A DSML for Mobile Phone Applications Testing

Youssef Ridene
LIUPPA, University of Pau
Avenue de l'université
64013 Pau, France
youssef.ridene@univ-pau.fr

Nicolas Belloir
LIUPPA, University of Pau
Avenue de l'université
64013 Pau, France
nicolas.belloir@univ-pau.fr

Franck Barbier
LIUPPA, University of Pau
Avenue de l'université
64013 Pau, France
franck.barbier@univ-pau.fr

Nadine Couture
ESTIA, LaBRI, University of Bordeaux
Technopole Izarbel
64210 Bidart, France
n.couture@estia.fr

ABSTRACT

Model-Driven Testing (MDT) is a relevant approach for the automation of software testing. This approach uses models to express and execute tests. These models are instances of metamodels describing a dedicated Domain-Specific Modeling Language (DSML). In this paper, the targeted domain is mobile phone applications. In fact, mobile devices have various hardware and software features like operating systems, screen parameters (size, touch screen or not...), keyboards (QWERTY, AZERTY...), presence of additional hardware (camera, voice recorder, accelerometers...), etc. In this paper, we describe an industrial platform (test bed) that includes an Eclipse-based DSML. A key characteristic of this DSML is its ability to cope with variability in the spirit of software product lines. We discuss this DSML as a component of a tool suite enabling the test of different devices having variable features.

Categories and Subject Descriptors

D.2.5 [**Testing and Debugging**]: Testing tools D.2.6 [**Software Engineering**]: Programming Environments – graphical environments

General Terms

Design, Languages, Verification.

Keywords

DSML, Model-Driven Testing, Embedded Mobile Applications, Software Product Lines, Variability.

1. INTRODUCTION

The "Internet of Things" refers to a network of objects in which each physical "thing" is a communicating device [1]. One of the greatest challenges behind this idea is that developers should write applications which can be deployed on a huge range of potentially different target devices. For instance, let's consider a fingerprint reader program that would be deployed on a laptop, on a tablet PC and on a mobile phone. The same program must properly run in each environment. However, the issue is that such systems do not have the same technical characteristics. Software applications must be adapted to target devices implying specific code: a part of the program would be common while several other parts must be adapted. Typically, each system provides its own graphical API, meaning that the fingerprint reader program must have specific code for each one. This generates a significant effort not only in terms of development but also in the verification and validation stages.

Although problems may be considered as solved with Java virtual machines, reality shows that unanticipated bugs occur in situations ("in situ"), *i.e.*, when applications are deployed in actual mobile phones. For instance, networks capabilities of each telecom operator may create non homogeneous behaviors. Thus, adaptations for the deployed software are expected accordingly. In short, reality demonstrates that the "write once, run everywhere" slogan is not true even for Java.

Over the last months, mobile phones have considerably evolved to become more and more sophisticated with increasing capabilities. The myriad of devices, operating systems, platforms and, above all, telecom operators' requirements have put forward the "mobile device fragmentation" syndrome [2]. Software versions must be tested in strong relation with real handsets so that they achieve the "expected behavior". In fact, testing is an increasing challenge for software providers due to the cost induced by buying hundreds of devices and carrying out large-scale test activities to validate each version of the application on each handset.

In this line of reasoning, this paper discusses "in situ" testing (see boldface line in table 1). We constructed a physical platform (test bed) and its associated middleware to share real devices and to carry out testing activities remotely using Internet.

Table 1 Paper's concerns and positioning

	Concern	*Actual or to-be-invented support*	*Research area*
	Application development based on standards like Java ME, iOS, Android, WebOS, Symbian OS, etc.	Emulators	Classical testing approaches
Paper's contribution	**Software product line engineering, device variability and heterogeneity management**	**Test beds**	***In situ* testing**
Paper's perspectives	Usages, nomadism, ubiquitous even "lost" software preventing traditional maintenance tasks and requiring continuous adaptation and self-adaptation	Built-in testing, components and applications management, self-managing systems	*In situ* testing, autonomic computing, software adaptation

In this paper, we propose a DSML resembling to UML Sequence Diagrams. It allows the description of test scenarios in which commonalities and variability between mobile phones can be expressed without great difficulties. Test scenarios use distinct execution paths according to the expressed variation points.

We look for a significant reuse of test material; the sharing of test data sets and scenarios across devices; the replay and tracing of tests in general to compare commonalities and variability across component/application behaviors.

To address and illustrate these issues, the paper is structured as follows: section 2 presents the importance and the specificities of mobile application testing due to device fragmentation. In section 3, we develop a vision that stresses Model-Driven Testing as a crucial technique. An industrial platform (test bed) and its associated control middleware developed by Neomades [3] are described. We explain the role, the positioning and the benefits of a DSML in this professional testing environment. In section 4, we sketch perspectives while section 5 serves as a conclusion.

2. MOBILE APPLICATIONS

Many projects were developed to bring a solution to device fragmentation. These frameworks rely on cross-compilers or virtual machines to generate adapted source code for targeted platforms. Large data sets about manufacturer specifications on all market devices (screen size, key codes, known bugs, supported multimedia codecs...) [4-5] are inputs of these tools. Such information is used at programming time to generate different binary files according to each mobile phone capabilities. For example, in the Netbeans IDE, such a configuration approach is managed with pre-processing directives. Although configuration actions are widely used within the development stage, this does not guarantee at all that released applications will have the expected behavior once deployed on hundreds of different mobile devices.

As described in Table 1, classical testing made by using emulators does not lead to first-class confidence and assurance about application behaviors. Residual bugs or anomalies come from actual execution conditions that cannot be setup as configuration data at development and "traditional" testing stages.

2.1 Context of mobile applications testing

Today, mobile application developers face a multitude of challenges due to the increased complexity of testing across different handsets, carriers, languages and locations. This is not only a technical problem. This is also a commercial problem: Ensuring a high-quality and successful user experience is crucial to the success of mobile applications [6]. Because of this, mobile companies need to be constantly aware of how their products behave once an application has been deployed on a given product. With mobile device manufacturers constantly invent and sell new devices, mobile applications testing must occur to allow developers to make necessary changes so that their applications work properly (for the newest devices especially). Each developed application needs to be tested on its target mobile phone before it can be accessible to final users. Thus, specialized skilled teams test every application on hundreds of devices.

2.2 Specificity of mobile applications testing

Even if a significant part of the program validation can be done during the development stage thanks to emulators like the Java ME Wireless Toolkit or the Android Software Development Kit, mobile application testing cannot be only done on desktop machines. In fact, emulators do not support all devices and platform extensions. For this reason, final tests must be carried out on real devices connected to live networks. For example, communication protocols like HTTPS cannot work properly on some mobile phones for various reasons. The only way to address this issue is to test the application using HTTPS and debug it at runtime on real handsets.

In [7], we may find detailed test categories. In general, mobile software testing activities aim to control different points, for example:

- Application launching: Once an application is loaded on the device, it must start and stop correctly and behaves as expected. The application must be stable on the device, *i.e.*, no frozen state, no exception, etc.
- Usability: User interfaces must be user-friendly independently of strict screen features. Other issues to address are for instance the fact that all texts must be free from spelling errors.
- Connectivity: If the application has over the air (OTA) communication capabilities then it must handle this correctly, *i.e.*, it must deal with network and server problems in a robust way.

2.3 Existing solutions and limitations

Mobile application validation is mainly based on actions performed by users (pressing a key or clicking on a screen to check navigability…). It also relies on visual and sonorous interpretations (check spelling, check layouts, check if music plays correctly…). The number of actions and their properties may change from a mobile to another (the key to be pressed, the area to click on...). Nowadays, the most used solution to reduce the cost of tests is to delegate this activity to low-cost offshore companies. For externalized projects, the main problem is the communication between developers and testers [8]. Another issue relates to network characteristics. In fact, some mobile applications can work properly on some public networks and cannot do so on others because of technical restrictions like blocked ports, bandwidth limitations, etc.

Some industrial solutions were developed to have a new way to test mobile applications. The most achieved project ("Device Anywhere") [9] is as follows: The tester may register a sequence of actions by performing them on a specific device. A script based on a proprietary language is then generated. The user can run it only on a similar mobile phone for the reasons we mentioned in section 1: Each mobile has specific hardware and software characteristics. For example, if the user registers a test on a mobile phone using left and right key validation, such scenario will not work on a touch screen device since this kind of mobile phone do not possess such keys.

The two major flaws of this solution are the use of a non-open language (far from a standard especially) and the absence of commonality/variability and genericity of scenarios across devices. The need for sharing scenarios between devices is indeed high with languages that are not so different from popular graphical languages like UML.

3. PROPOSITION

In the industrial sector, the process of testing is repetitive and annoying since testers have to carry out exactly the same procedure on each mobile and to manually report the results.

The proposed DSML is plugged into Eclipse on the top of a platform (software + hardware) enabling a connectivity between instances of the DSML's model elements on one side and devices on the other side.

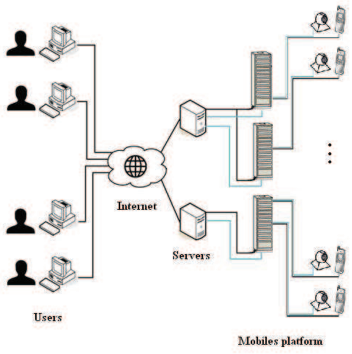

Figure 1 Global architecture of the platform

3.1 Platform

Figure 1 shows the global architecture of the solution we developed. The idea is based on a dedicated platform with real devices connected to Web servers. A video camera is placed in front of each device to capture the real user experience as displayed on the handset's screen. Hands-free sensors are used to capture audio outputs. Keys are electrically wired (as shown in Figure 2) in order to emulate inputs. We also developed a middleware to allow users to remotely connect with the servers and operate on a mobile phone from the installed devices.

A database was designed and filled in with all the relevant information about each connected phone. This database contains screen properties (sizes, touch screen or not…) keyboard mappings, available inputs and outputs, sensors (landscape mode, rotation…), navigation menus, etc.

Once the connection is established, the desktop application displays the layout of the real mobile phone and plays received video and audio data which are transferred in real-time from the remote handset. The user is also able to send inputs (keypad or touch screen) to the mobile phone using the mouse and the keyboard of her/his computer.

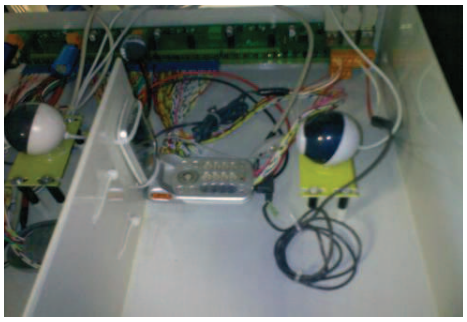

Figure 2 Sony Ericsson W300I connected to the platform

Mobile phones do not provide hardware or software interfaces to test applications behavior once deployed. It means that we cannot use software to simulate an event on the mobile phone (press a key, click on the screen…). For this reason such events are electrically "realized" (see above). Furthermore, phones may actually be connected with telecom operator networks to analyze at test time more realistic incoming events from these networks: incoming calls, SMS, MMS, etc.

3.2 The Domain-Specific Modeling Language

By providing this platform, we avoid mobile developers to buy all released handsets. In order to resolve testing repetitiveness and annoyance, a specific language (Figure 3) helps the tester to describe a test scenario and run it on available devices (Figure 4). Model-Driven Testing fosters quality of software in general [10-11-12]. Furthermore, using a DSML significantly facilitates test scenario design because testing is based on model elements closely related to the field of mobile phone concepts and does not need any extra programming skills. In fact, the DSML provides domain elements (mobiles, tester and mainly interactions between both of them: press keys, press pointer, rotate…) to let users describe expressive tests, concentrate on the test design and no more repeating it on hundreds of devices. A test scenario is typically a suite of actions performed by the tester on the phone: downloading an application, installing it, launching it, navigating in menus, validating user permission requests...

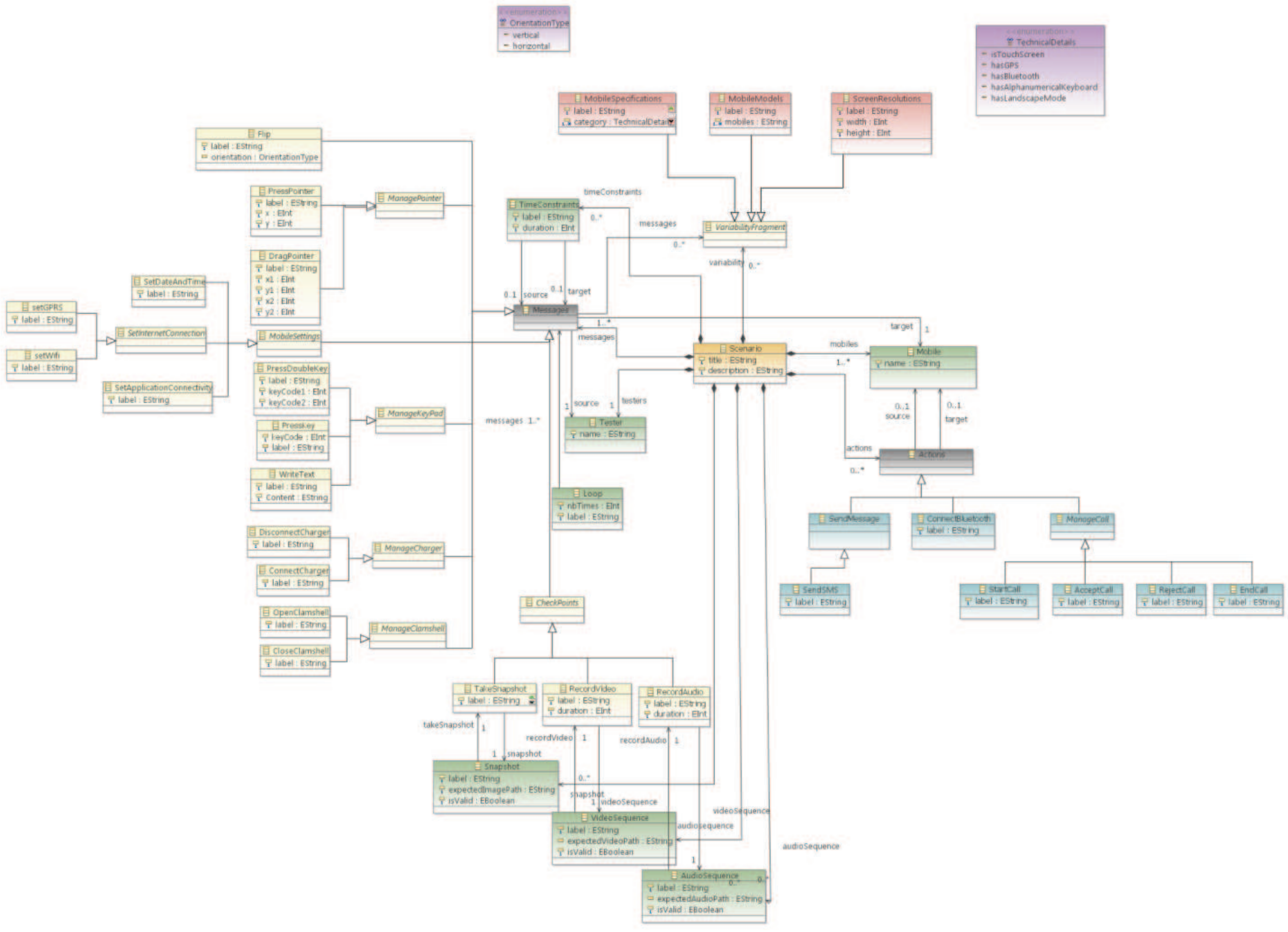

Figure 3 – The DSML's Metamodel

As evoked in section 1, our DSML is inspired from UML Sequence Diagrams. At this time, the DSML does not support all the UML notation adornments (*i.e.*, scenario operators like *alt* for alternative, *opt* for optional, etc.) but this work is in progress. Otherwise, the DSML and its associated editor provide several levels of test design which is made just by drag and drop of available elements of a model.

3.2.1 Basic domain elements

For a simple test scenario, a tester, a mobile and actions performed on the device are mandatorily required. The DSML provides an extended list of possible actions according to mobile phones capability (pressing on a key, touching the screen, dragging the pointer, etc.). In practice, a scenario contains at least two lifelines and at most three:

- A tester: this is a mandatory lifeline (no more than a tester in a scenario) which designates the user who performs actions on the mobile phones. The DSML provides a list with all the possible actions that can be performed. This is fully similar to the environment of a user with its preferred phone. These actions can be simple, e.g., pressing a key or combined, e.g., writing text.
- A first mobile: this is a mandatory lifeline which designates the mobile(s) under test.
- A second mobile: this is an optional lifeline which can be used to simulate interruptions on the mobile under test (SMS, incoming calls, Bluetooth connections…).

The DSML offers the *Message* metatype as a key interaction tool to express how the application is manipulated (and therefore tested) through external stimuli (users' inputs, SMS arrivals…). *Message* instances have properties which must be set by the user thanks to the DSML editor. For example, if the tester chooses the "pressKey" action, he/she must accurately define the key to be pressed.

With the help of the database, while executing the scenario, the test bed performs the right through the homemade middleware services.

3.2.2 Variability management

In the spirit of Software Product Line engineering: three different types of variability management are defined:

- The first one is defined according to the mobile model, i.e., Sony Ericsson W810, BlackBerry 8900, etc.
- The second one relies on the mobile's specifications. The DSML editor provides a predefined list (isTouchScreen, hasGPS, hasKeyboard…) from which the user can specify one or more specification type.
- The third one is dedicated to screen resolutions. The user can specify the resolution he/she wants. This is done by providing the width and the height of the mobile's screen.

These three types of variability come from specialists in mobile applications development. According to them, an application may be adapted from a mobile to another mainly due to these variations. In practice, the user is able to design specific actions according to these variability points.
We added these concepts, to the DSML, as metatypes (*MobileSpecifications*, *MobileModels* and *ScreenResolutions*) which inherit from a generic metatype called *VariabilityFragment.* The user can directly manipulate these concepts in a testing scenario. One or more messages can be assigned to a *VariabilityFragment.* For example, in Figure 4, we see a variability called TouchScreen (blue rectangle) to which we assigned a "pressPointer" action. This variability is an instance of the *MobileSpecifications* metatype set to the "isTouchScreen" value. With the help of the database, the test bed performs this action only on touch screen-enabled mobiles.

Thanks to these DSML's features, the tester has to design only one testing scenario for all the targeted handsets. Moreover, this scenario can be adapted for similar applications.

3.2.3 Interruptions

To test how applications behave when the phone receives an incoming call or an incoming message, the "send SMS" or "call" features let users triggering automated incoming events. By using a third lifeline in the testing scenario (Figure 4), the mobile under test can be tested in reaction to external interruptions.

3.2.4 Automated actions

In order to facilitate the test design and provide more high level components ready for use by the tester, we decided to automate some actions. For example, entering a text in a form is a suite of the "pressKey" action. So, we decided to provide an action called "writeText" which has just one property, i.e., the text to be typed. The test bed generates the right suite of the "pressKey" actions which differs from a device to another. The DSML provides a list of automated actions such as sending SMS or MMS, writing text, starting the application, setting date and time, etc.

We also added a *Loop* metatype which allows the user to assign a loop to one or more actions just by providing the number of times he wants to perform these actions. For example, sometimes, the tester has to press a specific key many times. So, instead of adding a lot of messages in the scenario, using the *Loop* metatype is only required.

3.2.5 Checkpoints and results validation

The DSML provides functions to monitor applications under test. In fact, it is possible to take screen snapshots, to capture video and audio clips from the actual device while operating. Three special *Message* subtypes are defined for that: "takeSnapshot", "recordAudio" and "recordVideo". With these facilities, the tester may want to see what happens after some actions, so he/she takes a snapshot of the screen (action 2 in Figure 4). Each checkpoint must be assigned to an instance of a *Result* metatype which can be chosen between *Snapshot*, *VideoSequence* or *AudioSequence*.

It is possible for the test designer to specify expected results linked to fixed checkpoints, *i.e.*, selecting the snapshot the tester has to wait for. The test bed returns the computed image once the scenario execution is finished. The DSML editor then displays the real and expected results so that the tester can validate or not test results (no automatic comparison occurs at this time).

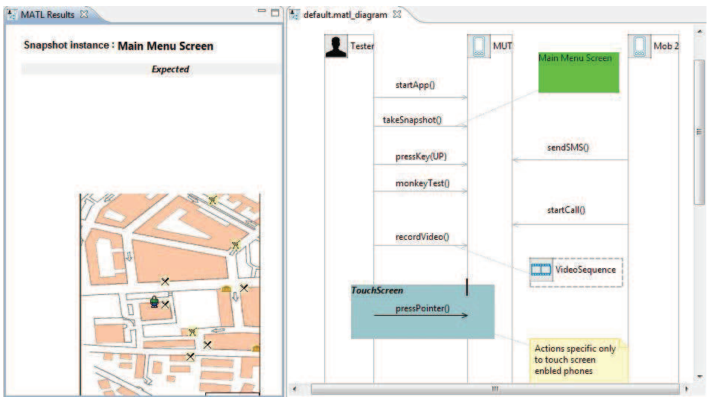

Figure 4 Scenario example

We specified and implemented our DSML on the test bed in order to provide an end-to-end tool suite for specifying and running tests on the platform. We used Eclipse that provides adequate tools for metamodeling and for designing DSMLs. We especially used the Eclipse Modeling Framework (EMF) to design the language. We also used the Graphical Modeling Framework (GMF), an Eclipse component to add a visual representation to the DSML and to generate a complete tool based on an Eclipse Rich Client Platform.

4. PERSPECTIVES

The ongoing investigations presented here are well-fitted for black-box testing. One of the limitations of this kind of test is the difficulty to precisely evaluate more deep features. For instance, it is not possible to determine the state of the application in which an anomaly or an error occurred. We identified two main problems to be addressed in order to enable a better interpretation of an occurring anomaly/error: (i) a fairly general lack of introspection capabilities of mobile applications and (ii) a difficulty to materialize the link between the application's specification and its tuned implementation. The latter does not always fully and/or properly conforms to the specification.

We propose to improve this platform by exploring two directions. We are on purpose working on an enhanced version to add to mobile applications Built-In Test (BIT) capabilities. BIT technology [13] was developed in the Component+ European project. It allows the addition of dedicated testing and configuration interfaces to component-based applications. The main idea is to embed in the application testing code that remains at runtime (this is equivalent to the idea of managed code promoted by the Microsoft .NET platform for example).

We also consider the change of the programming approach (ii) by incorporating models (*i.e.*, models@runtime [14]) into the mobile application. The aim is to decrease the risk of adding bugs during the implementation of the application: code is an exact mapping of the model using libraries like State Chart XML (SCXML)[1] or PauWare[2]. This enables autonomic capabilities [15] in the applications under test; this also allows the remote management of the component/application behavior at runtime. Benefits of this approach coupled with BIT capabilities are to manage and to run tests with a higher control and confidence of the tested application.

5. CONCLUSION

We have presented in this paper the issues related to wireless applications testing due to device fragmentation. A dedicated platform allowing remote access to real handsets is in a final phase of development. Due to its genericity, the proposed DSML can be easily plugged into any other industrial test bed. Testers are currently introducing the platform in their business test protocols and procedures. The DSML favors extensions of new test objects required by testers and their graphical rendering with a very customizable look & feel.

We observed very early adoption and adhesion about the integrated nature of the platform especially. Tests in emulators may be easily and seamlessly followed by equivalent tests for a given device.

We are more generally improving the DSML by creating more benchmarks. We accordingly obtain a smooth integration of the DSML in the industrial platform to offer a complete tool for specifying and running tests. Once the industrial version of the product fully finished and packaged, we intend to experiment and evaluate BIT and autonomic computing to move forward to a more sophisticated approach for mobile applications testing.

6. REFERENCES

[1] Craig W. Thompson, Smart Devices and Soft Controllers, IEEE Internet Computing, vol. 9, no. 1, pp. 82-85, Jan./Feb. 2005

[2] Sun Microsystems. J2ME Building Blocks for Mobile Devices. White Paper on KVM and the Connected, Limited Device Configuration (CLDC), 2000

[3] Neomades: http://www.neomades.com

[4] V. Agarwal, S. Goyal, S. Mittal, S. Mukherjea. MobiVine - A Middleware Layer to Handle Fragmentation of Platform Interfaces for Mobile Applications. IBM Research Report 09009, 2009

[5] Vander Alves , Ivan Cardim , Heitor Vital , Pedro Sampaio , Alexandre Damasceno , Paulo Borba , Geber Ramalho, Comparative Analysis of Porting Strategies in J2ME Games, Proceedings of the 21st IEEE International Conference on Software Maintenance, p.123-132, September 25-30, 2005

[6] Pei Hsia , David Kung , Chris Sell, Software Requirements and Acceptance Testing, Annals of Software Engineering, 3, p.291-317, 1997

[7] Unified Testing Criteria for Java Technology-based Applications for Mobile Devices, http://javaverified.com (2009)

[8] S. Overby. The Hidden Costs of Offshore Outsourcing, CIO Magazine, 2003

[9] Mobile complete: http://www.deviceanywhere.com/

[10] L. Apfelbaum, J. Doyle. Model Based Testing. Software Quality Week Conference, May, 1997

[11] Kärnä,J., Tolvanen,J-P., Kelly,S. Evaluating the Use of Domain-Specific Modeling in Practice, DSM, 2009

[12] Kelly, S. and Tolvanen, J-P.2008. Domain-Specific Modeling: enabling full code generation, John Wiley & Sons, ISBN 978-0-470-03666-2, 427p.

[13] F. Barbier and N. Belloir: Component Behavior Prediction and Monitoring through Built-In Test, proceedings of The 10th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems, Huntsville, USA, IEEE Computer Society Press, pp. 17-22, April 7-10, 2003

[14] Morin, B., Barais, O., Jézéquel, J.-M., Fleurey, F. and Solberg, A. 2009. Models@Run.time to Support Dynamic Adaptation. IEEE Computer 42(10), 44-51.

[15] J. Kephart and D. Chess: The Vision of Autonomic Computing, IEEE Computer 36(1), pp. 41-50, 2003

[1] SCXML: http://commons.apache.org/scxml/

[2] PauWare : www.PauWare.com/PauWare_software