Yet Another DSL for Cross-Platforms Mobile Development

Olivier Le Goaer[*]
LIUPPA, Université de Pau
64000 Pau, France
olivier.legoaer@univ-pau.fr

Sacha Waltham
LIUPPA, Université de Pau
64000 Pau, France
sacha.waltham@etud.univ-pau.fr

ABSTRACT

With the growing success of mobility, mobile platforms (iOS, Android, WindowsPhone, etc.) multiply, each requiring specific development skills. Given this situation, it becomes very difficult for software developers to duplicate their apps accordingly. Meanwhile, web-based applications have evolved to be "mobile-friendly" but it appears that this is not a silver bullet : the user experience and the overall quality is still better with native applications. Of course, cross-platform mobile development tools have emerged in recent years. This paper provides a survey of these tools and points out that a full-fledged language for mobile development is highly desirable. Consequently, we present a preliminary work on *X*MOB , a technology-neutral DSL intended to be cross-compiled to produce native code for a variety of platforms.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques; D.3.2 [**Programming Languages**]: Language Classifications—*Specialized application languages*

General Terms

Languages

Keywords

mobile, cross-platform, DSL, MDA

1. INTRODUCTION

We have definitively entered an era of mobility. Evidence of that is the phenomenal growth of mobile markets, in terms of both the number of devices sold (essentially SmartPhones and Tablets) and number of downloaded mobile applications. But this growth has been accompanied by a fragmentation of the mobile platform (or mobile OS), which seriously complicates the task of mobile apps developers. An application must be made available on as many platforms as possible in order to reach diverse user base. However the mobile platforms landscape is very quickly evolving : there are over ten different platforms (e.g., Android, iOS, Windows Phone, Blackberry, Firefox OS), and it is likely that in under a year a few of those will gradually disappear as others emerge. In addition, every platform comes with its own SDK and requires specific programming skills, in terms of both languages and API.

As such, this heterogeneity creates very important additionnal redevelopment costs for software developers, who are thus compelled to focus their skills on just a few platforms, usually only two or three. It is worth mentionning that this lack of sustainability in software development was already encountered in the past, and therefore the MDA [5] returns centre-stage with the slogan : "*Write once, run everywhere*".

Surprisingly, existing research attempts [6, 7, 8, 9, 11] have not used model-driven techniques to tackle the heterogeneity issue but rather to improve the development process for a given mobile platform or even a given mobile usage, along with a strong focus around the UML langage. In contrast, this research paper proposes a language dedicated to mobile development, named *X*MOB , that is designed to capture sufficiently common and high-level concepts to be shared by different platforms – present and yet but likely to come. This DSL will output native code skeletons, using the MDA approach, i.e., through model refinement (PIM – PSM – Source code), and the use of (meta)modeling languages standardized by OMG (MOF and UML2). The *X*MOB language is designed for computer programmers, but who no longer need to be experts in each mobile platform, which allows us to consider the two following uses :

1. From an agile development point of view, *X*MOB will allow for a faster prototyping, by several stakeholders, even including the end-customer.
2. From an academic point of view, *X*MOB will make mobile development more easily accessible to undergraduate students. It will also spare the teacher from having to choose only one platform during his course.

The rest of this paper is organized as follows : in Section 2 we contribute to the debate between a web-based or native-based strategy when it comes to the issue of developing a mobile app. Section 3 lists the existing cross-platform development solutions, and compares them through pragmatic criteria. Then we describe the key elements of our future *X*MOB DSL in Section 4 before concluding and giving our

[*]Member of the PauWare Research Group - `http://www.pauware.com`

perspectives in Section 5.

2. *WEBAPP* VERSUS *MOBAPP*

Generally speaking, web-based applications development (WebApps) is very popular and has logically adapted to a mobile use. WebApps considered "mobile-friendly" are most often obtained through frameworks such as *jQuery Mobile* [1], *jQT* [2] (formerly jQTouch), *Dojo Mobile* [3] or *Sencha Touch* [4]. At first glance, those applications seem ideal since they are inherently and effortlessly portable from one mobile OS to another, as they are executed through a web browser. Nevertheless, even Facebook creator Mark Zukerberg has recently recognized [1] making a mistake by neglecting native applications (MobApps) and the unparalleled user experience they offer. Indeed a MobApp is specific to the OS for which it is designed, and can be characterized by the following items :

- User Interface (UI) : Each OS is characterized by its own aesthetics and interaction modes, which are perfectly recognizable : the look'n'feel. Through potential rejection of an app on the official store, OS vendors show they are anxious to unify the UI of the downloaded applications, for a user-friendly and seamless feeling ;
- System Access : this includes the access to applications already installed on the system, the information they contain (address book, photo gallery, etc.), and listening to system events (resuming from sleep, evolution of battery power, receiving an SMS, etc.) ;
- Hardware Access : Smartphones and tablets of medium and high quality have a large collection of sensors (light, acceleration, pression, etc.) that make it possible to consider many exciting applications. We must not forget either a smart use of short-range wireless interactions such as NFC or Bluetooth ;
- Performance : mobile users are a more impatient, more demanding population than others, and hence mobile applications need to be able to run quickly, and steadily. Another level of indirection introduced by a browser at runtime is unacceptable in a lot of situations, where the application's reactivity is paramount, as in video games for instance.

Despite the capabilities of modern browsers (geolocalization, web storage, WebGL, etc.) and the evolution of the associated W3C standards (HTML5, CSS3, etc.), as other authors [3] we remain convinced that the cross-platform challenge is currently in the development of native applications, and not mobile web applications.

Several solutions have recently emerged to try and satisfy all or part of the previous items. The term XMT (Cross-platform Mobile development Tool) was suggested in [2] to qualify that kind of solution.

3. A PRACTICAL SURVEY ON XMT

Existing XMT comparisons are often scattered over unofficial sources like blogs' posts. Even surveys found in the academic literature [2, 4, 12, 13] are partial. In this regard, we propose in this section to gather existing solutions and to establish very pragmatic criteria for their comparison.

1. `http://www.jquery.com`
2. `http://www.jqtjs.com`
3. `http://www.dojotoolkit.org`
4. `http://www.sencha.com/products/touch`

3.1 Write once ? Run everywhere ?

Note that the slogan "Write once, run everywhere" is made up of two parts. The first one concerns which source language(s) the computer programmer will have to write his application only once. The second concerns which target platforms are supported, i.e., the ones that will be able to run the code.

3.1.1 Write

This criterion deals with the skills the programmer has to possess. We will distinguish three categories :

- Web-related languages : typically the triplet HTML / CSS / JavaScript, because their programmers' community is very large. The development method is web-like, but the result tends to get close to that of a native app ;
- Mainstream languages : we can cite Java, C++, or even Ruby. These general-purpose languages are supported by their own communities, and have already proved their qualities in the past ;
- Domain-specific languages : DSLs present the advantage of offering constructions dedicated to mobile software development. The problem with those languages is that they often lack clear documentation, making them harder to learn.

3.1.2 Run

This criterion deals with the technique used to be able to execute the same code on different targets. We will distinguish two classic methods :

- Interpretation : The interpreter bridges the constructions of the source language with calls to the hosting system's native functions. This assumes that such a bridge must be developped for each mobile platform. It can be installed in the manner of a virtual machine, or be packaged with each application installed onto the device ;
- Cross-compilation : The compiler operates a translation from the source language to several target languages. Each new platform requires to add a new transformation rule to the compiler. Nothing distinguishes eventually generated code from code written directly by an engineer competent with the target platform's SDK.

3.2 Selected tools

We decided to study twelve solutions found in the literature or available on their official websites : *Rhodes*, *LiveCode*, *Tabris*, *Apache Cordova* (formerly PhoneGap), *Titanium*, *Neomades*, *XMLVM*, *Canappi*, *APPlause*, *MoSync*, *Codename One* and *Marmelade SDK*. The strict application of our viewpoint on the debate of Section 2 excludes from this list solutions that do not ensure a native experience, like *MobL* [5], *Moscrif* [6], *Icenium* [7] or *Mobia Modeler* [6], and the family of Rich Internet Applications (RIA) frameworks (*Adobe Flex*, *Microsoft Silverlight*, *Vaadin/Google Web Toolkit*, etc.). Moreover, some XMTs can either generate standalone or client-server mobile applications (n-tiered architecture). Obviously only the part executed on the mobile terminal is taken into account. Let us have a more detailed look on these selected XMT below :

5. `http://www.mobl-lang.org/`
6. `http://moscrif.com`
7. `http://www.icenium.com/`

1. Rhodes (`www.rhomobile.com`) is a software product released by a company called Rhomobile Inc. Based on a MVC pattern (Model/View/Controller), views are written in HTML 5, and controllers in Ruby, then compiled into Ruby 1.9 bytecode. Applications are provided with a Ruby virtual machine, that will execute the bytecode.
2. LiveCode (`www.livecode.com`) introduces the principle of a very high level language where code is written in a near-English syntax and compiled to a C++ engine, providing an abstraction layer over the mobile Operating System.
3. Apache Cordova (`www.phonegap.com`) operates by interpreting HTML5/CSS3 code for views, and JavaScript code using an abstraction API, allowing system and hardware access (camera, accelerometer, etc.). The user interface is presented like a "chromeless web browser", used as a WebView (class UIWebView in iOS, class Webview in Android). Packaged binaries include the browser and the application sources.
4. Titanium (`www.appcelerator.com`) precompiles JavaScript code into a set of symbols, resolved based on the targeted mobile OS (compiled .o files for iOS, .class files for Android). When resolution is over, and the generated dependancy matrix can be understood by the front-end, the adequate back-end compiler is called to compile it into binaries. Those also include a JavaScript interpretor, packaged with the application for performance reasons, and reading dynamic code.
5. Tabris (`www.eclipsesource.com/tabris/`) is a Java-toolkit for the cross-development of native iOS and Android apps, relying on a web server (based on Eclipse RAP) which generates the UI in JSON form. A runtime engine (the Tabris client) will access the server, receive the UI description and render it using native components. The UI, server-side, is written in Java SWT ; the clients are available for iOS, Android, and basically anything that can read and render JSON and HTML.
6. Neomades (`www.neomades.com`) proposes a tool for developping cross-platform applications, in a license mode, or in a service mode. The JavaME code is compiled for each target platform.
7. XMLVM (`www.xmlvm.org`) is a tool allowing to directly recompile instructions in (front-end) Java or .Net CIL bytecode, rather than working at a source code level. Several back-ends allow to generate Java or CIL bytecode, or JavaScript, Python, Objective-C.
8. Canappi (`www.canappi.com`) is an approach allowing to generate code for Android, iOS, or Windows Phone (with optionnal PHP / MySQL files), using MDSL, a simple to learn and use language. The code generation uses the MDA approach.
9. APPlause (`applause.github.io`) is a DSL using the Xtend technology to generate native iOS, Android, or WP7 code. That code, Java, Objective-C, C#, or Python, is human-readable, and can be reworked before deployment.
10. MoSync SDK (`www.mosync.com`) is an IDE allowing to develop in C/C++ or in HTML5/JavaScript, complex mobile applications for multiple platforms. The C/C++ code is compiled into an intermediate language, MoSync IL, using a GCC back-end. Based on the target platform, this intermediate code is packaged with a runtime engine (for JavaME and Blackberry) which will interpret it. For Android, Symbian, Windows Mobile, etc. the process is similar ; the runtime recompiles the code into ARM bytecode, which is then directly executed. For iOS, the process is different still, as MoSync IL code isn't generated. It is instead translated into C, and an xCode project is generated.
11. Codename One (`www.codenameone.com`) is a Java-based platform, allowing to create real native apps, on several mobile platforms. Using plugins for NetBeans or Eclipse, a developer can build an application using Java and a Swing-like API, and a drag-and-drop GUI builder. Apps are then deployed, directly in Java for Android, BlackBerry, and JavaME devices ; for iOS the code is translated to C/Objective-C using XMLVM ; for Windows Phone a C# translator is used.
12. Marmelade SDK (`www.madewithmarmalade.com`) is mainly used as a game engine, as it can access the OpenGL ES API to render graphics, and directly supports features intended for game design. It comprises two layers : a low-level, C-based API, provides an abstraction layer that allows a programmer to access the device functionalities simply ; a high-level, C++-based API, is used for graphical (2D or 3D) rendering, resource management and HTTP networking. Marmelade apps are interfaced with the OS using an OS-specific loader.

3.3 Results and remarks

Table 1 summarizes the different elements introduced in sections 2 and 3.1, namely the language skills, the running technique in terms of interpretation (I) or cross-compilation (C), the look'n'feel, the system and data access, and the list of supported mobile platforms [8]. Performance is removed from the table, since it has to be evaluated in consideration of specific features (e.g., 2D/3D rendering), and transcompilation typically gives better results than interpretation. The meaning of the symbols used in the table are the following : yes (✓), no (✗), partial (/) and empty cell when unknown.

When looking at table 1, we can first draw a demarcation line between solutions that are full-native and the ones that are semi-native (typically HTML display + bindings to native calls). We observe a causal relationship between the type of solution and the underlying execution technique : full-native is obtained through transcompilation while semi-native is obtained through interpretation.

Secondly, most of the solutions above require exactly the same level of details for programming than native, but rather allow people to use well-known languages. This is undeniably a crucial aspect for software companies that have to reuse the skills of their in-place developers. In constrast, a DSL aims at actually decreasing the effort needed to write a program, surprisingly very complex.

Hence, we clearly argue in favor of transcompilation in this paper, along with a mobile-specific language. In this regard, our position is closely related to that of the Canappi or APPlause solutions, that provided interesting proofs of

8. We neglect the platform version for the sake of simplicity.

Table 1: Results of the survey on XMT in 2013. May be deprecated when reading this paper.

						Supported Platforms									
	Write	Run	Look'n'Feel	System Access	Hardware Access	iOS	Android	Blackberry	Windows Phone	Symbian	Bada	Firefox OS	Ubuntu Touch	Meego	WebOS
Rhodes	Ruby / HTML5	I	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗	✗
LiveCode	LiveCode	I	✓	✓	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗
Cordova	HTML5 / JS	I	✗	/	✓	✓	✓	✓	✓	✓	✓	✗	✗	✗	✓
Titanium	HTML5 / JS	I	✗	/	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗
Tabris	Java	I	✓	✓	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗
Neomades	Java	C	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗
XMLVM	Java / .Net	C	✓	✓	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗
Canappi	MDSL	C	✓			✓	✓	✗	✓	✓	✗	✗	✗	✗	✗
APPlause	APPlause	C	✓			✓	✓	✗	✓	✗	✗	✗	✗	✗	✗
MoSync	C++ / HTML5 / JS	C		✓	✓	✓	✓	✓	✗	✓	✗	✗	✗	✓	✗
Codename One	Java	C / I	✓	/		✓	✓	✓	✓	✗	✗	✗	✗	✗	✗
Marmelade SDK	C/C++	C / I	✗	✓	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗	✓

concept. But as it is well admitted that several languages may be competing (e.g., Java versus C++ versus C#), we decided to give rise to our own initiative : *X*MOB .

4. THE *XMOB* DSL : A ROADMAP

The purpose of *X*MOB is to abstract away all specificities from one platform to another. It aims at capturing their commonalities in terms of a high-level mobile dialect. Consequently, constructs of the language are directly related to the vocabulary of mobile devices such as receiving an SMS, notifications, screens and transitions, etc. Hence, a developer using *X*MOB will be able to efficiently describe an app using those common concepts.

4.1 Elements of design

The idea of a mobile application can be captured through different events, triggered by user actions or by the OS, also through a user interface, and data retrieved from various sources ; these events are highlighted in *X*MOB , along with interface description and data sources.

In the manner of the well-known MVC pattern which decouples and interrelates three major concerns, *X*MOB language is in fact divided into the three sub-languages below :

– xmob-data : deals with data sources and their access ;
– xmob-ui : deals with all elements displayed ;
– xmob-event : deals with the events linking every user interface and data elements together.

These languages allow the app developer to easily modularize each part of the application, writing code for different purposes in different files, which are thus easier to reuse.

4.1.1 xmob-ui

The xmob user interface language is used for describing the elements that the app will present. We accept in this language that, from a user point of view, an app can be broken down into a succession of screens, each of them containing different GUI elements (also known as Widgets), such as forms, buttons, lists, and such. Some of them are intended to be populated from data sources.

Xmob-ui is thought this way : here the developer specifies only visual elements, that will automatically be placed at compilation, linked together and such by functions called from other files. We distinguish screens from built-in screens which correspond to already available screens within applications like the dialer, the address book, map, media player, etc.

In this way, we can write a simple “hello world !” program, using only the xmob-ui language :

```
@launch
screen Hello {
  title:'My␣first␣screen'
  body {
    label:'Hello␣world!'
    list
  }
}

@builtin
screen Dialer
```

If we take a look at that snippet, there are various elements to be noticed. The annotation `@launch` specifies the following screen to be the one the application will launch. We open a `screen` with the identifier name `Hello` and the title `My first screen`, and put inside its `body` a `label` and a `list`. We use an implicit layout manager so that the label and the list will be, by default, centered on the screen. The annotation `@builtin` specifies that the screen is externally defined in a third-party mobile app.

4.1.2 xmob-data

Mobile apps often require retrieving data, be it from a database, a webservice or something else. Xmob-data allows the developer to work with a diverse set of inputs.

```
--Remote data source
cnnNews as datasource[access:remote,return:xml]{
  url:'http://cnn.com/newsFeed.php?date=2013-04-8'
}

--Local data source
cust as datasource[access:local,return:record]{
  entity Customer {
      String firstName
      String lastName
      Date birth
  }
}
```

4.1.3 xmob-event

As previously stated, an application is essentially constituted of a GUI, data to fill it, and events to glue everything together. We dealt with the two first considerations. An event can be any kind of user interaction or system alert that can be captured by an app.

Xmob-event is used to listen to any event and to call functions accordingly, using on/do rules. For instance, we have button1 on Screen1, and wish to link to Dialer when that button is pressed. A xmob-event file might look like this :

```
--User-defined event
on buttonpress[Screen1.button1] do {
  open Dialer
}

--System-defined event
on networkavailable[SYSTEM] do {
  close Dialer
  open Hello
  fetch cnnNews into Hello.lists[0]
}
```

Here, using the event `buttonpress` with the argument `Screen1 .button1` we define very simply the action to take whenever a user presses button1.

We distinguish system-defined functions (a predefined set) from user-defined ones, as well as system or user-defined events. System-defined functions include opening a different screen (as in the previous example), opening another application (like the system calendar, or the phone dialer), sending an SMS, etc. System-defined events are for instance an incoming call or SMS, a low-battery warning, whereas user-defined ones can be a pressed button, a particular gesture and so on. For instance, when the network becomes available again (event `networkavailable`) we wish to show the Hello screen populated (keywords `fetch into`) by a news feed. Of course, the underlying fetching technique (SAX Parser, DOM Parser, JSON Parser, Cursor Loader, Unserialization) depends on the return type declared in the xmob-data file.

4.2 *Xmob* crosscompiler

From a MDA point of view, one way to achieve crosscompilation is to refine an input model into output models, through model transformation techniques (one-to-many). That is the solution that supports the architecture of the *X*MOB crosscompiler, as depicted in Figure 1.

4.2.1 Models and metamodels

In accordance to the MDA vision, we have two levels of refinement :

– PIM : *X*MOB is independant from the different mobile OS so that engineers can focus on the essential, ignoring technical details at first ;

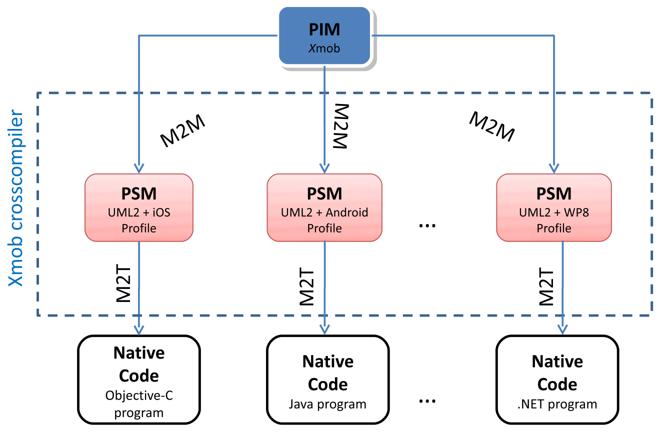

Figure 1: MDA-compliant architecture of the *X*mob crosscompiler.

– PSM : The technical details, specific to each mobile platform, are directly woven. Here we use the UML2 (class and activity diagrams) language with dedicated profiles.

4.2.2 Model transformations

Starting with *X*MOB and ending to native code, each transformation chain contains two model-to-model (M2M) transformations and one model-to-text (M2T) :

– PIM2PSM : translation of high-level DSL concepts into their equivalent. Matches are far from trivial ;
– PSM2PSM (optional) : iterative introduction of best practices (naming conventions, various refactorings, etc.) ;
– PSM2Code : production of source code corresponding to the elements in the PSM. At this level, matches are quite direct.

4.3 The *Xmob* solution delivery

In this section we describe the software development related to the *X*MOB roadmap and the way it will be used as a launch pad by developers of mobApps.

4.3.1 Implementation

In order to implement the solution described in this section, we rely on the Eclipse Modeling Framework (EMF) tooling. Here are the technical choices we have made :

– EMF provides native tools for metamodeling and modeling, plus the facility to build plug-ins inherited from Eclipse ;
– XTEXT is a lightweight tool for the development of one's own text editors and concrete syntax for languages ;
– KERMETA is a transformation engine well-suited for the support model-to-model transformations ;
– XPAND provides a template-based engine for model-to-text transformations.

4.3.2 Packaging

The idea is to distribute *X*MOB as an Eclipse IDE plug-in, in forms of dedicated perspectives and views, including a complete code editor (syntax highlighting, code completion and folding, customized outline, validation, etc.) and a generator module. This module will be itself extendable, dynamically loading each available transformation chain as an add-on (one per target mobile platform). Last but not least, wizards will provide assistance to the developers when creating a new XMOB project and when generating to the desired target platforms.

4.3.3 Features

The code skeleton produced by the XMOB plugin will be organized (i.e., directories and files) as expected by the targeted SDK, and ready to be imported as is in the advised IDE : an Eclipse+ADT project structure in the case of Android, an X-code project in the case of iOS, a QtCreator project in the case of Ubuntu Touch, and so on. Such projects ought to be reworked and complemented subsequently, manually. Finally, the IDE's configuration, compilation of the generated code and deployment of the binaries are out of the scope of the XMOB plugin, remaining the entire responsibility of the developers of the mobApps.

5. CONCLUSION AND ON-GOING WORK

Since the mobile market has become fragmented, mobile app development across a wide set of platforms is a big headache for developers. In this paper we provided a straightforward comparison of existing cross-platform tools that alleviate the development efforts, discarding the web apps which are too naive a solution for the time being. Then, we introduced a new language called XMOB dedicated to mobile software engineers, albeit not experts on every platform, and intended to be cross-compiled through model transformations, as prescribed by the MDA approach.

The very short-term perspective of this work is obviously to continue to refine the scope of XMOB as well as to start the development of its plug-in for Eclipse IDE. A mid-term perspective is to deal with the progressive migration of legacy material : the idea is to then turn XMOB into a pivot language when recovering from already coded mobile app, prior to regenerating towards further platforms.

6. REFERENCES

[1] *"The biggest mistake we made as a company was betting too much on HTML5 as opposed to native"*, Interview of TechCrunch Disrupt, San Francisco, USA, 2012.

[2] Julian Ohrt and Volker Turau, *Cross-Platform Development Tools for Smartphone Applications*, In Computer Journal, Los Alamitos, USA, 2012

[3] Andre Charland and Brian Leroux. *Mobile application development : web vs. native.* In Communications of the ACM, pp. 49–53 , Volume 54, Issue 5, May 2011.

[4] Andreas Sommer and Stephan Krusche. *Evaluation of cross-platform frameworks for mobile applications.* In Proceedings of the 1st European Workshop on Mobile Engineering, February 2013.

[5] Anneke G. Kleppe, Jos Warmer and Wim Bast, *MDA Explained : The Model Driven Architecture : Practice and Promise*, Addison-Wesley Longman Publishing Co., USA, 2003

[6] Florence Balagtas-Fernande, Max Tafelmayer and Heinrich Hussmann, *Mobia Modeler : easing the creation process of mobile applications for non-technical users*, Proceedings of the 15th international conference on Intelligent user interfaces, pp. 509–512, Hong Kong, China, 2010.

[7] Peter Braun and Ronny Eckhaus, *Experiences on Model-Driven Software Development for Mobile Applications*, in Proceedings of the 15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems, pp. 490–493, 2008

[8] A.G. Parada and L.B. de Brisolara, *A Model Driven Approach for Android Applications Development*, in Brazilian Symposium on Computing System Engineering, IEEE Computer Society Press, pp. 192–197, 2012.

[9] Min Bup-Ki, Ko Minhyuk, Seo Yongjin, Kuk Seunghak and Kim Hyeon Soo, *A UML metamodel for smart device application modeling based on Windows Phone 7 platform*, in Proceedings of TENCON 2011 - IEEE Region 10 Conference, pp. 201–205, 2011

[10] Wang Zongjiang, *The study of smart phone development based on UML*, Computer Science and Service System (CSSS), pp. 27916-2794, 2011.

[11] Frank Alexander Kraemer, *Engineering Android Applications Based on UML Activities*, In Model Driven Engineering Languages and Systems, pp. 183–197, LNCS, Springer, 2011

[12] Henning Heitkotter, Sebastian Hanschke and Tim A. Majchrzak, *Evaluating Cross-Platform Development Approaches for Mobile Applications*, In Web Information Systems and Technologies, pp. 120–138, LNBIP, Springer, 2013

[13] Manuel Palmieri, Inderjeet Singh and Antonio Cicchetti, *Comparison of Cross-Platform Mobile Development Tools*, In 4th International Workshop on Business Models for Mobile Platforms, IEEE Communications Society, October 2012, Berlin, Germany.