Automatic Code Generation for Cross-platform, Multi-Device Mobile Apps: Some Reflections from an Industrial Experience

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

Hamza Ed-douibi
AtlanMod team (Inria, Mines Nantes, LINA)
4, rue Alfred Kastler 44307 Nantes Cedex 3, France
hamza.ed-douibi@inria.fr

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

Jordi Cabot
ICREA - UOC. Internet Interdisciplinary Institute.
Brookhaven National Lab
Av. Carl Friedrich Gauss, 5. Ed. B3. E08860 Castelldefels, Spain
jordi.cabot@icrea.cat

Aldo Bongio
WebRatio S.r.l
Av.Piazza Cadorna, 10.
I-20123 Milano, Italy
aldo.bongio@webratio.com

ABSTRACT

With the continuously increasing adoption of mobile devices, software development companies have new business opportunities through direct sales in app stores and delivery of business to employee (B2E) and business to business (B2B) solutions. However, cross-platform and multi-device development is a barrier for today's IT solution providers, especially small and medium enterprises (SMEs), due to the high cost and technical complexity of targeting development to a wide spectrum of devices, which differ in format, interaction paradigm, and software architecture. So far, several authors have proposed the application of model driven approaches to mobile apps development following a variety of strategies. In this paper we present the results of a research study conducted to find the best strategy for WebRatio, a software development company, interested in producing a MDD tool for designing and developing mobile apps to enter the mobile apps market. We report on a comparative study conducted to identify the best trade-offs between various automatic code generation approaches.

Categories and Subject Descriptors

D.2.1 [**Software Engineering**]: Requirements/Specifications—*Methodologies*; D.2.2 [**Software Engineering**]: Design Tools and Techniques—*Computer-aided software engineering (CASE)*; D.2.6 [**Software Engineering**]: Programming Environments—*Graphical environments*

General Terms

Design,Languages

Keywords

Mobile application, Code generation, IFML, Cross-platform development, Multi-device development

1. INTRODUCTION

Software development organizations that aim to exploit the mobile market have plenty of new business opportunities through direct sales in app stores and delivery of B2E and B2B solutions. Nevertheless, the vastness and diversity of mobile devices and operating systems available on the market oblige companies to produce and deploy the same app several times, once for each of the different mobile platforms. Unfortunately, cross-platform and multi-device development is a barrier for today's IT solution providers, especially SMEs, due to the high cost and technical complexity of targeting development to a wide spectrum of devices, which differ in format, interaction paradigm, and software architecture.

The adoption of model driven development (MDD) can dramatically simplify multi-device development, reducing substantially cost and development time, so as to increase the profit of SME solution providers and at the same time reduce the price and total cost of ownership for end-customers. So far, several authors have applied model-driven techniques to specify application interfaces and user interaction (in a broad sense) for multi-device UI modeling: IFML [3], TERESA [2], MARIA [13], MBUE [11], UsiXML [15], and UCP [14]. A

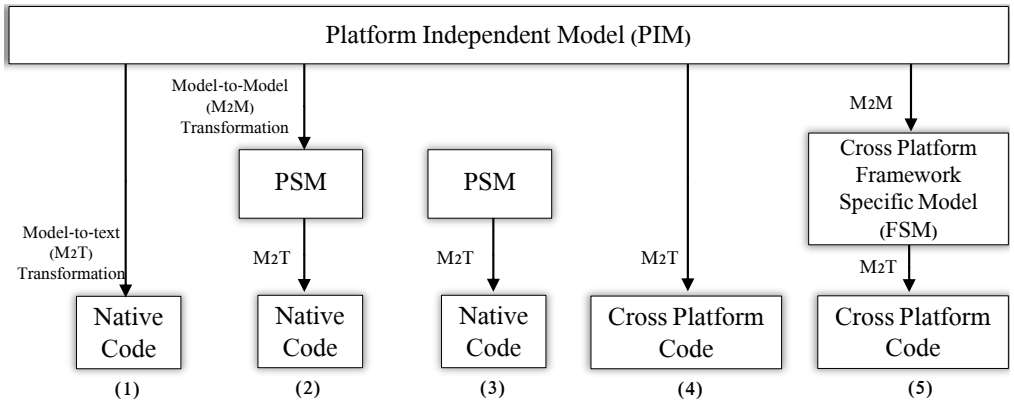

Figure 1: The different approaches towards the automatic generation of mobile applications from the models describing the application

few approaches specifically target mobile applications front-end specification such as Mobile IFML [4], MD2 [9], and MIMIC [5] to cite few of them.

Still, when following a MDD approach, several code-generation strategies are possible depending, both, on the abstraction level to be used when modeling the application and the abstraction level of the code to be generated. We will use the model driven architecture (MDA) as a reference framework to illustrate the different alternatives. MDA defines models at three different levels of abstraction: Computation Independent Models (CIM), Platform Independent Models (PIM), and Platform Specific Models (PSM). A set of mappings between each level and the subsequent one can be defined through model transformations. Typically, every CIM can map to different PIMs, which in turn can map to different PSMs but many other combinations can be followed, for instance skipping one of the levels.

This paper reports on the research study conducted (as part of the Automobile European research project [1]) to find the best mobile code-generation strategy for *WebRatio* with the goal of maximizing its chances to succeed in entering the mobile apps market. WebRatio is a software development company interested in producing a MDD tool, providing developers with full code generation capabilities, for designing and developing mobile apps. The generated apps shall be available on different platforms such as Android, iPhone, and Windows Phone.

This scenario is faced by many companies today and given the variety of strategies, choosing the wrong one can have dramatic consequences for the company. Roughly speaking a company should choose among the following general options (Figure 1):

(1) PIM-to-Native Code (NC). This option states that the app code is generated from a PIM describing the app. Cross platform is achieved by providing one native code generator for each targeted platform;

(2) PIM-to-PSM-to-NC. A global PIM is transformed into a set of PSMs that refine it for specific platforms. These lower-level PSM models are the input of the code-generator for the corresponding platforms;

(3) PSM-to-NC. It consists on defining directly the PSMs, one per each development platform and then generate the app code from these PSMs;

(4) PIM-to-Cross Platform Code (CPC). This option takes as input the platform independent model and generates the code required by the *cross platform framework* (such as PhoneGap, AppCelerator Titanium, and Xamarin) to produce the *cross platform apps*; and

(5) PIM-to-Framework Specific Model (FSM)-to-CPC. With respect to the PIM-to-CPC option, this approach introduces the FSM which gathers the information regarding the *cross platform framework* used to produce the apps. FSM is a PSM in which the *Platform* in the MDA terminology, is actually a Cross-Platform Framework for mobile apps development.

We present the results of a comparative study conducted to identify the best trade-offs between those alternatives. The study included the development of the code generators for native platforms (Android and iOS) and generators for cross-platform frameworks such as PhoneGap and AppCelerator Titanium. Mobile IFML [4] is used as the reference modeling language to express the PIM level but the discussion would be the same when using any other PIM language.

The rest of the paper is organized as follows: Section 2 presents the state of the art; Section 3 shows a running example; Section 4 presents the code generation options; Section 5 compares code generation strategies; and the Section 6 concludes.

2. STATE OF THE ART

Several approaches propose modeling languages for the different aspects of mobile platforms. For instance, Mobile IFML (Interaction Flow Modeling Language) [4, 3] is a platform independent modeling language designed to express the content, user interaction, and control behavior of the front-end of mobile apps. Also, Franzago et al. [6] defined a collaborative framework for the design and development of data-intensive mobile apps. Their approach is based on platform

[1] http://automobile.webratio.com/

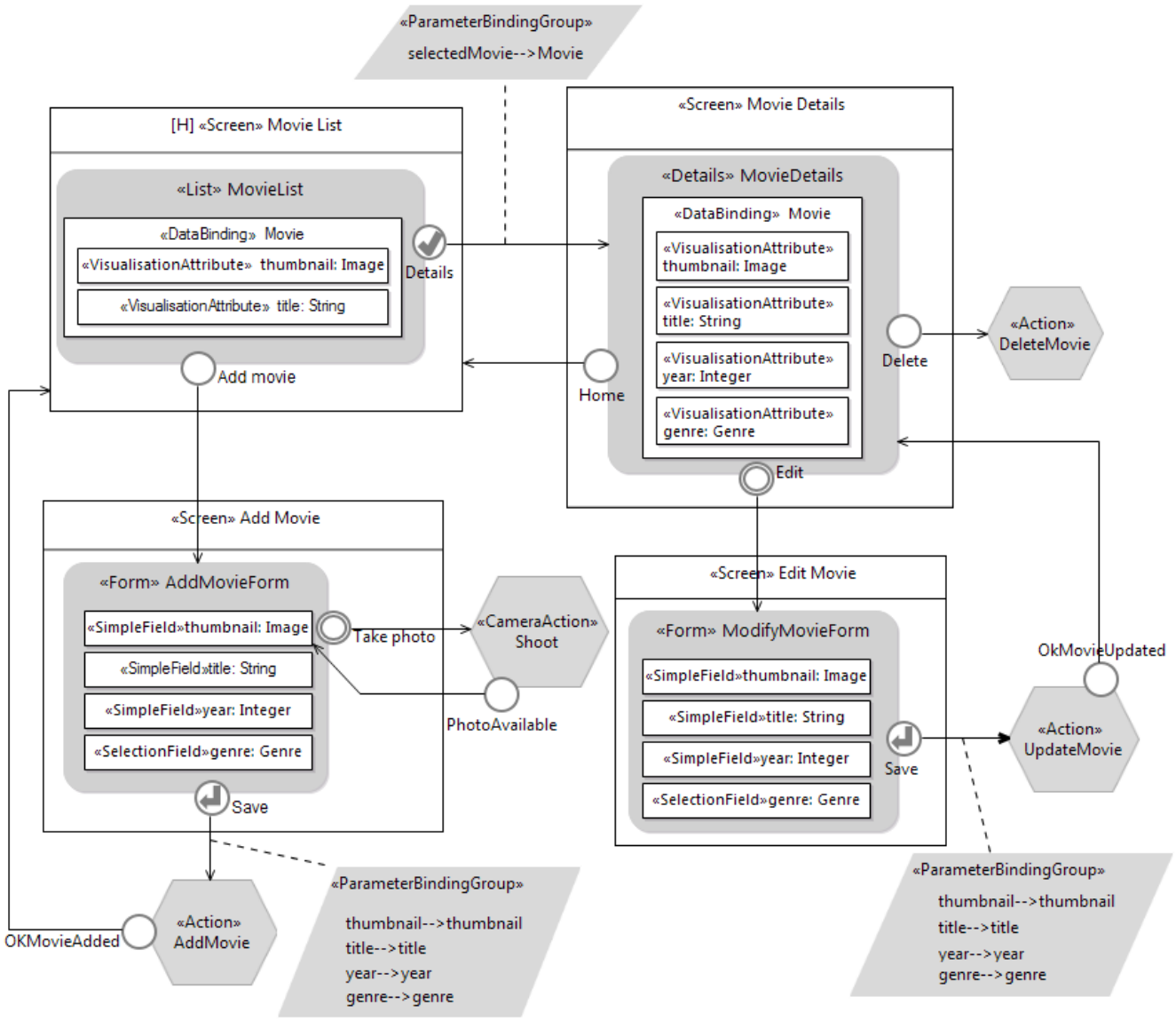

Figure 2: A piece of Mobile IFML model (PIM) describing Movies Manager app. It models the listing, the adding, the detail visualization and the editing of a movie. The model has been specified using the IFML editor (http://ifml.github.io/)

independent modeling languages allowing the specification of various viewpoints (navigation, content, user interface, and business logic) of a data-intensive mobile apps. Additionally, Heiko Behrens [1] defined a model driven development environment for the iPhone. His approach is based on a textual DSL allowing the description of the structure and behavior of data-centric mobile apps;

Besides proposing modeling languages, various approaches also offer code generation mechanisms for one or several platforms. Namely, MD2 [9] is an approach that focuses on the code generation (for Android and iOS) of data-driven business apps for tablets according to the MVC paradigm. MIMIC (MobIle MultImodality Creator) [5] is an approach that relies on the Mobile MultiModality Modeling Language (M4L), which is a language based on use of state machines to model input and output multimodal mobile interfaces, and proposes a code generation facility for Android platform. Also, Höpfner et al. [10] developed a Web based tool allowing the cross-platform generation of information-intensive mobile apps. All them are good examples of the code-generation ”families” mentioned above.

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. [12] 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, Win-

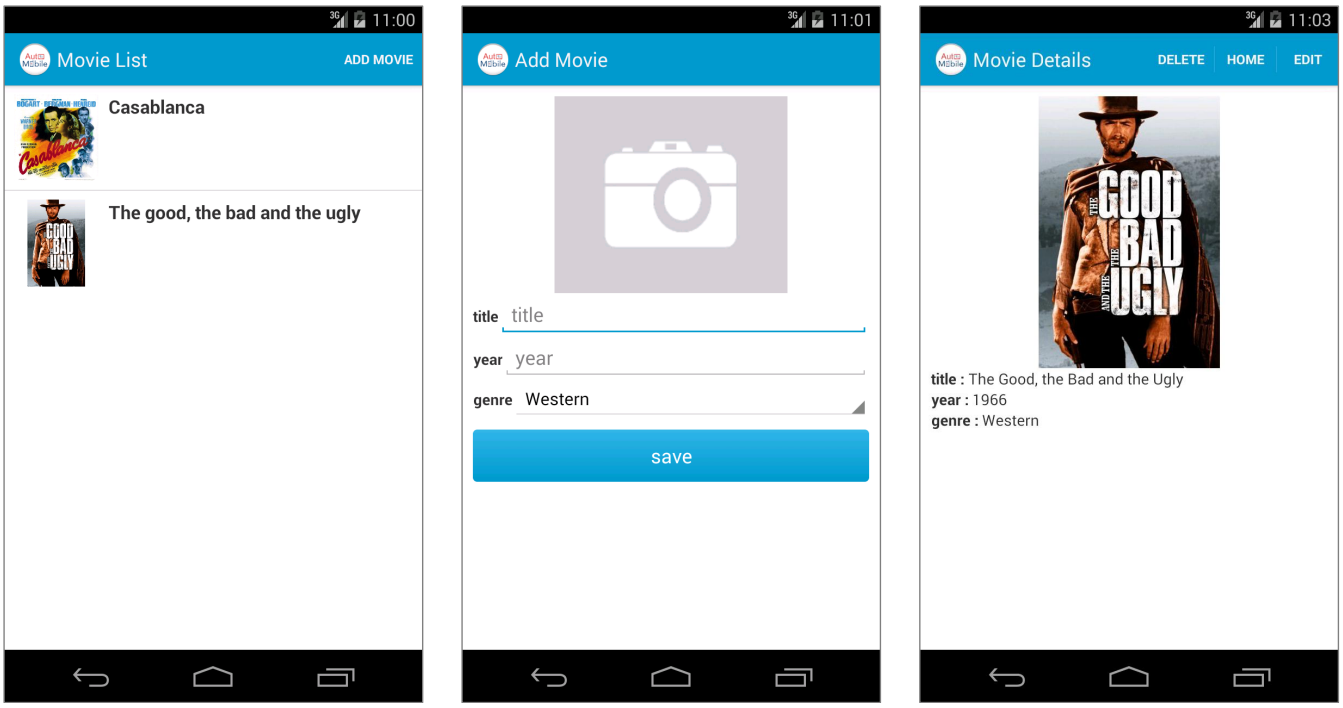

Figure 3: The UI of *Movies Manager* app (Android version): the home screen, which displays the list of movies, is shown on the left side. The screen in the middle allows the adding of a new movie while the last screen displays the details of a selected movie.

dows Phone, and BlackBerry) was unaffected when deployed on the respective native platforms. The research performed recently by Tor-Morten et al. [7] reviewed the mobile app development challenges and compared the issues and limitations of mobile platforms. Heitkötter et al. [8] 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 pretends to complement this discussion by offering a comparative study of model-based strategies for both native and cross-platform development.

3. APPLIED USE CASE

This section presents the app, *Movies Manager*, which is used to illustrate the different code generation approaches discussed in next sections. The domain model is specified in the UML class diagram while the user interaction is modeled using the Mobile IFML language.

3.1 Mobile IFML (Interaction Flow Modeling Language)

Mobile IFML [4] is a platform independent modeling language designed for expressing the content, user interaction, and control behaviour of the front-end of mobile apps. It is an extension of an OMG standard called Interaction Flow Modeling Language (IFML) [3], designed to address the specific requirements of mobile devices. A Mobile IFML model supports the following design perspectives: the *view structure specification*, the *view content specification*, the *events specification*, the *event transition specification*, the *parameter binding specification*, and the reference to *actions* triggered by the user's events. The effect of an event is represented by an *interaction flow* connection, which connects the event to the view container or component affected by the event.

3.2 The Movies Manager App Running Example

Movies Manager is a simple mobile app which keeps track of the movies watched by the user. The app allows the user to access a list of available movies, see the details of a movie, add a new movie, and finally delete a movie from the app.

The **Domain Model** of the Movies Manager app would be just a simple Movie class with the thumbnail, the title, the year of publication, and its genre as properties. Therefore, Figure 2 only shows the Mobile IFML model describing *the content and the interaction flows* of the app. The model specifying the app has four screens: (i) *Movie List* screen. Is the home screen of the app. It contains a *List*, Movie List, which displays the thumbnail and the title for each movie managed by the app. From it, a user can access the *Details* of a selected movie, via *details* event, or add a new movie by following the *add movie* event; (ii) *Add Movie* screen. It allows the user to add a new movie. It contains an entry *Form*, AddMovieForm, allowing the user to provide the details of the movie. The *Long press* event, *Take photo*, associated to the field *Thumbnail* allows the user to take a picture to use as the thumbnail for the movie he is entering. The *Take photo* event triggers a *Photo Camera action*,

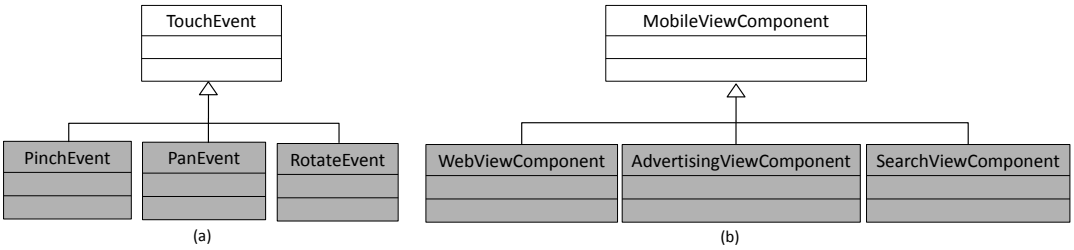

Figure 4: Mobile IFML for iOS: *(a)* Shows touch events which extend the *TouchEvent* class of mobile IFML to better address the iOS gestures. *(b)* Shows the view components which extend the *MobileViewComponent* class.

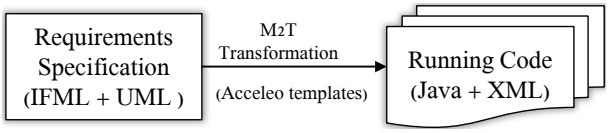

Figure 5: PIM-to-NC: the app's requirements are specified in Mobile IFML and UML models from which the running code is generated.

shoot. The *Save* event associated to the form allows user to submit the information to the system by invoking AddMovie *Action*; (iii) *Movie Details* screen. It displays the details of a selected a movie. It is associated with two events allowing the deletion and editing of a movie; and (iv) *Edit Movie* screen which allows the user to modify the information of the movie. Figure 3 shows a piece of the UI of Movies Manager app generated from the models described in this section.

4. CODE GENERATORS

This section describes in more detail the different approaches for the automatic code generation for mobile apps depicted in the introduction.

4.1 PIM-to-NC: Generation of the Native Apps from PIM

The PIM-to-NC approach, which states that the app code is generated from the platform independent model, is described in this section through an example generating the Android native code for the Movie Manager app.

An *Android App* is written in Java. The Android SDK tools compile the Java code into an APK (Android package), which is an archive file with an *apk* extension. The APK file is used by the android-powered devices to install the app. The essential building blocks of an Android app are called *App Components*. They are of four types: activities, services, content providers, and broadcast receivers. Each component has a distinct life-cycle and plays a specific role to define the behavior of the app. The *Android generator* depicted in figure 5, consists of a set of *Acceleo* [2] *templates* which take the app's models in input and generate the app code. The generated code consists of *Java Classes* implementing the app behavior and the *XML files* managing the layout of the app.

[2]http://www.acceleo.org

4.2 PIM-to-PSM-to-NC: Native Apps from PIM through PSM

The PIM-to-PSM-to-NC approach consists firstly in the specification of the app independently to the platform that will be used to implement the executable code and, in a second step, on creating different PSMs (one for each platform). A PSM is a model refined and annotated based on the specific characteristics of the platform to maximize the quality of the generated code. The running code is generated from those PSMs. Figure 4 shows a piece of the metamodel of the PSM language, *Mobile IFML for iOS*, designed as an extension of Mobile IFML to model the specific information regarding the behavior and structure of the app on iOS Platform. Similar PSM languages would be required for the other platforms.

The iOS extensions address:

- *Events.* The iOS platform supports a large set of gestures and provides rich APIs to implement the related events. The mobile IFML events have been extended to better reflect the events managed in iOS; 4*(a)* reports a fragment of this extension that details the Platform Independent TouchEvent into iOS specific PinchEvent, RotateEvent and PanEvent which directly map to the iOS classes UIPinchGestureRecognizer, UIRotationGestureRecognizer, and UIPanGestureRecognizer respectively.
- *View components.* The iOS platform provides many types of views to help present and organize app's content. The mobile IFML *MobileViewComponent* have been extended to better implement the iOS views; 4*(b)* reports a fragment of this extension that details the Platform Independent *MobileViewComponent* into iOS specific WebViewComponent, AdvertisingViewComponent, and SearchViewComponent which directly map to the iOS classes WebView, AdBannerView and SearchView respectively.
- *Mobile context.* For each sensor managed by iOS platform, we defined a *iOSContextVariable* extending the *MobileContextVariable* Mobile IFML class, capturing the readings of device's sensor.

The *Code Generator for iOS* depicted in figure 6, consists of a M2M and M2T transformations applied in sequence. The M2M, conceptualized through a set of *ATL* [3] rules,

[3]https://www.eclipse.org/atl/

Figure 6: PIM-to-PSM-to-native iOS code: the app's requirements are specified in mobile IFML and UML models. The IFML model is then mapped into an IFML-iOS model (PSM level) from which the app code (the Objective-C code) is generated.

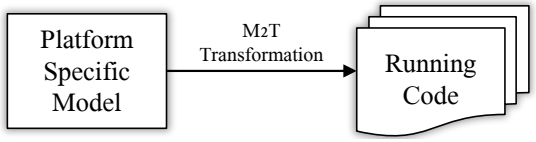

Figure 7: PSM-to-NC: the app's requirements are specified in a platform specific model from which the app code is generated.

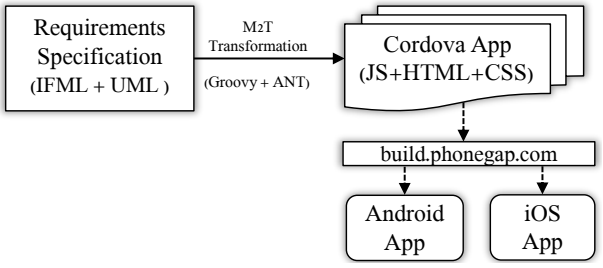

Figure 8: PIM-to-CPC: the process for code generation for PhoneGap. The app's requirements are specified in Mobile IFML and UML models. The M2T rules are applied to those models to generate the files required by *phonegap builder* to produce the app files.

is applied to the models describing the generic behavior of the app to produce a corresponding iOS model refined with iOS-specific information. The M2T transformation, conceptualized through a set of Xtend templates, is applied to that refined iOS model to generate the app code (*Objective-C code*).

4.3 PSM-to-NC: Native Apps directly from the PSMs

The code generation options presented in previous sections start from the PIMs and produce the app's code by applying a chain of M2M and M2T transformations. Nevertheless, another option is to skip the PIM level, specially when targeting a single platform since in that case there's no global model that can be reused across the target platforms.

The PSM-to-NC approach states that the app's requirements are directly specified in a platform specific-manner, i.e the model describing the app contains all the information required to implement an executable code in a given development platform (Figure 7).

4.4 PIM-to-CPC: Generation of Cross-Platform Apps from PIM

Providing a native implementation might not be always preferred due to the development cost, in terms of resources and time associated with the development activity for each platform, even if this problematic is soften when using MDD-based approaches. Using a target which is already *cross-platform* ***per se***, simplifies the options presented in previous paragraphs by reducing the time to market and the development costs at *the cost of losing the complete control of the native part of the apps.*

The PIM-to-CPC option, which generates the code for cross platform frameworks from a PIM is described through an example which generates the code for PhoneGap framework. *PhoneGap* is a web based mobile development framework, based on the open source Cordova project. It allows using the standard web technologies (HTML5, CSS3, and JS) for cross platform development. PhoneGap produces a binary app archive (ipa file for iOS, apk file for Android, xap file for Windows Phone, and so on) that can be distributed through standard app ecosystems such as iTunes Store, Android Market, Amazon Market, and Windows Phone Marketplace. The *PhoneGap generator* depicted in figure 8 takes as input the app's models and generates the JavaScript, HTML5 and CSS optimized for Phonegap framework. The generated code is wrapped in the Cordova container and then sent to the *Build PhoneGap* which produces the app files.

4.5 PIM-to-FSM-to-CPC: Cross-Platform Apps from PIMs through FSMs

The code for cross platform frameworks can be also generated from FSM (a PSM in which the *Platform* in MDA, is actually a Cross-Platform Framework for mobile apps development). The model describing the app contains in this case the specific information regarding the behavior and the structure of the app on a specific cross platform framework. Figure 9 shows the approach which generates the apps'code from a FSM derived from a PIM. The same result can be reached by using directly (without passing through PIM level if desired) a FSM allowing the specification of the app in terms of the targeted cross platform framework.

5. DISCUSSION

This section compares the code generation approaches presented in previous sections and provides the general guidelines for selecting the winning approach for a specific scenario. It ends by presenting a concrete case in which the guidelines have been applied to select the best approach for WebRatio based on the requirements and the expertise of the company.

5.1 Effort Analysis for the Different Approaches

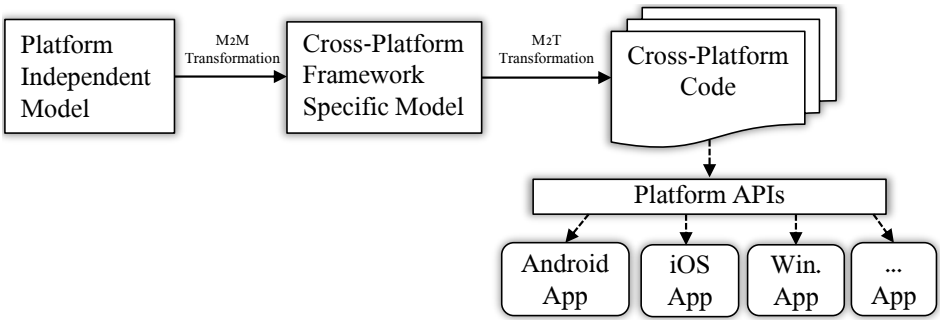

Figure 9: PIM-to-FSM-to-CPC: the app's requirements are initially specified in a PIM. The requirements are then refined in a FSM which specifies also the information about the targeted cross-platform framework. The code required by the framework to produce the app's files (one for each deployment platform) is generated from that FSM.

Table 1 provides a summary of the effort involved in the development of the needed languages and transformations for each approach. For the purpose of our analysis, it doesn't make sense to mention the exact number of days to complete each task since the exact effort is context (expertise of developer, tools, etc.) dependent. Based on our experience, we classify the effort in three categories: Low, Medium and High. Where *Low* represents the effort less than 10 man-days, *Medium* represents effort between 10 and 25 man-days while *High* corresponds to more than 25 man-days effort. For the *PIM language*, the effort depends on whether the language is designed from scratch or as an extension of an existing standard. Based on our experience in developing a new PIM-level language (the IFML standard [3]) and its mobile extension (the Mobile IFML [4]), developing a new PIM language is a *costly* task while extending an existing standard requires *low effort*.

Approach	**PIM**		**PSM**	**MTM**	**M2T**
	New PIM	**Ext.PIM**			
PIM-to-NC	H	L	-	-	H
PIM-to-PSM-to-NC	H	L	M	H	M
PSM-to-NC	-	-	H	-	M
PIM-to-CPC	H	L	-	-	M
PIM-to-FSM-to-CPC	H	L	M	H	L

Table 1: Effort analysis for code generation approaches. Each item is denoted with level of effort involved: low (L), medium (M), and high (H).

Our tests of all the five alternatives, considered in this research, showed that there is no approach better than others in absolute terms but allowed us to make the following classification of the best scenarios per alternative:

(1) PIM-to-NC. In this option, all the platform-specific details are embedded in the code generator. That makes this approach faster (since there is no need to generate the PSM). But embedding all the details of the app in the code generator reduces its flexibility. Typical data-driven native apps would be better suited for this option.

(2) PIM-to-PSM-to-NC. This approach could be better when having to develop native apps, with complex functionalities. In fact, the PSM level allows to benefit from the specificities of the platform at the level of models and thus allowing more tuning options with respect to option (1).

(3) PSM-to-NC. This option is better when developing native apps for one specific platform.

(4) PIM-to-CPC. This option is better when developing the apps in which cross platform richness is more important than high performance. The cross platform approaches are generally advisable for the companies with limited resources. Several researchers concur that it is unlikely that a single vendor will dominate the future mobile-centric world. Therefore, for these companies, it is better to loose in native functionalities to cover the entire market.

(5) PIM-to-FSM-to-CPC. Similarly to the approach (4), this option is suitable when developing the apps in which cross-platform richness is more important than high performance. The framework specific model allows to fully benefit from the cross platform framework specificity at the level of models. The option (5) is preferable to the option (4) when having a working team with low experience in the targeted cross-platform framework since is much more easier to control the app at model level than during the generation.

5.2 Industrial Experience: The Case of WebRatio

Webratio is an SME with a mature experience in model driven development for web application. The company has around 60 engineers specialized in web technologies and web application modeling using the IFML standard in a proprietary environment. The strong requirements for the company can be summarized as follows: (i) the tool shall produce the apps available for different platforms (initially iOS and Android shall be supported); (ii) the apps shall access the device's common sensors; (iii) the apps shall be distributed through the standard apps' ecosystems; and (iv) the typical customers of WebRatio are business users.

According to the classification of the best scenarios, the PIM-to-CPC and PIM-to-FSM-to-CPC approaches fit the requirements but the later requires an extra effort related to the PSM level which does not bring additional benefits

to Webratio (since it is specialized in Web technologies). We advised the *PIM-to-CPC* approach, using *PhoneGap* as cross-platform development framework for the following main reasons:

- Cross-platform is more important than high performance. The company is not interested in full native apps;
- Typical customers are business users that privilege practical and usable UI with respect to fancy graphics;
- Heitkötter et al [8] advised PhoneGap as the best cross-platform framework when the native UI is not a strong requirement; and
- The team is experienced in app modeling and in standard Web technologies.

The approach we advised to WebRatio is actually implemented in the company's newly released product *WebRatio Mobile Platform* (released February 17, 2015). Its implementation required a development effort of 200 man-days.

6. CONCLUSIONS

In this paper we presented different approaches to generate the code for mobile apps from the models describing those applications. We presented the results of a comparative study conducted to identify the best trade-off between different code generation strategies. Our study showed that there is no approach better than others in absolute terms but provided some useful guidelines that helped us to identify the best strategy for the WebRatio company in particular. The approach we advised to WebRatio is implemented in the company's newly released product *WebRatio Mobile Platform* (released Feb 17, 2015). Future works will cover more empirical studies, based mainly on the expected feedback from the two end-user companies (partners of the AutoMobile project), to validate and refine the solution in a variety of scenarios (company size and domain).

Acknowledgement. This work was funded by the AutoMobile EU 7th FP SME Research project (http://automobile.webratio.com).

7. REFERENCES

[1] Heiko Behrens. MDSD for the iphone: developing a domain-specific language and IDE tooling to produce real world applications for mobile devices. In *SPLASH/OOPSLA*, pages 123–128, 2010.

[2] 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.

[3] 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.

[4] 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.

[5] Nadia Elouali, Xavier Le Pallec, José Rouillard, and Jean-Claude Tarby. MIMIC: leveraging sensor-based interactions in multimodal mobile applications. In *CHI*, pages 2323–2328, 2014.

[6] Mirco Franzago, Henry Muccini, and Ivano Malavolta. Towards a collaborative framework for the design and development of data-intensive mobile applications. In *MOBILESoft*, pages 58–61, 2014.

[7] 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.

[8] Henning Heitkötter, Sebastian Hanschke, and Tim A. Majchrzak. Evaluating cross-platform development approaches for mobile applications. In *WEBIST*, pages 120–138, 2012.

[9] Henning Heitkötter, Tim A. Majchrzak, and Herbert Kuchen. Cross-platform model-driven development of mobile applications with md^2. In *SAC*, pages 526–533, 2013.

[10] Hagen Höpfner, Jonas Pencke, David Wiesner, and Maximilian Schirmer. Towards a target platform independent specification and generation of information system apps. *ACM SIGSOFT Software Engineering Notes*, 36(4):1–5, 2011.

[11] Gerrit Meixner, Kai Breiner, and Marc Seissler. Model-driven useware engineering. In Heinrich Hussmann, Gerrit Meixner, and Detlef Zuehlke, editors, *Model-Driven Development of Advanced User Interfaces*, volume 340 of *Studies in Computational Intelligence*, pages 1–26. Springer, Heidelberg, 2011.

[12] Gebremariam Mesfin, Gheorghita Ghinea, Dida Midekso, and Tor-Morten Grønli. Evaluating usability of cross-platform smartphone applications. In *MobiWIS*, pages 248–260, 2014.

[13] 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.

[14] David Raneburger, Roman Popp, Sevan Kavaldjian, Hermann Kaindl, and Jürgen Falb. Optimized GUI generation for small screens. In Heinrich Hussmann, Gerrit Meixner, and Detlef Zuehlke, editors, *Model-Driven Development of Advanced User Interfaces*, volume 340 of *Studies in Computational Intelligence*, pages 107–122. 2011.

[15] Jean Vanderdonckt. A MDA-compliant environment for developing user interfaces of information systems. In *CAiSE*, pages 16–31, 2005.