Tool Guide for RAPPT

Scott Barnett[a]

[a]*Deakin Software and Technology Innovation Lab (DSTIL), School of Information Technology, Faculty of Science, Engineering and Built Environment, Deakin University, Burwood, Victoria 3125, Australia*

Abstract

Mobile app generation can improve developer productivity and improve quality. RAPPT [1, 2] is one such tool that was developed to aide Android developers in building mobile apps by 1) generating code that looks like a developer wrote it, 2) providing high level abstractions for common components and 3) generating an app ready to deploy to a device. The core features of RAPPT and how RAPPT is to be used by developers are described in this report.

Keywords: Visual Notation, Mobile App Development, Domain Specific Languages, Code Generation

1. Introduction

RAPPT is a tool for generating the scaffold of an Android app. Developers specify the high level features of their mobile app such as the web services used, navigation pattern and then generate a working app. The code generated by RAPPT is designed to resemble code that was written by a developer so that the code is easier to work with than traditionally generated code. This report describes how RAPPT is to be used to build a simple mobile app and the core features of RAPPT. As a motivating example consider the MovieDB App shown in Figure 1 and described below.

MovieDBApp shows a list of popular movies that when selected enable a user to navigate to a screen showing the details of that movie. This app consists of three screens a *Popular Movies* screen for displaying the list of popular movies, an *About* screen to display copyright information, and a

Email address: scott.barnett@deakin.edu.au (Scott Barnett)

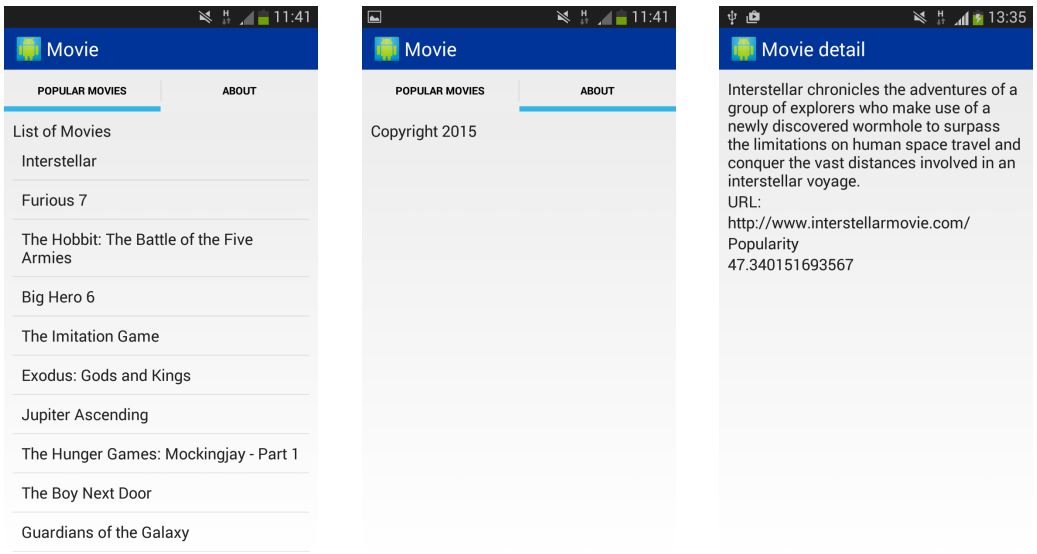

Figure 1: The *Popular Movies*, *About*, and *Details* screens for MovieDB, an app based on the MovieDB API.

Movie Detail screen to display the details of a selected movie. Screenshots for these screens are shown in Figure 1. The content to be displayed by this app will be provided by the freely accessible MovieDB API[1] and includes the following requirements:

1. A tabbar for navigation between the *Popular Movies* and *About* screens. Tabbar is a Mobile UI Pattern that displays tabs near the top of the screen for navigation (see Figure 1). The user can also swipe the screen to navigate between tabs.
2. Handle authentication with the Movie DB API e.g specify an authentication key.
3. Display a list of the popular movies from the Movie DB API on the *Popular Movies* screen.
4. Navigate to the *Details* screen from the *Popular Movies* screen by selecting a list item.
5. Pass the identifier for a movie when clicking on a list item in the *Popular Movies* screen and pass it to the *Details* screen so the details for the

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

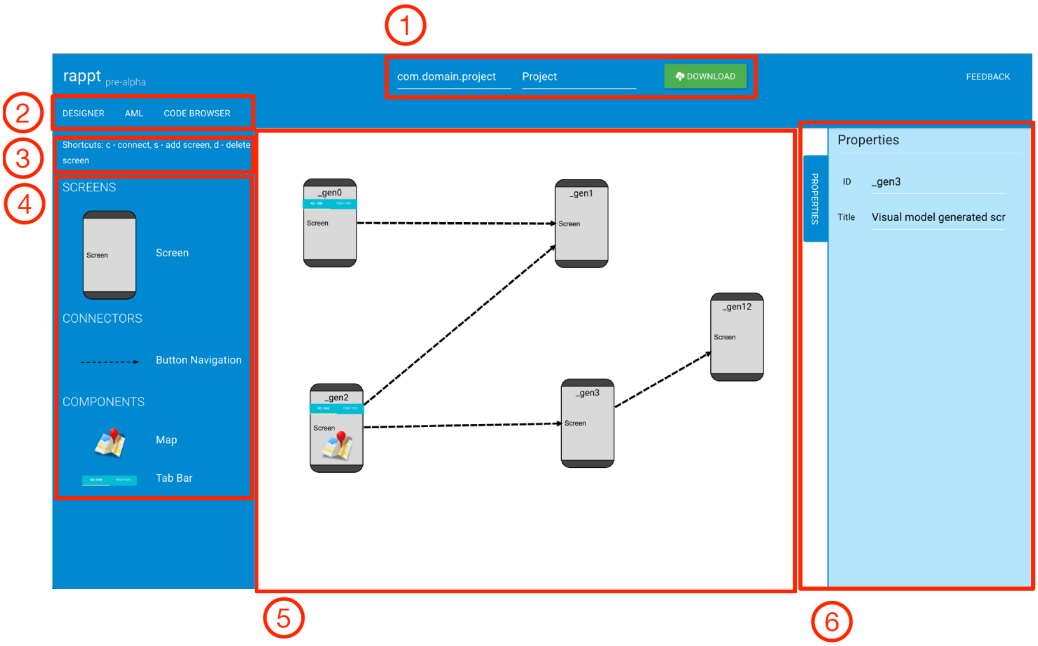

Figure 2: Screenshot for RAPPT's Designer. 1. Package, Project and Download, 2. Navigational Tabs, 3. Shortcuts, 4. Widget Pane, 5.Visual Editing Pane and 6. Properties Pane.

correct movie can be fetched.

6. Fetch the details for a selected movie from the MovieDB API and render the results to the screen on the *Details* screen.
7. Display a copyright message on the *About* screen.

2. RAPPT Tool Overview

2.1. Designer

In order to out explain how RAPPT is intended to be used we will use Peter, our fictitious developer. The first task facing Peter is that of modeling the navigation flow and specifying the screens that make up the MovieDBApp. When Peter opens up RAPPT he is prompted for the project name and package for the new app he is going to create. Once he has completed filling out these details Peter is presented with a blank *Designer* screen similar to the one shown in Figure 2. The *Designer* interface consists of 6 sections numbered in Figure 2 and described in more detail below.

1 This section contains the UI components for the *Package, Project and Download*. Peter can update the name of the project and package at anytime by clicking on the input fields contained in this section. Next to these fields is the Download button for downloading the source code for the generated Android app. These three fields are displayed for each of RAPPT's screens.

2 *Navigational Tabs* enable the developer to navigate between the Designer, AML code editor and the Code Browser. After Peter has completed specifying the app using the DSVL he can add additional details by going to the AML screen or preview the generated source code before downloading the app by navigating to the Code Browser section.

3 An overview of the keyboard *Shortcuts* for adding elements from the DSVL to the *Visual Editing Pane*.

4 The *Widget Pane* contains the elements that make up the DSVL ready for the developer to drag onto the *Visual Editing Pane*. For the MovieDBApp, Peter needs to drag three screens onto the designer one for the list of movies, displaying the details for a single movie and one for the app's about screen. Peter also drags the Tabbar component onto the list and about screens to specify the top level navigation pattern for the app. He then connects the list screen to the details screen specifying the navigation between these two screens.

5 The *Visual Editing Pane* is where the developer adds elements for describing the app and specifies the navigation flow by connecting screens together. This is the main area of interaction that Peter will use to specify the mobile app. In this area Peter can clearly demonstrate high level app concerns to non-technical stakeholders such as designers or clients.

6 An element's properties can be updated in the *Properties Pane* once the element has been added to the *Visual Editing Pane*. Properties are attributes that are not represented visually such as the element's ID. After Peter has updated the *Visual Editing Pane* with the three screens he then updates the title that will be shown for each screen as well as the screen's ID. It is important that Peter uses clear and easy to understand IDs as these are used in the code generation stage.

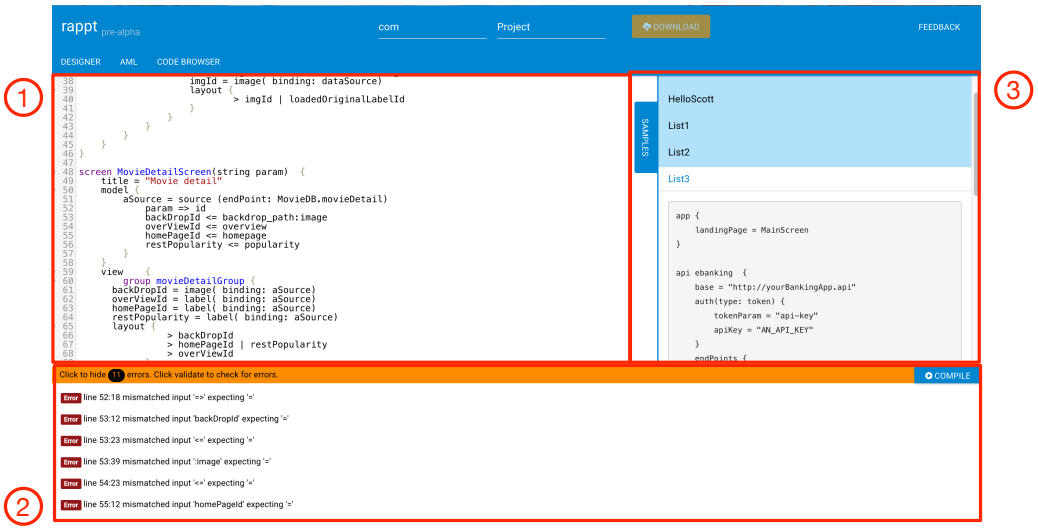

Figure 3: Screenshot of the AML editor: 1. DSTL Editing Pane, 2. Compiler Errors Pane, and 3. Samples Pane.

Now Peter switches to the AML tab to enhance the description of the MovieDBApp by specifying the Movie DB API and screen UI components.

2.2. App Modeling Language (AML)

The AML section is where Peter uses the DSTL to add details to the screens created on the Designer screen. This section is made up of three major panes: the editor where AML DSTL code is written, a Samples Pane that contains source code examples for various tasks to act as a guide to the developer and a Compiler Errors Pane that pops up when the project has errors. These sections are shown on the right side of Figure 3. Descriptions of each section below.

1 *DSTL Editing Pane* is where the developer edits the generated DSTL code. The developers uses the custom DSTL, AML, to enrich the details of the app. Here Peter needs to describe the Movie DB API specifying the location of the data, the structure of the data returned, the UI components for displaying a list and details of a movie, configure the event handlers to pass data from the list screen to the details screen and handle authentication. On the *Popular Movies* screen, Peter describes the API call for fetching the popular movies, the list and the UI

components to display the data using the samples as reference. Peter then describes the event handler for clicking on a list item and ensures that the ID for the selected movie is passed to the *Details* screen. This is needed to ensure that the *Details* screen shows the details about the correct movie that was selected in the list. In Android development this is known as the Master/Detail UI design pattern [3]. Next, Peter prepares the **Details** screen to make an API call using the passed Movie ID. Once done, he links the data in the response with the appropriate UI components for rendering to the screen. Peter does not need to worry about error handling, checking for a network connection, logging or dealing with concurrency, as RAPPT handles these by automatically generating the necessary code. Then Peter writes a short copyright notice on the About screen.

2 During the course of editing any syntax errors in the AML app description will be shown in the *Compiler Errors Pane*. Semantic errors preventing the generation of an Android app are also shown in this pane. An example semantic error is navigating to a screen that does not exist. These errors guide Peter in specifying the MovieDBApp ensuring that the generated code can compile and run.

3 The *Samples Pane* contains a list of working AML examples with different features such as Maps, Lists and UI components. These samples are intended to encourage reuse through copy-paste so that the developer can extend the DSTL code generated by the DSVL. Samples were created for each feature of AML to demonstrate how that feature should be used and how the different elements of the language can be combined. Peter copies the *api* block from the Api sample and updates the code to match the Movie DB API. From another sample Peter copies the *auth* block to the editor and configures the parameters to match the authentication details required for the Movie DB API. These samples provide multiple working examples that contain snippets of AML code and greatly help with development process by reusing already existing code. They will also be useful to beginner programmers, as they can view samples of different ways to utilize various capabilities of the tool.

Now that Peter has finished describing the MovieDBApp he is ready to view the generated code before downloading the app and adding the final polish.

2.3. Code Browser

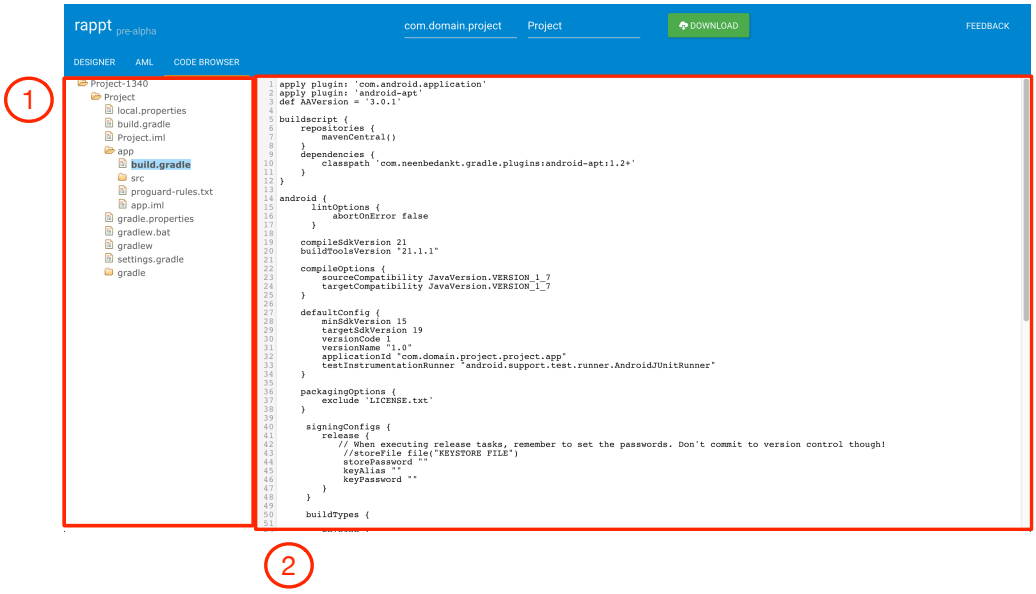

Figure 4: Code Browser. 1. Project Directory Pane and 2. Generated Code Browser.

The *Code Browser* enables developers like Peter to browse the generated source code in the *Code Browser* tab for the app before downloading the project but are unable to modify the output until they have downloaded the project. A conscious decision was made to not support round trip engineering so that our approach would not require any artifacts in the generated code such as annotations or additional libraries specifically for using RAPPT. The primary reason for this is that the generated code is the scaffolding for a new project and is intended to be heavily modified. Developers also have the freedom to switch tools and are not tied to using RAPPT once they have started using this tool. Developers extend and polish this scaffolding to produce the final app. A screenshot of the *Code Browser* is shown in Figure 4. Different sections of this code browser are marked and described below.

1 The *Project Directory Pane* shows the directory structure for the generated Android app. This project structure can be imported directly into an IDE for editing code.

2 *Uneditable Code Browser* displays the files generated by RAPPT. This view enables developers to gain an insight into the generated files. Peter

Figure 5: Complete source code for the MovieDBApp showing code generated by the DSVL, code reused from copy-pasting AML samples and manually edited DSTL code.

can view all of the generated files for the screens of the MovieDBApp as well as the networking code, the choice of 3rd party libraries and the architecture used to ensure that the app satisfies the concerns specific to mobile apps [4].

Now the initial version of the app is ready to download and deploy to a device. Peter downloads the generated code and runs the app on his device, thankful he does not need to manually write all of the boiler plate code automatically produced by RAPPT. The complete code for building a MovieDBApp is shown in Figure 5 and illustrates which parts of the DSTL were generated from the DSVL, copied from the samples section and edited manually.

3. Conclusion

RAPPT is a tool designed to improve Android developers productivity and simplify the initial work with starting a new mobile app. This report describes the core features of RAPPT and presents a case study on a fictional app as a guide to how the tool will be used.

References

[1] S. Barnett, R. Vasa, J. Grundy, Bootstrapping mobile app development, in: Proceedings of the 2015 IEEE/ACM International Conference on Software Engineering (ICSE 2015), IEEE, 2015, pp. 305–306.

[2] S. Barnett, I. Avazpour, R. Vasa, J. Grundy, A multi-view framework for generating mobile apps, in: Visual Languages and Human-Centric Computing (VL/HCC), 2015 IEEE Symposium on, IEEE, 2015, pp. 305–306.

[3] T.-D. Nguyen, 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.

[4] S. Barnett, R. Vasa, A. Tang, A conceptual model for architecting mobile applications, in: Software Architecture (WICSA), 2015 12th Working IEEE/IFIP Conference on, 2015, pp. 105–114. doi:10.1109/WICSA.2015.28.