社団法人 電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

A Prototype of Comparison Tool for Android Applications Based on Difference of API Calling Sequences

Tetsuya KANDA†, Yuki MANABE†, Takashi ISHIO†,

Makoto MATSUSHITA†, and Katsuro INOUE†

† Graduate School of Information Science and Technology, Osaka University,
1-5, Yamadaoka, Suita, Osaka, 565-0871, Japan
E-mail: †{t-kanda,y-manabe,ishio,matusita,inoue}@ist.osaka-u.ac.jp

Abstract For both users and developers, it is important to select an appropriate application from similar ones. There are some ways to know the differences in terms of features in several similar applications, but it is not easy. In this paper, we present a prototype of a comparison tool which identifies features corresponding to differences between API calling sequences extracted from two Android applications. We conducted a case study to evaluate the prototype tool. The result shows that the proposed tool achieves comparing applications.

Key words Comparing Software, API Calls, Android

1. Introduction

Today, people can get many applications from many software download sites like Android Market [1] and Apple App Store [2]. Also, they can get a massive amount of source code easily from source code repository such as SourceForge.net [3] and Google Code [4].

Among a lot of applications, some applications have similar features. For example, about 2,960 applications are hit by searching with a word "calculator" on Android Market as of June 2011. Some tools can categorize them into similar application groups [5], [6]. A result of categorization shows which applications are alike, but the differences among applications in a similar group are not clear for users. To select the most appropriate application for a user from a group of similar applications, a tool to compare the features of the applications is necessary.

There are several ways to compare features in similar applications: trial use of applications, reading documents, and comparing source code. However, these methods require a lot of effort of users, and it is difficult for users to determine criteria for comparison. Comparison based on trial use of applications is also difficult for users who don't know what features are provided by each application. Reading documents also have some problem. If documents are not well-formed, it is hard to read and compare them. We think that documents are insufficient for the goal because they don't always describe all features, or no documents are available from the start.

If source code of the applications is available, comparing them looks a good way to compare the features of the applications. UNIX diff [7] is a simple way to compare source code. Unix diff shows only lines which have been changed, and Semantic Diff [8] shows changed lines and its effects on dependence relation between variables. If users want to know differences between two versions of software, these tools are useful. However, these text-based comparisons might extract the whole source code as diff if their design is different even though they use the same programming language. Therefore, it is hard to compare applications in easily understood form by these methods.

In this research, we propose a comparison tool that identifies the differences between two applications in terms of their features. To compare the features of applications, we focused on Application Programming Interface(API) calls since software developers often combine existing APIs to implement a feature [9]. We supposed that a feature of an application is a sequence of API calls; therefore, we can extract the feature-level difference instead of the textual difference by comparing API calls.

In the next section we discuss the background, and in Section 3, we show how we compare API calls. In Section 4, we describe our prototyping tool and in Section 5, we describe a case study with the tool. In Section 6, we discuss

the related work, and finally we conclude with a discussion of future work.

2. Background

2.1 Platform-dependent software development and API

Platforms such as OS and hardware have characteristic features like devices and UI. For example, most software for Microsoft Windows uses the same GUI components, and many of smartphones have cameras and touch screens. Application Programming Interface (API) is a set of functions to communicate with a specific platform. Each platform provides APIs to access the features commonly called from a lot of applications, and developers can implement a new feature by calling needed API.

2.2 API callings and feature

Software developers construct an application by combining several features. Developers often make software for a specific platform. When the target platform provides a high level API, developers implement features by combining several API calls. In such a platform, API calls explain the feature of the application. While a single API call may directly correspond to a single feature, a sequence of API calls corresponds to a feature in most cases.

In this research, we compare API calls in applications to compare the features. Tamada et al. proposed to compare the order and the frequency of API calls to detect some software theft [10].

2.3 Android

Android, developed by Open Handset Alliance [11], is a platform for mobile devices. It includes an operating system, middleware, and some applications like a web browser. Most devices using Android can connect the Internet via mobile phone network or Wi-Fi, and they have touch panels, gravity censors, and cameras.

Android application developers use Android SDK, which is based on Apache Harmony (an open source Java implementation) excluding GUI classes such as Swing. Instead, Android SDK includes additional APIs to control a camera, a GPS device, a touch screen and so on. Android SDK also contains Google Maps API by default.

3. Proposal method using API calls

In this section, we describe our method to compare two Android applications. Our method has two phases: making a knowledge-base and comparing two applications. Figure 1 shows an overview of the method.

In this research, we focus on Java applications. We use a term "an API calling sequence" to represent a sequence of

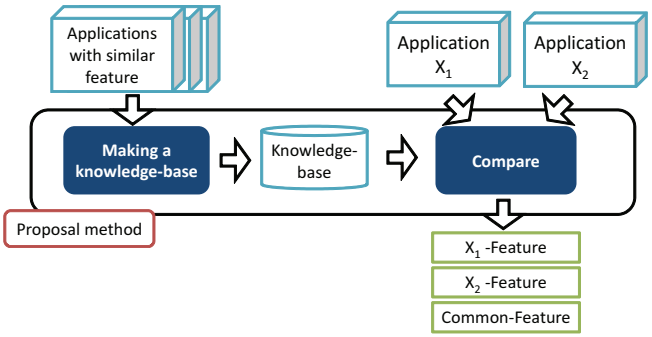

Figure 1 An overview of proposal method

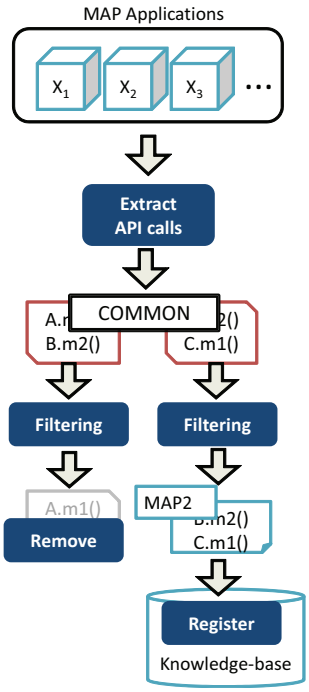

Figure 2 Making knowledge-base

method calls of API classes in a single method.

In making a knowledge-base phase, we name API calling sequences which explains characteristic feature and we define them as a knowledge-base. A knowledge-base consists of pairs of API calling sequence and its feature name. A knowledge-base helps to understand what feature is implemented by the API calling sequence.

In comparing phase, our method compare two applications, application X1 and application X2, and extract features that appear commonly(common-feature) and the features thgat appear in only one application(X1-feature, X2-feature).

3.1 The phase making a knowledge-base

Figure 2 shows the overview of the making a knowledge-base phase. This phase uses a set of the applications which have similar features as the input, and outputs a knowledge-base. Common API calling sequences extracted from applications are likely to implement the common features of

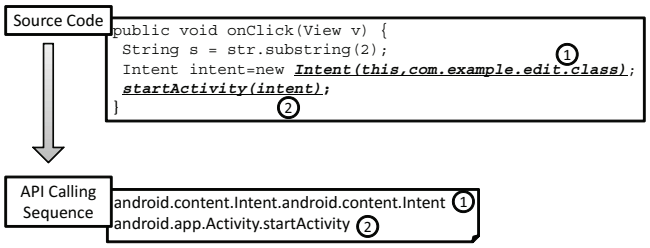

Figure 3 Extracting API calling sequences

the applications. Therefore, we can make a knowledge-base through naming API calling sequences which appear in two or more applications commonly (COMMON). In this research, we made a knowledge-base as following steps;

Step A Collect source code of applications that have a specific feature, and extract API calling sequences that communicate with Android platform.

Step B Compare API calling sequences and get COMMON.

Step C Filter COMMON and then give a feature name to each API calling sequence, then store them in a knowledge-base.

Figure 3 shows an overview of extraction of API calling sequences in Step B. API calling sequences are extracted according to the following steps. First, we analyze source code of application using MASU [12]. MASU is a plug-in platform for metrics measurement and it extracts method calls in analysis. Next, we extract method call whose API call is unique to Android from the result of analysis. Android SDK contains libraries from Java, libraries from Apache HttpComponents project, Android original API and so on. In this research, we selected methods whose fully qualified names start with "android." or "com.google.android" as Android API.

The result of API call extraction is a pair of an API call and a line number. We define this pair as API calling information. Then, sort API calling information in order of line numbers. If multiple API callings appeared in the same line, sort them in alphabetical order. We don't care about control flow statements and the order of methods in a class. Save API calling sequence with information of its owner file, class, and method at the same time.

To get COMMON, we make a table like Figure 4, and find out common sequences of API calls which include two or more API calls. Following information will be extracted as COMMON.

- API calling sequence
- Appearance information (API calling information of where the API calling sequence starts)

After the COMMON API calling sequences are extracted,

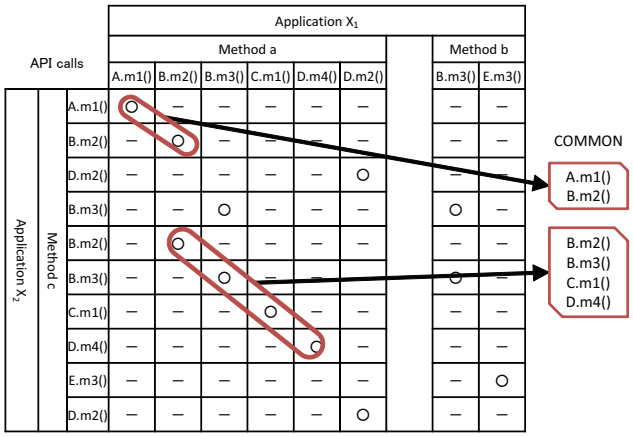

Figure 4 Extractiong COMMON

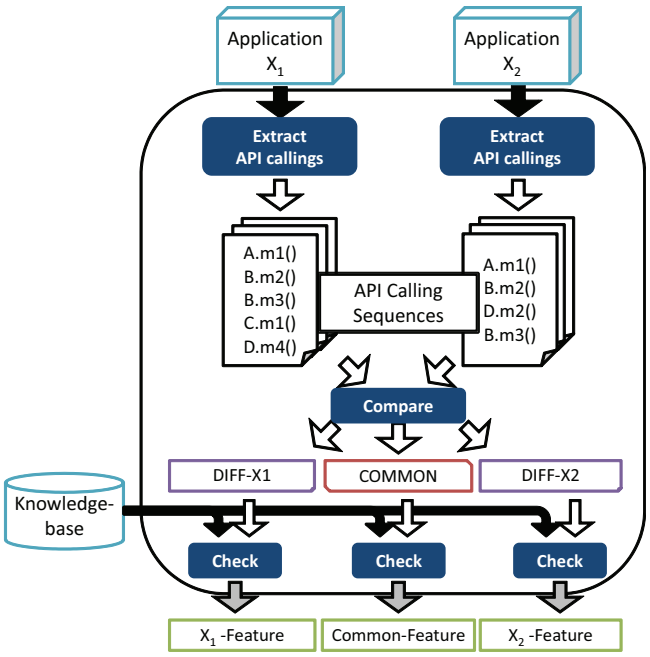

Figure 5 Overview of comparison

we conduct a manual filtering. The purpose of filtering is to remove API calling sequences that are not explaining specific feature. Criteria of filtering are:

- Control or get information using devices like camera
- Notification
- Whether using Intent (call for another application)

After the filtering, we assign a feature name to each API calling sequence. A pair of a feature name and an API calling sequence is stored into a knowledge-base.

3.2 The phase comparing two applications

Figre 5 shows the overview of comparison phase. Our method compares two applications as following steps:

Step 1 We analyze source code of two applications and extract API calling sequences. We extract only method calls of API classes in the target platform, i.e., Android SDK classes.

Step 2 We compare two API calling sequences and iden-

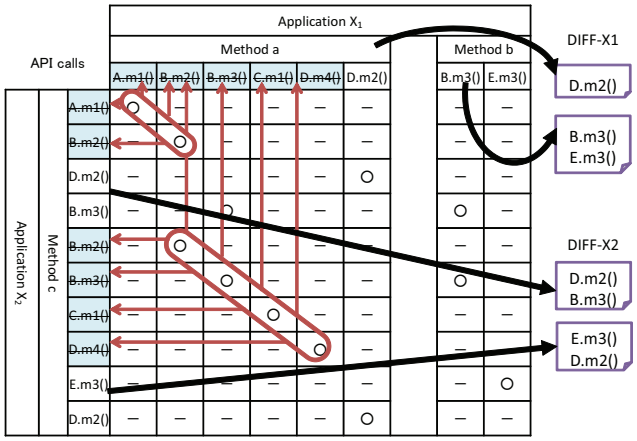

Figure 6 Extracting DIFF

tify the COMMON sequences. The differences are regarded as DIFF-X1 and DIFF-X2.

Step 3 We translate API calling sequences in COMMON and DIFFs into features according to a knowledge-base.

Procedure of extracting API callings and identifying COMMON are the same as the first phase making a knowledge-base. To get DIFF, mark API calls which extracted as COMMON like Figure 6. After the COMMON sequences are identified, the rest (unmarked) calling sequences are regarded as DIFFs. A long API calling sequence containing a COMMON sequence in the middle is divided to two API calling sequences involved in a DIFF.

We check COMMON and DIFF with a knowledge-base; if an API calling sequence is found in the knowledge-base, its corresponding feature name is output. Here is an example. A COMMON sequence is "ABCDE". A knowledge-base contains three features: feature X and its API calling sequence is "BC", Feature Y and its API calling sequence is "CB", Feature Z and its API calling sequence is "AC". Feature X is found in the COMMON sequence, but feature Y and Z are not found by our method.

The source code location of the calling sequence is also recorded for further analysis.

4. A prototype tool

We have developed a prototype tool implementing the proposed comparison approach.

A user of the tool should prepare a knowledge-base. The tool analyzes two selected applications and shows feature names corresponding to COMMON and DIFFs.

4.1 Selecting targets

On booting up the tool, startup window appears (Figure 7). Radio buttons are used for selecting whether the user want to compare and show the result or only analyze and not to show the result of comparison. Analyze only mode can be

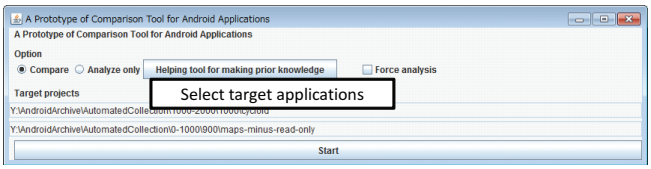

Figure 7 Startup window

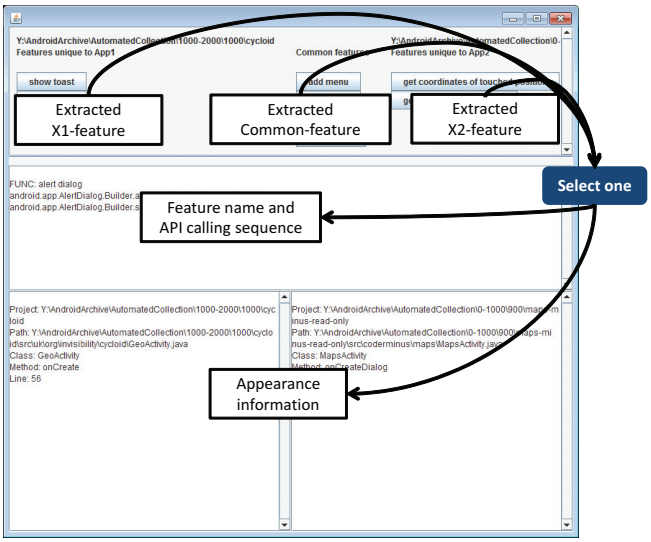

Figure 8 View for showing result of comparison

used for making knowledge-base. The tool uses existing result of analysis, so check "Force analyze" if the user wants to do analysis again. Click a label of the project name then appear a dialog to select a target project. The tool starts analysis and comparing when "start" button is clicked.

4.2 Showing result of comparison

Figure 8 is a view for showing result of comparison. The window has two parts of view and the upper part of the window is divided into three parts. X1-feature, Common-feature, X2-feature are showed in the left, center and right of the view respectively. One of the buttons is clicked, and then details are showed in the lower part.

5. Case study

We performed two case studies to answer following research questions:

RQ1 Is an API calling sequence corresponding to a feature of an application?

RQ2 Is our prototype tool able to clarify differences of two applications?

RQ1 concerns the validity of a knowledge-base, and RQ2 is related to the comparison phase.

5.1 Case study 1

To ensure that API calling sequence shows the feature of application, we collect some application and extract API calling sequences. The steps of this case study are as follows:

Step 1 We extracted API calling sequences from each of five applications.

Step 2 We saw feature names corresponding to each API calling sequences with knowledge-base, and ensure that an application is distinguished from the other application.

As a target of case study, we used 11 applications labeled "Map" in Google Code as applications with feature "map". This case study uses 5 applications to compare and the rest to make a knowledge-base. Table 1 and Table 2 shows a list of applications and its LOC.

Table 1 Application list for making a knowledge-base

Application name	LOC	#API calls
OpenGPSTracker	8122	1099
mapsforge	37326	1407
OSMandroid	3150	175
TripComputer	14487	825
shareyourdrive	2761	346
savage-router	1041	66
total	66797	3918

Table 2 Application list for comparison

ID	Application name	LOC	#API calls
App1	MapDroid	6387	1160
App2	cycroid	1278	761
App3	yozi	5348	159
App4	maps-minus	1785	218
App5	BigPlanetTw	4139	432
	Total	18937	2730

Table 3 The number of features appeared in target applications

#applications	0	1	2	3	4	5	Total
#features	5	8	3	0	6	1	23

Table 4 An example of differences of features in 5 applications

ID	App1	App2	App3	App4	App5
Alert Dialog	✓	✓	✓	✓	✓
Get Latitude and Longitude	✓		✓	✓	✓
Show Toast (pop-up message)	✓	✓		✓	✓
Set Latitude and Longitude	✓				✓
Submenu					✓

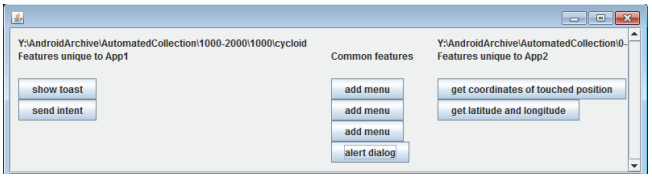

Figure 9 Comparison cycroid and maps-minus

We extracted 156 API calling sequences from 6 applications. We have assigned names to 23 API Calling sequences and stored into a knowledge-base.

Table 3 shows the result of Step 1. 80% of features in the knowledge-base was appeared in the targets. The result shows that API calling sequences stored in the knowledge-base are involved in many applications.

Table 4 shows a part of the result of Step 2: an example of differences of features in 5 applications, and Table 5 shows API calling sequences of some features. Some features are common to all five applications and some features appeared only in few applications. The result also shows that some detected differences did not correspond to map features; for example, "Alert dialog", "submenu" and "toast (pop-up message)". These features are associated with user interface. Table 4 shows the features involved in 5 applications; our approach successfully distinguished applications using API calling sequences.

5.2 Case study 2

The second case study is to investigate the detailed behavior of our tool. We reuse knowledge-base which we made in case study 1. Target applications are App2, cycroid, and App4, maps-minus.

Figure 9 is the window of the tool displaying the result of comparing App 2 and App 4. Our prototype tool can show common features and features appeared only one or the other application. Output of the tool is same as the result of case study 1.

5.3 Discussion

The result of case study 1 shows that a particular API calling sequence appears in two or more applications which

Table 5 Example of API calling sequences

Feature name	API calling sequences
Show toast(pop-up message)	android.widget.Toast.makeText android.widget.Toast.show
Set latitude and longitude	android.location.Location.setLatitude android.location.Location.setLongitude
Alert dialog	android.app.AlertDialog.Builder.android.app.AlertDialog.Builder android.app.AlertDialog.Builder.setTitle
Get latitude and longitude	android.location.Location.getLatitude android.location.Location.getLongitude
Submenu	android.view.Menu.addSubMenu android.view.SubMenu.setIcon

has a similar feature. It follows from this fact that our idea is reasonable; we can detect the differences of features by using API calling sequences.

The result of case study 2 shows that our tool can compare and distinguish features of two applications. Some features doubly detected because they have some patterns of sequence (like pattern ABC and pattern ACB, both of them implements the same feature).

Features found by the tool are considered that the application actually has them, but features which are not reported are not always considered that the application doesn't have them. API calls which are not registered in a knowledge-base are not found by the tool, so existing features may not be detected if the knowledge-base contains not enough data.

In the case study, we cannot distinguish domain-specific features from generic features, since we have used only "map" applications to make a knowledge-base.

6. Related work

To compare source code of two similar programs, many tools have developed. Unix diff [7] is a line-based text comparison tool and widely used for comparing source code. Semantic diff analysis [8] uses data flow pairs for showing the semantic effects of the modification. Comparing source code is effective for knowing difference of two versions of same application. However, these methods are not helpful in comparing applications developed individually.

MUDABlue [5] and LACT [6] categorize software automatically. These tools can classify applications with similar feature in same category, but we want to know detailed differences between applications in same category.

7. Conclusion

We have developed a prototype tool to compare two Android applications using API calling sequences extracted from the applications. The tool extracts COMMON and DIFF API calling sequences, and then translates them into common features and unique features of the applications.

We performed a case study with Map applications. The result ensures that API calling sequences are usable for comparing applications.

In the future work, we would like to support developers to give an appropriate name to a sequence of API calls. In addition, we would like to judge whether some API calling sequences explains the same feature or not.

Acknowledgment

This research was supported by Japan Society for the Promotion of Science, Grant-in-Aid for Scientific Research (A) (No.21240002) and Grant-in-Aid for Exploratory Research (No.23650015) .

References

[1] "Android Market," http://market.android.com/.
[2] "Apple - iPhone - Learn about apps available on the App Store," http://www.apple.com/iphone/apps-for-iphone/.
[3] "SourceForge.net," http://sourceforge.net/.
[4] "Google Code," http://code.google.com/.
[5] S. Kawaguchi, P.K. Garg, M. Matsushita, and K. Inoue, "MUDABlue: An automatic categorization system for open source repositories," Proc. 11th Asia-Pacific Softw. Eng. Conf., pp.184–193, Busan, Korea, Nov. 2004.
[6] K. Tian, M. Revelle, and D. Poshyvanyk, "Using latent dirichlet allocation for automatic categorization of software," Mining Soft. Repositories, 2009. 6th IEEE Int. Working Conf. on, pp.163 -166, may 2009.
[7] E.W. Myers, "An O(ND) difference algorithm and its variations," Algorithmica, vol.1, no.1, pp.251–266, March 1986.
[8] D. Jackson, and D. Ladd, "Semantic diff: a tool for summarizing the effects of modifications," Softw. Maintenance, 1994. Proc., Int. Conf. on, pp.243 -252, sep 1994.
[9] T. Xie, and J. Pei, "MAPO: Mining API usages from open source repositories," Proc. of the 2006 Int. Workshop on Mining Software repositories, pp.54–57, Shanghai,China, 2006.
[10] H. Tamada, K. Okamoto, M. Nakamura, A. Monden, and K. Matsumoto, "Design and evaluation of dynamic software birthmarks based on api calls," Information Science Technical Report NAIST-IS-TR2007011, ISSN 0919-9527, Graduate School of Information Science, Nara Institute of Science and Technology, May 2007.
[11] "Open Handset Alliance," http://www.openhandsetalliance.com/.
[12] A. Saito, G. Yamada, T. Miyake, H. Higo, S. Kusumoto, and K. Inoue, "MASU: Metrics assessment plugin-plutform for software unit of multiple programming languages," 2009 Int. Workshop on Empirical Softw. Eng. in Pract., 2009.