

CO7201 Individual Project
Department of Computer Science
University of Leicester



A GRAPHICAL PROGRAMMING LANGUAGE FOR THE ANDROID PLATFORM

By

FEIFEI HANG
(*fh77@leicester.ac.uk*)

Supervised By:

ARTUR BORONAT
&
STANLEY P.Y. FUNG

A Dissertation
Submitted to University of Leicester
in Partial Fulfilment of the Requirements
for the Degree of Master of Science
in Advanced Software Engineering
in the Department of Computer Science

Leicester, United Kingdom

September, 2011

WORD COUNT: 12760

DECLARATION

All sentences or passages quoted in this report, or computer code of any form whatsoever used and/or submitted at any stages, which are taken from other people's work have been specifically acknowledged by clear citation of the source, specifying author, work, data and page(s). Any part of my own written work, or software coding, which is substantially based upon other people's work, is duly accompanied by clear citation of the source, specifying author, work, date and page(s). I understand that failure to do this amounts to plagiarism and will be considered grounds for failure in this module and the degree examination as a whole.



(Feifei HANG)
15 September 2011

Acknowledgements

I am heartily thankful to my supervisor, Artur Boronat, and my second marker, Stanley Fung, whose encouragements, suggestions, supervisions and supports from the preliminary to the concluding level enable me to develop and understanding of the subject.

Meanwhile, I would never have been able to finish my dissertation without the strong supports from my parents, my families and my friends who were always there cheering me up and stood by me through the good times and bad. I would like to express my deepest gratitude to my parents, my families, my friends and each one of those who around me.

Finally, I would like to thank Xi Wang, Ming Lian, Chu Feng, Hao Zhu, Tiancheng Lu, Kai Ma and Yanliu Hu who kindly participated in the long distance focus group during this project, especially Xi Wang, who permitted me to post her code segment in the section 8 of this dissertation.

Feifei HANG
15 September 2011
in Leicester, UK

Abstract

This project focuses on the prototyping of applications in the domain of book library querying system on the Android platform. With the development of computer science, domain engineering and application prototyping have become much more important than ever before. In order to implement a graphical programming language or, more precisely, to build a domain-specific application prototyping system, a series of studies and researches were carried out on domain engineering and application prototyping in this project. Moreover, other issues such as the designs of graphical user interfaces and the implementations of the multi-language bridging middleware were also concerned.

As a result of a group of studies, researches, designs and implementations, a fully functionally GUI-based domain-specific application prototyping system, called DroidBuilder, was finally created. In this dissertation, each of the issues in this project such as the objectives, the design overviews and the implementations will be addressed in detail. At the end of this dissertation, an analysis on the limitations and further researches of this project will also be given based on the reviews of the entire project.

Keywords: Domain-specific language, application prototyping, the Android platform, domain engineering.

Table of Contents

1. Introduction	1
2. Motivation of Domain	2
3. A Literature Survey on Eclipse.....	3
4. Project Aim and Objective.....	5
4.1 Manifest Generator.....	6
4.2 Easy-to-use User Interface	6
4.3 Scratch-like Graphical Representation.....	7
4.4 Middleware.....	8
4.5 Domain Engineering.....	9
5. Challenges	9
5.1 Connections Between Android Devices and Remote Database	9
5.2 Automatic Prototyping.....	9
5.3 Development of Applications for Android Devices	9
5.4 Integration with Graphical Representations	10
6. Background & Technologies	10
6.1 JSON	10
6.1.1 JSON Basic Types	11
6.1.2 JSON on the Android Platform.....	12
6.2 Programming Technologies.....	13
6.2.1 Programming language - Java 6.0	13
6.2.2 Programming language - PHP 5.3.4	14
6.2.3 Programming language - Python 2.6.1	15
6.2.4 Java Swing.....	15
6.3 Project Tools.....	16

6.3.1 NetBeans	16
7. Design Overview.....	16
7.1 Main Packages.....	16
7.1.1 com.android.dsl - The Kernel of DroidBuilder	17
7.1.2 mods, gui, ctrl - Model-View-Contorller (MVC).....	18
7.1.3 keychain - Key-value Storage Engine	20
7.2 Component Integration.....	20
8. Code Blocks	21
8.1 Introducing of Code Blocks	21
8.2 Integration with DroidBuilder.....	23
8.2.1 Scratch-like Graphical Representation.....	24
8.2.2 Activity Builder	25
9. Domain Engineering.....	26
9.1 Domain Analysis	26
9.1.1 Domain Analysis - Book Library Querying System.....	26
9.1.2 Domain Analysis - Applications on the Android Platform.....	30
9.2 Domain Design.....	33
9.3 Domain Notation	34
9.3.1 Domain-Specific Language (DSL).....	35
9.3.2 Android Application	35
9.4 Domain Implementation.....	36
9.4.1 Domain-Specific Language (DSL).....	36
9.4.2 The Kernel.....	45
9.4.3 Domain Wizards in DroidBuilder.....	47
10. Bridging Middleware.....	48
10.1 Implementation.....	48

10.2 Data Receiving.....49

10.3 JSON Parsing.....51

11. Keychain Key/value Storage Engine.....52

12. Testing.....53

13. Further Research & Limitation.....54

14. Conclusion.....55

References56

Appendix.....58

Appendix 1 -- JSON document for data transfer.....58

1. Introduction

In the last quarters, the market share of Android platform kept growing and nearly reached a peak of 50 per cent according to the survey made by Gartner [1]. In other words, Android system can be found almost anywhere on many kinds of mobile devices, nowadays. As a result of being an open system [3], Android witnessed the fact of “the more market share Android reach, the more Android developers there are”. However, as a mobile platform, Android have shown its weaknesses on several aspects, especially on the length of application development life cycle. Even though more and more applications have already been built for the Android platform, the development process of Android applications still can be long and frustrate.

This MSc individual project (CO7201) is going to build a graphical environment, which helps programmers to create applications for the Android platform, in order to reduce the difficulties of developing Android application and to shorten the development life cycle of building software for Android devices. In this project, a requirement is that the final product should be domain specified so that people can easily prototype applications in the domain. I am supposed to concentrate on the domain of book library querying system, which is being commonly used in libraries in order to help customers finding books on shelves, on the Android platform. In this case, the main task of my individual project is not only to build a graphical system for prototyping Android applications, but also to work on several components of domain engineering, for example, domain analysis, domain design, and domain implementation.

Technically, the entire system will be organised by several different components: a group of front end graphical user interfaces, which help users to interact with the whole system; a kernel of the system, which can be seen as a domain-specific language and a set of compilers being designed to facilitate the developing of Android applications in the domain; a key/value engine, which is being used to be the storage of data that will be used by the entire system in runtime; and, a middleware, which is acting as a bridge between Android devices and the remote database that holding the data of libraries.

In this dissertation, I will describe the concepts and technologies used in the project as well as the design and implementation details of the entire system, and will address the contributions being made on domain engineering.

2. Motivation of Domain

Originally, the idea of focusing on the domain of book library querying system on the Android platform was from real life experience. People always complain that whenever they need to find books in libraries, they always have to use the computers or terminals in libraries to find out the information they need. Or, in better case, those information might be able to be queried on Internet. In other words, people then are able to query on the own computers at home. However, it is very obviously that no matter what kind of querying method does library provide, we still need to access the databases of libraries via computers, basically. In other words, it can be easily figure out that the traditional ways of book library querying still have very huge limitations. More precisely, the mobilities of the traditional methods are quite poor since we always need to have computers by hands even in the later situation.

One of the best solutions to the above scenario can be the use of mobile devices, for example, mobile phones, tablets, and even E-readers. Since more and more mobile devices are becoming Wi-Fi or 3G supported, we are able to surf the Internet with web browsers, like Opera Mini [2], wherever we are and whenever it is. Thus, it becomes possible for us to visit the websites of libraries through mobile devices when we need to find books. However, if the website of library that we access from mobile phones can be seen like an application, the performance of this app is much poorer than the native applications as well as the usability is much worse because we have to start up web browser at first, and then go to the website of the library, and then after a couple of steps we can finally query information with helps of search engine. Obviously, most of the above steps have to be manually repeated every time when we find books in library in that way. Therefore, a solution, which ensures both the usability and mobility, is deserved to be discovered.

As what have been mentioned about in the previous section, the Android platform has already been used in many kinds of mobile devices. Also, as one of the open systems, third-party applications are supported by the Android platform. Furthermore, according to the technical specifications of Android platform and Android devices

[4], Wi-Fi hardware access is one of the core component of this platform. In other words, each Android device should normally has the ability of connecting with Internet. Therefore, based on these advantages, which can also be described by the Figure 2.1, of the Android platform, this open system can definitely be a solution to the problem we met. Thus, the domain of this project has been eventually concentrated on book library querying system on the Android platform.

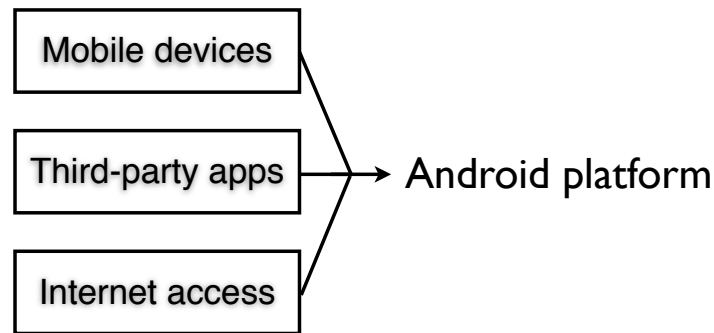


Figure 2.1 The Advantages of the Android platform

3. A Literature Survey on Eclipse

Before making decisions on what to build in this project, it is necessary to have a look on what existing systems have already been built, firstly. As the official integrated development environment of Android applications, Eclipse has been globally widely used. With helps of a bunch of very powerful functionalities provided by Google and Eclipse, programmers are able to manage and develop many different types of applications for the Android platform. However, as what has been already said, although it has been proved that Eclipse is capable for developing neither mini project or large system for the Android platform, the development life cycle is still relatively long. By studying on the functionalities of Eclipse, it can be easily pointed out that even though many helpful features are being provided by Eclipse, there is only few support on domain-specific development. In other words, Eclipse is a very powerful general purpose development environment, but is not a good choice when working in specific domains.

Therefore, it is reasonable to design the system which needs to be built for this project based on the lessons learnt from Eclipse. In order to have a clear picture of

both the advantages and shortcomings of Eclipse in the project domain, a diagram has been made as follow.

	Features	Description
Advantages	Android Package Handling	inc. package creating, signing, generating, and etc.
Shortcomings	GUI Building	A graphical toolkit for building GUIs.
	Manifesting	Users are able to define Manifest manually.
	Domain Specific	There is no features to support the specific domain in Eclipse.

Undoubtedly, the features on Android packages handling are quite essential since the purpose of this project is to provide a system for the Android platform. Thus, these functionalities should be definitely kept in the system that will be built. For the feature of GUI Building, it is not a real shortcoming in the domain specific system, logically. Because the system should be able to help users to prototype Android applications with graphical user interfaces. Otherwise, the prototypes made by the system will become meaningless and useless as it is unreasonable to image that the users of those prototypes will be able to use non-GUI or text-based applications on their mobile devices. However, when prototyping applications for the specific domain, it is much reasonable if users are being asked to bind default prototype GUIs rather than building the look and feels. Thus, a feature of binding GUIs is much expected in stead of the functionality of building GUIs.

Regards to the feature of manifesting, programmers are benefited by defining the manifest documents of their Android applications as more features can be brought in and be supported. On the other hand, modifying manifest documents can be risk if programmers cannot completely handle with the behaviours of their applications. For example, by modifying the manifest document, Android applications are able to access Internet without any notification to users. Assume that the permission of Internet connection has been accidentally added by programmer, the Android application will then be able to access Internet during its whole life cycle. As a result of that, users may loose their money if the Internet accesses are made through prepaid or post paid connections. Thus, in stead of providing the ability of modifying the manifest document, the system is expected to generate manifest documents, which are corresponding to the behaviour of the prototypes, automatically.

Finally, as a general purpose development environment, the specific domain is not originally supported by Eclipse. Thus, a group of features which support the domain that being focused on in this project should be provided by the system (inc. reusable models/frameworks for prototyping Android applications and databases).

In conclusion, based on the above literature survey on Eclipse, the expected features in the early designing of the system are being summarised as follow.

Features	Description
Android Package Handling	inc. package creating, signing, generating, and etc.
GUI Binding	A graphical toolkit for binding GUIs.
Manifesting	Auto-generated Manifest XML documents.
Domain Specific	Reusable domain-specific models / frameworks.

4. Project Aim and Objective

The main task of this project is to implement a graphical programming environment which can help programmers to prototype domain-specific application for the Android platform. The runnable Android applications are basically organised by a group of JAVA source codes and a couple of XML documents which define the configurations of the Android applications. Therefore, the programming environment should be able to help programmers not only to prototype the behaviours, which are being define in JAVA source codes, of Android application, but also to generate default configurations in XML documents, like Manifest. In order to make users having fabulous user-experiences, this system should also provide well designed and easy-to-use interface that allow users to easily build their prototypes with helps of this system. More precisely, in some components of this system, the user interfaces should be smart in order to save the time of users. For instance, once programmers make changes on the prototypes of their Android application, the system should be able to automatically identify whether the programmers want to confirm or cancel the changes in stead of being told by the buttons clicks of users. Besides, graphical representations which represent the workflow and life cycle of the prototypes will be

displayed during the behaviour defining procedures in order to keep users always be clear about how their prototypes will work like.

Because this project is also working on a specific domain, domain engineering is necessary to be carried out in order to find out the commonalities and the variabilities of the domain. Moreover, based on the domain engineering, the system should provide an reusable framework, a domain specific language, and a set of compilers of the language. Finally, in order to establish connections between Android devices and remote databases of libraries, a middleware should also be able to be generated by this system according to connection configurations defined by users.

4.1 Manifest Generator

Manifest Generator will take the information (inc. project name, project package, Android Activity name and Android GUI name) of the prototype that users are working on as input, then generate Manifest XML documents as output. The data on the left is the input information to the program based on the configurations of prototype while expected output XML document is shown on the right.

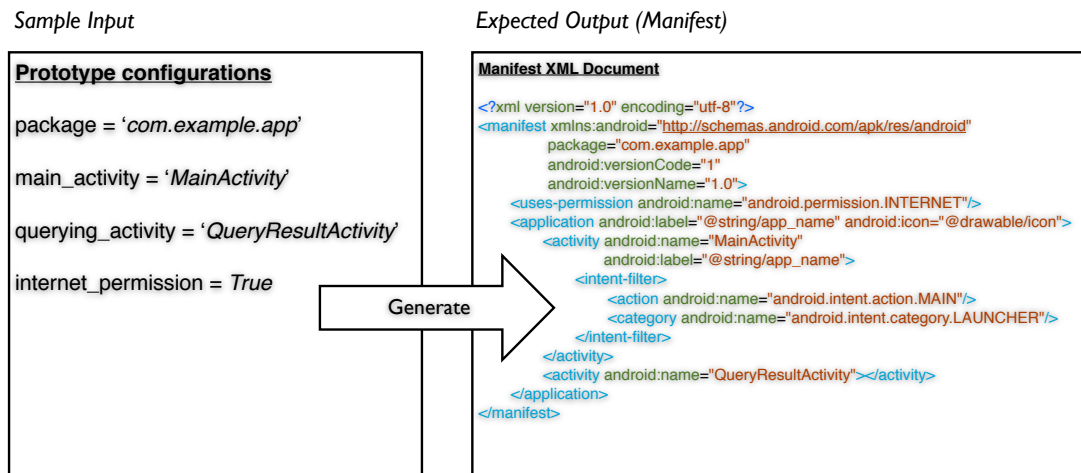


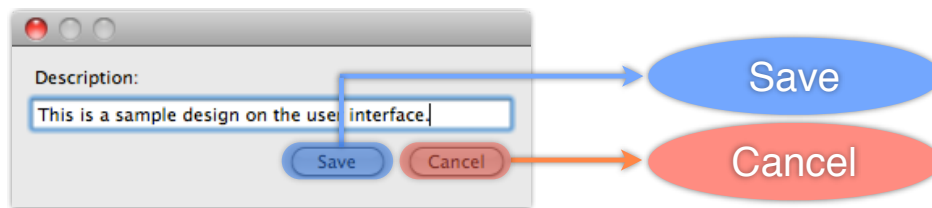
Figure 4.1 Sample input and expected output for *Manifest Generator*

4.2 Easy-to-use User Interface

In order to provide an user-friendly system to programmers, the GUI of the system has to be well designed. A smart and easy-to-use user interface will not only bring users a fabulous user experience, but also facilitate the interacting between users and the system. As a prime example of the current interface paradigm [5], Macintosh brings us a potential solution to this problem. Based on the Human Interface

Guideline of Apple Inc. [6], a well designed user interface or user interface component should minimise the steps that users will spend to achieve their goals. Thus, in some parts or components of the GUI of this system, the interfaces should be handleable as much as possible. Figure 3.2 shows a expected human-computer interaction procedure based on the idea of easy-to-use user interface while making a comparison with the traditional way that we interact with applications.

Traditional HCI procedure



Expected procedure

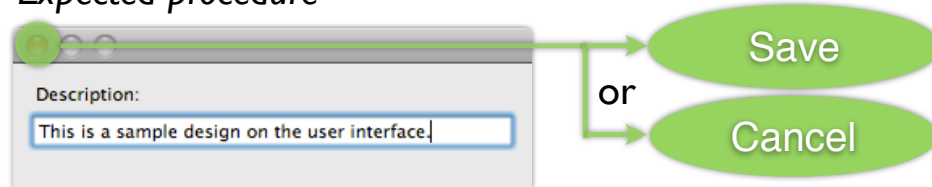


Figure 4.2 Traditional and expected HCI procedure

Based on a well made and easy-to-use user interface, the system should be able to automatically identify the behaviour or activity that users want to performance rather than being told by users inputs or clicks. As an example shown in Figure 3.2, the system understands in which case the changes made by users should be saved or cancelled.

4.3 Scratch-like Graphical Representation

In order to show an overview on the work flow and life cycle of the Android applications that users are prototyping in the system, a Scratch-like graphical representation [7] will be displayed to represent the behaviours of the prototype. Moreover, the graphical representation should also indicate different types of behaviours (inc. loops and conditions) for users. The source code on the left is the input to the program while expected graphical representation is shown on the right.

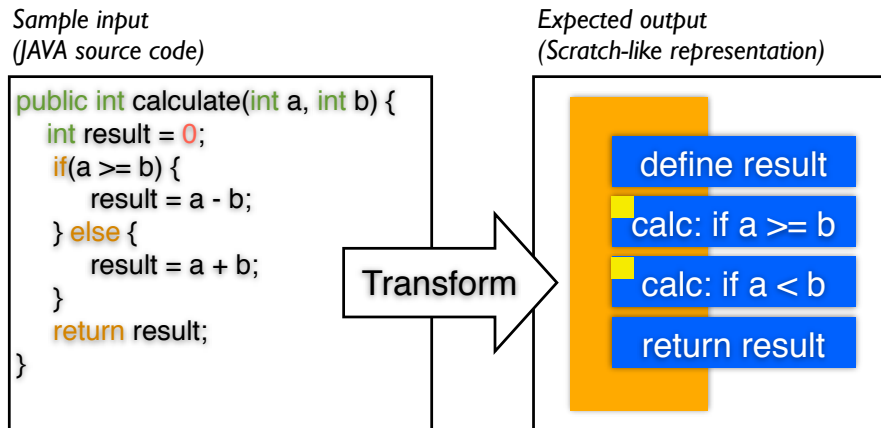


Figure 4.3 Sample input and expected representation for the system

The concept of Code Blocks has been introduced into the graphical representation. Details of Code Blocks will be described in the section 8.

4.4 Middleware

Because of the domain of book library querying system on the Android platform, it is necessary to make connections between Android devices and remote database of libraries in the runtime. However, according to the current design of the Android platform, remote database connecting and accessing are not directly and natively supported by any existing version of Android systems. Thus, a middleware will be used to transfer data between Android devices and the remote databases. Essentially, the middleware should be a two-way bridging program which asks and sends back feedbacks from remote databases after receiving querying information from Android devices.



Figure 4.4 Bridging workflow of *Middleware*

Bridging procedure involves several main steps: Receiving from Android device - Querying database - Feedback returning - JSON retrieving (on Android devices). The technical details will be described in section 10.

4.5 Domain Engineering

As one of the most important parts of the entire system, several key components have been covered by domain engineering. Generally, the domain engineering will concentrate on domain analysis and domain implementation in order to produce a reusable domain model as well as a domain specific language. Since the domain engineering can be one of the major topics of this project, it will be addressed in section 9 in details.

5. Challenges

This section will briefly describe the main challenges during this project, including functional challenges as well as unfunctional ones.

5.1 Connections Between Android Devices and Remote Database

As what has been mentioned about in section 4.4, even in either the current version of Android SDK or the Android platform itself, there is no abilities existing for interacting with remote databases, natively. Thus a reliable middleware, who transfers data between Android devices and remote databases, is one of the key to success.

5.2 Automatic Prototyping

Essentially, as the most important aim of the project, prototyping deserves to be spent more efforts than others. However, even though the entire project is focusing on only one specific domain, slight differences still exist in the applications which are being prototyped by the system. In other words, in order to meet and fulfil the aim of project, the correctnesses of automatic prototyping have to be seriously ensured.

5.3 Development of Applications for Android Devices

When working on relatively new technologies, like Android, the experiences of developing and building Android applications are always lack. Thus efforts should be spent on learning Android SDK and building Android applications. Additionally, by comparing with developing applications for computers, testing and debugging of Android applications are always more difficult. Hence it is necessary to build the right Android applications in right way.

5.4 Integration with Graphical Representations

In order to produce user-friendly GUIs, some graphical representations, like UML diagrams, might be helpful when representing workflows or behaviours of Android applications. However, it is not easy to find the way to integrate the system with a graphical representations since a poor designed integration of graphical representation may mislead the programmers who use the system. As a simple and clear graphical representation of behaviours and workflows, the Scratch-like graphical representation has been chosen in this project.

6. Background & Technologies

This section will give a brief description of JSON (JavaScript Object Notation) as the format being used in data interchange by middleware (see section 4.4) when bridging connections and transferring data between Android devices and remote databases. In addition, retrieving of JSON on the Android platform will be briefly introduced in chapter 1.2 and will be described in details in section 10.3. The basic structure and format of JSON document can be found in the appendix.

6.1 JSON

JavaScript Object Notation (JSON) is a light-weight text-based data-interchange format and standard for representing simple data structures and associative arrays, called objects[8]. Typically, the JSON format is used for transmitting structured data over a network connection, and is being primarily used to transmit data between a server and web applications as an alternative to XML. Originally, JSON is being built on two structures: (1) a collection of name/value pairs. (2) an ordered list of values. In

modern programming languages, these structures are realised as an object and an array, respectively. As a lightweight data-interchange standard, JSON can be read and write by humans as easy as being parse and generated by machines.

In this project, the data bridging happened on middleware is based on the second structure. Because when querying database, it is reasonable to expect that the feedback or results from database can be either a single name/value pair from data records or a group of name/value pairs. In this case, it is ideally to struct and parse the data collections as an ordered list of objects and each object contains several name/value pairs corresponding to the querying result of database. With helps of the native API `json_encode` [9] of PHP, the feedback collections received from database can be automatically struttred to a JSON document by middleware. In other words, the generating of JSON document will not be a point to be concentrated on in this project while parsing and retrieving of the JSON document will be mainly focused on.

6.1.1 JSON Basic Types

Even though the generating of JSON documents is not being concentrated in this project, it is still necessary to understand the basic types or forms of JSON being used in this project in order to efficiently and correctly retrieve data in the system. This subsection very briefly describes the two basic types of JSON.

Object: An object is an unordered set of key:value pairs which is separated by comma while being enclosed in curly braces. The key of the pairs must be string. According to the instruction of JSON [8], the graphical representation of the format of JSON object is shown in Figure 6.1.1.1.

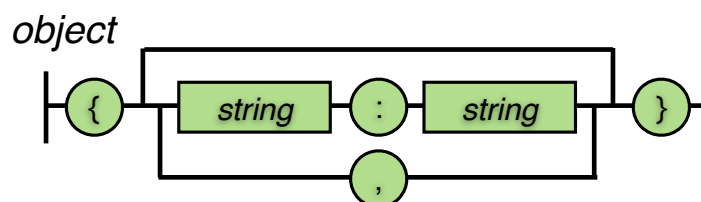
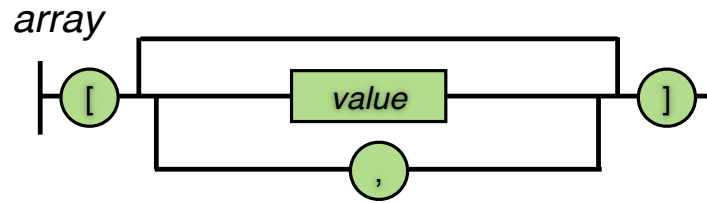


Figure 6.1.1.1 JSON *object* [8]

Array: An array in JSON is an ordered collection of values. An array begins with [and ends with] while separates values by , (comma). In addition, the values in an array are not necessarily to be the same type. The graphical representation of array is shown below.

Figure 6.1.1.2 JSON *array* [8]

As what has been described in section 6.1, the form of returning data being transferred by the middleware in this project will be built on the structure of an ordered array which contains name:value pairs inside. Therefore, it can be understood as a set of JSON objects involved in a JSON array. Figure 6.1.1.3 shows the format and relationship of the JSON elements in this project. To see example of a full JSON document being used in this project, please refer to *Appendix [Appendix 1]*.

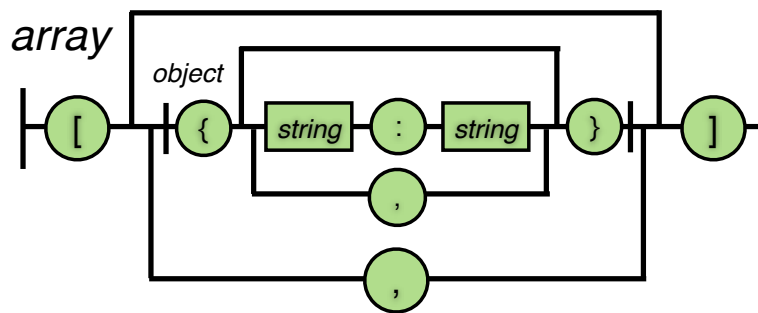


Figure 6.1.1.3 The format of JSON document being used in this project

6.1.2 JSON on the Android Platform

Based on the Dalvik virtual machine [10] of the Android platform, a collection of Android APIs has been involved in the Android SDK while parts of traditional Java libraries have been kept for the platform. As one of the native components of Android, the package of *org.json.** provides abilities for parsing JSON data on Android devices. By identifying the name of the expected JSON elements (inc. JSON array and JSON object), data can be easily navigated and retrieved with helps of the subpackages of *org.json.JSONObject*, *org.json.JSONArray* and etc.. Thus, the last step of the procedures of feedbacks responding, receiving and parsing between remote database, middleware and Android devices can be implemented natively on the Android platform. The whole procedure of JSON data transferring in this project

is shown in Figure 6.1.2. The technical details on implementing the JSON parsing is described in section 10.3.

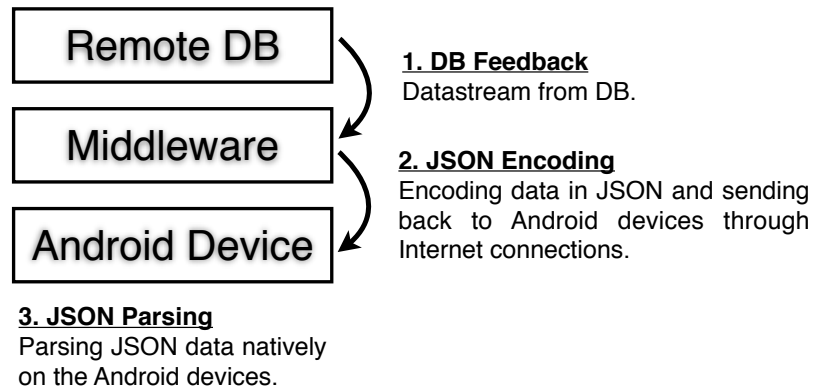


Figure 6.1.2 The procedure of JSON data transferring

6.2 Programming Technologies

This section will describe the programming languages and the technologies used in this project. In order to make the system be able to run on any platforms, Java, PHP and Python have been chosen as the programming languages in the major components of the entire system.

6.2.1 Programming language - Java 6.0

Java 2 Platform Standard Edition is a major feature release. As a very powerful general purpose programming language, several extensions have been introduced to JDK 1.6. In addition, JDK 1.6 has become the default version of Java Development Kit on several platforms, like Mac OS X Snow Leopard, therefore there is no need to worry about the compatibility of JDK 1.6 on those platforms. This is the major reason why I choose it for developing this project. Also, some other features such as Java Generics are being widely used in the implementation of this project. An example taken from my code (*CodeContainer.java*) which uses the feature of Java Generics is shown below.

```

private ArrayList<String> arrayList;
private int currentLine;

public String pop() {
    if(this.arrayList.size() == 0 || this.currentLine >= this.arrayList.size()) {
        return null;
    } else {
        return this.arrayList.get(this.currentLine++);
    }
}

```

Figure 6.2.1 Java code segment -- implement *DroidBuilder* using JDK 1.6

As what can be seen from the above figure, the declaration on the first line reads as “ArrayList of String arrayList”. The code using generics is clearer and safer.

As the features of JDK 1.6 used in programming, this system must be compiled with JDK 1.6 and run with the corresponding Java runtime library. The source codes may not be successfully compiled with a lower version of JDK. If attempts are being made on running this system with JRK 1.5, it may rise exceptions or errors.

6.2.2 Programming language - PHP 5.3.4

As one of the built-in programming language, PHP 5.3.4 has been used to implement the middleware in this project. Based on the features of PHP 5.3.4, the program can be easily implemented by using the natively APIs such as *mysql_connect* and *json_encode* to handle the connections with remote MySQL database and to generate JSON data. Moreover, as a server-side HTML embedded scripting language [11], PHP is ideal for developing the middleware which runs on server. This is another reason why PHP has been chosen as the programming language in the implementing of middleware.

The below example taken from the code of *query_subject.php* in the middleware shows the usages of the features of MySQL Connecting and JSON Generating.

```

<?php
mysql_connect("127.0.0.1","root","");
mysql_select_db("demo_db");

$query = mysql_query("SELECT * FROM book_list WHERE subject LIKE '%" . $_REQUEST['subject'] . "%'");
while($entry = mysql_fetch_assoc($query))
    $output[]=$entry;
print(json_encode($output));
mysql_close();
?>

```

Figure 6.2.2 PHP code segment -- implement *Middleware* using PHP 5.3.4

We can see that the features of MySQL Connecting and JSON Generating are widely used in the implementation of the middleware in this project. Based on the description in section 6.1, since JSON data can be automatically generated by calling the natively API of `json_encode`, we do not need to focus more on the building of JSON document.

As the above features of PHP 5.3.4 contributed a lot on the implementation of middleware, the middleware has to be run with PHP 5.2.0 onward. A lower version of PHP such PHP 5.1.6 will not be able to run the source codes and will cause errors.

6.2.3 Programming language - Python 2.6.1

In order to make DroidBuilder more reliable and powerful, especially to store the runtime configurations of the entire system, a very lightweight key-value storage engine, called Keychain, has been designed as another core component of the entire system. Based on the design principles of Keychain, the source code should be as simple as possible in order to provide opportunities to users being able to maintain and even modify Keychain. Thus, Python has been used to be the programming language of Keychain. During the development of Keychain, the standard library *md5* was used to improve the security of data.

```
import md5
...
# Generate md5 value
md = md5.new(_name).hexdigest()
...
```

Figure 6.2.3 Python code segment -- generating MD5 value in Python 2.6.1

The above code segment taken from my code (*keychain.py*) shows the usage of the MD5 standard library in the implementation of Keychain. The MD5 library is mainly supported by Python 2.5 onward. Thus, a lower version of Python will not be able to use the MD5 library and will throw an exception.

6.2.4 Java Swing

Java Swing (Swing) is one of the frameworks for developing graphical user interfaces in Java. [12] Swing is designed to be fully featured user interface development kit. As a result, we can take its rich and flexible features to create Swing-based desktop application. Additionally, as one of the native package of J2SE, Swing (*javax.swing*)

can be run on any platform directly without referring additional libraries to create GUI components.

In order to build the Swing-based user interfaces involved in DroudBuidler efficiently, NetBeans (this will be discussed in next section) has been chosen as the IDE of this system.

6.3 Project Tools

This section will describe the Integrated Development Environment (IDE) that used within this project. With helps of this tool, it will be easier to develop the system more quickly and efficiently and improve the performance and reliability.

6.3.1 NetBeans

NetBeans [13] is an open source cross-platform Integrated Development Environment that provides excellent development tools and features for developing Java projects. It contains a Java project explorer, a Java debug perspective, a Java editor, refactoring tools, revision control plugins and Java compiler. As a Swing-based platform, NetBeans also provides a very powerful visual editor for creating Swing components in Java projects. As we have already chosen Swing (*javax.swing*) to be the GUI framework in the beginning of this project, there is no doubt that NetBeans will be the best choice of being the development tool in this project.

7. Design Overview

This section will show the design and framework of the entire system. It will give a brief description of all packages in the system as well as the responsibilities of each one.

7.1 Main Packages

The entire system contains five main packages as follow.

Package	Description
com.android.dsl	The kernel of DroidBuilder
gui	Graphical User Interface
ctrl	Controllers of DroidBuilder
mods	Models of DroidBuilder
org.gnuer.lib	Keychain key-value storage engine

7.1.1 com.android.dsl - The Kernel of DroidBuilder

As a domain-specific programming environment, domain engineering has been carried out in this project (this will be addressed in section 9 in details). As a result of that, a domain-specific language (DSL) is designed and built in order to facilitate the developing of Android applications in the domain of library querying system on the Android platform. By integrating with DroidBuilder, this DSL has become the kernel of DroidBuilder when prototyping and generating source codes for Android applications. Technically, the kernel of DroidBuilder is organised by a collection of lexers and interpreters which take the code of the DSL as tokens and then convert (eval) into Java source code.

Essentially, the kernel contains two groups of compilers inside. Generally speaking, *com.android.dsl.Tojava* is used to convert the DSL to generate Java source code while *com.android.dsl.Xml* is being used to build the codes for defining the GUIs of Android applications. Each compiler generally contains a lexer, an interpreter and a buffer for storing the input code and tokens. The technical details of the kernel will be discussed in section 9.3.1 to 9.3.2.

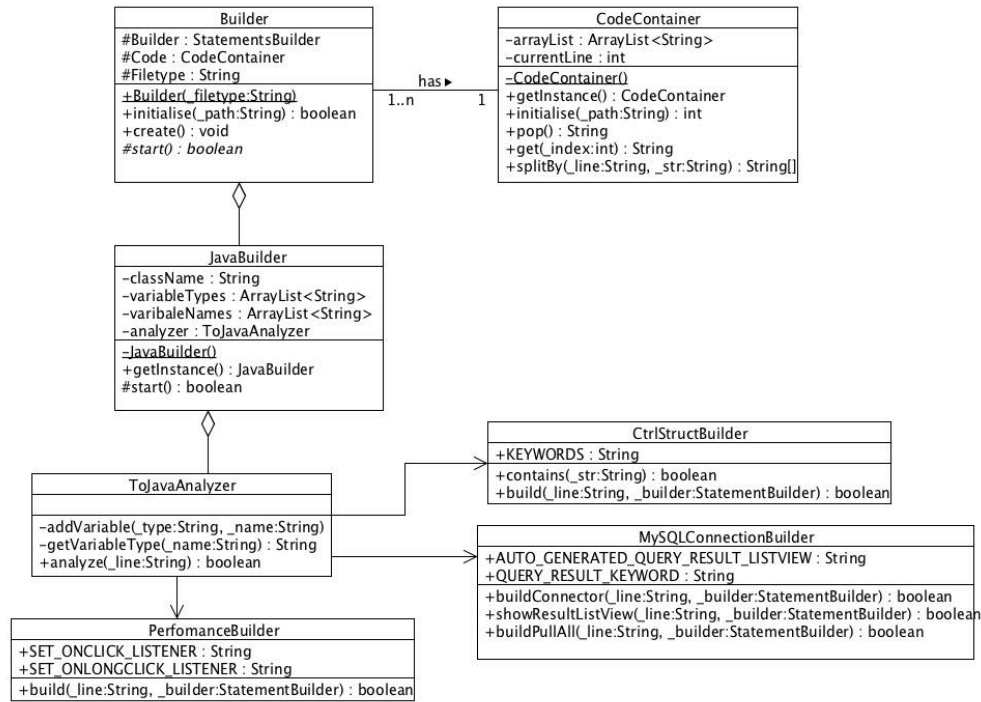


Figure 7.1.1 class diagram of *Tojava* compiler

As an example, Figure 7.1.1 shows the class diagram of *com.android.dsl.Tojava* compiler in the kernel. From the diagram, it is noticeable that class *CodeContainer* is the code buffer of the compiler. Each instance of *JavaBuilder* contains one code buffer in order to store the tokens of input source codes tokenised by instance of *ToJavaAnalyzer* which acts as the lexer of the compiler. After tokenising the input, the interpreters (inc. *CtrlStructBuilder*, *MySQLConnectionBuilder* and *PerformanceBuilder*) will eval the tokens into Java source code. The *com.android.dsl.Xml* compiler uses the same structure and workflow in design and implementation.

7.1.2 mods, gui, ctrl - Model-View-Contorller (MVC)

As the the major components of DroidBuidler, the packages of *mods*, *gui*, and *ctrl* are designed and implemented based on the concept of MVC (Model-view-controller) [14]. All Swing GUI classes such as main window and other dialog windows are placed in the package of *gui* while models and controllers are placed in *mods* and *ctrl*, respectively. In most user interface components of DroidBuilder, the view classes are working together with corresponded models and controllers. As a good example, the view class *IDEFrame.java* runs with the model of *ProjectProperties.java* in order to store and share the basic information of Android project with other GUI classes while

using the controller *AndroidOpt.java* to perform core actions which take the data inside of the instance of *ProjectProperties.java* as the input.

WelcomeFrame.java is the first window started up by *DroidBuilder.java* which is acting as the launcher of DroidBuilder and is placed in default package. Through the *JButtons* on the panel of *WelcomeFrame.java*, users are able to be redirected to either *IDEFrame.java* or *NewProjectFrame.java*.

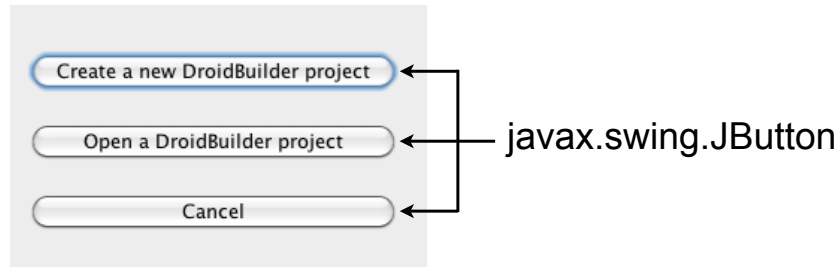
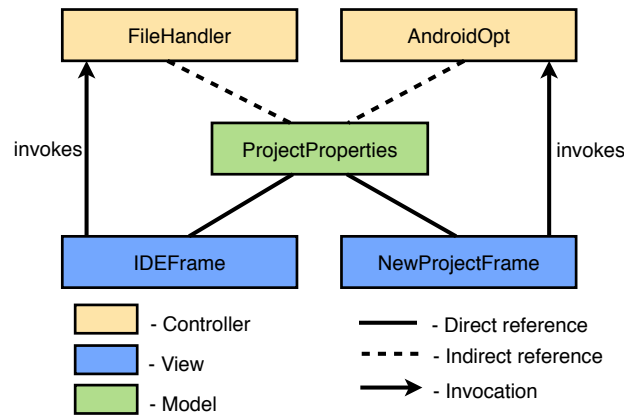


Figure 7.1.2.1 JButtons in *WelcomeFrame.java*

As the first wizard window of DroidBuilder, *NewProjectFrame.java* guides users to create new Android project with helps of the controller *AndroidOpt.java*. A group of *JTextFields* and *JButtons* is displays for users to input the basic information of Android project (inc. project name and project package). As one of the most importance controllers, *AndroidOpt.java* is also used to sign the Android package, to generate Android Manifest and to invoke the kernel which is described in the previous section. Taking the advantage of MVC, *AndroidOpt.java* is not only used by *NewProjectFrame.java* but also connected with the main window *IDEFrame.java* in order to sign and generate Android packages.

In the implementation of DroidBuilder, models and controllers are usually cross referred by views in order to maximise the reusability of classes under the concept of MVC. The cross references have also improved the maintainability of the system.

Figure 7.1.2.2 MVC relationships on *mods.ProjectProperties*

We can see that in the relationships over model *ProjectProperties* have been referred as the data members by view *IDEFrame* as well as *NewProjectFrame*. As a result, the instances of *ProjectProperties* in the both views will always be up-to-date after being updated either in *IDEFrame* or in *NewProjectFrame*. Indirect reference refers to the parameters transfer happened when a method of a controller is being called by views. Specific data members of the model will be the input data to the method.

7.1.3 keychain - Key-value Storage Engine

The Keychain Key-value storage engine inside of package *org.gnuer.lib* is originally designed for storing configurations of the entire system for runtime querying and loading. Essentially, Keychain is developed in Python in order to bring users opportunities for self modification and maintenance. To integrate the Python-based program with DroidBuilder which is built in Java, the Java class *org.gnuer.lib.JKeychain* is implemented to be the bridge between DroidBuilder and Keychain with helps of the free project Jython [15].

7.2 Component Integration

The following component diagram shows the modules and their dependencies related to DroidBuilder.

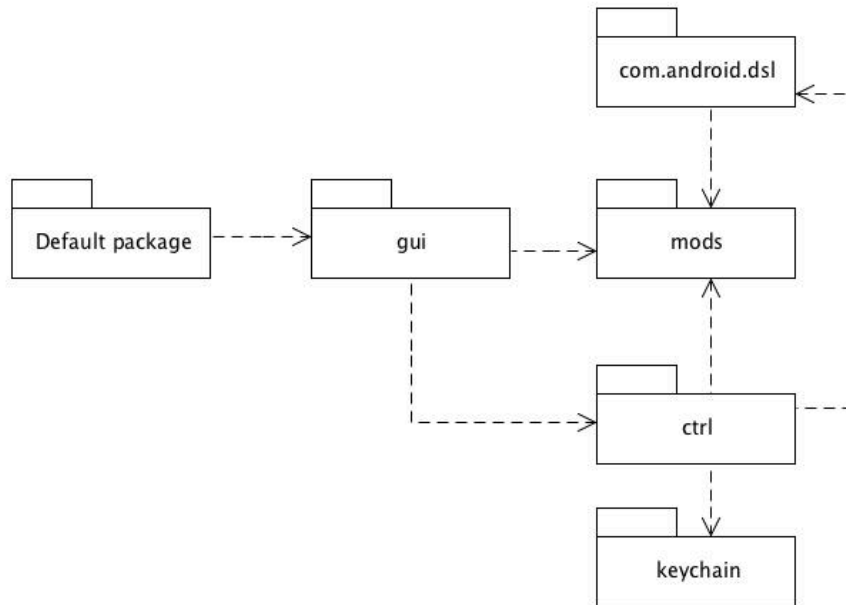


Figure 7.2 Package Dependencies of DroidBuilder

From the above diagram, the package relationships are defined as follow: Default package needs package *gui* to create user interfaces and package *gui* needs *mods* and *ctrl* to store runtime data and to interact with other packages. Package *mods* is also needed by *com.android.dsl* to get runtime input to the kernel of the system. Meanwhile, package *keychain* is needed by *ctrl* to query and load configurations to the entire system.

8. Code Blocks

This section will describe the concept of Code Blocks in details. As an important concept introduced into this project, a discussion on the integration with DroidBuilder will also be addresses.

8.1 Introducing of Code Blocks

As mentioned in chapter 4.3, a concept of Code Blocks has been introduced into this project. Essentially, Code Blocks are used to be a solution to an existing problem of programming. Traditionally, when implementing complex classes of applications, programmers are always lost in large numbers of code. Although comments in code are helpful in some cases, people still frequently have no idea about what a large

scope of code is doing, especially when they are trying to maintain the existing source files.

Based on a range of studies on the source codes produced by a focus group, the traditional ways to comment code have very limited capacity to complex classes or methods. An example code segment [16] taken from the focus group is shown below.

```
# Implementation of scanner
# Scan one character at the time until finding something to parse
while index < len(_code):
    chunk = _code[index:]
    identifier = ''
    # Matching if, print, method names, etc.
    items = re.findall(r"\A([a-z]\w*)", chunk)
    if items:
        identifier = items[0]
        if identifier in KEYWORDS:
            tokens.append(['' + ', '.join([':' + identifier.upper(), '' + identifier + '']) + ''])
        else:
            # Indentation
            dedents = re.findall(r'\Aend', chunk)
            if dedents:
                current_indent -= 1
                tokens.append(['' + ', '.join([':DEDENT', str(current_indent)]) + ''])
                tokens.append(['' + ', '.join([':NEWLINE', '"\n"']) + ''])
                index += len('end')
                continue
            else:
                tokens.append(['' + ', '.join([':IDENTIFIER', '' + identifier + '']) + ''])
    index += len(identifier)
    continue
```

Figure 8.1 A code segment using comments [16]

We can see that although the author of the above codes used comments in her source code, the procedure of the scope of code still can be confusing. In other words, the traditional way that we comment on code is to describe “what will happened in the following code” rather than “what is happening in the following code”. Based on this point of view, the concept of Code Blocks has been introduced.

Comparing with the other technical components implemented in this project, Code Blocks is about the way that we organise and program source code. As described by its name, Code Blocks encourages people to place the different parts of code (inc. different method, different control flows and different loops) in separated blocks which indicated by names or brief descriptions. Once the blocks of code that we need have been made, we can organise them in the order of the performances that the programs are expected. Generally, the way that we build applications with Code Blocks is much like building a house with bricks. The bricks are made firstly, and then construct the building. In addition to the concept of Code Blocks, several terminologies are introduced as follow.

Code Blocks: Essentially, the term Code Blocks refers to the concept itself. However, when using this concept in software development, Code Blocks means the collection of defined Code Block.

Code Block: A single block of code is called Code Block. A Code Block usually contains a series number of the Code Block, the description or the name of the Code Block and a Code Segment.

Code Segment: A scope of code that the Code Block should perform at runtime.

A graphical description of Code Blocks is shown in the below diagram. It also shows the way that we use Code Blocks when building applications.

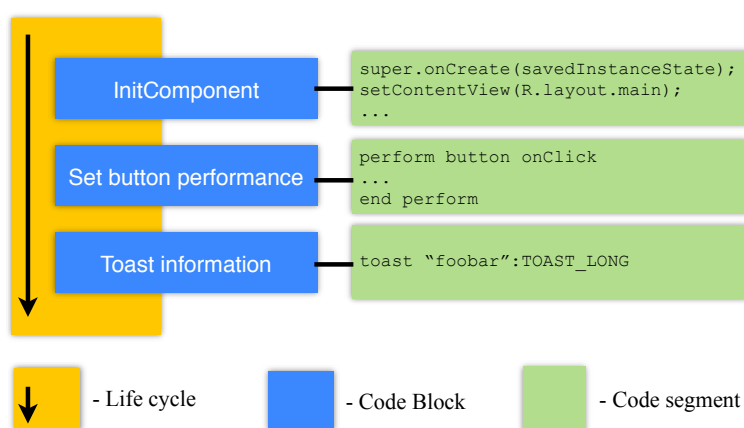


Figure 8.2 Code segment using *Code Blocks*



We can see that by using the concept of Code Blocks the work flow of the above sample code is clear and readable. According to the diagram, the runtime procedure of this code segment can be read as “Initialing component first and then setting the performance of button and then toasting information”. Obviously, in this way, the readability and maintainability of source code are improved with helps of Code Blocks.

8.2 Integration with DroidBuilder

As an important concept, Code Blocks has been completely integrated with DroidBuilder in this project. A set of graphical toolkits are provided for users to fully experience the advantages of Code Blocks. Technically, the concept of Code Blocks is used widely in the components of DroidBuilder such as Android Activity Builder

and Scratch-like Graphical Representations. By comparing with traditional Integrated Development Environments, DroidBuilder encourages users to build Android applications by defining, ordering and organising the different scopes of code which perform specific tasks rather than coding the source code traditionally.

8.2.1 Scratch-like Graphical Representation

As mentioned in section 4.3, a Scratch-like Graphical Representation [7] is chosen to represent both the application life cycle of Android applications and the work flows of Android activities. As a linear graphical representation which built on a number of parent nodes and child nodes, the Scratch-like is provided opportunities for fully integrating with the concept of Code Blocks. Based on the structure of Code Block, the brief description of each Code Block is ideal for being the identifier of each node in a Scratch-like diagram. By ordering and organising the Code Blocks, the corresponded real-time life cycle or work flow diagrams will be displayed for users to monitor or track the behaviours of their Android applications. In addition to the graphical representation on the work flows of specific methods, the mark  (loop) and  (logic) are used to indicate what kind of behaviour the specific scope of code is going to perform.

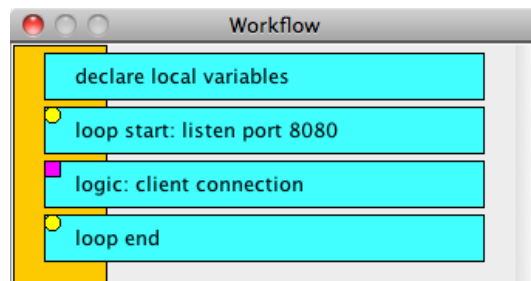


Figure 8.3 A sample workflow diagram generated by *DroidBuilder*

The above figure shows a workflow diagram using the Scratch-like graphical representation in DroidBuilder. We can see that by combining the Scratch-like graphical representation and the concept of Code Blocks, the behaviour and workflow of the sample is shown clearly as the above workflow can be understood as “declaring local variables first, and then starting the listen on port 8080. If client connections are detected, do something.”. Logically, the more detailed descriptions on Code Blocks given by the users, the more information can be displayed by the Scratch-like graphical representation.

8.2.2 Activity Builder

As the component designed for defining the behaviours of Android applications more precisely, Activity Builder is completely designed with the concept of Code Blocks in order to help users prototyping and building their Android applications more efficient while ensuring the readability and maintainability of the code. Users are strongly encouraged to build their classes and methods by ordering and organising the Code Blocks they defined in Activity Builder.

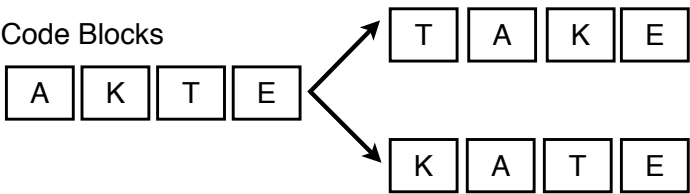


Figure 8.4 Using Code Block in Activity Builder

As what can be easily pointed out in the diagram above, with helps of Code Blocks, we can not only efficiently create the expected programs in Activity Builder, but also build other programs based on the same Code Blocks, quickly. In other words, when using Activity Builder to define the behaviours of Android applications, it is much like producing objects in a product line. We firstly build different modules (Code Block) and then assemble them in the form we expect.

The mechanism of organising Code Blocks in Activity Builder is shown in the diagram below.

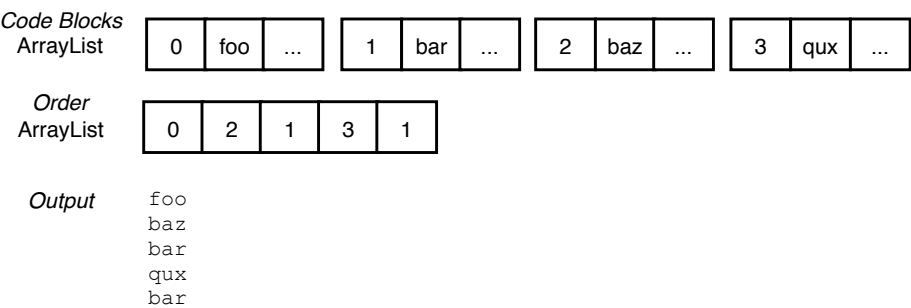


Figure 8.5 Organise Code Blocks in Activity Builder

As said in section 8.1, a single Code Block contains three parts: an unique series number, an identifier such as a brief description and a scope of code. As shown in the diagram above, the defined Code Blocks are initially stored in an array list. When defining the behaviours of Android application in Activity Builder, a copy of the

series numbers will be stored in another array list in the order we need. Eventually, the Code Segments of the Code Blocks will be organised according to the order declared in the Order array list in order to generate the expected output.

9. Domain Engineering

As mentioned in section 4.5, this project is also based on a domain engineering since I am supposed to concentrate on the domain of library querying system on the Android platform. In order to be domain-specific, domain engineering was carried out. This section will describe the domain engineering in this project in details.

9.1 Domain Analysis

As the first phase of domain engineering [17], domain analysis was used to abstract the domain model in this project. Based on the specific domain of library querying system on the Android platform, this step was reasonably divided into two parts: analysis on the domain of library querying system and analysis on the domain of applications on the Android platform. Considering with the scope of this project, the ingredients of the domain model in this project were sketched as follow.

Domain scoping: finding out the boundaries of the domain.

Commonality analysis: considering both the commonalities and variabilities of the application in the domain based on a group of case studies.

Domain dictionary: defining the terms used in the domain. As the Android platform has already been provided a high number of terms with itself, domain dictionary will not be concerned on the domain of applications on the Android platform.

Requirements: defining the requirements of the reusable components that will be built for the domain. Typically, these requirements will be the commonalities captured in commonality analysis.

9.1.1 Domain Analysis - Book Library Querying System

As mentioned in last section, the domain analysis needs to be carried out based on a couple of case studies in order to find out the commonalities and variabilities in the domain. To collection the information, I focused on the websites of the libraries at three universities in different countries: University of Leicester, UK [18]; James Cook University, Australia [19]; Beijing University of Technology, China [20].

By studying the querying webpages of these libraries, I became able to point out the basic functional features of them.

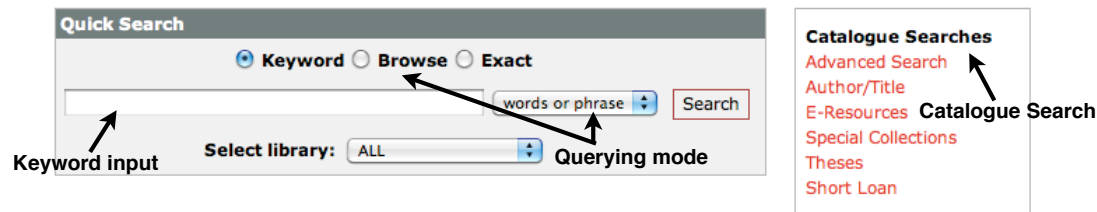


Figure 9.1.1.1 Screenshot from University of Leicester Library [18]

As we can see from the webpage of University of Leicester Library shown in above diagram, the querying service in this library basically consists of a *keyword field* which allows users to input the keyword to be queried with, a selection of *Catalogue Searches* which helps users to search books under specific catalogues (inc. authors, publishers and subjects) and an option on the *querying mode* to scale the scope of searching on data records.

Regards to James Cook University, the services provided by its library can be seen in the screenshot below.

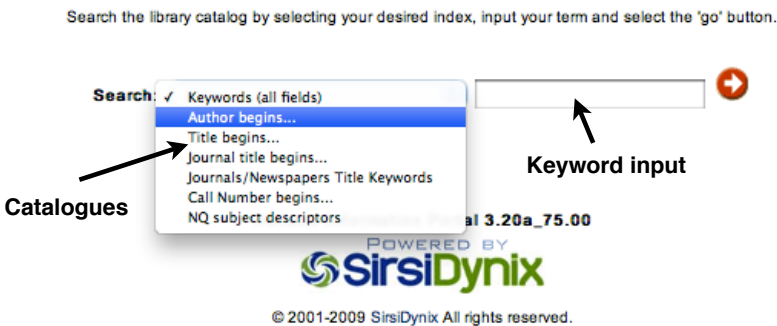


Figure 9.1.1.2 Screenshot from James Cook University [19]

We can see that the library of James Cook University provides a pair of very simple but useful components in their service. A *keyword input field* is used to get the keywords from users while an option on the *searching catalogues* is available. By

using this service, users are able to find the books either written by a specific author or with a specific calling number.

Finally, another concentration has been made to the functional components provided by the library of Beijing University of Technology. The screenshot below is taken from the book querying webpage of Beijing University of Technology Library [20].



Figure 9.1.1.3 Screenshot from Beijing University of Technology [20]

It is not very difficult to figure out that the basic functional components involved in the webpage of Beijing University of Technology Library are quite similar to the ones provided by James Cook University Library. A *keyword field* is used to get input from users while a couple of options on *catalogues* are designed for specifying the querying scopes.

Therefore, based on the above studies of the existing book querying webpages, the commonalities and variabilities on the front-end components are summarised as follow.

Commonalities	
Keyword input field	A text field which allows user inputing the querying keywords such as a part of book title or an exact name of a book.
Catalogues	A group of options which help users to identify the belongings of keywords (inc. whether the keyword refers to the title of books or the name of the author).
Variabilities	

Querying mode	A set of selections on the accuracies of querying results.
---------------	--

Regards to the database, since neither the design nor the structures of the databases used by the libraries is released to public, it is impossible to study on the commonalities and variabilities between these databases in this project. However, based on the above studies and summaries, a very simple but reasonable design on the database used in the domain can be described as follow.

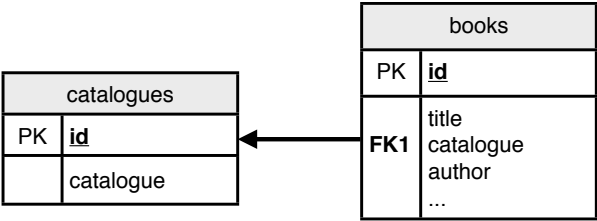


Figure 9.1.1.4 The basic database structure in the domain

We can see that the design on the database is quite simple in the domain. The very basic database consists of a table of *catalogues* which contains information of a specific catalogue such as all the subjects of the books that available in the library and a table of *books* which involves basic information of all the books. Although the above design on the database is quite basic, the real structures can be definitely modified or improved by programmers since the aim of this project is to help users to prototype their applications in the domain.

When defining the terms covered by the domain, I focused on the interactions between the front-end application run on the Android platform and the remote databases. Briefly, the three terms below haven been defined in this domain.

Query: querying MySQL database through the middleware (please refer to section 10) and store the results in a ordered list.

Pull: retrieving the expected collection of data records from the list updated by “query”.

Hence, based on the above information defined or collected from case studies, the domain model is described as follow.

Domain Model - Book Library Querying System		
Domain Scoping	A set of frond-end components which interact with a remote database in order to querying data record based on the input from users	
Commonality Analysis	<u>Commonalities</u>	<ol style="list-style-type: none"> 1. Keyword input field: A text field which allows user inputing the querying keywords such as a part of book title or an exact name of a book. 2. Catalogues: A group of options which help users to identify the belongings of keywords (inc. whether the keyword refers to the title of books or the name of the author).
	<u>Variabilities</u>	Querying mode: A set of selections on the accuracies of querying results.
Domain Dictionary	<ol style="list-style-type: none"> 1. Query: querying MySQL database through the middleware and store the results in a ordered list. 2. Pull: retrieving the expected collection of data records from the list updated by “query”. 	
Requirements	<ol style="list-style-type: none"> 1. The commonalities should be implemented in the implementation of the domain. 2. Regards to the variability, the MySQL keyword “LIKE” can be used by on the logic: $R_{like} \supseteq R_{all}$. (The result collection by using “LIKE” consists of the result collection of an exact data record.) 	

9.1.2 Domain Analysis - Applications on the Android Platform

Comparing with the previous domain analysis, the study on the applications on the Android platform concentrated more on the technical implementations of Android applications. According to the guideline of Android development [4], each runnable Android application originally contains a manifest document, a couple of xml documents which define the GUIs of the application, a group of Java classes used to define the behaviour of the Android application and several configuration files generated by Android SDK.

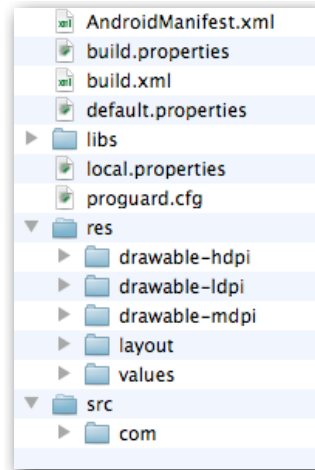


Figure 9.1.2.1 Android application file structure

The above diagram shows the default file structure of an Android application. Generally, the files with the extension name of *.properties* under the root and the files inside of the directory of *libs* are automatically generated by Android SDK. In other words, those files were not concentrated in the domain analysis in this project. Besides, the *drawable* folders under *res* are used to be the place to hold the drawable resources such as the icons of the Android application. Thus, they also have not been focused in this project.

Hence, the file of *AndroidManifest.xml* were taken into account in the domain analysis as well as the directories of *layout*, *values* and *src*. Logically, all the files and directories were defined as the commonalities of this domain since all of these must be contained by each single Android project in order to make application valid and runnable. The *AndroidManifest.xml* was chosen as the start point since it contains the core configurations of the related Android applications. A very basic *AndroidManifest.xml* is specified below.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.app"
    android:versionCode="1"
    android:versionName="1.0">
    <application android:label="@string/app_name"
        android:icon="@drawable/icon">
        <activity android:name="HelloWorld"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category
                    android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

```

        </activity>
    </application>
</manifest>

```

According to the above example document taken from an Android project generated by Android SDK, concentrations were made to three key elements: *manifest*, *activity* and *intent-filter*. Essentially, the responsibilities of the elements are defining the basic information such as name of package of the Android project, specifying the name of the main activity (the very first activity used to interact with users when an Android application is started up) of the Android project and indicating what kind of behaviours the Android application is going to perform, respectively.

By understanding the tasks of the above elements, they were defined as the variabilities of *AndroidManifest.xml* since they will refer to different information in different Android projects.

Comparing with *AndroidManifest.xml*, the files involved in *layout* and *values* shown lots of similarities on the structures of the documents since they are also XML-based. Regards to the contents, it is unreasonable to dig out the commonalities and variabilities inside of those documents in details in this section, because different Android applications consist of different layouts and values. Therefore, the files contained by *layout* and *values* were directly identified as parts of variabilities of this domain. For the same reason, source codes involved in *src* were also treated as another variability.

Thence to this domain analysis, the domain model was defined as follow. Notice that because a plenty group of terms on Android development have already been introduced by Android project, there is no need to provide more in the domain dictionary.

Domain Model - Applications on the Android Platform		
Domain Scoping	The valid and runnable GUI applications on the Android platform.	
Commonality Analysis	<u>Commonalities</u>	The file structure of each Android project.
	<u>Variabilities</u>	<i>AndroidManifest.xml</i> , package <i>layout</i> , package <i>values</i> and package <i>src</i> .
Domain Dictionary	N/A	

Requirements	<ol style="list-style-type: none"> 1. The commonalities should be always kept in all the prototypes of Android application. 2. The elements involved in variabilities should be automatically prototyped by the system based on the basic configurations of each Android project.
---------------------	---

9.2 Domain Design

In order to tell how the reusable components can be derived when prototyping the Android applications in the domain, domain design was carried out based on the two domain models mentioned in last two sections. In some literatures this part of output in domain design is also called *production plan* [21].

As mentioned section 9.1, the entire domain focused by this project essentially contains two parts which will be logically separated at the runtime of the entire system. More precisely, the first part of the domain is covered by the Android applications prototyped by DroidBuilder while the later one is contained in the design of the programming environment.

However, when producing a fully functional prototyping system like DroidBuilder, it is extremely necessary to find a way that making the two domains work together. Generally, the logic of the solution used in this project is to prototype the domain-specific Android applications in the domain of book querying system by using the domain-specific prototyping environment, DroidBuilder, which is in the domain of prototyping GUI applications for the Android platform. The diagram below shows the relationship between the two domains in a graphical way.

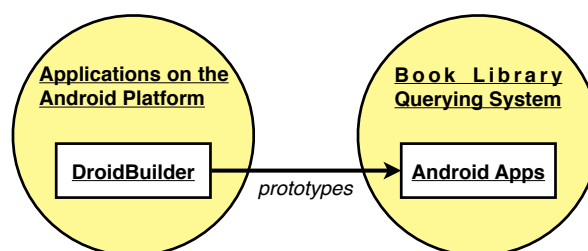


Figure 9.2.1 The relationship between the domains in the project

Therefore, based on the above relationship and the domain models, the way of using the reusable components when prototyping applications in the domain focused by this project was specified as “With helps of several dialogs or wizards in DroidBuilder, users will be able to define the variabilities contained in the domain model of Applications on the Android Platform and the domain model of Book Library Querying System. Based on the defined commonalities and variabilities, the reusable domain components will be used to create the prototypes of Android applications.”. Alternatively, this production plan can be also graphically described by the following diagram.

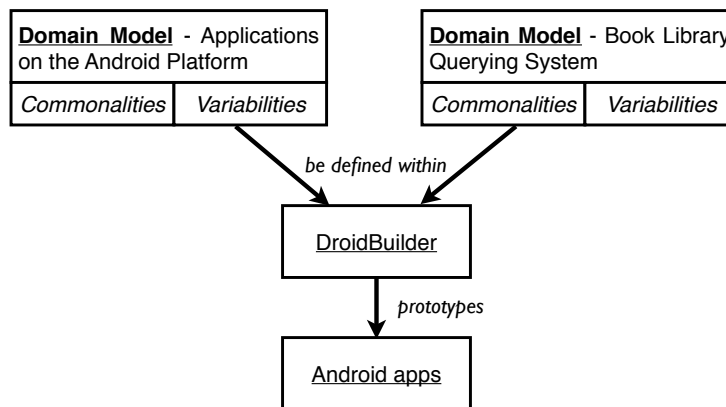


Figure 9.2.2 Production plan on the domain models

9.3 Domain Notation

Domain notations are used to provide a uniform way to represent the concepts used in the domains [21]. Based on the outputs of domain analysis and domain design, the notations were carried out in order to map the domain models, concepts and designs with the final outputs of the domain engineering in this project. In other words, the domain notations will specify how the domain components and final product will work.

In this project, the phase of domain notion was concentrated on the domain-specific language and the Android applications prototyped by DroidBuilder. In order to provide such a notation that is familiar for the public, UML (Unified Modelling Language) has been chosen in this case. More precisely, the designs of the domain-specific language and the prototyped domain-specific Android application were described by state machines.

9.3.1 Domain-Specific Language (DSL)

As said in section 7.1.1, a domain-specific language was designed and implemented during this project. In order to build the DSL, more precisely, the compiler, the below state machine was used to describe the expected workflow of the DSL compiler.

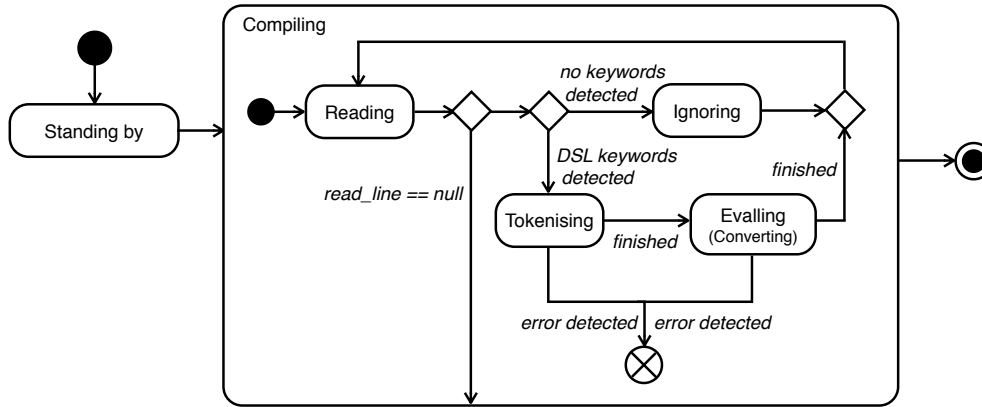


Figure 9.3.1 State Machine of the domain-specific language

As we can see from the above diagram, the entire procedure of compiling contains five states or phases. Once DroidBuilder has been started up, the DSL will come to the state of standing by. When users begin to compile their codes, the compiler will read through each line of those code until the end of files. If the current line of code contains pre-defined keywords, the compiler will begin to tokenise and eval the code, otherwise, the current line of code will be ignored and be kept in the outputs. In the case of errors are arisen in either tokenising or evalling, the compiling process will be interrupted.

Therefore, the domain-specific language can be implemented based on the domain notation. The implementation details will be discussed in section 9.4.1.

9.3.2 Android Application

In order to prototype domain-specific application on the Android platform, the domain notation which describes the excepted Android application is quite necessary. Based on the domain models and the designs mentioned about so far, the domain notation of domain-specific Android application was declared as follow.

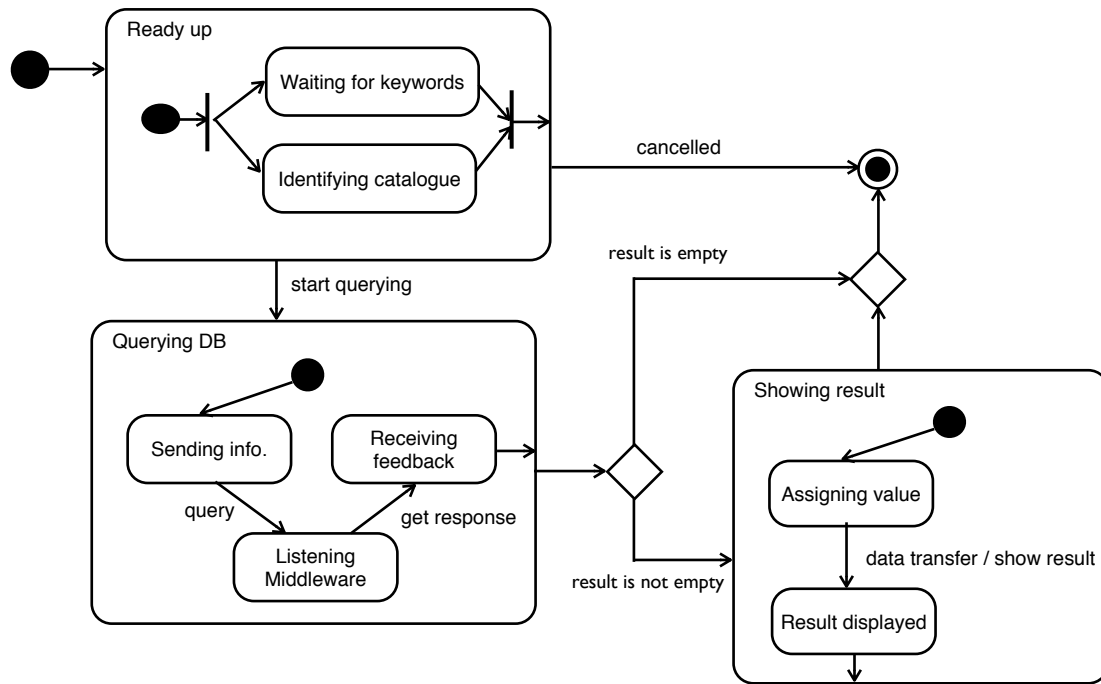


Figure 9.3.2 State Machine of the prototyped Android application

Technically, each domain-specific Android application prototyped by DroidBuilder covers three main states: Ready Up, Querying Database and Showing Result. The state of Ready Up is completely based on the domain model described in section 9.1.1. Essentially, the components in either Waiting for keywords or Identify Catalogue are basic human-computer interactions. In case of cancelling the interactions in the first state, the Android application will exit. Otherwise, the application will begin connecting and interacting with remote middleware in order to query the remote database to get expected data records. After all, the expected data will be assigned to another Android GUI, which is used to display the final result, in the application.

9.4 Domain Implementation

This section will describe the details on the domain implementation in this project. Based on the outputs of domain analysis, domain design and domain notation, the domain implementation in this project covered the implementation of the components and tools. More precisely, this comprised implementing domain-specific language and prototyping tools in DroidBuilder.

9.4.1 Domain-Specific Language (DSL)

Although the aim of the entire project is focusing on the prototyping of Android applications in domain, a DSL (domain-specific language) can still strongly benefit

the entire system. For example, by using the domain-specific language, the difficulties of developing Android applications can be certainly reduced while the development life cycle is dramatically shortened. The below code segments taken from the source code of an Android application prototype of mine are doing the exactly same things. Each of them is used to load the user interface of an Android application while registering an audio service and a button event in this Android activity. It is not difficult to point out how efficient it can be when defining Android activities by using this DSL.

```
task activity
package com.example.app

use android.media.AudioManager

activity MainActivity
  def AudioManager audio
  on create R.layout.MyLayout
    setButtonClickListener()
    service audio AUDIO_SERVICE
  end create

  fun void setButtonClickListener()
    def Button toggleButton
    bind toggleButton R.id.toggleButton
    perform toggleButton onClick
      // do something
    end perform
  end fun
end
```

The above code is written in the DSL while the following is in Java.

```
package com.example.app;

import android.media.AudioManager;
import android.app.*;
import android.content.*;
import android.os.Bundle;
import android.view.*;
import android.widget.*;

public class MainActivity extends Activity {
    AudioManager audio;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.MyLayout);
        setButtonClickListener();
        audio = (AudioManager) getSystemService(AUDIO_SERVICE);
    }
}
```

```

    private void setButtonClickListener() {
        Button toggleButton;
        toggleButton = (Button)findViewById(R.id.toggleButton);
        toggleButton.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                // do something
            }
        });
    }
}

```

As what can be found out in the above example, the basic idea of in the design of the domain-specific language is to modularise the traditional Java code used to define the components, the layout and the behaviours of Android applications under the domain models specified in section 9.1 in order to facilitate Android applications developing or prototyping. As the core elements of a lightweight domain-specific language, a group of terms, keywords and syntax were designed based on the domain models in this project.

In order to make the DSL functional, the compiler of the domain-specific language was implemented. By comparing with the traditional language such as Java, this compiler was designed to convert the DSL code to traditional Java code rather than bytecode or machine code to ensure the compatibility with Java. In other words, users are benefited since they can use either the DSL or Java to program their Android applications in DroidBuilder.

Essentially, the compiler of the DSL is originally organised by a lexer and a interpreter. Generally, when a piece of code is being compiled, the code will firstly be tokenised by the lexer into tokens and then be interpreted to Java code by the interpreter. The diagram below shows the working procedure of the compile designed and implemented in this project.

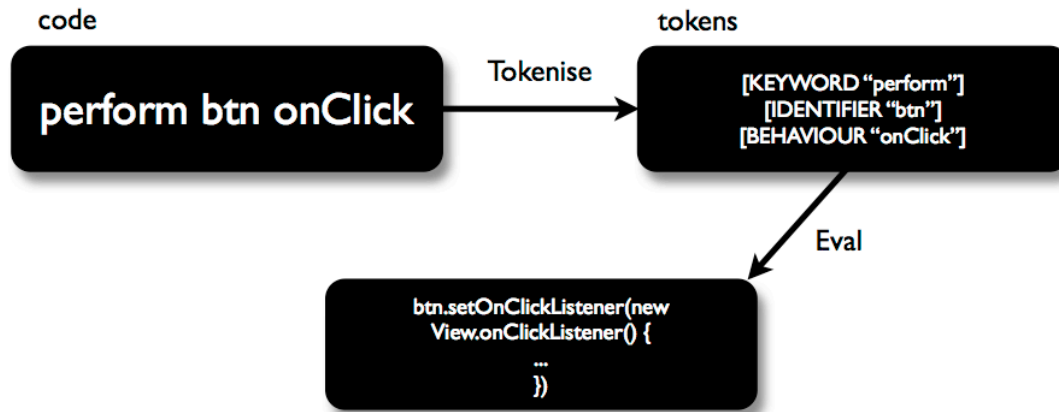


Figure 9.4.1.1 The workflow of the compiler implemented in this project

In order to describe the details of the compiler, the following code segment is taken from a source file of mine written in the DSL.

```

...
perform button onClick
Toast.makeText("Hello world!!!",
Toast.LENGTH_LONG).show();
end perform
...

```

As mentioned, when compiling the above code, the lexer will tokenise the code into tokens at first. Technically, the lexer firstly removes all the whitespace and line breaks in the beginning of each line of code. As a very brief example, if there is a line of code written like:

0	1	2	3	4	5	6	7	8
			f	o	o	b	a	r

the lexer will ignore the first three whitespace and begin to tokenise the code from position 3. The lexer splits the remaining code into tokens by whitespace. The first token will be used as an identifier which indicates whether the current line of code is in the DSL or in Java by matching with the keywords defined for the DSL. For instance, in the code segment example above, the first line of code “perform button onClick” is indicated as a line of code in the DSL since the lexer takes the code as *[KEYWORD “perform”] [ELEMENT “button”] [ELEMENT “onClick”]* while *perform* is one of the keywords of the DSL. By contract, the second line of code will be recognised as Java code, because after tokenising, the tokens will be, although they look weird, *[KEYWORD “Toast.makeText(”] [ELEMENT “\“Hello world!!!*

\", \"] and etc.. Obviously, the keyword in these tokens is not involved in the defined keyword list of the DSL, thus this line of code will be treated as a line of Java code.

Therefore, the tokens of each line of the DSL code will be interpreted, or say eval, by the interpreter. In order to maximise the performance and speed, the interpreter was designed and implemented to eval each group of tokens directly after tokenising by the lexer. In other words, it is much like the lexer and interpreter were designed to translate the target while reading it rather than to translate the target after reading it.

Since the DSL was designed to be Java-compatible to allow users prototyping and programming their Android applications in both the DSL and Java, Java was chosen as the target-language of the output produced by the interpreter in this project. Essentially, to achieve the objective, a group of Java code has been mapped with the defined keywords of the domain-specific language. When interpreting a specific token, the DSL code will be translated into to the equivalent code in Java. The following diagram shows the procedure and mechanism of the interpreter in this project.

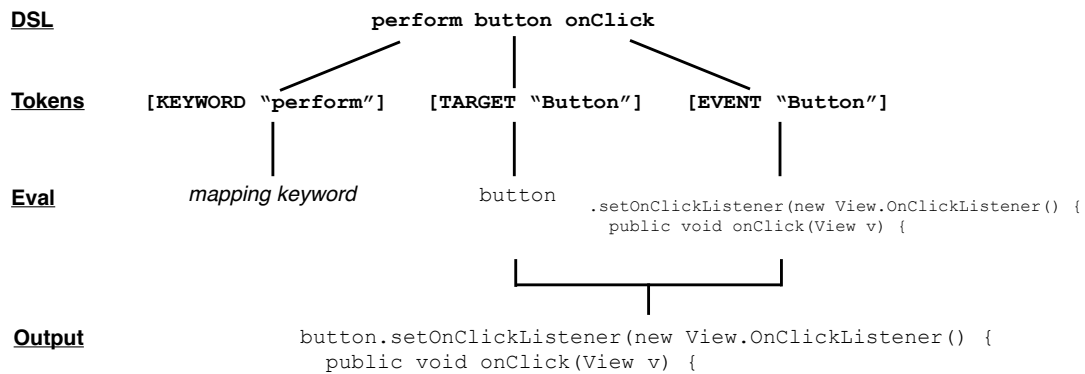


Figure 9.4.1.2 An interpreting example

We can see that the steps done by interpreter in the above example are *Eval* and *Output* while *Tokens* is processed by the lexer. Thus, by designing and building the lexer and interpreter, the compiler of the DSL was created. Therefore, the domain-specific language was implemented.

Besides, as a fully functional domain-specific language, the grammar is one of the most important component to be discussed with. Initially, based on the understanding of the DSL compiler, the language is completely compatible with Java. Thus, any kind of syntax or grammar is supported by the DSL.

Because the language was designed to focus on specific domain, the following grammars were implemented to support and handle the domain.

task activity||layout: Based on the understanding of Android, every single application running on this platform mainly consists of two parts of components: *activity* and *layout*. Hence, *task* was designed to identify the type of components that users are creating, and was required to be the first line of code in every source file in the DSL. For instance, when users are going to implement the behaviours of Android applications, *task activity* will be used. Technically, the keyword *activity* or *layout* in the *task* statement can be seen as the parameter.

Besides, when using the *task* statement, a scope of code will be covered by the statement. Which means, *task* starts a new indentation of code. Thus, an *end* tag is required at the end of the code sections in order to correspond to the *task*. The diagram below shows the pattern of the *task* statement.

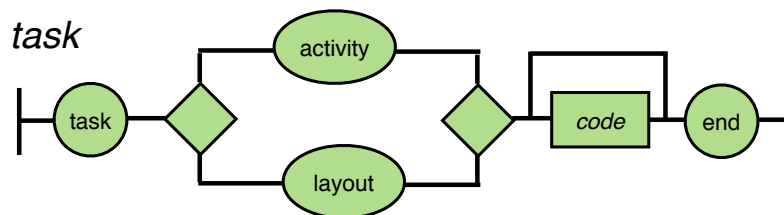


Figure 9.4.1.3 DSL *task* statement

layout [LAYOUT_MODE]: Comparing with the previous parameter *layout* in *task* statement, the *layout* here is used to define the type of layout that the Android application is going to use in its GUI. The value of the parameter *[LAYOUT_MODE]* can be any standard layout on the Android platform, such as *AbsoluteLayout*, *LinearLayout* and *RelativeLayout*. Inside of the *layout* section, users are able to declare their layouts with helps of other elements of DSL such as *control*. As another end-needed element, an end tag is also required in *layout*. The diagram below shows the pattern of the *layout* statement.

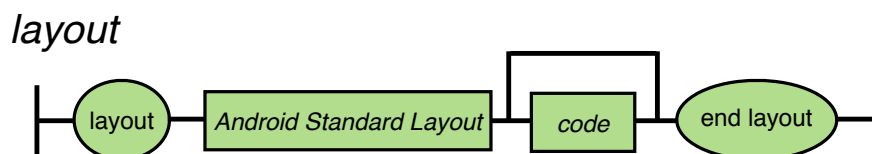


Figure 9.4.1.4 DSL *layout* statement

control [*Android_Widget*]: The statement of *control* was designed to define each widget in the GUI of Android application. The value of *Android_Widget* can be any type of Android widgets. Inside of the section of *control*, users are able to make configurations on the widgets. For example, if a *TextView* which displays “Hello World” is required, the DSL could be:

```
control TextView
    android:text = "Hello World"
end control
```

As we can see that the configuration of the widget can be directly written in the format of XML. The pattern of the *control* statement can be also described by the following diagram.

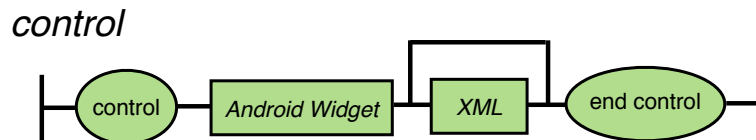


Figure 9.4.1.5 DSL *control* statement

def: The *def* statement in the DSL is responsible for declaring variables or data members in the Android activity. By default, the access type of each data member declared by *def* is protected. In order to make the data members public or private, *def* + and *def* - can be used, respectively.



Figure 9.4.1.6 DSL *def* statement

The above diagram shows the pattern of *def* statement in the DSL. As a Java-friendly language, the data type in the *def* statement can be any valid type in Java.

fun: The *fun* statement is quite similar with the *def*. The *fun* statement was designed to define the methods in classes. *fun*, *fun* + and *fun* - stand for the access type of protected, public and private, respectively. By comparing with the *def*, the *fun* statements will lead a scope of code for implementing the method. Thus, an end tag is required at the end of *fun* section. The pattern of *fun* is shown below.

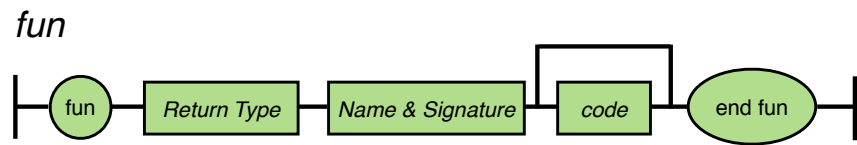


Figure 9.4.1.7 DSL *fun* statement

bind: As a graphical platform, Android has provided a way to define GUI components for the expected application. The *bind* statement in the DSL was built to bind an instance of widget or a data member with an existing raw resource defined in the layout of the Android application. For instance, suppose that a button, which is called *testButton*, has already been declared in the layout. By using the statement of “*bind myButton R.id.testButton*”, the *Button* instance *myButton* will be bind with the *testButton*. *R.id* is the default package which stores the id of each widget defined in the layout. The pattern of the *bind* statement is as follow.

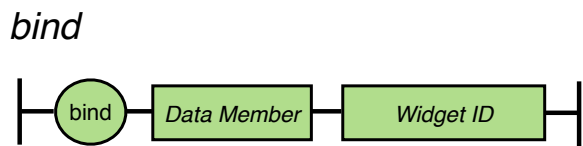
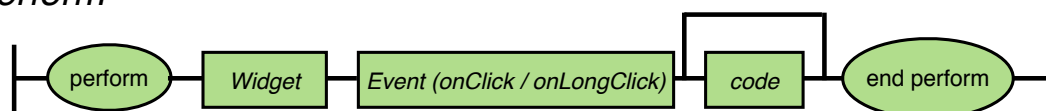


Figure 9.4.1.8 DSL *bind* statement

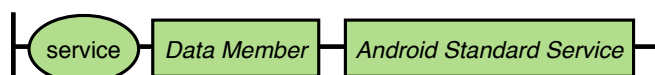
perform: After binding the widget instances with existing resources, the performances of each widget can be defined by using *perform*. In the current version of the DSL, only the events of “*onClick*” and “*onLongClick*” are supported. As one of the most importance components in the DSL, the *perform* statement will strongly benefit users by reducing the workload of development.

JAVA	DSL
<pre>toggleButton.setOnClickListener (new View.OnClickListener() { public void onClick(View v) { ... } })</pre>	<pre>perform toggleButton onClick ... end perform</pre>

As can be easily pointed out in the above comparison, the *perform* statement helps programmers to define the behaviours of Android applications much faster and more efficient. The pattern of the *perform* statement is shown below.

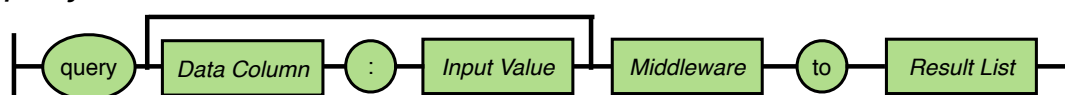
performFigure 9.4.1.9 DSL *perform* statement

service: As a multi-functional platform, Android provides a wide group of services natively. The *service* statement helps programmers to declare and bind an instance of service in their applications. The usage of the *service* statement is quite similar with *bind*. The pattern can be described as follow.

serviceFigure 9.4.1.10 DSL *service* statement

query: Since the project was being concentrated on the domain of book library querying system on the Android platform, a couple of elements, which help programmers to query remote database, were designed and implemented in the DSL. *query* is the first one will be used when querying data. Essentially, the statement of *query* was designed to query the remote MySQL database through the PHP middleware generated by DroidBuilder and then store the feedbacks in a list of result. When using the query statement, programmers need to provide the querying information (inc. the data column and its querying value), the location of the middleware and the name of the result list as the parameters. For instance, by executing the statement of “query id:usr_input http://localhost/php to queryResultList”, the expected data can be queried with id by the value of usr_input through the middleware at http://localhost/php, and will be saved in the result list of queryResultList.

The pattern of the query statement can be seen in the below figure.

queryFigure 9.4.1.11 DSL *query* statement

pull: After getting the feedbacks from the database, the *pull* statement will help programmers to extract the expected data from the result list into an ArrayList. For instance, by using “pull queryResultList.title:String titleList”, each data attribute of title will be extracted from queryResultList into titleList as String. The pattern of the pull statement can be specified as follow.

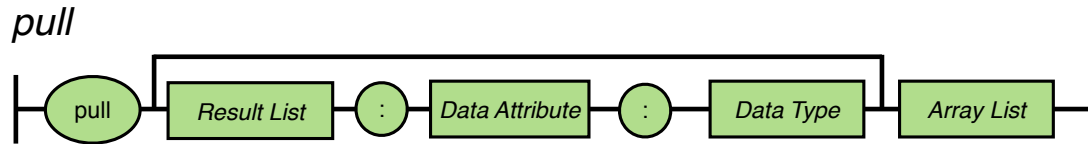


Figure 9.4.1.12 DSL *pull* statement

show: Eventually, after pulling the expected data into the array list, all the results in the array list will be displayed by using the statement of *show*. In stead of writing a large number of lines of code in the traditional way, the DSL allows users the display the results by providing only three parameters: the name of the array list of results, the activity that invokes the *show* statement and the activity used to display the results. For example, the results in resultList can be displayed in the activity of ResultViewActivity through MainActivity by using “show resultList MainActivity:ResultListViewActivity”.

The pattern of the show statement is shown below.

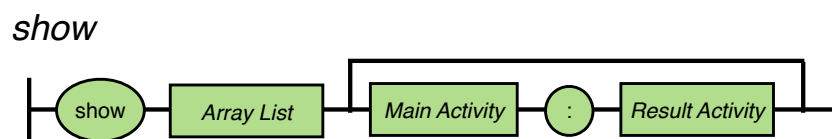


Figure 9.4.1.13 DSL *show* statement

Based on the aim of the project, the DSL and its compilers are used as the kernel of the entire system in order to allow users to prototype of their applications more precisely. For example, once the core components, based on the domain models in section 9, of the Android applications are prototyped, users will still be able to define the behaviours of their applications with helps of the DSL in details.

9.4.2 The Kernel

In order to relate the domain-specific language with DroidBuilder, the kernel of the entire system was implemented based on the DSL and its compilers. Essentially, the kernel is used to establish the connections between DroidBuilder, the domain-specific language and Android SDK in order to make the entire system becoming able to build, compile, sign and generate the Android projects. Technically, as the core of the kernel, a group of interfaces from the compilers of the DSL were implemented to be tightly connected with DroidBuilder. In other words, the compilers were designed to be easily and natively invoked by DroidBuilder by calling the specific interfaces.

Since the technical details of the domain-specific language, which is the most important component of the kernel, is discussed in last section, only the responsibility of the kernel will be addressed in this chapter.

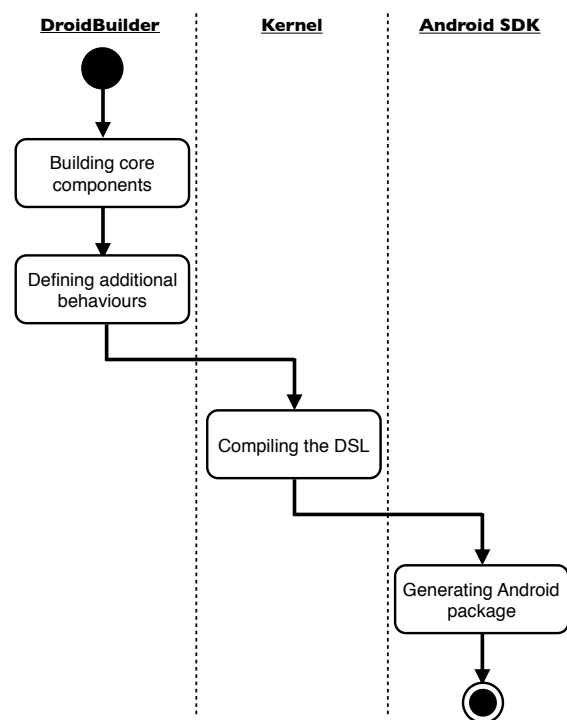


Figure 9.4.2 Activity diagram of prototyping Android applications using DroidBuilder

According to the above activity diagram, it can be pointed out that the kernel is acting as the middleware between DroidBuilder and the Android SDK. Originally, the kernel is used to compile the DSL code generated by the prototyping wizards in DroidBuilder and then invoke the Android SDK to compile the entire Android project. In other words, the responsibility of the kernel can be also understood as to make the reusable components of the domain-specific Android applications produced by DroidBuilder understandable to the Android SDK.

9.4.3 Domain Wizards in DroidBuilder

In order to help users handling with the reusable components or domain models defined in this project, a collection of wizards and dialogs were created as parts of the user interfaces of DroidBuilder. Technically, the domain wizards were designed to guide users defining the basic information used by each of the domain models mentioned in section 9.1 since the expected prototypes will be, therefore, specified based on those data given by users.

As an example, the following screenshot taken from DroidBuilder shows how the querying catalogues which mentioned in section 9.1 are defined with helps of the domain wizards. In this way, each of the reusable components and domain models designed and implemented in the domain engineering in this project becomes able to be easily defined, reused and organised in DroidBuilder.

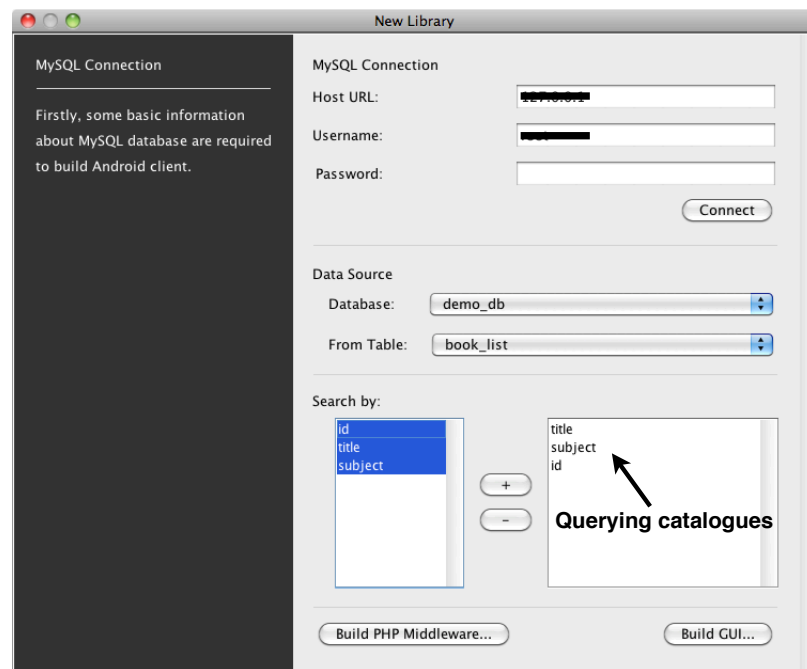


Figure 9.4.3 Defining catalogues using domain wizard

As we can see, the wizard shown in above diagram takes the input from users as the querying catalogues which will be used as a part of raw data when prototyping domain-specific Android applications. As a result, this part of the reusable domain component will be automatically prototyped and implemented in the Android application prototypes as shown below.

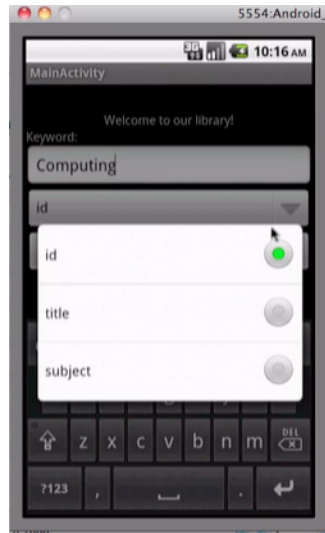


Figure 9.4.4 Querying catalogues in an Android app prototyped by DroidBuilder

10. Bridging Middleware

This section will describe the implementation of the bridging middleware designed and implemented in this project. Moreover, the mechanism used to retrieve JSON data on the Android applications prototyped by DroidBuilder will also be addressed.

10.1 Implementation

As mentioned in section 4.4, a middleware was designed to bridge the data transfers between Android applications prototyped by DroidBuilder and remote database in this project. Technically, the bridging middleware is a group of programs written in PHP, and is typically running on the server. The implementation of the bridging middleware was divided in two parts: receiving data from Android devices and sending back the querying result from database. By making comparison between these two steps, the implementation of the later one is very simple since a couple of extremely helpful APIs have already been natively involved in the standard library of PHP. The below code segment taken from the implementations of a bridging middleware shows how efficient it is when using the native APIs of PHP to query database and to generate JSON data.

```

<?php
mysql_connect("127.0.0.1","user","");
mysql_select_db("demo_db");

$query=mysql_query("SELECT * FROM books WHERE subject LIKE '%".
                    $_REQUEST['subject']."%'");

while($entry=mysql_fetch_assoc($query))
    $output[]=$entry;

print(json_encode($output));
mysql_close();
?>

```

Obviously, when the middleware in the above example is invoked by client, a connection with the specific MySQL database will be created by using the methods of *mysql_connect* and *mysql_select_db* at first. Then a query on the database will be proceed by calling *mysql_query*. Notice that, a built-in data member of PHP, *\$_REQUEST*, is also used here in order to receive the input from remote clients. This will be discussed in details in the next chapter.

Once the variable *query* has been assigned with the querying result, a loop of *mysql_fetch_assoc* callings will be used to fetch each row of the data records into *output*. Finally, the connection with database will be closed by *mysql_close* after sending back the JSON encoded data to the remote client by using *print* and *json_encode*, respectively.

Additionally, due to the simplicity of the source code, the speed of the PHP programs can be very fast at runtime. Therefore, the performance of the middleware and even the client applications can be ensured.

10.2 Data Receiving

As mentioned in last chapter, one of the most complex technical issues in the bridging middleware is on the data receiving. More precisely, this problem is about how the input data can be send to the middleware from the prototyped Android applications.

At very early stage of this project, the data receiving was planned to be resolved by using Java Socket. However, since the bridging program was finally designed to be written in PHP, Java Socket was no longer the best choice in this case. In stead of using Java Socket, the APIs in the package of *org.apache.http* in the Android SDK were used to solve the problem. Essentially, *org.apache.http* provided a chance of making remote connection with and sending input data to the PHP bridging programs

via HTTP connections. Within the HTTP connection, each of the PHP program will be invoked as a HTTP post which will be assigned with the input data as entities.

Moreover, by using *org.apache.http*, the Android applications will not only be able to send data to the bridging programs, but also have the ability to receive the feedbacks. The following code segment taken from an Android application prototyped by DroidBuilder shows the implementation of data receiving by using *org.apache.http*.

```
import org.apache.http.*;
...

//Data to send
ArrayList<NameValuePair> nameValuePairs = new
ArrayList<NameValuePair>();
nameValuePairs.add(new BasicNameValuePair("id",usr_input));

//http post
try{
    HttpClient httpclient = new DefaultHttpClient();
    HttpPost httppost = new HttpPost("http://127.0.0.1/php/
query_id.php");
    httppost.setEntity(new UrlEncodedFormEntity(nameValuePairs));
    HttpResponse response = httpclient.execute(httppost);
    HttpEntity entity = response.getEntity();
    is = entity.getContent();
}catch(Exception e){
    Log.e("log_tag", "Error in http connection "+e.toString());
}

//convert response to string
try{
    BufferedReader reader = new BufferedReader(new InputStreamReader
(is, "iso-8859-1"), 8);
    StringBuilder sb = new StringBuilder();
    String line = null;
    while((line = reader.readLine()) != null) {
        sb.append(line + "\n");
    }
    is.close();
    result = sb.toString();
}catch(Exception e) {
    Log.e("log_tag", "Error converting result " + e.toString());
}
```

As shown in the above code segment, in order to identify the input data from the Android client, an ArrayList of NameValuePair which contained by *org.apache.http.message.BasicNameValuePair* is used to store the Name/value pair which will be sent to PHP bridging programs. In order to invoke the PHP program, instances of *org.apache.http.client.HttpClient* and *org.apache.http.client.method.HttpPost* are declared. Meanwhile, the HttpPost instance is assigned with the URL of the PHP programs. Therefore, the remote

bridging programs will be executed by the Android clients through the interface of execute of the *HttpClient* instance.

Regards to the response of the remote PHP programs, the feedbacks will be firstly returned to an instance of *HttpResponse* which is used to retrieve the data into an inputstream with helps of *HttpEntity*. Hence, the results can be easily converted to strings by using *BufferedReader*.

10.3 JSON Parsing

As mentioned in section 6.1, the feedbacks that the Android client received from the PHP bridging middleware are encoded as JSON data. In other words, although the delivered querying results can be converted to the forms of string as shown in the previous chapter, those data are still more or less helpless to users. Thus, a solution of parsing the JSON data was highly deserved in the implementation of this project.

Fortunately, by using the package *org.json* officially involved in the Android SDK, the JSON data can be parsed very quickly. The code segment example below shows how a JSON data is parsed in a prototyped Android application.

```
import org.json.JSONObject;
import org.json.JSONArray;
import org.json.JSONException;

...
ArrayList<String> list = new ArrayList<String>();
try {
    JSONArray array = new JSONArray(result);
    for(int index = 0; index != array.length(); ++index) {
        JSONObject json_obj = array.getJSONObject(index);
        list.add(json_obj.getString("title"));
    }
} catch (Exception excpt) {
    ...
}
...
```

It is quite clear that when parsing a JSON data by using the package of *org.json*, an instance of *JSONArray* is declared and was assigned with the querying result received from the PHP bridging programs to be the initial data. When parsing the specific data we need, the value can be easily retrieved from each *JSONObject* in the *JSONArray* based on the given value of key.

Obviously, the algorithm of the parsing procedure is tightly based on the understanding of the forms of JSON discussed in section 6.1.1. The Pseudo-code of the parsing procedure is also shown below.

```

array: JSONArray
resultList: ArrayList
object: JSONObject

// Pseudo-code of JSON parsing

Procedure parsingJSON (queryingFeedbacks : String)
    // Assign the querying feedbacks to the instance of JSONArray
    array := create a new JSONArray instance with queryingFeedbacks

    // Retrieve the JSONArray
    For each name:value pair in the instance of JSONArray
        object := create a new JSONObject instance with
                     array.getJSONObject(loop_index)

        add object.getString(expected_name) to ArrayList resultList
    End loop

```

11. Keychain Key/value Storage Engine

When building a large-scale system such as DroidBuilder, it is reasonable to design and implement a key/value storage engine which can be used to store the runtime configurations of the entire system. As mentioned in section 6.2.3, Python was chosen as the programming language of Keychain in order to bring users opportunities to modify and maintain the engine however they wish.

The design of Keychain is very simple, when the key/value storage engine is used the entire system, Keychain will firstly generate an unique MD5 value for DroidBuilder to be its identifier. By doing so, the security of the data stored in Keychain can be seriously ensured. Meanwhile, a dictionary will be created for DroidBuilder to be the place for saving its configurations. Once a specific configuration needs to be loaded by DroidBuilder, Keychain will go back to the dictionary mapped with the given MD5 value and retrieve the expected data for DroidBuilder. Additionally, in order to improve the performance of Keychain, this engine was implemented to load the specific dictionary into RAM after initialisations.

However, when doing multi-language programming in a project, it is extremely necessary to build a middleware for the components programmed in different programming languages. Fortunately, with helps of the free project of Jython [15],

Python programs became possible to be bridged with Java applications. By using the Jython under the package of *org.python*, Java applications will not only be able to invoke a Python modules, but also have the chances to receive the returned value of the Python methods.

```
import org.python.core.*;
import org.python.util.*;
...

public String getKeychainList() {
    PyFunction func = (PyFunction)interpreter.get
("getKeychainList", PyFunction.class);
    return func.__call__().toString();
}
```

The above code segment taken from the Python-Java bridging program built for Keychain in this project shows how a Python method can be executed inside of Java.

Therefore, the entire system, which is mainly implemented in Java, will be able to be benefited by Keychain with helps of the middleware built for Keychain.

12. Testing

Since NetBeans was used as the project tool of this project, JUnit, which is an unit test framework for Java, was chosen for testing the domain-specific programming environment. The implementation contains several test cases. As a feature of JUnit, those test cases are able to be executed together from a test suite. The example below shows how the method *start* of class *XmlBuilder* in package *com.android.dsl.ctrl* can be tested.

```

package com.android.dsl.ctrl;
import junit.framework.TestCase;

public class TestXmlBuilder extends TestCase {
    XmlBuilder goodXml = null;
    XmlBuilder errorXml = null;

    protected void setUp() throws Exception {
        super.setUp();
        goodXml = XmlBuilder.getInstance();
        // Prepare to compile an error-free source file
        goodXml.initialise("/TEST_XML_OK");

        errorXml = XmlBuilder.getInstance();
        // Prepare to compile a source file which contains
errors
        errorXml.initialise("/TEST_XML_ERROR");
    }
    ...
    public void testXmlCompiling() {
        assertTrue(goodXml.start());
        assertFalse(errorXml.start());
    }
}

```

As shown in the above code segment, two instances of *XmlBuilder* were created in the *setUp()* method for testing. The instance *goodXml* was created with a source file, which contains no errors, while *errorXml* was initialised with another source file which contains errors. *assertTrue()* and *assertFalse()* are used to compare the actual return values and the expected results. In case that bugs are contained in the method, an exception will be thrown.

In order to ensure the correctness of this system, all the test cases must be passed in the testing phase.

13. Further Research & Limitation

There are still many limitations and improvements can be found and made on the entire system in the future. For example, in the current version, a Scratch-like graphical representation is used to represent the activities and workflows of Android applications. The common used UML diagram is not supported by the system right now. In other words, users may not completely understand the displayed diagrams in some cases. Besides, this programming environment does not have a very powerful GUI builder. Thus, users are not able to easily create elegant user interfaces for their prototypes. Hence, an upgrade version of the entire system should provide

capabilities that not currently supported while improving the performance of the system in the future.

This individual project concentrates on only one domain. This means the current version is not suitable for prototyping the applications which are not in the specific domain. However, based on the researches and studies on domain engineering in this project, the domain analysis, domain designs and domain implementations on other domains will be easily carried out in order to make the entire system to be multi-domain supported. This is what can be done in the future.

14. Conclusion

This paper represents a series of preliminary studies on application prototyping, domain engineering and system design and implementation. It is well accepted that domain engineering such as domain-specific language is acting as a more and more important role in computer science. It is also very important to produce an environment or a framework for prototyping domain-specific applications. The entire system implemented in this project provides functionalities to prototype fully functional applications in the domain of book library querying system on the Android platform. Users will be able to prototype runnable Android applications in a very short period of time. It will dramatically reduce the difficulties of developing Android application and shorten the development life cycle. However, what I have done so far is just the tip of the iceberg of the issue. More researches still need to be carried out in the future.

References

- [1] Gartner (2011). *Android Market share to near 50 percent*. Available from: http://news.cnet.com/8301-13506_3-20051610-17.html (last accessed: 30/07/2011)

- [2] Opera Software ASA. *Opera Mini & Opera Mobile Browsers*. Available from: <http://www.opera.com/mobile/> (last accessed: 19/08/2011)

- [3] Google Inc.. *Android Open Source Project*. Available from: <http://source.android.com/> (last accessed: 19/08/2011)

- [4] Reto Meier (2010). *Professional Android 2 Application Development (1st Ed.)*. Wiley Publishing, Inc.

- [5] Don Gentner and Jakob Nielsen (1996). *The Anti-Mac interface*. Communications of the ACM 39, 8, pp. 70-82.

- [6] Apple Inc.. *Mac OS X Human Interface Guidelines*. Available from: <http://developer.apple.com/library/mac/documentation/UserExperience/Conceptual/AppleHIGuidelines/index.html> (last accessed: 30/07/2011)

- [7] MIT Media Lab. *Scratch Project*. Massachusetts Institute of Technology. Available from: http://info.scratch.mit.edu/About_Scratch (last accessed: 30/07/2011)

- [8] JSON.ORG. *Introducing JSON*. Available from: <http://www.json.org/> (last accessed: 20/08/2011)

- [9] PHP. *Manual: json_encode*. Available from: <http://php.net/manual/en/function.json-encode.php> (last accessed: 30/07/2011)

- [10] DalvikVM. *Dalvik Virtual Machine Insights*. Available from: <http://www.dalvikvm.com/> (last accessed: 20/08/2011)

- [11] PHP. *PHP: Hypertext Preprocessor*. Available from: <http://www.php.net/> (last accessed: 20/08/2011)

- [12] M Hoy, D Wood, M Loy, J Elliot (2002). *Java Swing*. O'Reilly & Associates, Inc. Sebastopol, CA, USA.

- [13] Oracle Corporation. *NetBeans IDE*. Available from: <http://netbeans.org/> (last accessed: 20/08/2011)
- [14] Microsoft. *Model-View-Controller*. Available from: <http://msdn.microsoft.com/en-us/library/ff649643.aspx> (last accessed: 20/08/2011)
- [15] The Jython Project. Jython. Available from: <http://www.jython.org/> (last accessed: 21/08/2011)
- [16] X WANG. *The Code Blocks Focus Group*. James Cook University, Queensland, Australia
- [17] Krzysztof Czarnecki and Ulrich W. Eisenecker (2000). *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley.
- [18] University of Leicester Library, UK. Available from: <https://library.le.ac.uk/uhtbin/cgiisirs/22mg2dIa7e/DWL/107450017/60/502/X> (last accessed: 22/08/2011)
- [19] James Cook University, Australia. Available from: <http://hip.jcu.edu.au/> (last accessed: 22/08/2011)
- [20] Beijing University of Technology Library, China. Available from: <http://libaleph.bjut.edu.cn:8991/F> (last accessed: 22/08/2011)
- [21] Maarit Harsu (2002). *A Survey on Domain Engineering*. Institute of Software Systems, Tampere University of Technology. Available from: <http://practise2.cs.tut.fi/pub/papers> (last accessed: 18/08/2011)

Appendix

Appendix 1 -- JSON document for data transfer

This example is the JSON document being used to transfer database querying feedbacks from server to Android applications through middleware in this project. It describes a very simple collection of data records under the catalogue of Computing. Each record contains four attributes: title, author, publisher and catalogue.

```
{
  "": [
    {
      "title": "Java Java Java",
      "author": "Ralph Morelli & Ralph Walde",
      "publisher": "Prentice Hall",
      "catalogue": "Computing"
    },
    {
      "title": "Thinking in Java",
      "author": "Bruce Eckel",
      "publisher": "Prentice Hall",
      "catalogue": "Computing"
    },
    {
      "title": "Professional Android 2 Application
      Development",
      "author": "Reto Meier",
      "publisher": "Wiley Publishing, Inc.",
      "catalogue": "Computing"
    }
  ]
}
```