Approaches in Development of Multi-platform Mobile Applications: State of the Art

Zlatko Stapić[1], Luis de Marcos Ortega[2], José María Gutiérrez Martínez[2]

[1] University of Zagreb, Faculty of Organization and Informatics
Pavlinska 2, 42000 Varaždin, Croatia
zlatko.stpaic@foi.hr
[2] University of Alcalá, Computer Science Department
Ctra. Barcelona km 33.6, 28871, Alcalá de Henares (Madrid), Spain
{luis.demarcos, josem.gutierrez}@uah.es

Abstract. Software development teams are faced with the lack of portability and reusability during the development of mobile applications for two or more target platforms. The development of mobile application for second and every subsequent platform usually means a new project with a need to repeat almost all phases defined by a chosen methodology but without the possibility to reuse already defined artifacts. The results and efforts of scientific and professional community have important drawbacks which, along with stated fragmentation problem make the paradigm "code ones – run anywhere" useless for mobile application development. This article aims to summarize these approaches and efforts and to critically observe their advantages and disadvantages. The results show that new approach is needed in solving this reusability, interoperability and development efficiency problem.

Keywords: mobile development, multi-platform, fragmentation problem

1 Introduction

The development of mobile applications differs from development of traditional desktop or web applications in several important aspects [1], [2]. According to Rahimian and Ramsin [1] among other challenges, the designer of a software system for mobile environments has to cope with portability issues, various standards, protocols and network technologies, limited capabilities of devices and strict time-to-market requirements. Additionally, development of mobile systems is a challenging task with a high level of uncertainty, and according to Hosbond [3], it is the result of two main set of challenges that should be addressed in the domain of mobile systems development. These are *business* related challenges (e.g. tough competition, conflicting customer interests, establishment of revenue-share models etc.) and *development* specific challenges (e.g. rapidly changing technology, lack of standardization, integration with existing systems etc.).

In respect to multi-platform mobile applications development which is in focus of this paper, further research made clear the existence of specific issues that must be addressed in order to overcome these challenges while developing multi-platform mobile applications. First issue that should be addressed is the usage of methodology [1], [2], [4]. Classic or agile software development methodologies should be adapted for development of mobile applications as the existing ones do not cover the specific mobile targeted requirements [4]. Another issue is the use of platform specific and dependent development environments. These environments are not interoperable in a single way [5]. Additionally, an important issue is a number of different (specific) devices which are based on the same platform [5], [6], [7]. This includes various hardware implementations and operating systems capabilities with support on different API levels [5] and which are based on different programming languages [6]. This problem is also known as *fragmentation problem* [5], [6], [7], which says that a fragmentation of APIs exists even within a single platform. According to Manjunatha et al. the fragmentation problem forces the developers of mobile applications to focus on only specific platforms and versions [6]. As the development of mobile applications primarily aims the wide range of users, development for only specific platforms and versions is not an option and the development teams reach for different solutions to this problem. The ideal (i.e. still nonexistent) solution would be to code once and to deploy (run) the same code to all target platforms. The fragmentation problem is a result of mobile industry being continuously highly technology-driven, which means that the focus is on innovation instead of standardization. This problem was recognized several years ago by Hosbond [3]. Subsequently, testing becomes a greater problem as simulated or emulated devices usually do not provide full functionality or are incapable of creating a real life test scenarios [7]. The testing on physical devices is usually too expensive if used to cover up all important devices and their capabilities. Finally, the deployment and maintenance phases should not be forgotten too, while both of them bring a fresh set of specific requirements that are mainly defined by mobile device producers and their application stores.

This paper will deal with the approaches to solve stated *fragmentation problem.* As there are a number of different projects and solutions offering a quick deployment to different target platforms, this paper will summarize these approaches and discuss their advantages and disadvantages. In order to do that, chapter two will introduce the problem of multi-platform mobile applications development, chapter three will present existing solutions and chapter four and five will give discussion and conclusion from the authors' point of view.

2 The Problem of Multi-platform Development

Let us presume the real business scenario in which a development company wants to produce a classic business or non-business application that should be runable on a several different mobile platforms and devices. The standard approach would be to create several different teams, each team targeting one specific platform, to adopt several development methodologies or at least different methods, each of them applicable for a specific platform and to produce characteristic outputs which will

satisfy the requirements specified by the mediatory application stores or markets (see Fig. 1). More experienced teams will probably try to perform as many as possible unique activities that should be similar or same across all platforms, or will even try to perform whole Model Driven Development approach through all phases except in creation of Platform Dependent Model and its implementation.

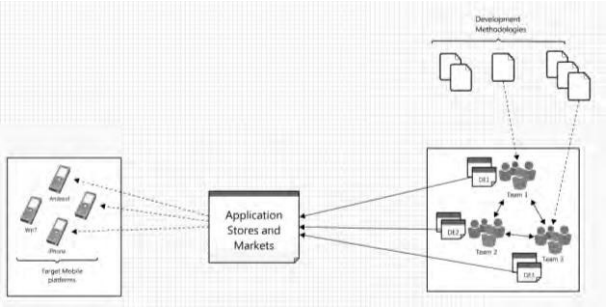

Fig. 1. The big picture of the problem

But, the big question still remains. Is it possible to make this process easier in the sense of development, interoperability and reusability? Is it possible to code once and run on different target platforms?

Unfortunately, it is not possible to code once and to run on any mobile device. This slogan, according to Ridene et al., is not true even for Java [7]. The trends in mobile industry show us that this will not be possible in the short-term future, as mobile platforms are still closed, locked-in [6], and devices are dependent on them.

3 Existing Solutions

As in the past year or two the problem of mobile applications development for multiple target platforms became important in the scientific and in the professional community, different results are visible in the form of several existing systems and projects that fairly enough enable the development teams to develop for several target platforms. These approaches are enumerated in this chapter.

3.1 Using Mediatory Transform Engine

First approach defines a use of a mediatory language or just mediatory transform engine to code for several target platforms. This approach is also called a *cross-compilation* and can be defined as process of generating native code to run on multiple platforms [8]. The most influential projects implementing this approach are MobiCloud [6], [9] from Kno.e.sis Research Group [10], Rhodes [11], Amanquah & Eporwei code generator [12] et cetera. As Fig. 2 shows, reaching for this solution will bring some improvements to development teams. First of all, project team or project teams will use a single proprietary or open-source programming language and will try

to implement desired functionality. The mediatory transform engine will then produce a platform specific code which can be tested and deployed through specific application store or market.

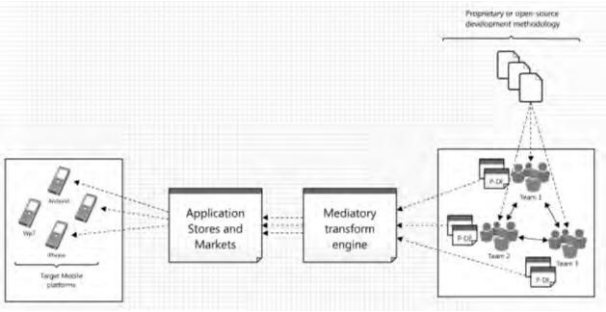

Fig. 2. Architecture of some existing solutions

There are several examples of systems with described functionality. Some of them (e.g. MobiCloud) use their own domain specific language (DSL) to transform into platform specific source (or even executable) code. The other systems [12] transform code written in well-known languages to specific source (or executable) code.

3.2 The Use of Native Application Adapters

Another possible solution to the given problem could be the introduction of adapter applications (adapters) as native applications for every target platform [5]. According to Agarwal et al. this is one of two main techniques for handling fragmentation. As standardization of APIs in mobile world is still not possible, the usage of programming techniques whereby the interface calls are wrapped, i.e. abstracted, in distinct modules which are then ported across the platforms, is left as the other solution. For example, the same authors are proposing a MobiVine as a solution to handle fragmentation of platform interfaces. Specifically, the authors have identified that the fragmentation of mobile platform interfaces results in (1) different syntax and semantics, (2) usage of platform specific data structures and properties, (3) throwing platform specific exceptions and (4) it is also characterized by inconsistencies in implementation by different vendors. This has bearing on the portability of mobile applications across multiple platforms.

The authors of MobiVine evaluate the usage of MobiVine as middleware layer and they discuss the achieved improvements in terms of improving platform and language portability, reducing code complexity, making maintenance easier and making performance by a negligible fraction slower. But they also conclude that MobiVine framework should be extended to cover other platform interfaces (like working with contact list information), to include other platforms, and to make the concept of proxy model broader by studying its applicability to other forms of mobile fragmentation, e.g. screen size and resolution.

Another well-known wrapper is PhoneGap [13]. The applications written in HTML/HTML5, CSS/CSS3 and JavaScript are wrapped with PhoneGap and then

deployed to multiple platforms. The developers could use free, open-source framework to access some of the native APIs.

After the Adobe Corporation acquired the original PhoneGap's creator Nitobi Company, they also announced that they will offer developers the choice of using two powerful solutions for cross-platform development of native mobile apps, one using HTML5 and JavaScript with PhoneGap and the other using Adobe Flash® with Adobe AIR®. On the other side, original PhoneGap approach is not changed and as the application takes on extra complexity, more involved logic will require spending more time on application behavior with specific devices. Even when the same code base is used when developing for multiple platforms, the separate prepare & build and sometimes porting steps should be performed to produce the version targeting multiple platforms. According to [14] more complicated applications are keen on "surprising" developers during the porting process. Finally, there are many different guides and recommendations that should be followed while developing this way [14], and we generally can conclude that taking all of them into consideration means learning a new programming and development style which is as difficult as learning a new programming language from the scratch.

So, generally, the adapter-based approach requests that the adapters should be pre-developed and published to the specific application store, or as in the case of PhoneGap, deployed along with the application [13]. The general idea of creating adapter is to create a *platform specific application* that will bi-directionally convert the specific interfaces of the target platforms (left-side) into one unique interface that could be used to communicate with different applications (single, right-side). Every single adapter converts a different target interface to unique (same) interface, which means that one application really could be imported into one or more different adapters and run under one or more different platforms. The mentioned application could be stored on any web server or even on a cloud as it is shown in Fig. 3.

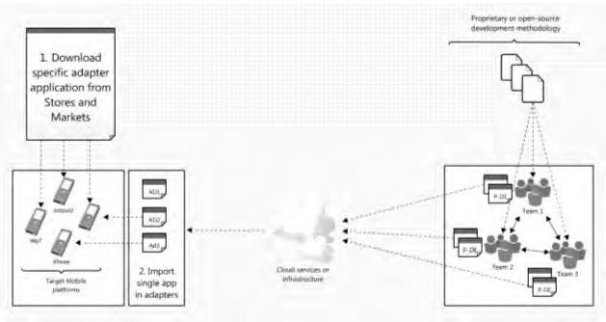

Fig. 3. Architecture of some possible solutions

3.3 Other Approaches

Finally, there are other attempts and efforts that are undertaken to over-come mobile platform and interface diversity and fragmentation. These efforts include for example:

creation of extensions to Java platform, through Java Specification Requests (JSRs) (like JSR 248: Mobile Service Architecture [15] or JSR 256: Mobile Sensor API [16]), or the development of Wholesale Applications Community (WAC) APIs and applications (Apps).

JSRs are designed to provide the set of APIs for specifically targeted use (e.g. for mobile service architectures or mobile sensors). But, according to Agarwal et al. [5], along with standard Java Micro Edition (Java ME), mobile platform developers in practice choose to include different set of JSRs and that results in diversity even among these devices. On the other side, WAC is an open, global alliance of leading companies in the mobile telecommunication industry with a goal of providing a different operator network APIs through single cross-operator API platform. Specifically, this platform currently offers WAC Apps framework and WAC Payment API. WAC Apps aims to help create the mobile apps quicker by using existing, familiar web technologies and tools through direct access to mobile device functionality. WAC Payment API aims to enable developers, through the set of developed Software Development Kits (SDKs) for multiple platforms, to be able to access the operator billing capabilities through single API. Although this API is useful in some cases, currently it covers only payment options and could be used for Android, PhoneGap, PHP and JavaScript/HTML5 platforms [17]. WAC announced that they plan to launch additional network APIs over time to provide the developers with further opportunities to create richer applications [18].

Another well known approach to solve platform fragmentation is to use only web technologies and web standards. The applications developed in this way are executed by the Internet browser on the device [8]. This approach became really popular, and it is good for a specific set of mobile applications that do not require any specific device capabilities which are used only by executing the native applications.

4 Discussion

All stated approaches and projects that implement them have their advantages and disadvantages. The idea of having mediatory transform engine that transforms source code to specific platforms depends on the efforts that were put into the development of the transform engine. The engine depends on specific platforms and available APIs, and by definition, DSL caters only to a specific domain [6]. Even if there is possibility to enrich the engine with transformation procedures to all existing APIs, there is an important problem of platform incompatibilities. For example, it is not possible to use multithreading in Windows Phone 7 but in the other platforms it is possible and even desirable, or Android does not provide thread sync mechanisms as Symbian does. The other drawbacks of this approach could be the necessity to learn a specific DSL, the boundaries defined by the use of any specific languages, the lack of control of generated source code, the lack of control of user interface design [6], the problems with testing and so on and so forth.

On the other side, there are two possible scenarios that could be implemented by adapters' developers. (1) The adapters could be 100% aligned by the mean of common interface and this scenario will reduce the number of teams (presented in

Fig. 3) to one. This would be a great achievement, but on the other side there is one big drawback too. The functionality of the future applications will be reduced to the common features that all target platforms support and to the common features that are implemented into the adapters for all target platforms. This brings us to the problems presented in previous paragraph and this also makes this scenario rather unlikely to be feasible. (2) The other scenario will introduce some differences in the adapters by the mean of common (right-side) interface. If the mentioned interface will not be the same for all platforms, the use of such adapters will provide a more specific functionality on mobile applications, a more feasible scenario, but also will bring the need to develop more or less different applications for each target platform.

Almost all drawbacks stated for existing solutions that introduce transform engine are also present in this possible solution. The mentioned PhoneGap [13] platform allows the development of native applications with web technologies (HTML/HTML5, CSS/CSS3 & JavaScript) enriched with the given set of APIs. According to PhoneGap Documentation (from 15th Oct 2012) this platform supports *back button event* only on *Android* platform despite the fact that such event exist in several other platforms as well. Although there is some space for research in this area, especially in the field of interface transformation, the improvements that will bring the process of development of demanding applications for multiple target platforms through this approach are also hardly achievable and even feasible.

5 Conclusion

As it can be seen, there are several rather different approaches that scientists and experts are taking to solve the problem of developing for multiple platforms. Each one of them has its own advantages and disadvantages. But still, one issue remains and is common to almost all of these approaches. It is impossible to create a transform engine, or adapter application that will use all advantages of all target platforms and that will provide the range of possibilities as native development environments do. Also, if we want to preserve the capability of teams working on the open-source projects, it is necessary to give them the possibility to work in a native development environment and to develop by using a programming language they prefer most.

In order to provide such possibilities, we encourage the development of new approaches that will enhance interoperability among teams working on the same application but on different (and native) development environments. The work on the native development environments will provide the teams with the full advantages of using the native APIs, the native test environments and the native generators of the executable code.

Acknowledgments

The research presented in this paper was conducted as a part of a PhD research and is partially founded by the scholarship granted to the first author from Croatian Science Foundation.

References

1. V. Rahimian and R. Ramsin, “Designing an agile methodology for mobile software development: A hybrid method engineering approach” in Proceedings of Second International Conference on Research Challenges in Information Science, RCIS (2008), Marrakech, pp. 337–342 (2008)
2. A. C. Spataru, “Agile Development Methods for Mobile Applications” PhD Thesis, University of Edinburgh, The University of Edinburgh, Edinburgh (2010)
3. J. H. Hosbond, “Mobile Systems Development: Challenges, Implications and Issues” in Mobile Information Systems II, vol. 191, J. Krogstie, K. Kautz, and D. Allen, Eds. Springer Boston, pp. 279–286 (2005)
4. H. J. La and S. D. Kim, “A service-based approach to developing Android Mobile Internet Device (MID) applications” 2009 IEEE International Conference on Service-Oriented Computing and Applications (SOCA), vol. 00, no. MID, pp. 1–7 (2009)
5. V. Agarwal, S. Goyal, S. Mittal, and S. Mukherjea, “MobiVine: a middleware layer to handle fragmentation of platform interfaces for mobile applications” in Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware, New York, NY, USA, pp. 24:1–24:10 (2009)
6. A. Manjunatha, A. Ranabahu, A. Sheth, and K. Thirunarayan, “Power of Clouds in Your Pocket: An Efficient Approach for Cloud Mobile Hybrid Application Development” 2010 IEEE Second International Conference on Cloud Computing Technology and Science, no. 2, pp. 496–503 (2010)
7. Y. Ridene, N. Belloir, F. Barbier, and N. Couture, “A DSML For Mobile Phone Applications Testing” in Proceedings of 10th Workshop on Domain-Specific Modeling in SPLASH, France (2010)
8. L. Maaløe and M. Wiboe, “A Platform-Independent Framework for Application Development for Smart Phones” (2011)
9. Services Research Lab and Metadata and Languages Lab, “Cloud-Mobile Hybrid Application Generator” MobiCloud (2011), http://mobicloud-classic.knoesis.org/.
10. Kno.e.sis Research Group, “Welcome to Kno.e.sis” (2011), http://knoesis.wright.edu/.
11. Rhomobile, Inc., “Smartphone Enterprise Application Integration, White paper” (2011), http://tiny.cc/rhomobile.
12. N. Amanquah and O. T. Eporwei, “Rapid application development for mobile terminals” in 2nd International Conference on Adaptive Science & Technology (ICAST), Accra, Ghana, pp. 410–417 (2009)
13. PhoneGap, “Take the pain out of compiling mobile apps for multiple platforms” PhoneGap Build (2011), https://build.phonegap.com/
14. A. Lunny, Phonegap beginner’s guide: build cross-platform mobile applications with the PhoneGap open source development framework. Birmingham, UK: Packt Publishing Limited (2011)
15. E. Bektesevic and E. Rysa, “JSR 248: Mobile Service Architecture” The Java Community Process(SM) Program - JSRs: Java Specification Requests (2008), http://jcp.org/en/jsr/detail?id=248.
16. P. Niemela, “JSR 256: Mobile Sensor API” The Java Community Process(SM) Program - JSRs: Java Specification Requests (2009), http://jcp.org/en/jsr/detail?id=248.
17. WAC Application Services Ltd, “WAC Payment API SDKs” WAC Payment API Resources - Developer Website (2012), http://www.wacapps.net/sdks.
18. WAC Application Services Ltd, “WAC APIs” WAC APIs - Developer Website (2012), http://www.wacapps.net/wac-apis.