2015 IEEE/ACM International Conference on Software Engineering (ICSE 2015), Florence, Italy, 18-23 May 2015 © IEEE 2015

Bootstrapping Mobile App Development

Scott Barnett, Rajesh Vasa, John Grundy
Centre for Computing and Engineering Software Systems & Software Innovation Lab
Swinburne University of Technology, Melbourne, Australia
Email: sbarnett@swin.edu.au, rvasa@swin.edu.au, jgrundy@swin.edu.au

***Abstract*—Modern IDEs provide limited support for developers when starting a new data-driven mobile app. App developers are currently required to write copious amounts of boilerplate code, scripts, organise complex directories, and author actual functionality. Although this scenario is ripe for automation, current tools are yet to address it adequately. In this paper we present RAPPT, a tool that generates the scaffolding of a mobile app based on a high level description specified in a Domain Specific Language (DSL). We demonstrate the feasibility of our approach by an example case study and feedback from professional development team. Demo at: https://www.youtube.com/watch?v=ffquVgBYpLM**

I. Introduction

Modern IDEs for mobile app development have relatively primitive code generation functionality. This is especially true when starting a new mobile project, where code generation offers a large benefit. A developer has four options when starting a new project: a) Use the IDE to create a project skeleton that contains the build files and one or two code files, b) Use a custom generation script that provides a wider range of project types and libraries e.g. Android bootstrap[1], c) Copy code from a similar previous project, or d) Use a Model Driven Development (MDD) approach involving a VML [1] or a DSL [2] to generate an app. Out of these options MDD tools provide the most code generation, and theoretically offer the higher productivity gains. However, developers choose to use a combination of the other three approaches when building an app. One explanation for this is that code generation tools often have a bad reputation amongst developers due to their limited flexibility and poor quality output.

In addition to limitations within the tooling, developers have to contend and deal with two more common distractions when building a typical data driven app – a) The existing programming language, frameworks and SDK require many repeating blocks of boilerplate code, and b) Functionality implemented throughout an app is similar – in that data driven apps tend to mostly get data from a source and render it to the screen. Modern libraries alleviate these issues but they can only do so much. Similar functionality in an app such as rending a list with data involves many components that need to be wired up correctly and each instance has slight differences e.g each list displays different data. A data-driven app is an app that predominantly passes data between one or more APIs and screens displaying the necessary information to the user. All Data-driven apps typically perform the following tasks:

- Connect to an online API to send and receive data.
- Retrieve content, usually predominantly from a single API.
- Pass data between screens to allow users to perform business workflows e.g. transfer money between bank accounts.
- Validate user input and format data to display to the user.
- Capture some amount of input from the user and either pass it to an API or store it locally.

These tasks are repeated through out the entire app. The end result from developing a simple data-driven app that fulfils the list above is copious amounts of boilerplate code - often hundreds if not thousands of lines of code - that is to be written by hand. Furthermore, a high quality app requires developers to consider the concerns of a mobile app shown in Figure 1. Each of these concerns have an influence on the design and implementation of the app requiring careful attention from developers. The choice on the type of app to build also impacts the concerns to consider. There are three types of mobile apps, native, hybrid, or a mobile web app and the decision is determined by a range of constraints, including technical, cultural background of engineers, as well as the business requirements. A non-trivial proportion of teams choose to build a native mobile application typically motivated by user experience, or performance objectives. Our research focuses on improving the quality and productivity of those native app developers by removing the need to write boilerplate code.

We present RAPPT[2], a bootstrapping tool for professional Android app developers inspired by MDD techniques. RAPPT generates scaffolding which is built upon and extended by the developer to finish the app. Care has been taken to ensure that generated code conforms with standards of high quality code written by a professional. Developers describe an app using a high-level Domain Specific Language (DSL) from which RAPPT infers what needs to be generated. RAPPT needs to understand many Android concepts shown in Figure 1 in order to make smart inferences. Based on this large number of concepts we decided to only target a single platform. Our approach provides both the full capabilities of the Android platform and the productivity benefits anticipated by MDD.

II. Related Work

MDD has long been a an area of study that has promised increased developer productivity and reduced costs [3]. In

[1]http://www.androidbootstrap.com/

[2]http://rappt.io/

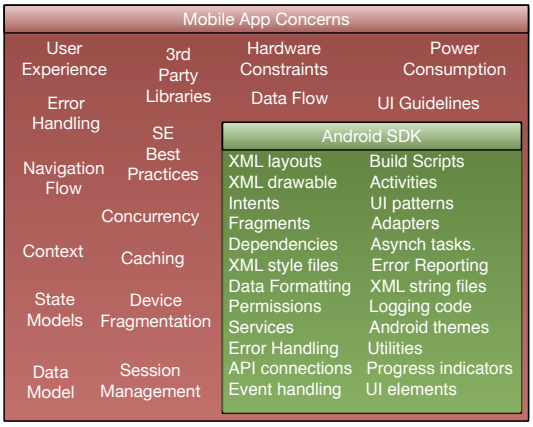

Fig. 1. Concerns facing mobile app developers.

the context of mobile apps, Khambati et al. [4] created a Visual Modelling Language to generate apps in the context of a health care plan. The visual language was intended to be used by health care professionals rather than developers and the generated app code was not open for modification. More recently, Nguyen et al.[5] developed a framework for generating the master/detail Android design pattern. The focus of this research was a small sub component of a final app rather than creating a fully working app. In recent work Ribeiro et al.[6] use a UML based approach to generate mobile apps. The authors describe a multi-stage MDD approach to app development that can generate a lot of boilerplate code for multiple platforms. Other researchers also view MDD, through the use of DSLs, as a way to address the issue of platform fragmentation [7], [8], [9]. These approaches focus on supporting a feature set that is common across all target platforms. There are also a number of commercial products for developing native apps Titanium[3] and Xamarin[4] being two of the more popular ones. Both tools provide an alternative language, for the purpose of cross-platform development, to build Android apps (Javascript in Titanium and C# in Xamarin). For developing an Android app the same level of effort is required as using the Android SDK as these framework APIs map 1-to-1 with the Android API.

Cross-platform frameworks, and most MDD approaches can handle inconsistencies between platforms in one of three ways: 1) Only support functionality common to all platforms, 2) Pick a base platform as the standard and implement missing features on the other platforms or 3) Provide a means to specify platform specific behaviour often with a plugin architecture for each platform. These options either limit the capabilities or greatly increase the complexity of using the tool. RAPPT avoids these limitations by focusing on a single platform. As a result RAPPT permits the inclusion of high level constructs specifically for Android development and the full capabilities of Android APIs are still available to developers by modifying generated code. In effect, we focus on a platform specific DSL, as in we consider the platform (e.g. Android) as part of the domain allowing us to generate native apps that map better to developer expectations.

[3]http://www.appcelerator.com/titanium/
[4]http://xamarin.com/

III. RAPPT

RAPPT differs from existing mobile app development tools and MDD solutions for the mobile application development domain. Key novel aspects of RAPPT include:

- RAPPT is aimed at improving productivity and quality for professional app developers. Most other app development tools target inexperienced developers or even non-technical end-users [10], [11] – limiting their scope.
- Many MDD solutions require generated code to be re-generated from their models or greatly limit (or prohibit) code modification [7]. RAPPT code is designed to be edited by software engineers.
- Rapid prototyping tools for mobile apps are designed to generate throw-way prototypes[5] again often limiting generated code modification. RAPPT code is designed to be incorporated into the finished app.
- MDD approaches for web and mobile app generation tend to try and support all features in the DSL, to obviate the need for modifying generated code, often limiting generated app flexibility and generality. RAPPT is designed specifically for generated code modification so that very high quality professional apps can be built leveraging RAPPT.

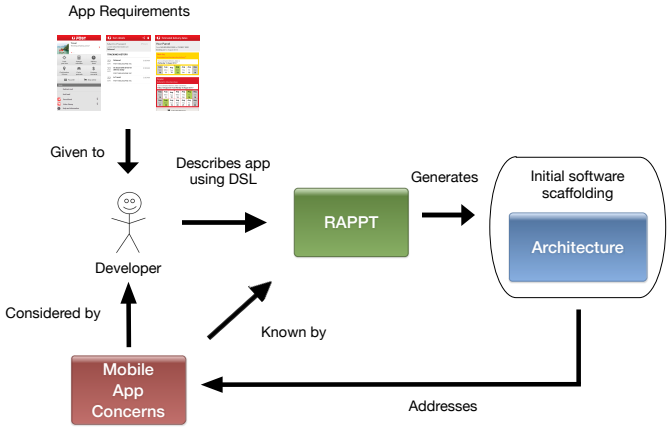

Fig. 2. RAPPT leverages an inference engine to assist developers in generating the scaffolding for a new Android app.

Developers describe the app using a high-level Domain-Specific Language (DSL) that allows them to express the app design declaratively and at a very high level, see Figure 2.

The RAPPT DSL was designed to be used by professional app developers, embody all of the key concerns of app development outlined in Figure 1, support high-level, declarative specification of all Android app features, and be used in professional quality code generation. Professional developers

[5]https://www.fluidui.com/

modify the generated code to add fine-grained tailoring of UI, layout, complex screen or processing logic, and use of third party (non standard) APIs.

IV. RAPPT Usage Example

We walk through an example of how RAPPT is used to bootstrap mobile app development. The example app is MovieBase[6] and can be downloaded from the Google Play Store[7]. MovieBase is a data-driven app that displays movie data from The Movie DB API[8]. Two screens showing the popular movies and the details of a movie are shown in Figure 3 on the left.

Consider Peter, a fictitious Android app developer who has been tasked with building the MovieBase app. On receiving the design for the app Peter begins to describe the app at a high level using RAPPT. Peter starts by describing app-wide features such as the landing page and global navigation pattern. Next Peter describes the data source for the app, The Movie DB API. RAPPT supports a couple of different authentication methods and the inference engine decides which to implement based on what information is specified by the developer. In this case, The Movie DB API requires all requests to be made with an API key which would be copied into the "api key" string. DSL code for the app and the API is shown in Listing 1. Peter then proceeds to describe all of the screens. The code for the screen displaying the popular movies is displayed in Listing 2.

```
app {
  landing-page MoviesScreen
  drawer navDrawerId {
    tab movieScreenTab "Movies" to MoviesScreen
    tab aboutTab "About" to AboutScreen
  }
}

api MovieDB "https://api.themoviedb.org/3" {
  api-key api_key "<api key>"
  GET popularMovies "/movie/popular" {list}
  GET movieDetail "/movie/{id}"
}
```

Listing 1. RAPPT DSL description code for the app wide details and API for the MovieBase app.

When Peter has finished describing all of the screens he generates the scaffolding for the app. The new project can be imported directly into the official Android development IDE, Android Studio[9]. Peter then builds the app and deploys it to his phone to get an idea of what was generated. RAPPT generates all code, build scripts and adheres to the conventions of the platform. Now that the code has been generated Peter sets about finishing off the app by building upon the scaffolding thankful he doesn't need to write everything from scratch – yet again! MovieBase required approximately 83 lines of code to implement in RAPPT generating 35 project files including source, resources and layouts or 1400 lines of code.

[6]https://play.google.com/store/apps/details?id=de.linuxwhatelse.android.moviebase&hl=en

[7]https://play.google.com

[8]http://docs.themoviedb.apiary.io/

[9]https://developer.android.com/sdk/installing/studio.html

```
screen MoviesScreen "Movies" {
  group moviesLayout {
    on-load {
      call MovieDB.popularMovies
    }
    list results : list {
      row rowId {
        label title : string
        label ratingId "Rating:"
        label vote_average : string
        label releaseDateId "Released on:"
        image poster_path : image
        label release_date : string
        button toDetail "Details" {
          to MovieDetailScreen pass idParam id : string
        }
      }
    }
  }
}
```

Listing 2. RAPPT DSL describing the popular movies screen.

V. RAPPT Architecture and Implementation

Internally RAPPT consists of three major components, the parser, an inference engine and a code generator. The relationship between these components is shown in Figure 4 and consists of four major steps. 1) RAPPT takes, as input, a description of the app to generate written in a custom DSL. 2) From this description an Abstract Syntax Tree (AST) is created and then fed into the inference engine. 3) The inference engine

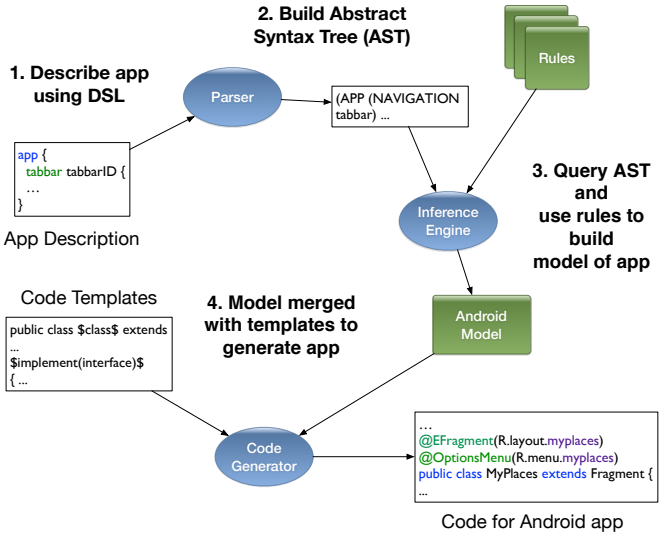

Fig. 4. The internal process of RAPPT that converts the DSL description to an Android project.

is responsible for constructing a model of the Android app to generate. The Android model is constructed by querying the AST and applying rules that determine implementation details. Inferring which Android components make up a given screen and which dependencies need to be added to the generated project are two example rules. There are also more complex rules that check if the app needs to implement a sophisticated UI pattern or check for a network connection. 4) Once the Android model has been created it is then passed to the

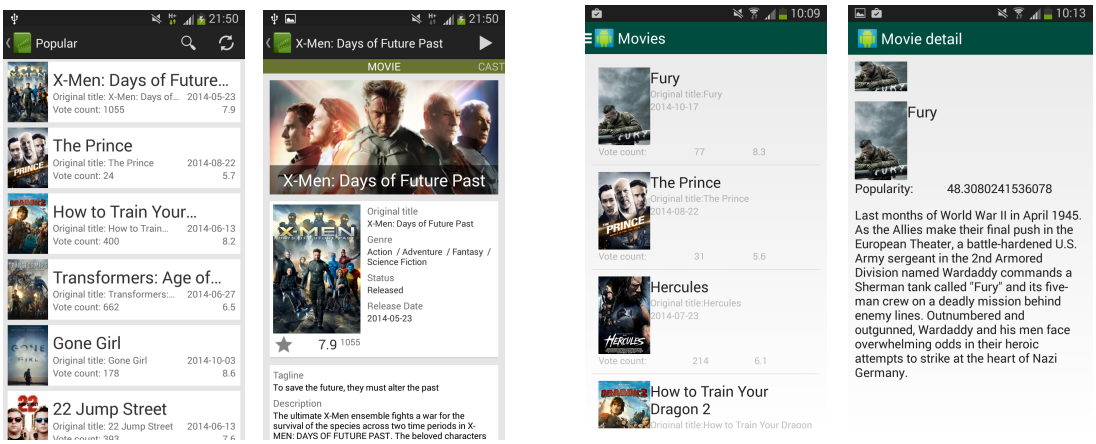

Fig. 3. MovieBase example screens (left) and RAPPT generated screens (right).

code generator component that also accepts a collection of templates. The Android Model is then used to populate the templates that constitute the Android app scaffolding.

A rich meta-model that captures all of the elements of an app was designed to underpin the RAPPT DSL. This meta-model was created from studying a wide range of existing apps that also informed the design of DSL and the tool chain. RAPPT and its meta-model were refined progressively by building many apps that highlighted gaps to be closed.

VI. Preliminary Evaluation

RAPPT was used in the development of the commercial app, Prompa available for download from the Google Play Store[10]. Informal feedback from developers indicated that the generated code was easy to work with and that RAPPT provided a significant productivity improvement. The main limitation cited was that RAPPT cannot be used after the code has been modified. In future work, we plan to evaluate the scalability of the DSL, the robustness of the underlying meta-model, and run trials with software engineers to ensure that the language is learable, and the generated code quality meets expectations, and finally ensure that the tool chain and the approach fits within the workflow of a professional mobile application developer.

VII. Conclusion

State of the art mobile app IDEs offer little code generation for bootstrapping development of a new app. Data-intensive apps in particularly would benefit from better code generation tools as they contain a significant portion of boilerplate code. Additionally, app developers must also strive to satisfy a number of often conflicting concerns of a mobile app. To address these concerns we present RAPPT an MDD based tool that generates the scaffolding for an Android app. RAPPT generates standard Android code maintaining the full capabilities of the framework. Initial feedback from developing the Prompa app showed that it increases developer productivity and is suitable for commercial projects. A more rigorous evaluation is planned to verify the concerns of a mobile app and to quantify the productivity increase.

[10]https://play.google.com/store/apps/details?id=net.prompa.production.release&hl=en

Acknowledgement

The first author would like to thank Swinburne University for scholarship support. We thank Prompa and SSI/NICTA lab members for their feedback on RAPPT.

References

[1] D. L. Moody, "The physics of notations: Toward a scientific basis for constructing visual notations in software engineering," *IEEE Transactions on Software Engineering*, vol. 35, no. 6, pp. 756–779, 2009.

[2] M. Fowler, *Domain-specific languages*. Pearson Education, 2010.

[3] M. Brambilla, J. Cabot, and M. Wimmer, "Model-driven software engineering in practice," *Synthesis Lectures on Software Engineering*, vol. 1, no. 1, pp. 1–182, 2012.

[4] A. Khambati, J. Grundy, J. Warren, and J. Hosking, "Model-driven development of mobile personal health care applications," in *Proceedings of the 2008 23rd IEEE/ACM International Conference on Automated Software Engineering*. IEEE Computer Society, 2008, pp. 467–470.

[5] T.-D. Nguyen and J. Vanderdonckt, "User interface master detail pattern on android," in *Proceedings of the 4th ACM SIGCHI symposium on Engineering interactive computing systems*. ACM, 2012, pp. 299–304.

[6] A. Ribeiro and A. R. da Silva, "Evaluation of xis-mobile, a domain specific language for mobile application development," *Journal of Software Engineering and Applications*, vol. 7, no. 11, p. 906, 2014.

[7] H. Heitkötter, T. A. Majchrzak, and H. Kuchen, "Cross-platform model-driven development of mobile applications with md 2," in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*. ACM, 2013, pp. 526–533.

[8] O. Le Goaer and S. Waltham, "Yet another dsl for cross-platforms mobile development," in *Proceedings of the First Workshop on the Globalization of Domain Specific Languages*, ser. GlobalDSL '13. New York, NY, USA: ACM, 2013, pp. 28–33. [Online]. Available: http://doi.acm.org/10.1145/2489812.2489819

[9] D. Steiner, C. Ţurlea, C. Culea, and S. Selinger, "Model-driven development of cloud-connected mobile applications using dsls with xtext," in *Computer Aided Systems Theory-EUROCAST 2013*. Springer, 2013, pp. 409–416.

[10] J. Danado and F. Paternò, "A prototype for eud in touch-based mobile devices," in *Visual Languages and Human-Centric Computing (VL/HCC), 2012 IEEE Symposium on*. IEEE, 2012, pp. 83–86.

[11] F. T. Balagtas-Fernandez and H. Hussmann, "Model-driven development of mobile applications," in *Automated Software Engineering, 2008. ASE 2008. 23rd IEEE/ACM International Conference on*. IEEE, 2008, pp. 509–512.