model that takes the advantage of MSC and its diagrams to describe the game rules. An adjusted model was built by combing the characteristics of mobile game software. Using GFT and the mapping rules between GFT and the TTCN-3 core language, we showed some patterns that can better facilitate the generation of TTCN-3 test suite and test case. The proposed approach was illustrated using the pattern of a specific game software, the Fruit Machine game, to generate a test case. In order to show the pattern's effectiveness, we discussed an implementation of the test tool built on top of some open source tools and interfaces that are defined in the TTCN-3 standards.

We are currently working on several experiments on mobile game testing and developing specialized tools using the framework discussed in this paper.

References

[1] BusinessWeek. Global video game market set to explode. http://www.businessweek.com/innovate/content/jun2006/id20060623_163211.htm, 2006.

[2] Michael Ebner. TTCN-3 test case generation from message sequence charts. In *In Workshop on Integrated-reliability with Telecommunications and UML*, 2004.

[3] Ibrahim K. El-Far and James A. Whittaker. Model based testing for real-the inhouse card case study. In John J. Marciniak, editor, *Encyclopedia on Software Engineering*, volume 1, pages 825–837. Wiley, 2001.

[4] Wireless Application Forum. Wireless profiled TCP. http://www.wapforum.org.

[5] Wireless Application Protocol Forum. WAP architecture: Wireless application protocol architecture specification. http://mobile.yqie.com/service/wap/source/wap/download/spec-waparch-19980430.pdf, 1998.

[6] Jens Grabowski, Dieter Hogrefe, György Réthy, Ina Schieferdecker, Anthony Wiles, and Colin Willcock. An introduction to the testing and test control notation (TTCN-3). *Computer Networks*, 42:375–403, 2003.

[7] Jens Grabowski, Anthony Wiles, Colin Willcock, and Dieter Hogrefe. On the design ofthe new testing language TTCN-3. In R.L. Probert H. Ural and G. von Bochmann, editors, *Testing of Communicating SystemsTools and Techniques*, volume 13. Kluwer Academic, 2000.

[8] Michael Holden. Video games are tops in UK entertainment. http://www.msnbc.msn.com/id/27554817, 2008.

[9] ITUT. ITU-T Recommendations Z.140-142: The Testing and Test Control Notation version 3 (TTCN-3), 2002. Geneva, Switzerland.

[10] Organization of China Investing Consultation. Prediction report of 2008-2010 years network game market. http://www.ocn.com.cn/reports/2008436soujiyouxi.htm.

[11] Wolfgang Prenninger and Alexander Pretschner. Abstractions for model-based testing. *Electronic Notes in Theoretical Computer Science*, 116:59–71, 2005.

[12] Alexander Pretschner, Heiko Ltzbeyer, and Jan Philipps. Model based testing in incremental system development. *Systems and Software*, 70:315–329, 2004.

[13] Alexander Pretschner, Oscar Slotosch, Ernst Aiglstorfer, and Stefan Kriebel. Model based testing for real-the inhouse card case study. *Journal on Software Tools for Technology Transfer*, 5(2):140–157, 2004.

[14] ISO Information Technology. Open systems interconnection, conformance testing methodology and framework, 1991.

[15] Mark Utting. Position paper: Model-based testing. In *IFIP Working Conference on Verified Software: Theories, Tools, Experiment*, 2005.

[16] Pulei Xiong, Robert Probert, and Bernard Stepien. An efficient formal testing approach for web service with TTCN-3. In *Proceedings of the 13th International Conference on Software, Telecommunications and Computer Networks*, 2005.