Establishing Common Architectures for Porting Mobile Applications to new Platforms

Tilmann Stehle, Matthias Riebisch
{stehle, riebisch}@informatik.uni-hamburg.de
University of Hamburg, Department of Informatics

1 Introduction

Currently the market of mobile operating systems is divided between several platforms and developers have to target more than one in order to achieve a large number of users [4]. Reimplementing an existing application for a second platform is no trivial task, though. Developers need to learn the second platform's API, concepts and paradigms such as an app's life cycle, the platform's caching mechanisms and the like. On the one hand, different technologies and frameworks ease cross-platform development, such as Xamarin [8], PhoneGap and many others [5]. On the other hand, reimplementing an existing app with such a framework and discarding the original one is likely to break the maturity level and the users' acceptance in consequence. Additionally there are reasons not to use tools and languages other than those supported by the operating system producers: The native API is maintained continuously and the IDEs are strictly aligned to the latest technology.

This paper introduces a porting approach that aims at easing the maintenance of the original and the emerging implementation for the target operating system. Furthermore it contributes to the preservation of the flexibility of native development and the maturity level of the existing original app. This is achieved by restructuring the first app in order to subject most of the code to a semi-automated porting. It establishes a common architecture of the original and the ported implementation. This way it reduces the maintenance effort compared to a complete reimplementation. The following sections introduce this process and shortly specify related work. Finally, the currently conducted evaluation and future work is sketched.

2 The Mobile Porting Process

The goal of the process proposed in this section is to establish a common code base or at least a common structure for as much platform independent code as possible. This way, developers can conduct future maintenance tasks in a structurally equivalent way for both implementations and an experience portability [7] is established.

In the first step, both platforms are analyzed with regard to the API they provide for OS functionality. Considering Android as the original platform, the classes *Activity* as well as *AsyncTask* are exemplary parts of this platform API. They are used and subclassed to provide user interaction and multithreading.

Secondly, the original app's architecture is analyzed in order to find dependencies such as usage and inheritance relationships to the platform API as well as to external libraries, that are not available for the target platform. Furthermore, one has to assess, which functionality is intended to behave differently on the target platform. An example of such inherently platform specific functionality regularly is the user interface, since users of different platforms are used to different interaction and design concepts. The necessity of the identified platform dependencies is assessed and the effort for removing or isolating them from platform independent functionality is estimated in the third step.

During the fourth step, avoidable or even inappropriate platform dependencies are removed from the original implementation, for example by replacing a superfluous external library from the class path or by using a local variable instead of the platform's configuration mechanism.

In the subsequent separation step, one extracts as much platform independent code from code with platform dependencies into new, portable artifacts. This is done by building a common interface for the operating system functionality, which is implemented by platform specific code for both platforms. The platform independent codeparts have to be moved to new platform independent artifacts, which only have dependencies to the newly created interface. The interface can be implemented by adapters that either use or inherit from the specific platform classes. Depending on the considered structure one can argue, if inheritance or delegation is the right choice ([3], p.20; [6]). As a result of this separation, more platform independent functionality is freed from platform dependencies. At the same time, platform dependent code artifacts are isolated as recommended by [7]. It is important, that existing tests and documentation artifacts are adapted during the conducted changes in order to protect the existing functionality. A tool support for the refactorings can help to avoid the introduction of bugs in this phase.

After this separation, the platform independent code is transformed to the target language in the sixth step. In order to save effort for reimplementation and maintenance, it is highly desirable to use automatic code transformation tools, if available. These can also be used to translate future code changes that are conducted during maintenance. The least tool support that should be used is to

extract structural models such as class diagrams from the original code and to create class skeletons in the target language automatically.
After the (semi-)automatic translation, the isolated platform specific code is reimplemented for the target platform. The new implementation provides the common interfaces developed in the separation phase.
In order to use platform specific functionality in platform independent code (e.g. to use multithreading in business logic), instances of platform independent classes must get access to platform dependent ones. Corresponding instantiation statements can be added to the transformed independent code, or dependency injection mechanisms are introduced to both implementations in order to extract the wiring of both code categories to a single platform dependent component. As a third option, platform dependent classes can instantiate and pass platform specific objects to platform independent ones, that they create. In consequence of this wiring step, the target platform implementation becomes runnable. A comparison of those wiring concepts is subject to future works.
As a final step, traceability links between the original code artifacts, corresponding diagram elements and their target platform pendants are established to represent dependencies in an explicit way, as an important part of design information. These links may as well be creates automatically during code translation.
The process can be applied incrementally to single features of the application, thereby augmenting the target implementation run by run. This way, general problems concerning the later process steps (e.g. inappropriate translations) will become visible, the second platform version becomes testable and the customer is able to see results much earlier.

3 Related Work

In [1], common abstract architectures among several platform implementations are discussed the separation of the user interface and data manipulation components from others is suggested. The authors do not address porting though.
[9] describes a process for porting desktop applications to an iOS-Device. In contrast to the process described here, it states the necessity of using a common programming language and consequently does not cover the benefits of a common structure of two implementations. Furthermore it does not discuss an incremental application of the process.

4 Evaluation and future work

Currently, the process is applied to test projects in order to find proper decoupling mechanisms that separate platform dependent from platform independent code. Additionally, a case study is undertaken, which applies it to a real world Android application that is being ported to iOS. In this contexts, an MVVM-like pattern (cf. [2], similar to MVC) has been applied by extracting presentation logic from *Activity* classes into *ViewModel* classes and business logic further down into model classes. The iOS-version reimplements the user interface, which is wired to the ported *ViewModel*. The asynchronous initialization of *ViewModels* has been subject to the separation of technical OS-functionality (multithreading) from business logic (model initialization) in this context.
Further research has to develop decoupling mechanisms for other commonly used framework functionality and to suggest corresponding refactorings leading to a common structure. These have to be analyzed with regard to the effects they have on possible future maintenance strategies. Additionally there is a need for refactoring tools that ease the establishment of that structure. Prototypes of those tools will be developed to show the applicability of the process in a larger scale. A categorization of dependencies with regard to the effort that is needed to remove or isolate them, is desirable as well.

References

[1] ALATALO, P., JRVENOJA, J., KARVONEN, J., KERONEN, A., AND KUVAJA1, P. Mobile application architectures. In *Product Focused Software Process Improvement*, M. Oivo and S. Komi-Sirvi, Eds., vol. 2559 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2002, pp. 572–586.

[2] FURROW, A. Introduction to mvvm. http://www.objc.io/issue-13/mvvm.html, 6 2014.

[3] GAMMA, E., HELM, R., AND JOHNSON, R. E. *Design Patterns. Elements of Reusable Object-Oriented Software.*, 1st ed., reprint. ed. Addison-Wesley Longman, Amsterdam, 1995.

[4] GARTNER. Gartner says worldwide traditional pc, tablet, ultramobile and mobile phone shipments on pace to grow 7.6 percent in 2014, 01 2014.

[5] HEITKÖTTER, H., HANSCHKE, S., AND MAJCHRZAK, T. Evaluating cross-platform development approaches for mobile applications. In *Web Information Systems and Technologies*, J. Cordeiro and K.-H. Krempels, Eds., vol. 140 of *Lecture Notes in Business Information Processing*. Springer Berlin Heidelberg, 2013, pp. 120–138.

[6] KEGEL, H., AND STEIMANN, F. Systematically refactoring inheritance to delegation in java. In *Proceedings of the 30th International Conference on Software Engineering* (New York, NY, USA, 2008), ICSE '08, ACM, pp. 431–440.

[7] MOONEY, J. Strategies for supporting application portability. *Computer 23*, 11 (Nov 1990), 59–70.

[8] PETZOLD, C. *Creating Mobile Apps with Xamarin. Forms*. Microsoft Press, 2014.

[9] SCHMITZ, M. *Strategie für die Portierung von Desktop-Business-Anwendungen auf iOS-gestützte Endgeräte*. BestMasters. Springer Fachmedien Wiesbaden, 2014.