Summarizing Services of Java Packages

Maen Hammad, Anas Abuljadayel, and Mohammad Khalaf

***Abstract*—Program comprehension is essential for code maintenance and evolution activities. It saves time and efforts of developers who want to perform any code changes. It also minimizes the chances of introducing bugs. Textual summaries for source code provide great help to code understanding activities. This paper presents an approach to automatically generate textual summaries for services implemented in java packages. The summary is generated by analyzing the source code of methods defined the package. Each method represents a service provide by the package. Each service is summarized as a natural language textual description. The generated summary for a method mainly includes the used data and the names of invoked methods. Summaries of all methods defined in a package are refined and integrated to be reported as a comprehensive summary for the services provided by the package. The generated summaries are useful in different ways. They can be used by developers in their maintenance activities. They also can be useful for the documentation purposes.**

***Index Terms*—Program comprehension, software maintenance, source code summarization.**

I. Introduction

Program comprehension is essential to software maintenance. Understanding the current structure of the source code, its design and its behavior is the key to performing corrective, adaptive and/or corrective maintenance. The most important part of program comprehension is to understand the source code. Developers who understand the source code can perform maintenance tasks in less time and efforts. Source code may not be clearly written, organized or commented. In the worst case, the code is understandable only by the developer who wrote it. Another issue that affects program comprehension is the lack of documentations. Many software projects don't have a well written documentation or sometimes there is no documentation at all.

The problem under consideration in this paper is how to support program understanding efforts for the source code of java services. Services are mainly implemented by the methods of packages. Understanding the service helps developers in their maintenance tasks, especially perfective maintenance. To update or add new services to a subsystem, it is necessary to understand the current services of the subsystem under consideration.

Source code of projects, that have with no or weak documentation, need to be examined by developers to understand it. Manual browsing and analyzing the source code of methods, to understand the services they provide, takes lot of time and efforts. A solution that may be very useful for developers is to provide them with textual summaries for the source code of services. Reading a textual description about the structure and the behavior of the services of subsystems saves developers' time and efforts.

Another benefit from textual summaries for java services can be utilized for educational purposes. One possible application is to hide the source code of services and show the summaries for students. Then, students are asked to implement the main structure of services based on the textual summaries. In this case, students can practice how to implement the functional requirements of the system.

Recently, there are some proposed approaches on automatically generate summary comments for source code artifacts such as java methods [1], parameters [2] and java classes [3]. These approaches helped in simplifying the source code to those developers who didn't wrote it by generating these summary comments. In this paper, we present an approach to automatically summarizing the services of java packages from the source code of methods. The generated summary is a natural language textual description about the services of a java package. The approach examines the source code of classes and methods to extract useful information that are used in the summary. Names of methods, used data and invocations are the main components of the generated summaries. So, the final result is a descriptive summary for the package that includes the main services provided by the package.

The proposed approach automatically analyzes the source code of a specific package to generate the summary. It is a light weight approach and can be realized as a plug-in tool to automatically document the services of java packages. In this paper, we illustrate the idea and detail the proposed approach.

The paper is organized as follows. Section II discusses the related work in the area. Section III presents the proposed approach followed by our conclusions and future work in Section IV.

II. Related Work

The closest related work to our approach is the work done by Sridhara *et al*. [1]. They presented a text generation technique that takes a java methods signature and body as an input and outputs a natural language text that summarizes what actions are done by the method. Our approach is a light weight approach that is focused on the group of methods defined in the package. In [2], Sirdhara *et al*. presented a technique to generate comments that provide an overview of the role of a parameter inside a method, and connect that parameter with the methods main intent so that it gives a description of the methods functionality and the parameter

Manuscript received November 12, 2014; revised February 5, 2015.

The authors are with the Department of Software Engineering, The Hashemite University, Zarqa, Jordan (e-mail: mhammad@hu.edu.jo, abuljadyel1990@gmail.com, hamoodnooman@gmail.com).

role in fulfilling that functionality.

Moreno *et al*. [3] proposed a technique to automatically generate natural language descriptions for java classes, presuming no documentation of the code exists. The tool determines the class and method stereotype and uses them in conjunction with heuristics to select which information to be included in the summary. The tool takes a Java project as input, and for each class, it outputs a natural-language summary.

A review research in automatic summarizing in the last decade is presented in [4] which concludes that automatic summarization techniques has made steady progress, with better evaluation and better tools & applications, but their research stated the poor motivation of summarizing systems in relation to the factors affecting them.

Moreno *et al*. [5] considered that summary is based on the stereotype of the class; they proposed J Summarizer, which is an Eclipse plug-in that automatically generates natural language descriptions of Java classes. The tool takes as input a Java class and produces a short, text-based description of that class, which is inserted as a Java doc comment into the class.

Haiduc *et al*. [6] mentioned that a combination of automated text summarization techniques is more reliable for source code and helps in better program comprehension. They focused on investigating the suitability of several summarization techniques, mostly based on text retrieval methods, to capture source code semantics in a way similar to how developers understand it.

Dragan *et al*. [7] presented an approach to automatically determine a class stereotype; this stereotype is based on the frequency and distribution of method stereotype in the class. They implemented a tool that automatically reverse engineers a class's stereotype and re-documents the class. The tool can analyze an entire system and re-document it efficiently.

Hill *et al*. [8] presented a novel approach that automatically extracts natural language phrases from source code identifiers and organizes them in a hierarchy. They proposed an algorithm to automatically extract and generate noun, verb, and prepositional phrases from method and field signatures, capturing word context of natural language queries. These phrases naturally form a hierarchy that allows the developer to quickly identify relevant program elements by reducing the number of relevance judgments, while the phrases help the developer to formulate effective queries.

Sridhara *et al*. [9] presented an automatic technique for identifying code fragment that implement high level abstraction of actions and expressing them as natural language description, their approach was the first for identifying code fragments of statement sequences, conditionals and loops that can be abstracted as a high level action. There are also other approaches that are based on modeling for automatic summarization of source code as in [10].

III. The Approach

Java projects consist of a set of packages. Each source package mainly contains a set of classes and/or interfaces that have related or similar functionalities. In software design, each package is considered as a subsystem that has related set of services or functions provided to the users.

In a general view, the proposed approach automatically extracts a textual summary for the services provided by a java package. Services are mainly realized by the methods that are implemented in the package. So, the source code of each method is analyzed to extract a textual summary about the service it provides.

The automatic identification of services is based on extracting and processing the methods of classes defined in the package. The source code of each method is extracted and analyzed. The summary of a method is generated from its contents. The contents include local variables, reference types, and methods' invocations. The generated summaries of methods are integrated and refined to provide a complete summary for the services of the package.

The process of generating the textual summary for a specific package is detailed as follows and summarized in Fig. 1:

1) The input is the source code of a java package.
2) The source code is transformed into the XML markup format srcML.
3) The names of classes defined in the package are extracted by parsing srcML.
4) The data fields and the methods of each class identified in Step 1 are extracted.
5) For each method identified in Step 2, the following information are extracted:
   - Defined local variables.
   - Used and defined references.
   - Names of invoked methods.
6) A textual summary is generated from the extracted information for each method.
7) The generated summaries for each package are integrated and refined.

The process starts by reading the source code of a java package. In the first step, the source code is transformed into the srcML [11] format. srcML is parsed to extracted all syntactic information about the source code. It is parsed to extract names of classes, methods, data fields, invoked methods…etc. Then, a summary is generated for each method. Finally, all summaries of a specific package are refined and integrated as one summary for the services provided by that package.

The following subsections detail the proposed approach and illustrate the idea with examples.

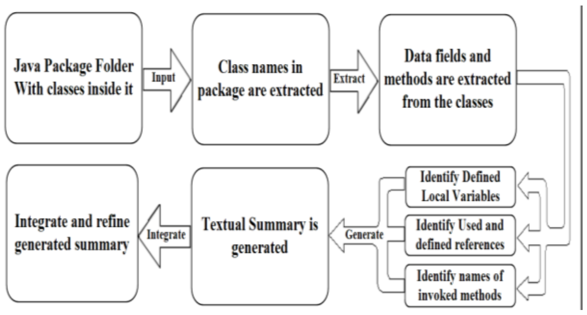
Java Package Folder With classes inside it
Input
Class names in package are extracted
Extract
Data fields and methods are extracted from the classes
Identify Defined Local Variables
Identify Used and defined references
Identify names of invoked methods
Generate
Textual Summary is generated
Integrate
Integrate and refine generated summary

Fig. 1. Process steps of the proposed approach.

A. The Input

The process starts with a complete java package that contains the java source files. The source code of each java file is analyzed to generate the summaries for package. To ease the process of extracting information from source code, it is converted to the XML representation srcML [11]. srcML is a XML representation in which each code element is tagged with its syntactic information. Once the code is presented in srcML, we can parse it to extract all information needed about classes and methods. More about srcML representation is detailed in [11] and the supporting tools can be freely downloaded from the URL (http://www.srcml.org/).

B. Identification of Classes and Methods

Classes and methods are extracted by parsing the srcML representation of the source code.

The parsing process is applied by using a set of XPath queries that parse the XML representation of the source code. In the first step, classes are identified and extracted. Then for each identified class, all its data fields and methods are extracted. For each method, all its defined local variables and used references are identified by parsing the srcML representation for each method. The names of invoked methods are also extracted for each method.

All extracted information from the source code is used in the process of summary generation. The proposed process is detailed in the following section.

C. Generating a Summary for a Method

First of all, the names of methods, classes and reference types are split using camel case as in [12]. For any method, a summary is generated from its name, local variables, used reference types and invoked methods. Then, all summaries for a specific package are integrated.

Fig. 2 shows a snapshot from class *SpacerPanel* that is part of the ArgoUML open source project. The class is defined in the package *org.argouml.swingext*.

As shown in the figure, the class contains two methods. This means that it provides two different services in the package. The first method *getMinimumSize* is split into "getMinimumSize" and the second method is split into "getPreferredSize". Both methods, in the figure, use data fields from the class with no local variables. The two methods also do not invoke any other methods.

```
public class SpacerPanel extends JPanel {
privateint w = 10, h = 10;

public Dimension getMinimumSize() {
return new Dimension(w, h);
}

public Dimension getPreferredSize(){
return new Dimension(w, h);
}
}
```

Fig. 2. A snapshot from *SpacePanel* class.

For each method, its local variables and method invocations are included in the generated text for the service. The generated summaries for the two methods in Fig. 2 are:

- **The service is:** getMinimumSize**for**Spacer Panel. **The service uses the attributes of** Spacer Panel**:** w **and** h**.** **The servicereturns**Dimension.
- **The service is:**get Preferred Size**for**Spacer Panel. **The serviceuses the attributes of** Spacer Panel**:** w **and** h. **The service returns**Dimension**.**

The bold texts are templates that included in all generated summaries. The name of the service is the words of the method's name followed by the words of the class. The names of used data fields and local variables are also included in the summary. The data fields used in the method are preceded with "**the attributes of**" in the summary followed by the class name.

The class name is also split into words based on camel case naming convention. On the other hand, local variables are preceded with "**local data**" in the summary and reference types are preceded with the word "**types**".

```
public class LeftArrowIcon implements Icon {
public void paintIcon(Component c,
Graphics g, int x, int y) {

intw=getIconWidth(),h=getIconHeight();
g.setColor(Color.black);

Polygon p = new Polygon();
p.addPoint(x + 1, y + h / 2 + 1);
p.addPoint(x + w, y);
p.addPoint(x + w, y + h);
g.fillPolygon(p);
}
}
```

Fig. 3. Source code for *paintIcon* method.

Fig. 3 shows another example for method *paintIcon* that is defined in class *LeftArrowIcon*in the package *org.argouml.swingext*. The method uses local variables, references and invokes some other methods. The generated summary for the method, based on the proposed approach, will be as follows:

- **The service is:** paint Icon**for**LeftArrowIcon. **The service uses local data:** w, h, x **and** y**. The service uses types:** Icon, Component, Graphics, Color**and** Polygon**. The service**get Icon Width, get Icon Height, set Color, add Point **and**fill Polygon.

The summary includes the name of the method at the beginning. It also lists both local variables (w, h, and y) and used reference types (Icon, Component, Graphics, and Polygon). Local variables are distinguished by the words "**localvariable**". Finally, the names of invoked methods are included after the words "**The service**". Including invoked methods enhance the generated summary and make it more meaningful. Actually, invoked methods participate in shaping the behavior of the service.

D. Integration and Refinement

After all methods that are defined in a package are summarized, the generated summaries are refined and then integrated. The refinement includes the followings:

- Summaries for abstract methods are ignored because they have no implementation and they will be overridden in some subclass. The same is applied on the methods of interfaces.
- Summaries of overloaded methods are grouped in one summary as one big service.
- Summaries for constructors are ignored because constructors are mainly used for initializing data fields.

Finally, all summaries for the package are integrated and reported as one big summary for the services provided by the package. The final summary also includes the following information about the package:

- The number of abstract, final and concrete classes
- The total number of interfaces.
- The total number of methods.

IV. Conclusions and Future Work

The paper proposed an approach to automatically generate summaries for services of java packages implemented by methods. The approach automatically parses the source code of classes and methods inside the package. Then it extracts useful information from the parsed code and used them to generate natural language summary for the given package. The summary is focused on the services provided by the package. The approach is a light weight approach in which to complex processing is needed to generate the summary.

The proposed approach helps in program comprehension tasks. It helps developers to get a brief description about the service provided by the method before examining its implementation. Another benefit is the automatic documentation of the source code, especially the behavior of methods. The summaries can be useful for educational purposes to help students in implementing functional requirements.

We are currently working on realizing the approach as an Eclipse plug-in tool. We are also working on improving the generated summary by including more descriptive information about the importance of the service, how it is connected to other services and its impact on external services.

References

[1] G. Sridhara, E. Hill, D. Muppaneni, L. Pollock, and K. Shanker, "Towards automatically generating summary comments for java methods," in *Proc. the IEEE/ACM International Conference on Automated Software Engineering*, 2010, pp. 43-52.

[2] G. Sridhara, L. Pollock, and K. Shanker, "Generating parameter comments and integrating with method summaries," in *Proc. the 19th IEEE International Conference on Program Comprehension (ICPC'11)*, 2011, pp. 71-80.

[3] L. Moreno, J. Aponte, G. Sridhara, A. Marcus, L. Pollock, and K. Shanker, "Automatic generation of natural language summaries for java classes," in *Proc. the 21st IEEE International Conference on Program Comprehension (ICPC'13)*, 2013, pp. 23–32.

[4] K. Sparck-Jones, "Automatic summarising: The state of the art," *Information Processing and Management: An International Journal* vol. 43, issue 6, November 2007, pp. 1449-1481.

[5] L. Moreno, A. Marcus, L. Pollock, and K. Shanker, "J summarizer: An automatic generator of natural language summaries for java classes," in *Proc. the 21st IEEE International Conference on Program Comprehension*, 2013, pp. 230-232..

[6] S. Haiduc, J. Aponte, L. Moreno and A. Marcus, "On the use of automated text summarization techniques for summarizing source code," in *Proc. the 17th Working Conference on Reverse Engineering*, 2010, pp. 35-44.

[7] N. Dragan, M. Collard, and J. Maletic, "Automatic identification of class stereotypes," in *Proc. 26th IEEE International Conference on Software Maintenance (ICSM'10)*, 2010, pp. 1-10.

[8] E. Hill, L. Pollock and K. Shanker, "Automatically capturing source code context of nl-queries for software maintenance and reuse," in *Proc. the 31st IEEE International Conference on Software Engineering (ICSE'09)*, 2009, pp. 232–242.

[9] G. Sridhara, L. Pollock, and K. Shanker, "Automatically detecting and describing high level actions within methods," in *Proc. the 33rd IEEE International Conference on Software Engineering (ICSE'11*), 2011, pp. 101–110.

[10] P. Eddy, A. Robinson, A. Kraft, and C. Carver, "Evaluating source code summarization techniques: replication and expansion," in *Proc. the 21st IEEE International Conference on Program Comprehension (ICPC'13*), 2013, pp. 13–22.

[11] M. L. Collard, H. H. Kagdi, and J. I. Maletic, "An XML based lightweight C++ fact extractor," in *Proc. the 11th IEEE International Workshop on Program Comprehension (IWPC'0*3), 2003, pp. 134-143.

[12] E. Hill, Z. P. Fry, H. Boyd, G. Sridhara, Y. Novikova, L. Pollock, and K. Vijay-Shanker, "AMAP: Automatically mining abbreviation expansions in programs to enhance software maintenance tools," in *Proc. the 2008IEEE International Working Conference on Mining Software Repositories (MSR),* 2008.

Maen Hammad is an assistant professor in the Software Engineering Department at the Hashemite University, Jordan. He completed his Ph.D. in computer science from Kent State University, USA in 2010. He received his master degree in computer science from Al-Yarmouk University, Jordan and his B.S. in computer science from the Hashemite University, Jordan. His research interest is software engineering with focus on software evolution and maintenance, program comprehension and mining software repositories.

Anas Abuljadayel is a master student in the Software Engineering Department at the Hashemite University, Jordan. He completed his B.S. in software engineering from the Hashemite University, Jordan in 2012. His research interest is software engineering with focus on software testing and software maintenance.

Mohammad Khalaf is a master student in the Software Engineering Department at the Hashemite University, Jordan. He completed his B.S. in software engineering from the Hashemite University, Jordan in 2013. His research interest is software engineering with focus on software maintenance and cloud computing.