Development of Sensor Based Applications for the Android Platform: an Approach Based on Realistic Simulation

Pablo Campillo-Sanchez[a], Juan A. Botía[b], Jorge J. Gómez-Sanz[a]

[a] Software Engineering and Artificial Intelligence, Universidad Complutense de Madrid

[b] Information Engineering and Communication Department, Universidad de Murcia

KEYWORD

Context-aware services
Agent based social simulation
Testing
Mobile Devices
Android

ABSTRACT

Smart phones are equipped with a wide range of sensors (such as GPS, light, accelerometer, gyroscope, etc.) and allow users to be connected everywhere. These characteristics offer a rich information source for creating context-aware applications. However, testing these applications in the lab, before their deployment, could become a hard task or impossible because of sensors correlation, too wide testing area or an excessive number of people involved. This work aims to solve these problems carrying out the testing in a simulator, simulating the world in which the application user is immersed into. Tester controls her avatar and the avatar has a simulated smart phone that is connected with the user's smart phone. Applications under test are installed on the real smart phone and are compiled with a library that replaces standard services of the sensors by others that offer data sensor from the simulator (depending on the simulated smart phone context) instead of real world.

1 Introduction

Mobile applications have experimented a new revolution in the last years. And such revolution has been pushed by two related but, in principle, contrary forces. They are two different operating systems, iOS and Android. iOS is a closed operating system which is devoted to the mobile devices manufactured by Apple. Thus, the possibilities for developing in such applications are imposed by the Apple policies for open applications development. Android is the operating system designed by Google. And it follows a radically different philosophy of development: open source and Java based development. Moreover, Android is the second most used operating system for smartphones, it is more popular than BlackBerry and iOS, but it is expected that it will overtake to the number one, the Symbian OS by Nokia [GARTNER, 2010]. So, the mobile software development study is centered on Android mobile applications because the most users are benefited and the group follows an open source philosophy.

Recently, Smart phones are equipped with a set of sensors, such as GPS sensor, accelerometer senor, gyroscope sensor, camera, microphone and etc. All this hardware allows to get context information about the user is involved into, such as date and time, location, activity. This information is used to develop context-aware applications that offer services to users depending on their needs. Despite the promising potential of using mobile phones as context source devices to make context-aware applications, some problems emerge that need to be solved.

One of the central problems on context-aware application development is verification and validation of such applications by testing. Testing software is the process of executing a program in order to find

errors in the code [MEIER, R. 2010]. Such errors must then be debugged. According to the IEEE Standard Glossary of Software Engineering Terminology [IEEE 1990]: "Validation is the process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirement".

The main challenge of testing is to generate a set of values for each sensor, useful and meaningful for each test. The problem increases with the number of sensors and their correlation. For example, suppose a mobile service that offers indoor location information to others based-location applications like in this work [CHON, J. *et al*. 2011]. The service uses the digital compass to get orientation changes and the accelerometer to deduce displacements. To test the service we could generate several sceneries defined by a sequence of values, accelerations for accelerometer and radians for digital compass. Given an initial location and the sensor's values, the service estimates a final location. But it could be easier and more useful if the values are generated indirectly by moving a user and his smart phone in a simulated environment. In this way, the test set could be formed by a list of rooms a user has to visit. Other way could be to define an autonomous user behavior that uses a phone in his natural environment. Anyway, the last two options to define tests are much more natural, comprehensible and realistic than the first one that directly defines a displacement as a sequence of acceleration and radian values.

This work is focused on testing of applications or services for smart phones through a simulator where the environment and interactions are modeled. Considering an Android service or application as the system under test (SUT), some of the errors may be found by using a Unit approach [OSHEROVE, R., 2009]. Concretely, UbikSim [UBIKSIM, 2013] is used as simulator. It has been designed to simulate environments, devices and people interacting with real ubiquitous software [CAMPUZANO, F. *et al*. 2011]. It is already been focused as an ubiquitous computing environments simulator which tries to alleviate the particularities of testing services and applications whose behavior depends on both physical environment and users. Moreover, UbikSim offers a world editor that offers an easy way to create environments and the agents that interact in it.

Using the simulator, not only a sequence of values can be simulated, but it is generated indirectly through defining concrete simulated environments from the reality. It offers two main advantages: (1) software testing is not defined by the component level as a sequence of sensor values, but from stress concrete situations in the virtual world that are more natural and realistic. (2) A graphical representation of the situations means that a set of final users can understand and, therefore, they can validate the application behavior more easily while the testing process is performed simultaneously. So, since the user is getting involved in early stages, we achieve a user centering development philosophy in a natural way.

In this paper the contents are exposed as follows. Section 2 is related with the challenges. At the next section 3, it is exposed a complex example application to test. Section 4 covers SUT testing by simulation. And, finally, the conclusions and future work are treated.

2 Challenges

The most common Android development tools [MEIER, R., 2010] are composed by SDK (Software Development Kit) and ADT plug-in (Android Development Tools), both supported by Google and open source. The SDK includes the Android APIs (Application Program Interface), development tools and the Android Virtual Device Manager and Emulator. ADT plugin extends the SDK functionalities to an IDE (Integrated Development Environment).

The Android software stack is composed by several layers. The Application Framework Layer is the most important. It provides the classes used to create Android applications and a generic abstraction for hardware access and manages the user interface and application resources. The following subsections contain some Android services that are offered by classes of such layer. They can be used as a source of context for developing context-aware applications. Also, it is explained why the existing testing tools are insufficient to test each application based on such services, and even more if we want to simultaneously test an application based on a correlated set of those services.

2.1 Location-based SUT

Mainly on a smart phone, location services are obtained through CellID or GPS. The second one is the

preferred method as it is more precise but it only works outdoor. CellID approximates your location based on the urban cell you are in, and this could employ too many meters, but it works indoor also. Location services on Android are obtained through the *LocationManager* class.

An interesting facility on Android is a hook which allows the programmer to simulate in the emulator, different locations a user passes through, when debugging the application. In this way, the user is not moving, but the emulator virtually does. But the test is not realistic because consists in a file with coordinates and the simulation reproduces constant velocity and straight line displacements. It could be more realistic if the coordinates were given by simulated person displacements in his natural environment. Furthermore, a based-location service including inertial devices could not be tested by this tool.

2.2 Sensor-based SUT

Sensors on Android are managed in a similar way that location, through a *SensorManager* class which is the one giving access to all the sensors of the phone. Exists a wide list of the sensor-types currently available; note that the hardware on the host device determines which of these sensors are available to the SUT.

There are third-party tools which help working with sensors on Android. For example, the Sensor Simulator [SENSIM, 2013] is a stand-alone application of OpenIntents and it lets simulate moving the mobile and the corresponding sensors by only moving the mouse.

Another, Samsung Sensor Simulator [SAMSENSIM, 2013] lets simulate the registers of sensors, obtained by a simulated mobile. It also lets connect to a real device to log real registers from it. But again, those tools are complicated either to manage or to generate the sceneries composed by incompressible sequences of values.

2.3 Audio and Video-based SUT

Audio and video are another source for context sensing. They are interesting by means of their processing, e.g. image processing to detect objects or recognize commands by speech. Android offers this type of services that support data processing through *Camera* and *AudioRecord* classes for video and image processing, respectively.

The emulator allows using a microphone and a webcam to test applications that use those resources. But, it is not considered a tool to feed the SUT with artificial images and sounds that defines scenery for testing.

3 A hard to test SUT example

In this section is studied a type of application which is complex to test. It uses an indoor location-based service (ILBS) to develop an augmented reality (AR). It is conceived to make museum tours more attractive and educational by locating POIs (Point Of Interesting). AR applications need to know the location and orientation of the phone in order to show the POIs on the screen. There are no problems outdoor because the GPS give us the location, but it is not as easy at indoor. In fact, a lot of techniques which try solving this problem are based on different technologies (WiFi, Bluetooth, ultrasound, inertial sensors) or a mix.

There exist different variants of AR, this paper defines it as a term for a live indirect view (through mobile screen) of a physical real-world environment whose elements are augmented by virtual phone-generated POI icons - creating a mixed reality. The augmentation is in real-time and in semantic context with environmental elements.

In order to show the POI icons, each one has a coordinate location and the smart phone gets its own location and orientation from indoor location system. A practical positioning and tracking solution for users in indoor environments relies on both an accelerometer and a digital compass. When a user starts to move, classification data acquired from both sensors are used to approximate the user's location. But the mechanism is needed to get an initial position and to solve accumulated sensor errors. So, several QR (Quick Response) codes, with location information, are distributed in the museum.

The AR application is developed for the Museo Arquelógico de Murcia (MAM) [MAM, 2013]. The museum visitors download it and they can browsing through MAM identifying POI, e.g. archeological pieces, next exhibition hall, toilets, etc. So, a user can identify POIs around him with his smart phone and gets information about them pushing on each POI icon or reaches them physically.

Usually, testing stages are divided in several tasks depending on modular functionality of the SUTs. For this example: (1) friendly graphic user interface (GUI), (2) read QR codes with smart phone camera, (3) the correct QR codes content depending their location, (4) the right location of the POIs and their content, (5) the error of the predictions of the indoor location system based on both an accelerometer and a digital compass and (6) the location of the QR codes depending the error of the predictions. Given these tasks, the fifth is the harder to test. Due to the correlation of sensor values, accelerations and angles. And this type of tests is hard to generate and manage with the actual tools available, as we have seen.

To test indoor AR applications it is used a pilot test that is formed a set test in a real environment, a museum in this case. But it is expensive due to it requires to deploy the infrastructure (QR codes, internet access), probably at least an exhibition hall must be closed, time and money to manage and coordinate people. By its cost, a pilot test is usually performed at the end of the development process. At the same time, it implies more costs because the detected errors are more expensive to resolve in this stage than in early ones.

4 SUT testing by simulation

Context-aware SUTs are harder to test in a lab as the use of sensor values are more correlated. A simulation-based testing is proposed where environment and its elements related (people and devices included) with the SUT are modeled and simulated. So, first a model of the world and the related elements have to be created in order to the SUT could be tested using a smart phone and the simulator.

4.1 Modeling Elements

In this stage, the simulated world, where the SUT of smart phone will be involved, is modeled. It includes: environment, people and devices. The world is modeled using an UbikSim editor. **¡Error! No se encuentra el origen de la referencia.** shows the editor and a model of an exhibition hall of the MAM. This tool offers an easy way to create environments by dragging elements from the catalog (panel located at top-left) to the panel of edition. It already has some elements but we can create new ones quickly. In addition, a 3D view of the model is available on the bottom panel.

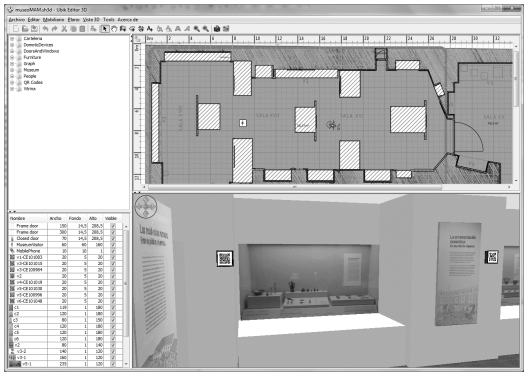

Fig. 1 MAM Exhibition hall modeled with UbikSim.

Each world model represents a test configuration. So, first a basic environment is created (e.g., composed by exhibition hall, furniture and pieces of art) and then, others more complex to test specific functionalities. For instance, to test the indoor location system, it is shown the predicted tracking on screen to check if the error gotten is acceptable. Other sceneries could be composed by different locations of the QR tags.

4.2 Testing process

In this stage, a user or developer tests the SUT installed in a smart phone (or emulator) in a simulated world where the user interacts using a keyboard and a mouse. UbikSim is used as simulator and it has several main features [GARCIA-VALVERDE, T. *et al*. 2009]. UbikSim offers basic models for physical environments (e.g. offices building floors), for humans (e.g. professors in universities) and for sensors (e.g. presence, pressure and open door sensors) that are already developed and validated.

Fig. 2 shows the main schema of the elements and their interactions needed to test the SUT. At left side, it is the simulator and it includes simulated smart phone (SSP) and simulated user (SU). SU carries SSP and it has simulated sensors that register context information from simulated environment (e.g. simulated temperature sensor registers ambient temperature) and SU actions (e.g. simulated accelerometer registers user displacements). At the right side, we find the real elements. A real user (RU) tests the SUT installed in a real smart phone (RSP) that receives sen-

sor values from the simulated environment through SSP like it comes from the real world. RU interacts with the simulated environment with the keyboard and mouse like a computer game such as Counter Strike [COUNTER, 2013].

The example exposed in section 3 will be used in order to illustrate how to test an application. It supposed that the simulator represents the predicted location by RSP and the SUT is completely developed. So, the SUT is installed on the RSP. Also, complete scenery is already created and available to be simulated.

Once the simulator is started with the scenery and the SUT is installed on the RSP, the test can be begun by the RU. To start, RU could to move his SU until a QR tag using the keyboard, then RU activates the QR reader from his RSP to decode the tag. In order to perform this task, the SUT needs to get images from the camera, instead it receives images (containing the QR tag if it is focused) that are displayed on the PC screen by simulator. The tag is identified and processed to get the location information, after this, it is displayed in the simulator. By this way, the user tests easily if the QR tag contains a correct location.

Once the SUT gets its position, the RU can activate the AR from the RSP to identify POIs. RU sees them on RSP screen and can test how the POI icons change by rotating his SU using keyboard. Therefore, RU can check if POI icons are correctly displayed in our RSP screen and also, RU can review the content of a POI pushing its icon.

Finally, by moving SU and consequently its attached SSP that sends simulated acceleration and orientation changes to the RSP. RSP tries to predict the SSP location from those simulated values and, at the same time, RU checks how varies the predictions on the screen. In addition, RU can check if it affects to AR too much depending on if the POIs are located more or less correctly on the RSP screen.

As we have seen, UbikSim contributes to test by means of the displays. The simulation displays are very useful to observe that the application behavior is appropriate. UbikSim works on MASON [MASON, 2013] and can use its features as, for example, inspectors. They are a means to graphically visualize the evolution of variables of interest for the simulation. A large number of inspectors for various simulation variables can be used and monitored dynamically as the simulation evolves. They can be used to check that such variables take always reasonable values, such as estimated locations.

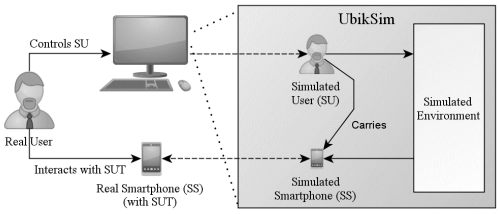

Fig. 2 Proposal of interactions within a mobile applications testing scenario based on simulation.

5 Conclusions

Currently, existing tools are insufficient to test context-aware applications that make extensive use of their sensors whose values have a correlation. This paper proposes an approach to test this kind of applications using simulation. The approach simulates the whole environment where the application will be deployed, such as physical space, people or sensors. In contrast, both the SUT and the smart phone where the SUT is installed are real. Therefore, the user interaction with the SUT is realistic, giving a real experience that is fundamental for validation.

This work is a contribution to the engineering process of smart phone application in general and AmI systems in particular. It has three main advantages. (1) The software testing is not defined by the component level as a sequence of sensor values, but from stress concrete situations in the virtual world that are more natural and realistic. (2) A graphical representation of the situations means that a set of final users can understand and, therefore, they can validate the application behavior more easily while the testing process is performed simultaneously. (3) Some tests can be performed in early stage of software development process because a pilot test is not needed as it was needed before. When an error is detected early, it is easier and cheaper to fix. (4) Developers neither need to learn any new application programming interface (API) nor change source code of the application.

Future works include a deep study about defining a graphical modeling language that allows users to specify how a system (included smart phone) should react against defined situations. The language will be defined for a specific domain, Parkinson Disease. Doctors, caregivers and relatives have to be able to use the notation. Therefore, it will have to be simple

enough. Finally, these models will be translated automatically in a simulation in order to users can validate the model defined themselves.

6 Acknowledgment

This work has been supported by the Spanish Ministry of Science and Innovation in the scope of the Research Projects TIN2011-28335-C02-01 and TIN2011-28335-C02-02. Facultad de Informática, Campus Universitario de Espinardo, 30100 Murcia, Spain.

7 References

[BADGETT, T. *et al*. 2004] BADGETT, T., MYERS, G.J., SANDLER, C., and THOMAS, T.M. *The art of Software Testing*. Wiley, 2nd edition, 2004.

[CAMPUZANO, F. *et al*. 2011] CAMPUZANO, F., GARCIA-VALVERDE, T., GARCIA-SOLA, A., and BOTIA, J. *Flexible simulation of ubiquitous computing environments*. In Ambient Intelligence – Software and Applications, volume 92 of Advantages in Intelligent and Soft Computing, Springer, 2011, Berlin / Heidelberg, pp. 189-196.

[CHON, J. *et al*. 2011] CHON, J., and CHA, H *Lifemap: A smartphone-based context provider for location-based services*. IEEE Pervasive Computing, volume 10, 2011, pp. 58-67.

[COUNTER, 2013] Counter Strike website: http://www.counter-strike.net, last accessed March 1, 2013.

[GARCIA-VALVERDE, T. *et al*. 2009] GARCIA-VALVERDE, T., SERRANO, E., BOTIA, J., GOMEZ-SKARMETA, A., CADENAS, J.M. *Artificial societies immersed in an ambient intelligence environment*, in: Workshop W31 Social Simulation of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, 2009.

[GARTNER, 2010] GARTNER Corporation. *Gartner says worldwide mobile phone sales grew 35 percent in third quarter 2010; smartphone sales increased 96 percent*. http://www.gartner.com/it/page.jsp?id=1466313, Nov 2010.

[IEEE, 1990] I.O Electrical and E. E. IEEE 90: *IEEE standard glossary of software engineering terminology*, 1990,
Artificial Intelligence and Application, J&J Editors, 2012. Spain

[MAM, 2013] MAM website: http://www.murciaturistica.es/museos/museos.inicio?museo=museo-arqueol\%F3gico-de-murcia-(mam)\&id=1, last accessed March 1, 2013.

[MASON, 2013] MASON website: http://cs.gmu.edu/~eclab/projects/mason/, last accessed March 1, 2013.

[MEIER, R., 2010] MEIER, R. *Professional Android 2 Application Development*. Wrox Press Ltd., Birmingham, UK, 1st edition, 2010.

[OSHEROVE, R., 2009] OSHEROVE, R. *The art of Unit Testing: With Examples in .Net*. Manning Publications Co., Greenwich, CT, USA, 1st edition, 2009.

[SAMSENSIM, 2013] SAMSUNG Sensor Simulator website: http://developer.samsung.com/android/tools-sdks/Samsung-Sensor-Simulator, last accessed March 1, 2013.

[SENSIM, 2013] SENSORSIMULATOR website: http://code.google.com/p/openintents/wiki/SensorSimulator, last accessed March 1, 2013.

[UBIKSIM, 2013] UbikSim website: http://ubiksim.sourceforge.net, last accessed March 1, 2013.