MobileTest: A Tool Supporting Automatic Black Box Test for Software on Smart Mobile Devices

Jiang Bo
School of Computer Science and Engineering, Beihang University, Beijing, China
gongbell@gmail.com

Long Xiang
School of Computer Science and Engineering, Beihang University, Beijing, China
long@buaa.edu.cn

Gao Xiaopeng
School of Computer Science and Engineering, Beihang University ,Beijing, China
gxp@buaa.edu.cn

Abstract

With the development of mobile computing and pervasive computing, smart mobile devices such as PDAs or smart-phones are gradually becoming an indispensable part of our daily life. However, as the software running on these devices becomes more and more powerful and complex, the testing of these mobile applications poses great challenges for mobile application vendors and phone manufacturers. In this paper, we introduce MobileTest, a tool supporting automatic black box test for software on smart mobile devices. The objectives and the design of the testing tool are thoroughly discussed. The paper also adopts a sensitive-event based approach to simplify the design of test cases and enhance the test cases' efficiency and reusability. Finally, we conducted an experiment in a real testing project. Measurement data of the testing process shows that MobileTest can effectively reduce the complexity of automatic test on smart mobile devices.

1. Introduction

The goal of pervasive computing is to enable everyone to compute and communicate anywhere at anytime [1]. And smart mobile devices, such as PDAs, smart-phones, are undoubtedly the most promising portals for us to achieve these goals. And by "smart" it means the devices have an embedded OS such as Windows Mobile, Symbian, Linux and Palm. It is these embedded OS that make third-party application possible and popular. Currently, there are abundant applications on smart mobile devices ranging from mobile-commerce, multimedia, games, location based services, PIM application to standard applications like SMS, MMS and voice calling. However, when developing mobile applications, the third-party software vendors as well as phone manufactures face a dilemma.

At one hand, end users have high expectation on the quality of the software on smart mobile devices. Users require that their smart mobile devices to be reliable and stable. They will not be comfortable with a mobile phone that crashes and loses personal data as often as a PC. This requires the phone manufacturer and third-party software vendors to guarantee the high quality of their products. And testing is their most important tool to achieve this.

On the other hand, the testing of the mobile application on smart mobile devices is difficult.

Firstly, the environment of target devices is complex as they have to interact with end users, wireless signals and other devices in a context-sensitive way [2].

Secondly, the diversity of the devices and platforms reduces the reusability and maintainability of test cases.

Thirdly, the devices are highly resource constrained in terms of processing ability, memory capacity and communication ability. Thus, a testing approach that is highly intrusive to the target system under test may affect the actual result of the testing.

Finally, the behaviors of smart mobile devices are highly interactive. The devices constantly accept activations from the users and send responses back for user to take further actions. Since it's hard to predict a user's actions, many of the usage scenarios are hard to be automated.

2. Survey on the automatic testing tools for mobile application

Many automatic testing tools are developed to help the software vendors and phone manufacturers get out of the dilemma. In this section we will briefly summarize the merits and drawbacks of related black box testing tools for mobile applications and the related research in this area.

TestQuest Pro is a non-intrusive automated test solution that provides comprehensive support for a wide range of electronic devices [3]. Simulating the presence of a "virtual" user, TestQuest Pro executes predefined actions and compares the output to valid states to determine whether the test was successful. [4]

The merits of TestQuest are summarized as follows. First, TestQuest Pro adopts its innovative Test Verb technology [5]. The test verb is a high level abstraction of the actions for the target device, so it can provide maximum portability of the test cases.

Second, TestQuest Pro provides strong image comparison, text recognitions and audio data comparison capabilities for test results verification. This is essential for testing GUI applications automatically and non-intrusively.

Third, TestQuest Pro provides different agents applications on different devices to activate the target device and return the screenshots back [6, 7].

However, several problems limit TestQuest Pro's ability to a higher level of testing automation.

Firstly, TestQuest Pro solely depends on image comparison and text recognition to determine the state of target. Using a busy-waiting approach, image comparison is resource consuming and often ineffective for detecting states changes. And this may also reduce the reusability of the test scripts since GUI are prone to change.

Secondly, TestQuest Pro did not provide a synchronization mechanism for testing several devices' interaction. The testers have to estimate the time of waiting, which may lead to unexpected testing failures.

Thirdly, TestQuest Pro fails to take environmental factors into consideration.

Digia AppTest is another testing tool that allows developers and testing engineers to quickly record, edit and run effective tests [8]. One of the most userful features of Digia Apptest is that it can test critical mobile software attributes such as memory consumption, error management and fileserver behaviors. Digia AppTest can be distributed over the air directly to the target device, with specific test scripts installed.

The drawbacks of Digia AppTest is that it is solely an application running on the smart-phone, this makes the fully automatic testing of interactive functions very hard without human intervention. For example, it is hard for Digia AppTest to test two phones that use push-to-talk service to communicate with each other.

Several researchers have worked on testing context-sensitive softwares [9]. Ichiro Satoh brought forward an approach for testing context-sensitive networked applications by emulate the physical mobility of portable computing devices through the logical mobility of applications designed to run on them [10]. This approach is helpful for testing mobile applications that interact with environments. But it only supports applications running on JVM within an emulator. It is not applicable to applications using native APIs of target devices.

The Agent technology is also very useful when applied to software testing. Jeongeun Choi and Byoungju Choi [11] brought forward an Agent based testing tool. And Yu Qi, David Hung and Eric Wong brought forward an Agent based Web application testing method [12]. They use the Agent technology to reduce the complexity of testing Web applications. Although their solutions prove effective in a specific area, their uses of Agent technology are not well adapted to test mobile applications on smart mobile devices.

To address the above problems, a sensitive-event based testing approach is brought forward in this paper and the design of the system is well considered to enhance the efficiency of testing mobile applications.

The following sections are organized as follows. Section 3 discusses the objectives of our system - MobileTest. In Section 4, the design of MobileTest is presented and the sensitive-event based approach is brought forward. Section 5 uses an experiment conducted in real testing projects to show the effectiveness of our system. Section 6 concludes the paper and discusses future works.

3. The objectives of MobileTest

The objectives of MobileTest are as follows:

- Good support for interactive operations test, volume test, multiple states test, boundary test and multiple task tests.
- Minimize the usage of image comparison and text recognition for state determination.
- Create test cases library and provide schedule mechanism for regression test and smoke test.
- Accommodate for as many devices as possible and provide adapters for future devices to be added.
- Provide flexible storage support for test cases, configuration data and results.
- Strong result verification abilities. Such as image comparison, OCR, audio and video data verification.
- Provide mechanisms to control the environment of the target device so that it's possible to test the device's behavior in different environments.
- Support script generation and execution visualization.

4. Design of MobileTest

In this section, we will briefly summarize the design of various components of MobileTest. MobileTest is an automatic black box testing tool for smart mobile devices. It can build sophisticated, maintainable and reusable test cases library for testing system level and application level software on a variety of smart mobile devices.

4.1. MobileTest architecture

The architecture of the MobleTest is shown in the Figure 1 below. The system uses a layered design to reduce the complexity of the whole system. Each layer provides services to upper layer with the support of lower layer. In this way, the test control layer can be separated from the peculiarities of the underlying devices.

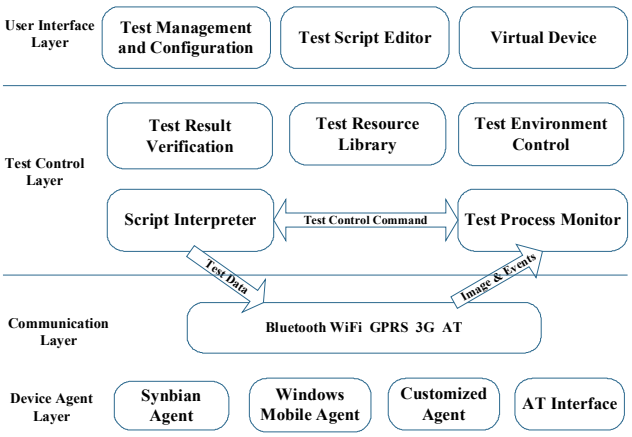

Figure 1. MobileTest Architecture

The system is composed of four layers:

The User Interface Layer interacts with testers. It runs on the workstation.

Test Control Layer executes the test scripts, sends simulated operations to target devices, receives screenshots and sensitive events from the target devices and further controls the test process according to sensitive events. This layer also resides on the workstation.

The Communication Layer connects the Test Control Layer and the Device Agent Layer. It runs on both the workstation and target devices.

The Device Agent Layer receives commands from upper layer, executes them and sends the status of the target back.

4.2. Basic usage scenario

The basic usage scenario of MobileTest is shown in Figure 2. And the main process is as follows: ① The testers make configurations on the target device and the test environment. ②The testers make test plans and write test scripts in the script editor manually or generate test scripts using the virtual devices. ③The scripts are scheduled to run. ④ When the scripts are interpreted, the script interpreter uses the uniform interfaces provided by communication layer to send simulated keys or other input information to the Device Agent running in the target device and then the script interpreter suspends itself. ⑤The Device Agent simulate the keys in the target device and return the screenshots and results to the process monitor module via communication module. The Device Agent may also notify the test process monitor about the sensitive events automatically. ⑥ After receiving feedback, the test process monitor will activate the script interpreter to continue interpreting next statement and save the test results into the test resource library.

4.3. Sensitive-event based testing approach

In this section, we present the sensitive-event based testing approach to achieve the objective of testing interactive operations among several devices as well as supporting volume test, multiple states test, boundary test and multiple task tests[13].

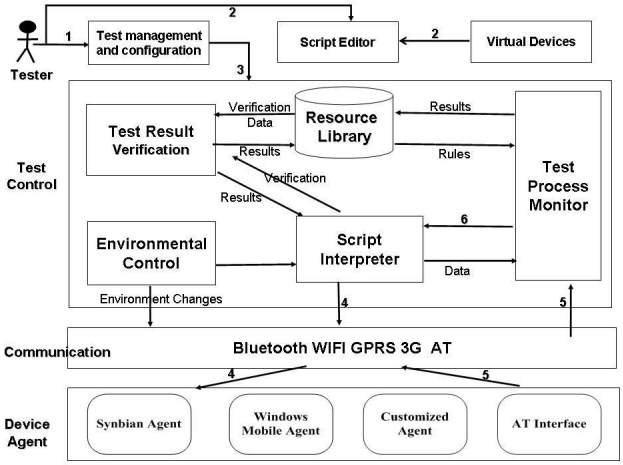

Figure 2. The basic usage scenario

4.3.1. Definition. *Definition 1(Sensitive Event)*: Sensitive event is any event occurs in the target device under test that is related to the current goals of testing.

Examples of system events are "inbox_full", "incoming_call", "call_end", "inbox_emppty", "disk_full", "phonebook_full", "memory_low", etc.

Sensitive-event based approach is to capture the sensitive events from the target device in real-time and notify them to the test control layer. The control layer is responsible for taking further actions to change the testing process according to the rules.

In our system, the agent running on target system is responsible for capturing and notifying the sensitive events. And the test process monitor receives these events and controls the interpreter and other modules to handle them. To enable sensitive-event based testing, the script interpreter should be able to suspend its interpretation and resume when sensitive events arrive.

4.3.2. Support for interactive operation testing. Testing interactive operation requires the system to be able to control several devices simultaneously and provide a mechanism for the synchronization between several devices.

The communication module provides the ability to connect to several devices simultaneously using Bluetooth, WiFi or 3G. And to provide synchronization between different devices, the agent programs running on the target device capture events such as incoming call, SMS arrival and ad-hoc connection request from other devices, and send them to test control layer as sensitive events in real-time.

4.3.3. Support for volume test & boundary Test. Volume test means to send extremely high volumes of data to target devices. Typical volume test includes: [14]

Table 1. Volume test cases

	Description
1	Fill in as many entries as possible to the inbox until it's full
2	Keep creating new drafts until inbox is full
3	Keep creating new files until disk space is emptied

Boundary test aims to test the target device's function when it's in the process of a long operation or during state transition. Typical boundary test includes:

Table 2. Boundary test cases

	Description
1	Receiving a voice call when deleting Call History
2	Receiving an SMS when emptying IN_BOX
3	Activate alarm or reminder when downloading a MMS

During volume test and boundary test, the system needs to be notified when target devices have changed to specific states such as in box full, memory low, etc. In our system, the agent program keeps watching the occurrence of states related events and captures them when they appeared.

4.3.4. Support for multiple states test and multiple-task test. Multiple states test is to check the target devices' function when it is in different states [15, 16]. Typical test cases are shown in Table 3.

Table 3. Multiple states test cases

	Description
1	Receiving a call when it is in charging sate
2	Test the function of application in low memory state
3	Test the function of application when battery is low
4	Test the function of networked application when signal is weak

Multiple tasking is an essential feature of smart mobile devices. For example, you can read a SMS while listening to the music stored in the mobile phone. Most of the operating systems of smart mobile devices support multiple tasks processing. Multiple-task test tries to verify that multiple applications can run on the target devices simultaneously. Typical test cases are shown in Table 4.

During multiple states tests, the system must first set the target to a specific state such as battery full, battery low, charging or weak signal. And in multiple-task tests, the system must ensure an application is running in a specific state such as in foreground, in background or sending data.

It is impossible or very hard for image comparison to achieve these objectives, but using the sensitive-event approach, we can keep sensing and change the devices' state until it satisfies the testing requirements.

Table 4. Multiple-task test cases

	Description
1	When sending file using Bluetooth, insert data cable
2	When using VOIP application, receive a voice call
3	Switch between multiple applications quickly
4	When composing a new message, receive an incoming call

4.3.5. Reduction of image comparison and text recognition for state determination. Image comparison may lead to extra maintenance work when reusing test cases. Suppose we are going to fill in the drafts with new messages until it is full. If using the image comparison, the test case has to keep capturing screenshots and make regional comparisons to find the Drafts_Box_Full dialog. This is resource consuming as images must be transmitted continuously. Further more, when the target devices changes, the reference images may be invalid. Our system solves this problem by using the sensitive events as the decision criteria. For the above example, when the draf_box is full, the agent program will capture this event immediately and send it to the test control layer. We do not have to capture and compare the images anymore.

4.3.6. Support for Exception Handling. During the testing process, it's possible for exception condition to occur. For example, system crash, no memory or battery empty. It's nearly impossible for image compassion approach to capture these events. In our system, these exceptions are all defined as sensitive events. So when exception happens, the agent program will capture them and send them to the test control layer to take further actions.

4.4. Test case management and scheduling

One of the advantages of automatic testing is its efficiency in large scale regression test and smoke test. To achieve this objective, we manage our test scripts in testing projects and schedule the tests according to testing plans. As exceptions and failures may occur during a large regression test, we must ensure the following tests won't be affected by previous failures or exceptions. To solve the problem, we write our testing schedules as scripts. So we can use the script language and our script interpreter to gain a refined control over the scheduling process.

4.5. Script Generation and Testing Visualization

Script generation visualization requires the system to record user operations and generate test scripts in the script editor accordingly. And testing visualization is to show the current status of the target device in a visual manner. These are crucial for enhancing the usability of the tool and we use virtual devices to meet these requirements.

The virtual device simulates the GUI of the target devices. When we click the buttons or tap a position on the virtual UI, the virtual device will map the operation into a script statement and add the statement into the script editor. And during testing execution, the screenshots of the target device will be captured and sent to virtual devices for displaying in real-time.

4.6. Test resource management

The test data generated in automatic test is huge. The system must manage the configuration information, test plans, test cases, test data, test results and other test resources. The test cases will be retrieved often. And the test results should be stored in a manner suitable for displaying, measurement and even data mining. We use a database to achieve this objective. We identify the relation of the data, define data tables for each kind of testing resources and use a uniform interface to access data.

4.7. Test results verification

Test results verification is a critical step for a close-loop testing. Since the main source of output of a smart mobile device are screen images and audios, it's important to verify their correctness. We rely on image comparison, OCR and audio comparison for test result verification.

Since the information on the screen of a smart mobile device may change over time, such as the clock, we use the regional image comparison and regional text recognition abilities to filter unnecessary parts of the screenshots.

4.8. Communication module

The Communication Layer has the following design considerations. Firstly, it supports many communication media. Secondly, it provides a uniform interface for the test control layer despite the underlying diverse smart mobile devices so that the upper test framework can work on virtual operations. Thirdly, it uses an adapter mechanism to allow new devices to be added easily.

4.9. Device agent

This layer is made of specific agent programs running on smart mobile devices. The device agent receives the control command from the test control layer, performs corresponding operations, such as simulating key presses and stylus click, on the smart devices and returns the sensitive events and screenshots back. We use the software agent technology to reduce the complexity of interacting with the target device [17]. Currently, we have device agents support Symbian and Windows Mobile platform.

4.9.1. Decision Rules for the Device Agent. Device Agent uses the following rules to decide its actions when it captures events and exceptions. The decision rules are stored in the built-in embedded databases of the device.

- Sensitive Event Report Decision Rules (SERD-RULE)

This decision rule decides whether the Device Agent will report the status information to the upper layer as sensitive event. It makes the Device Agent only send the status related to current testing goals to upper layer.

$$\forall x, y, DeviceStatus(x) \wedge TestGoal(y) \wedge Member(x, Relevant(y)) \Rightarrow Report(x)$$

- Exception Handling Rules (EH-RULE)

This decision rule decides how to deal with the exceptions occur during testing. It tells the Device Agent to handle the exceptions according to the pre-defined solutions.

$$\forall x, y, Excetion(x) \wedge Action(y) \wedge Solution(x, y) \Rightarrow DoAction(y)$$

4.9.2. Software Agent Characteristics. The Device Agent has following typical characteristics of a Software Agent.

The first one is Autonomy. With the Device Agent, the testers do not have to care about the specifics of target device. The Device Agent is a good abstraction of the target device. The upper test framework only need to perform corresponding virtual operations via communication module and the Device Agent will map these virtual operations to real operations on the devices.

The second one is social ability. The Device Agent can communicate and cooperate with other components of the testing system in a social manner.

The last one is intelligence. The Device Agent does not send all information it can acquire to the test control layer, as this may lead to unnecessary transmission and processing burdens. Instead, it filters the information according to the SERD-RULE and only sends the ones that closely related to the current testing goals. And the device Agent can also report and help handle exceptions happened during testing according to EH-RULE. For example, if the battery of the devices is low, it will notify the test control Agent to stop testing. And if the application crashes, test control layer may ask device agent to help restart the system.

5. Evaluation

5.1. The Design of the experiment

A comparison experiment is conducted in China Telecommunication Lab by experienced testing engineers in a real testing project. In the experiment, three testing teams are founded and there are two testing engineers in each team. All the engineers have more than one year's experience on testing mobile application. So the difference of different teams is very small.

The three teams are required to test three different phone models of the same platform: Nokia 6630, Nokia 6680 and Nokia 6681. They are all based on Symbian OS 8.0. The UI of these phones are nearly the same and the platform difference is negligible. We use the same Symbian platform to make the comparison of experiment results more persuasive. But their testing tools are different. The first team used TestQuest Pro. The second team used MobileTest. The third team performs testing manually. The test case specifications are based on China's national telecommunication industry standard on the test method of digital cellular mobile telecommunication device [18].

All the teams are given the same tasks. On the first phase, their work is to perform the same test cases specifications ranging from basic function test, volume test, multiple state testing, multiple task testing and boundary testing. They are required to automate as many test cases as possible as well as fulfilling the test tasks. This experiment is designed to compare the effectiveness of the different tools when testing different types of test cases.

On the second phase, they are required to exchange their phones and execute the same test cases as that of the phase one using the automated test cases created in the first phase. This is to compare the maintenance cost of the test cases generated by different testing tools

We also measure the time and bugs of both the automatic testing tools and manual test to determine the effectiveness of the automatic testing tools.

5.2. Result analysis

5.2.1. The First Phase. From the measurement data of the first phase (Figure 3), we can see that for function tests, TestQuest has a higher rate of test cases automation. This is due to its strong test result comparison technology. But when it comes to volume test, multiple state test, multiple task tests and boundary test, the MobileTest is superior.

This is because in these tests, some states of the target devices are hard to capture using images comparison or OCR technology used by TestQuest Pro. But the sensitive-event based testing approach makes the detection of these states much easier.

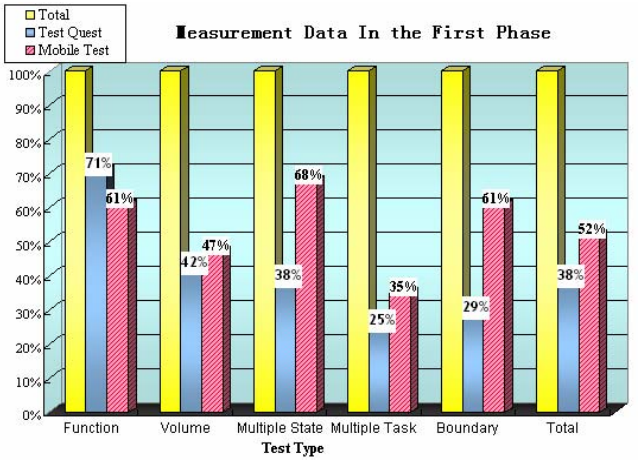

Figure 3. Measurement data of the first phase

5.2.2. The Second Phase. Note in the first phase, the total number of automated test cases generated by TestQuest is 352 and the total number of automated test cases generated by MobileTest is 477. From the measurement data of the second phase (Figure 4), we can see that when the phone model is changed, 173 out of 352 of the test cases generated by TestQuest Pro need to change. But only 98 out of the 477 of the test cases generated by MobileTest require maintenance. So the automated test cases generated by MobileTest are more reusable and efficient.

This is because the reference images used by image comparison approach are often invalid after the phone model is changed. In contrast, most of the test cases using the sensitive-event based approach can remain unchanged.

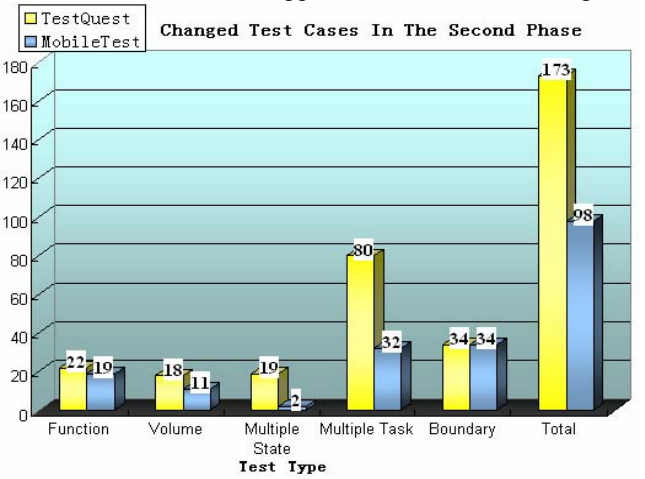

Figure 4. Measurement data of the second phase

5.2.3 Test Efficiency and Effectiveness. From Figure 5, we can see that in the first phase, the test time of using Test Quest or MobileTest is longer than manual test. That is because writing new automated test cases takes a lot of time. But in the second phase, the advantages of using automated test tools are apparent.

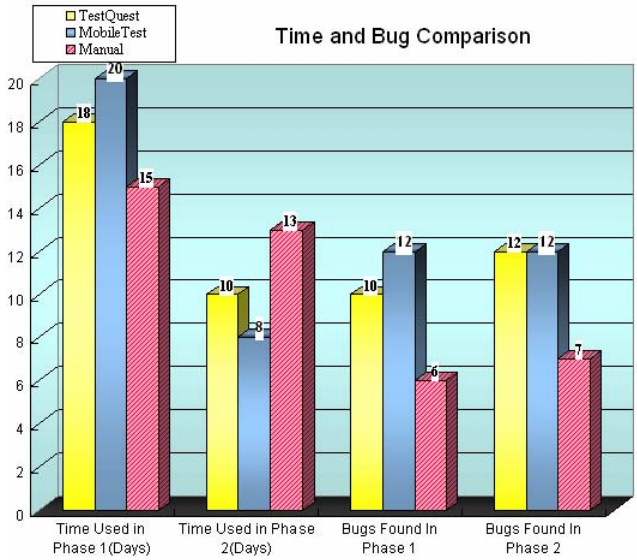

Figure 5. Time and bug comparison

And we can see that the time used by TestQuest is shorter than MobileTest in phase 1. This is because the TestQuest is a mature commercial tools and it is easier to use than MobileTest. But the test case maintenance effort in phase 2 makes TestQuest fall behind MobileTest finally.

Another result is that the automated tests find more bugs than manual testing. This is because some testing cases are too time-consuming or impossible for manual test. For example, testing the maximum volume of the phone book, out box or disk are missions impossible for manual test. As a result, manual testers finally choose to give up these testing tasks but it is just these test cases that generate bugs.

6. Conclusions and future works

In this paper, we introduced MobileTest, a tool to perform automatic black box test for software running on smart mobile devices. The tool adopted a sensitive-event based approach to simplify the design of test cases and enhance the test cases' efficiency and reusability. It also uses software agent technologies to simplify the system design. An experiment is conducted in real testing project that shows effectiveness of our tool and approach.

Future works will focus on the extension of our testing tool to support the whole software testing process. And we are also trying to use the test environmental control component to control a small base-station for simulating different wireless signals. This is helpful for testing the mobile devices under different network conditions.

7. Acknowledgments

We owe our thanks to the School of Computer Science and Engineering and the Department of System Engineering of Engineering Technology of Beihang University for the joint project on automatic mobile application testing. And we must also thank the China Telecommunication Lab for the opportunity they provided to conduct the experiment.

8. References

[1] Lothar Merk. *Pervasive Computing Handbook*. Springer-Verlag Telos, Germany, 2001.

[2] G. Chen and D. Kotz. A survey of context-aware mobile computing research. Technical Report TR2000-381. Dartmouth College, Hanover, New Hampshire, 2000.

[3] TestQuest. TestQuest Pro Guide. Technical Report, http://testquest.iexposure.com

[4] TestQuest. Automating the Functional Testing of Wireless Handset Software. *TestQuest White Paper*. http://www.superinst.com/downloads/products/testquest

[5] TestQuest. Developing a Low Cost, Test Automation Solution with TestQuest TestVocabulary/TestVerb Technology. *TestQuest White Paper*. http://testquest.iexposure.com

[6] TestQuest. TestQuest Interfaces for Symbian OS™. TestQuest Product Data Sheet. http://www.testquest.com

[7] TestQuest. TestQuest Interfaces for Windows Mobile™. TestQuest Product Data Sheet. http://www.testquest.com

[8] Tommi Välimäki. Test Automation in Symbian Smartphone Development. Presentation. SysOpen Digia Plc. http://www.tol.oulu.fi/projects/wg4/minutes/Test_Automation_Symbian1.ppt

[9] I. Satoh. A testing framework for mobile computing software. *IEEE Transactions on Software Engineering*, 29(12):1112–1121, Dec. 2003.

[10] T.H. Tse, Stephen S. Yau, W.K. Chan, Heng Lu, T.Y. Chen. Testing Context-Sensitive Middleware-Based Software Applications (2004). *Proceedings of the 28th Annual International Computer Software and Applications Conference (COMPSAC 2004)*

[11] Jeongeun Choi, Byoundju Choi. Test Agent System Design. *Proceedings of 1999 IEEE International Fuzzy System Conference Proceedings*.1999.

[12] Yu Qi, David Kung, Eric Wong. An Agent-based Testing Approach for Web Applications. *Proceedings of the 29th Annual International Computer Software and Application Conference*, pp1-3, 2005.

[13] Gu Le and Shi Jiulin. *Introduction to Software Testing Technology*. Tsinghua Press. Reading, China, 2004.

[14] Ricky Junday, Sanjeet Matharu. Symbian Signed Test Criteria 2.11.0. Symbian Ltd. https://www.symbiansigned.com

[15] Forum Nokia. Nokia Test Criteria for Symbian C++ Applications V1.6. http://www.forum.nokia.com

[16] Forum Nokia. Nokia Test Criteria for Java™ ME Applications V 1.0. http://www.forum.nokia.com

[17] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (Second Edition)*.Prentice Hall. USA, 2002.

[18] YDC023-2003 800MHz CDMA 1X *Digital Cellular Mobile Telecommunication Device Test Method: Mobile Station Part One*, China Communication Industry Standard