A Multi-view Framework for Generating Mobile Apps

Scott Barnett, Iman Avazpour, Rajesh Vasa, John Grundy
Swinburne Software Innovation Lab (SILAB)
School of Software and Electrical Engineering, Faculty of Science, Engineering and Technology
Swinburne University of Technology, Hawthorn, Victoria 3122, Australia
Email: {sbarnett, iavazpour, rvasa, jgrundy}@swin.edu.au

Abstract—This paper demonstrates a multi-view framework for Rapid APPlication Tool (RAPPT). RAPPT enables rapid development of mobile applications. It employs a multilevel approach to mobile application development: a Domain Specific Visual Languages to define the high level structure of mobile apps, a Domain Specific Language to define behavioural concepts, and concrete source code for fine grained improvements.

I. INTRODUCTION

Despite the popularity of mobile applications (apps), building an app is a non-trivial exercise. Implementing an app necessitates knowledge of data modelling, database design, error handling, concurrency, platform APIs and 3rd party libraries. Mobile apps also have domain specific concerns that influence software architecture that need to be addressed [1]. Current IDEs and tools provide limited to no automation for addressing these concerns. In addition, building high quality apps requires careful attention to user interface design and to user experience. Thus, mobile app developers have to understand and reason about many different concepts to ply their craft.

In previous work, we presented the Rapid APPlication Tool (RAPPT), an MDD approach to automate some of the decision making process for developers [2]. In this paper, we demonstrate a multi-view framework for building apps with RAPPT. RAPPT’s multi-view framework aids professional app developers by incorporating multiple levels of abstraction into three different views: a DSVL, Domain Specific Textual Language (DSTL), and raw source code. Each view is at a different level of abstraction and is targeted to address different concerns of a mobile app. Developers use the DSVL to describe the high level features of the app then use the DSL to provide more details. Finally the developer edits the raw source code to add the final polish.

II. RAPPT’S APPROACH

RAPPT’s interface consists of three screens — a Designer for using the DSVL, a code editor for our App Modelling Language (AML) DSTL, and a Code Browser for viewing the generated app. The major parts of the interface are shown in Figure 1. (1) This section contains the UI components for the Package, Project and Download. When starting a new project with RAPPT the developer is prompted for the package and project name. This information can be updated by editing these text fields. Next to these fields is the Download button for downloading the source code for the generated app. (2)
are much more receptive to visual tools.

The PSM is then passed to the (G) Code Generator that generates the project structure for a new mobile app project. The Code Generator’s job is to map elements from the PSM to code templates. Generated code represents a working app that can be compiled and run on a device for early testing. The generated code (H) is designed to be modifiable and can be imported directly into an IDE. We have put measures that ensures the generated code resembles that written by a professional developers, e.g. use latest libraries, formatting and structuring. Developers modify the generated code to add the final polish to their app (I). This Final Polish may include custom styles and business logic as neither of these two concepts are covered by the DSVL or DSTL. Now the app is ready to be distributed in an app store (J).

III. Conclusion

We have introduced a new multi-view framework for RAPPT, to simplify the process of modelling and generating mobile apps. RAPPT provides multiple views, a DSVL and a DSTL, at different levels of abstraction. We have done an acceptance testing with 20 users. The results of this evaluation demonstrated acceptance of the approach among software and mobile app developers. It also indicated further improvements to the tooling aspect of the approach.

Acknowledgment

The authors wish to acknowledge support provided by Kiandra IT. NICTA is funded by the Australian Government through the Department of Communications and the Australian Research Council through the ICT Centre of Excellence Program. The authors wish to acknowledge support provided by NICTA.

References

