Model Driven Development Approaches for Mobile Applications: A Survey

Conference Paper *in* Lecture Notes in Computer Science · August 2016

DOI: 10.1007/978-3-319-44215-0_8

2 authors:

Eric Umuhoza
Carnegie Mellon University
11 PUBLICATIONS 73 CITATIONS
SEE PROFILE

Marco Brambilla
Politecnico di Milano
274 PUBLICATIONS 3,196 CITATIONS
SEE PROFILE

Model Driven Development Approaches for Mobile Applications: A Survey

Eric Umuhoza and Marco Brambilla

Politecnico di Milano. Dipartimento di Elettronica, Informazione e Bioingegneria
Piazza L. Da Vinci 32. I-20133 Milan, Italy
{eric.umuhoza,marco.brambilla}@polimi.it

Abstract. The usage and development of mobile applications (referred to as apps) are experiencing exponential growth. Moreover, the vastness and diversity of mobile devices and operating systems oblige the software companies, that want to reach a wide audience, to develop and deploy the same app several times, once for each targeted platforms. Furthermore, the dilemma between browser-based and native user interfaces remains relevant and challenges the capacity of organizations to meet the increasing demand for mobile apps. The adoption of model driven development (MDD) can simplify the development of mobile apps, reducing significantly technical complexity and development costs. Several researches have applied MDD techniques to address these challenges. In this paper, we define a set of criteria to assess the current model driven approaches to mobile apps development. We classify those approaches according the defined classification schema and present the current trends and challenges in this field. The survey shows a preference of code generation over model interpretation and of native apps over cross-platform ones.

1 Introduction

Nowadays, mobile devices are becoming the most common computing device. A vast array of features has been incorporated into those devices to address the different demands of users spanning from games to serious business. Mobile application usage and development is experiencing exponential growth. According to Gartner, by 2016 more than 300 billion applications will be downloaded annually. The number of apps that are available in the online markets has reached unseen numbers. In fact, by July 2015, the Google Play store counted 1.6 million of available apps while Apple's App Store counted 1.5 million [3]. In parallel with these numbers, the market also expects an increase in the number of global smart-phones users, which is expected to surpass 2 billion by 2016 [1] in comparison with 1.4 billion users estimated in 2013 [3]. From those statistics we can expect a healthy market of mobile apps that would be powered by a steady increase in the number of mobile device users, which as of today have, on average, 41 apps installed on their devices [2]. Furthermore, the motivation of the software development companies to continue producing more and better apps is supported by recent industry figures, according to which global mobile app revenues are projected to surpass 76.52 billion U.S. dollars in 2017. ABI research forecasts in 2018, app revenues will be worth 92 billion U.S dollars [25].

The mobile domain presents new challenges to software engineering. It is experiencing an intense competition among software and hardware providers who are continuously introducing innovative operating systems and increasingly powerful mobile devices into the market. Thus, in addition to an increasing request of mobile apps, developers must provide applications that work on all platforms, at least on the most popular ones (Android, iOS, and Windows Phone). Moreover, the dilemma between browser-based and native interfaces remains relevant and challenges the capacity of organizations to meet the demand for mobile apps.

So far, several authors have applied model driven techniques to the development of various aspects of mobile application. Model driven development (MDD) is a development paradigm that uses models as the primary artifact of the development process, with the aim of supporting as many phases as possible, covering also *executability*, i.e., the possibility of getting executable applications out of modeling efforts. Usually, in MDD the implementation is (semi) automatically generated from the models which allows for gains in productivity and quality of the software to be built [7]. This survey overviews the main state-of-the art model driven approaches to the development of mobile applications that support executability, and classifies them based on:

i) *The phases of development process covered*: Requirements analysis, design, implementation, and testing;
ii) *The aspects of mobile application covered*: App data, user interaction, user interface, and business logic;
iii) *Model driven techniques applied*: Type of modeling language, model transformations, code generation or model interpretation;
iv) *Mobile application developed*: Native, hybrid, or mobile web;
v) *Supported mobile platforms*: Android, iOS, or Windows Phone.

The rest of the paper is structured as follows. Related work is studied in Section 2. Section 3 describes the classification criteria. Section 4 introduces the surveyed approaches and classifies them according to our classification schema. In Section 5 we present our analysis of the trends and evolutions. And Section 6 concludes.

2 Related work

This research is the first one that attempts to assess various model-driven approaches to the development of mobile applications with aim of providing their classification and identifying current challenges and trends. In this work we surveyed only the approaches that apply model driven techniques to the development of mobile applications, covering the phases of development down to model execution either in terms of code generation or model interpretation. Thus, this section assesses the existing works that apply the model driven techniques to other concerns of mobile applications or that are not focusing their attention to executability.

Those works can be divided into two different clusters. On one hand we encounter a corpus of researches that apply model driven techniques to specify application interfaces and user interaction (in a broad sense) for multi-device UI modeling. Among them we can cite: TERESA(Transformation Environment for inteRactivE Systems representations) [6], based on a so-called *One Model, Many Interfaces* approach to support model based GUI development for multiple devices from the same ConcurTaskTree (CTT) model; MARIA [24], another approach based on CTT; UsiXML (USer Interface eXtended Markup Language) [33]; IFML (Interaction Flow Modeling Language) [8], a platform independent modeling language designed to express the content, user interaction, and control behavior of the front-end of software applications; and Unified Communication Platform (UCP).

On the other hand we find a collection of works that propose model driven solutions to address non functional requirements of mobile apps. In this cluster we encounter works that apply model driven techniques to automate mobile applications testing. Namely, MobiGUITAR (Mobile GUI Testing Framework) [5] is a model driven testing framework which uses the state machines to test Android apps. Also Ridene et al. [26] proposed MATeL (Mobile Applications Testing Language), a DSL allowing the modeling of test scenarios. Other researches apply model-driven approaches to address the issues of power consumption [22, 29]. Thompson et al. [30] developed a model driven tool, SPOT (System Power Optimization Tool), which automates power consumption emulation code generation.

Few research works that use model driven approaches only for the purpose of modeling mobile apps can be found. Mobile IFML [9] is a PIM designed for expressing the content, user interaction, and control behavior of the front-end of mobile apps. Recently, a variant of Mobile IFML has been applied in the tool WebRatio Mobile Platform to model the user interactions of cross-platform mobile apps [4, 31].

Currently, the hot debate in the research and the development community in general is whether to go down the native app route (developing mobile apps for a specific operating system) or cross-platform (developing apps that work across multiple platforms). Several works conducted research studies to compare the different development platforms using metrics such as complexity and user experience. For instance, Mesfin et al. [23] conducted a comparative evaluation of usability of cross-platform apps on the deployment platforms. They observed that the usability of crossword puzzle app developed with PhoneGap (for Android, Windows Phone, and BlackBerry) was unaffected when deployed on the respective native platforms. The research performed by Tor-Morten et al. [16] reviewed the mobile app development challenges and compared the issues and limitations of mobile platforms. Heitkötter et al. [17] evaluated the cross-platform development approaches for mobile apps. However, all these studies are comparing the different development platforms (native and cross-platform) at the programming level. This paper complements this discussion by offering a classification of model-based approaches for both native and web-based mobile applications development.

3 Dimensions of Analysis

In this section, we define a list of criteria for evaluating model-driven approaches to the development of mobile applications. This set of criteria will guide our discussion of the surveyed approaches in the next sections. For a better understanding, the list of criteria has been structured into perspectives: development process, aspects covered, techniques, and generated apps.

3.1 Development Process Phases

The development process dimension considers the main phases of the development process of the app: requirements analysis, design, implementation, and testing. The aim is to check which phases of the process are directly covered by each analyzed approach.

- **Requirements analysis.** This criterion examines whether the approach in question covers the requirements analysis phase.
- **Design.** This criterion examines whether the approach in question covers the application design phase.
- **Implementation.** This criterion examines whether the approach in question addresses the coding phase.
- **Testing.** This criterion examines whether the approach in question covers the testing phase.

3.2 Covered Mobile App Aspects

Aspects covered perspective sums up criteria relating to the layer of the app covered by the MDD approaches.

- **Content.** This criterion examines whether the approach in question covers the aspects related to the logical structure and operation to the data managed by the application.
- **Business logic.** This criterion examines whether the approach in question covers the aspects related to the internal behavior of the application.
- **User interaction.** This criterion examines whether the approach in question addresses the aspects related to the interactions between user and the application.
- **GUI.** This criterion examines whether the approach in question covers the graphical user interface of the application.

3.3 Model-Driven Development Techniques Applied

This dimension regroups the criteria related to the model-driven techniques used, e.g. visual modeling, code generation, or model-to-model transformations.

- **Modeling language.** This criterion assesses how the applications are modeled in the approach under exam. It indicates whether the approach uses a graphical and / or textual concrete syntax.

- **Multilevel code generation.** This criterion examines whether the approach in question uses intermediate models (e.g. platform independent model which maps to different platform specific models from which the code is generated) to describe the app requirements or produces the finally code directly from the initial models (e.g final code generated from a platform independent model). This implies the use of model-to-model (M2M) transformations too.
- **Executability.** This criterion examines how the approaches addresses executability, i.e., it determines whether the approach relies on code generation or on model interpretation to execute the running apps. Notice that our analysis only focus on approaches that cover executability in some way.

3.4 Generated Apps Perspective

The generated apps dimension examines whether the approach generates native, hybrid (aka., cross-platform), or web apps. Native apps consist of applications developed for a specific target platform, using a programming language or framework provided by the platform itself (e.g., Objective C, or Java), and compiled as an executable software for that platform. Hybrid or cross-platform apps are designed and developed once and executed on multiple platforms, typically thanks to HTML-based code, that is wrapped inside some kind of Web browsing technology and delivered as executable applications. Finally, by web apps we mean actual web sites, developed on purpose to be consumed mainly from a mobile device (mobile-first development).

4 Overview of MDD Approaches

In this section we introduce the surveyed approaches based on the classification schema defined in Section 3. Those approaches are grouped into: researches approaches and commercial solutions.

4.1 Research Approaches

MD2 MD^2 [18] is a model-driven framework for cross-platform development of data-driven mobile apps. In MD^2, the application is firstly described in a platform independent model through a textual DSL. Then, a code generator (one for each platform of interest) transforms the PIM into source code, the business logic of the app along with necessary files to implement the the GUI, for the corresponding platform. The MD^2 framework provides in addition a code generator which creates a server back-end based on the data model of the application.

MobML MobML is a collaborative framework for the design and development of data-intensive mobile apps [15]. The framework is composed of three components: modeling languages, synthesizers (code generators) and a collaboration tool. It offers four platform independent modeling languages each of which addressing a different concern of mobile application: (i) *Navigation*, which describes a mobile app as a collection of views and a set of navigation flows; (ii) *UI*, which describes the graphical interface of the app as a collection of graphic elements, which some extensions to represent the components of a particular target platform; (iii) *Content*, which models the data managed by the app; and (iv) *Business logic*, allowing the modeling of the internal operations of the app, and the interactions that occurs between the user and the app. A synthesizer receives as input the four different models of a mobile app and produces the source code for the targeted platform.

MobIle MultImodality Creator (MIMIC) MobIle MultImodality Creator (MIMIC) [13] is a model driven framework that enables the modeling and automatic code generation of multimodal mobile applications. MIMIC relies on the Mobile MultiModality Modeling Language (M4L), a language based on use of state machines to model input and output multimodal mobile interfaces. The M4L and its graphical editor have been specified through Obeo Designer[1]. The mobile interaction modalities supported by the framework include the tactile, speech, and proximity. The framework generates multimodal interfaces for Android, iPhone and Web.

Applause Applause[2] is a toolkit for creating cross-platform mobile apps. It consists of a DSL to describe mobile apps and a set of code generators that use these models to generate native apps for iOS, Android, Windows Phone and Google App Engine. *Applause2*, the second version of the framework, is expected to cover all the aspects of a mobile app.

JUSE4Android JUSE4Android [11] is a model driven tool that allows the automatic generation of business information systems (BIS) for Android. The apps are specified through annotated UML class diagrams from which the running code is generated.

Francese et al. Francese et al. [14] prose a model-driven approach for the development of multi-platform mobile apps based on finite-state machine. The proposed development environment provides a finite-state machine editor (an Eclipse plug-in) and a generator, which produces source code starting from the model (Data flow, control flow, and user interaction) of the mobile app. The business logic of the application is written in JS. The editor allows developers to call device native features accessed through PhoneGap.

MAG:Mobile Apps Generator MAG is a model driven development approach to generate mobile apps for multiple platforms. The MAG approach is based on UML. The application requirements are modeled through use case diagrams, UML class diagrams are used to model the structure of the app while

[1] www.obeodesigner.com/
[2] https://github.com/applause/applause

UML state machine diagrams are used for behavioral modeling. The mobile domain specific concepts are included in the application models thanks to the mobile UML profile [32]. MAG allows the developer to automatically generate the business logic code of the app from those models while the GUI of the app is developed separately.

Vaupel et al. Vaupel et al. [34] propose a modeling language and an infrastructure for model-driven development of mobile business apps that support the configuration of user roles variants. Following this approach, different user roles are not combined in one app but lead to several app variants that may be configured after code generation. An app model consists of a data model defining the underlying class structure, a GUI model containing the definition of pages and style settings for the graphical user interface, and a process model which defines the behavior facilities of an app in form of processes and tasks.

Chi-Kien Diep et al. Chi-Kien Diep et al [12] propose an online model-driven IDE which provides developers with a platform-independent GUI design for mobile apps. This approach consists on visually designing the GUI of the application once, as an abstract model, and transforming it several times targeting different specific platforms.

Mobl Mobl [19] is a DSL based on standard web technologies for the development of mobile web applications. It integrates languages for user interface design, styling, data modeling, querying and application logic into a single, unified language.

Rule-Based Generation of Mobile User Interfaces (RUMO) RUMO [27] is a model driven development framework for multi platform user interface generation. The UI development in RUMO starts with the creation of a platform independent model which describes the basic structure of the user interface. Then, a set of rules has to be defined through a rule DSL in order to introduce application specific constraints such as usability or guidelines to guard the generation of the targeted user interfaces. Once the rules have been successfully checked, the PIM is finally transformed into target specific user interface (UI for Android, iPhone or for Win Phone). The platform specific ui generation involves the template mechanism which uses the predefined templates. Each template file is responsible of creating the source code for the desired platform. Furthermore, RUMO allows the creation of different versions of the template files as to address the issue of different versions of the same platform.

WL++ WL++ is a model driven code-generation framework for developing multi-platform mobile apps as clients to existing RESTful back-ends [28]. WL++ framework is based on IBM Workinglight platform, Backbone[3], and the WL++ app modeling plugin. WL++ inputs the specification of an existing back-end

[3] http://backbonejs.org

service in the form of the APIs it exposes or the schema of the resource it serves, and infers the data model of the application. Then the developer enriches that application model by adding the proper views and the navigations among them. The source files are finally generated from those models through a set of predefined templates.

AXIOM AXIOM [20] is a model-driven approach for the development of cross platform mobile apps. It uses Abstract Model Tree (AMT), a consistent model representation, as the basis for model transformations and code generation. In AXIOM, the requirements of the application are are firstly described in platform independent intent models (interaction and domain perspective) using AXIOM's DSL. Those intent models are then enriched with structural decisions and refined with platform-specific elements during a multi-phase transformation process to produce the source code for native apps: from requirements models to platform independent model (PIM), from PIM to platform specific models (PSMs), and finally from PSMs to running code.

4.2 Commercial Solutions

Mendix App Platform The Mendix[4] App Platform enables business users and developers to build and deploy multi-channel apps with a MDD platform. The core library is offered as a set of standard services in the platform while specific libraries required for an optimal UI and user experience are built once and then offered as reusable widgets. In Mendix App Platform, the apps are built by defining visual models for the various app components such as the domain model, user interactions and business logic. Those models are then executed in a runtime environment. In addition, the platform provides core capabilities for non-functional application requirements, such as performance, scalability and security. Mendix combines model-driven development, the support of native device functions with the integration of Adobe PhoneGap to generate cross platform, hybrid apps.

IBM Rational Rhapsody IBM Rational Rhapsody[5] integrates with Rational Team Concert to offer modeling capabilities for Android applications and the visual representation of the Android framework API that developers can reference from within Rational Rhapsody. The references are then generated into Java code to automate the manual coding task. Additionally, Rational Rhapsody can read the AndroidManifest.xml file to visualize activities, services, broadcast receivers, main activity and content provides specified for better understanding of the application. The Rational Rhapsody Debugger also enables runtime animation of the class diagrams created for an Android app.

WebRatio Mobile Platform WebRatio Mobile Platform[6] is a model-driven development tool for the development of mobile applications [9]. The tool is

[4] www.mendix.com
[5] http://www.ibm.com/developerworks/
[6] www.webratio.com

based on the mobile-extended version of IFML standard [4]. WebRatio Mobile provides three integrated environment: (i) the modeling environment allowing the specification of user interactions through IFML diagrams and application content model through UML or ER diagrams; (ii) the development environment for supporting the implementation of custom components; and (iii) the layout template and style design environment, which allows the customization of UI through HTML 5, CSS and JavaScript. The code generated by the tool consists of ready-to-deploy cross-platform mobile apps, based on the PhoneGap.

Appian Mobile Appian Mobile [7] is part of the integrated Appian BPM Suite designed for mobile applications following the *write-once, deploy anywhere* architecture. In Appian Mobile, application designers can simply drag-and-drop to design mobile process patterns in Appians Process Modeler. Using a graphical Business Process Modeling Notation (BPMN) modeler, even business users can model and orchestrate processes, define and update rules, create forms and enable them to render natively in mobile apps. The *User Experience* is powered by Appian Self-Assembling Interface Layer (SAIL) which allows designers to create a single user dynamic interface definition, then deploy to native mobile client applications on major device platforms and across major web browsers.

4.3 Classification

Table 1 summarizes the classification of the the surveyed approaches according to the classification schema defined in Section 3.

5 Trends and Outlook

Starting from the analysis we performed, this section identifies the current trends and suggests an outlook on the future of the Mobile MDD field.

5.1 Multilevel Code Generation Approaches

The surveyed approaches can be grouped into two clusters depending on whether they apply intermediate modeling (and thus model-to-model transformations) or not. When following multilevel code generation, the app is firstly specified in a platform independent manner, then the PIM is transformed into different PSMs, one for each platform of interest, from which the running code is generated. Only two approaches adopt this strategy: AXIOM (see Section 4.1) and the approach proposed by Chi-Kien Diep et al. (see Section 4.1). Therefore, despite the complexity of the mobile context and the number of possible target platforms, most of the approaches do not consider efficient to have some intermediate platform-specific models, which are in some sense seen as an excessive burden.

[7] www.appian.com

Table 1: Model-driven approaches to the development of mobile applications and classification of their main characteristics.

Approach	**Process**				**Aspects**				**Techniques**					**Apps**			**Platforms**[a]				
	Requirements	Design	Implementation	Testing	Content	Business logic	User interaction	GUI	Graphical lang. notation	Textual lang. notation	Multilevel code gen.	Code generation	Model execution	Native	Hybrid	Mobile Web	Android	iOS	Windows Phone	Black Berry	Web
Research Approaches																					
MD^2		✓	✓	✓	✓	✓	✓	✓		✓		✓		✓			✓	✓			
MobML		✓	✓	✓	✓	✓	✓	✓		✓		✓		✓			✓	✓			
MIMIC		✓	✓				✓	✓	✓			✓		✓		✓	✓	✓			✓
Applause		✓	✓		✓	✓	✓	✓		✓		✓		✓		✓	✓	✓	✓		✓
JUSE4Android		✓	✓		✓	✓	✓	✓	✓	✓		✓		✓			✓				
Francese et al.		✓	✓				✓	✓	✓	✓		✓			✓		✓	✓			
MAG	✓	✓	✓				✓		✓			✓		✓			✓		✓		
Vaupel et al.		✓	✓		✓	✓	✓	✓	✓			✓		✓			✓	✓			
Chi-Kien Diep et al.		✓	✓					✓	✓		✓	✓		✓			✓	✓	✓		
Mobl		✓	✓		✓	✓	✓	✓		✓		✓		✓			✓	✓			
RUMO	✓	✓	✓				✓	✓	✓	✓		✓		✓			✓	✓			
WL++		✓	✓		✓	✓	✓	✓	✓	✓		✓		✓			✓	✓			
AXIOM	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓			✓	✓			
Commercial Solutions																					
Mendix App Platform	✓	✓	✓		✓	✓	✓	✓	✓				✓		✓	✓	✓	✓			✓
IBM Rational Rhapsody	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓		✓			✓				
WebRatio Mobile Platform		✓	✓		✓	✓	✓	✓	✓			✓			✓	✓	✓	✓			✓
Appian	✓	✓	✓		✓	✓	✓	✓	✓			✓		✓		✓	✓	✓		✓	✓

[a] The *Platforms* column reports the platforms actually supported by the approach, but in some cases those are only a subset selected to describe the approach.

5.2 Single Level Code Generation Approaches

This category groups the approaches which skip one or more levels of the model driven architecture. *Rhapsody* and *JUSE4Android* approaches specify directly the app in a platform specific models (PSMs) from which the android code is generated (PSM-to-Java Code). This is common when targeting a single platform since in that case there is no global model that can be reused across the target platforms. The remaining approaches like (MD^2, MobML, MIMIC, WebRatio, and Mobl) directly generate the code from the platform independent models

PIMs skipping the PSM level. This is either obtained through actually generating cross-platform (hybrid) code, or through multiple generators of native code.

5.3 Development Process

All surveyed approaches apply MDD to Design and Implementation phases. Only 23% of the approaches covers the testing phase, and only 35% covers the requirements phase. However, it is interesting to note that 3 out of 4 analyzed commercial platforms cover requirements. This means that this is deemed a very important phase for production. Overall, only 11% apply MDD to all phases of the development process.

5.4 Mobile App Aspects

Some approaches cover only few aspects of mobile applications. User interactions and GUI are the most covered aspects (94%) while only 70% of surveyed approaches address content and business logic aspects. This is explained by the fact that the most crucial points of mobile apps are related to user interaction. Therefore, the approaches specifically target these aspects with custom modeling solutions. Viceversa, more consolidated aspects like content and business logic can be covered with traditional modeling languages and approaches (E.g., UML, BPM and so on).

5.5 Executability

Model driven development approaches are commonly used to generate the final code either for a single concern or for all aspects of a mobile application. In fact, 94% of surveyed approaches apply *code generation*: the running code is generated from the high level models, while only the 6% relies on *model interpretation*. When following model interpretation, a generic engine is implemented and the model is interpreted by that engine, thus, model interpretation does not requires to generate the code to create a working application from a model. The code generation is preferred to model execution mainly for the following reasons:

i) Code generation is easier to start with and allows reusing existing programming artifacts. The developers can start using code generation by turning existing code into templates and replacing parts of the code with tokens which will be replaced by model information;
ii) The generated implementation is easier to understand. The generated code is produced in a standard programming language that any developer can understand, while for model interpretation one needs to understand the generic implementation of the interpreter and the semantics of the model;
iii) A code generator is usually easier to maintain, debug, and track because it typically consists of rule-based transformations, while an interpreter has a generic and complex behavior to cover all the possible execution cases;
iv) Code generation provides an additional check for errors since the generated code needs to be compiled.

v) Generated apps are typically more easily accepted and integrated within an enterprise setting, because the generated code can be aligned with the company standards.

5.6 Native, Cross-platform or Web Applications

Web apps are the less popular solution in MDD for mobile. Adoption of native and cross-platform apps is supported among others by the following advantages [21, 10] with respect to mobile-accessible web applications:

i) *Availability.* Users can easily find and download apps of their choice from the app stores and marketplaces;
ii) *Offline.* With respect to web sites, apps have the ability to run offline;
iii) *Safety and security.* Apps have to get the approval of the app store they are intended for.

More precisely, the model-driven community is focusing its attention on native apps development. In fact, 82% of surveyed approaches target the development of native apps. The main reason of this is the perceived higher quality of the obtained applications, in terms of performance, usability, and capability of exploiting and integrating with the most advanced features of mobile devices.

This trend could be very interesting for the software development companies, especially to the SMEs. Indeed SMEs with limited resources, are currently obliged to go down the hybrid route, and thus, loosing some of the advantages of native apps in order to reach a large audience at a sustainable cost.[8] Adopting a MDD approach would allow these company to get the same benefit obtained through native development, but with the productivity of cross-platform development.

5.7 Cross Platform Development

When following MDD, with code generation, cross platform can be reached either by providing a code generator for each of targeted platforms or by generating the code required by the cross platform tool (like PhoneGap, Appcelerator Titanium, and Xamarin) to produce cross platform apps [31]. Basically, those code generators receive in input the same model describing the application and produce the code for the corresponding platforms. More than 88% of surveyed approaches target the development of at least two platforms.

5.8 Lack of Standard Mobile Modeling Language

The modeling language is a fundamental building block of each model driven development approach. Almost each approach relies on its own DSL either defined from scratch [13] or from the existing standards [31, 32]. However, no specific standard has been devised for the mobile domain. This means that the community looses all the advantages offered by the standards. This should be one direction to target in the near future.

[8] Gartner predicts that more than 50% of mobile apps deployed by 2016 will be hybrid.

6 Conclusions

In this paper we presented the results of a survey on model-driven approaches to the development of mobile apps. We have classified those approaches based on the covered development phases, adopted MDD techniques, type of developed apps, and supported platforms. The current trends are in line with the ones of mobile apps, i.e., preference of native applications (more than 80% of the cases), and prefer to apply code generation rather than model interpretation for the executability of apps.

References

1. emarketer report. `http://www.emarketer.com/`. (April 2016).
2. Flurry. `http://www.flurry.com/`. (April 2015).
3. Statista. `http://www.statista.com/`. (April 2015).
4. Roberto Acerbis, Aldo Bongio, Marco Brambilla, and Stefano Butti. Model-driven development of cross-platform mobile applications with web ratio and IFML. In *MOBILESoft 2015*, pages 170–171, 2015.
5. Domenico Amalfitano, Anna Rita Fasolino, Porfirio Tramontana, Bryan Dzung Ta, and Atif M. Memon. Mobiguitar: Automated model-based testing of mobile apps. *IEEE Software*, 32(5):53–59, 2015.
6. Silvia Berti, Francesco Correani, Giulio Mori, Fabio Paternò, and Carmen Santoro. Teresa: a transformation-based environment for designing and developing multi-device interfaces. In *CHI Extended Abstracts*, pages 793–794, 2004.
7. Marco Brambilla, Jordi Cabot, and Manuel Wimmer. *Model-Driven Software Engineering in Practice*. Synthesis Lectures on Software Engineering. Morgan & Claypool Publishers, 2012.
8. Marco Brambilla, Piero Fraternali, and et al. The interaction flow modeling language (ifml), version 1.0. Technical report, Object Management Group (OMG), http://www.ifml.org, 2014.
9. Marco Brambilla, Andrea Mauri, and Eric Umuhoza. Extending the Interaction Flow Modeling Language (IFML) for Model Driven Development of Mobile Applications Front End. In *MobiWIS*, pages 176–191, 2014.
10. Luis Corral, Alberto Sillitti, and Giancarlo Succi. Defining relevant software quality characteristics from publishing policies of mobile app stores. In *MobiWIS 2014*, pages 205–217, 2014.
11. L. P. da Silva and F. Brito e Abreu. Model-driven gui generation and navigation for android bis apps. In *MODELSWARD 2014*, pages 400–407, Jan 2014.
12. Chi-Kien Diep, Quynh-Nhu Tran, and Minh-Triet Tran. Online model-driven ide to design guis for cross-platform mobile applications. SoICT '13, pages 294–300, 2013.
13. Nadia Elouali, Xavier Le Pallec, José Rouillard, and Jean-Claude Tarby. MoMM '14, pages 52–61, 2014.
14. Rita Francese, Michele Risi, Giuseppe Scanniello, and Genoveffa Tortora. *PROFES 2015*, chapter Model-Driven Development for Multi-platform Mobile Applications, pages 61–67. Cham, 2015.
15. Mirco Franzago, Henry Muccini, and Ivano Malavolta. MOBILESoft 2014, pages 58–61, 2014.

16. Tor-Morten Grønli, Jarle Hansen, Gheorghita Ghinea, and Muhammad Younas. Mobile application platform heterogeneity: Android vs windows phone vs ios vs firefox OS. In *AINA*, pages 635–641, 2014.
17. Henning Heitkötter, Sebastian Hanschke, and Tim A. Majchrzak. Evaluating cross-platform development approaches for mobile applications. In *WEBIST*, pages 120–138, 2012.
18. Henning Heitkötter, Tim A. Majchrzak, and Herbert Kuchen. Cross-platform model-driven development of mobile applications with md2. SAC '13, pages 526–533, 2013.
19. Zef Hemel and Eelco Visser. Declaratively programming the mobile web with mobl. OOPSLA '11, pages 695–712, 2011.
20. Xiaoping Jia and Christopher Jones. AXIOM: A model-driven approach to cross-platform application development. In *ICSOFT 2012*, pages 24–33, 2012.
21. William Jobe. Native apps vs. mobile web apps. *iJIM*, 7(4):27–32, 2013.
22. Imre Kelényi, Jukka K. Nurminen, Matti Siekkinen, and László Lengyel. Supporting energy-efficient mobile application development with model-driven code generation. In *ICCSAMA*, pages 143–156, 2014.
23. Gebremariam Mesfin, Gheorghita Ghinea, Dida Midekso, and Tor-Morten Grønli. Evaluating usability of cross-platform smartphone applications. In *MobiWIS*, pages 248–260, 2014.
24. Fabio Paternò, Carmen Santoro, and Lucio Davide Spano. Maria: A universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments. *ACM Trans. Comput.-Hum. Interact.*, 16(4), 2009.
25. ABI Research. Abi research (march 2013) application revenues coming from either smart phone or tablets. http://mobithinking.com. (October 2013).
26. Youssef Ridene and Franck Barbier. A model-driven approach for automating mobile applications testing. ECSA '11, pages 9:1–9:7, 2011.
27. Andreas Schuler and Barbara Franz. Rule-based generation of mobile user interfaces. pages 267–272, 2013.
28. Eleni Stroulia, Blerina Bazelli, Joanna W. Ng, and Tinny Ng. WL++: code generation of multi-platform mobile clients to restful back-ends. In *MOBILESoft 2015*, pages 136–137, 2015.
29. Chris Thompson, Douglas C. Schmidt, Hamilton A. Turner, and Jules White. Analyzing mobile application software power consumption via model-driven engineering. In *PECCS 2011*, pages 101–113, 2011.
30. Chris Thompson, Jules White, Brian Dougherty, and Douglas C. Schmidt. Optimizing mobile application performance with model-driven engineering. In *SEUS 2009*, pages 36–46, 2009.
31. Eric Umuhoza, Hamza Ed-douibi, Marco Brambilla, Jordi Cabot, and Aldo Bongio. Automatic code generation for cross-platform, multi-device mobile apps: Some reflections from an industrial experience. MobileDeLi 2015, pages 37–44, 2015.
32. Muhammad Usman, Muhammad Zohaib Z. Iqbal, and Muhammad Uzair Khan. A model-driven approach to generate mobile applications for multiple platforms. In *APSEC 2014*, pages 111–118, 2014.
33. Jean Vanderdonckt. A MDA-compliant environment for developing user interfaces of information systems. In *CAiSE*, pages 16–31, 2005.
34. Steffen Vaupel, Gabriele Taentzer, Jan Peer Harries, Raphael Stroh, René Gerlach, and Michael Guckert. Model-driven development of mobile applications allowing role-driven variants. In *MODELS 2014*, pages 1–17, 2014.