ATLANTIS PRESS

Advances in Engineering Research, volume 113

International Forum on Mechanical, Control and Automation (IFMCA 2016)

An Approach for Extracting UML Diagram from Object-Oriented Program Based on J2X

Haoqing Zhang[1,a]

[1]School of Computer Science and Technology, Jilin University, Changchun 130012, China

[a]zhanghq_jlu@qq.com

Keywords: Reverse Engineering, UML, Class Diagram, Sequence Diagram

Abstract. Reverse engineering software system to extract UML model can help developer understand the structure and behavior of system. An approach for reverse extracting UML class diagram and sequence diagram from Object-Oriented program was proposed in this paper. Different from other extracting methods, our method is based on the intermediate language J2X which is a kind of markup language and stores semantic and structure information of programs. The method first transforms source code to J2X representation, then uses OFG(Object Flow Graph) and CFG(Control Flow Graph) analysis method to extract UML model based on J2X. The method can extract more accurate relationship between classes using OFG analysis. The method has a good accuracy and performance in extracting UML model according to the experiment in a set of test cases.

Introduction

Reverse engineering is the process of analyzing a subject system to identify the system's components and their interrelationships and create representations of the system in another form or at a higher level of abstraction[1]. It is of great significance to improve the efficiency of program understanding and system maintenance. UML(Unified Modeling Language) is a kind of high level abstraction and is de facto standard for modeling object-oriented system. It provides powerful modeling mechanism, can be used to model requirements, design, implementation and testing of software development process. So to reverse engineer software system to extract UML model can help developer understand the structure and behavior of the system and then facilitate software maintenance and reconstruction.

In the past few years, many reverse extraction methods for UML models have been proposed[2-6]. An approach to reverse engineer UML sequence diagrams from execution traces based on formal transformation rules was proposed in [3]. Aimed to extract sequence diagram, the authors in [4] defined a UML extensions(UML Control-Flow Primitives) to capture general flow of control and then gave a algorithm for mapping the control flow to sequence diagram. In [5] the authors presented a method combining static and dynamic analyses to extract sequence diagram. In general, these methods can be divided into two categories, static analysis and dynamic analysis. The former extracts the structure information of software by the analysis of source code, the latter analyzes the behavior of system by testing run (white box testing and black box testing), so it can capture the runtime information of system.

The method in this paper is a kind of static analysis methods and mainly focuses on the extraction of UML class diagram and sequence diagram. Class diagram is used for modeling the structure of Object-Orient software, while sequence diagram is used for modeling the behavior of system. Different from other reverse extraction methods, our method is based on the intermediate language J2X which is a kind of markup language and store semantic and structure information of programs. The method first transforms source code to J2X representation, then extracts UML diagram based on J2X representation.The advantages to use J2X as a intermediate of reverse extraction as follow.

1)It's more convenient to storage, exchange and extract semantic and structure information of programs using J2X because it is a kind of XML and XML is good at this.

2)J2X can isolate high-level reverse methods from low-level differences of programming language. So it's possible to use same reverse method such as OFG to extract program in different program language based on J2X.

Using J2X as a intermediate, we extracted the basic information about syntax and structure of program and then constructed UML class diagram using these information. And also discussed about the aspect to extract more accurate class relationship using the object flow analysis method. Furthermore, a method to construct UML sequence diagram combining control flow analysis and object flow analysis was presented in this paper. Finally, we used a set of test case to verify the correctness and accuracy of method and evaluated the results.

Table 1 A subset of J2X elements

The Element in J2X	Description
<compilation_unit> </compilation_unit>	Compilation Unit
<package> </package>	Package Statement
<import> </import>	Import Statement
<interface> </interface>	Interface Declaration
<class> </class>	Class Declaration
<class_body> </class_body>	Class Body
<field> </field>	Field Declaration
<method> </method>	Method Declaration
<block> </block>	Statement Block
<type> </type>	Type Identifier
<name> </name>	Identifier
<while> </while>	While Statement

J2X and Approach Overview

This section first introduces the intermediate language J2X, then outlines the method framework of reverse extracting UML diagram proposed in this paper.

J2X. The key of reverse engineering is to extract the information used to construct high-level abstraction. Direct information extraction method based on source code generally gets AST(Abstract Syntax Tree) using compilation techniques, then visits AST to get useful information. It's usually complicated and can only get limited information.

In order to storage, exchange and extract semantic and structure information of programs easily, we defined a intermediate language named J2X based on AST. Abstract syntax tree (AST) is a tree to represent syntax structure of source, each node in the tree represents a syntax element in source code, and program's nested structure is reflected by the hierarchical structure of the tree. As a kind of XML, the J2X can reflect the structure of abstract syntax tree naturally. In fact, the logical structures of XML document is a kind of tree-like hierarchical structure.

According to syntax rules and abstract syntax tree structure of program, we defined DTD (Document Type Definition) to regulate the elements and elements' structure of J2X document. There are totally 89 elements in the DTD, Table 1 only shows a part of them. A example of J2X representation of a short Java code segment is shown in Fig.1. You can notice that the J2X

representation is very long compared with the source code, it is unavoidable because there are lots of semantic labels in J2X. Because of adding lots of semantic labels in program elements, J2X can storage, exchange semantic and structure information of programs easily, it also shield underlying differences of program language. More important, using many mature XML parser like Dom4j, we can extract information we needed efficiently.

To transform source code to J2X representation is not difficult. Firstly parse the source code to generate AST, then visit AST to get needed syntax node, finally add semantic labels for these node and output them to XML file.

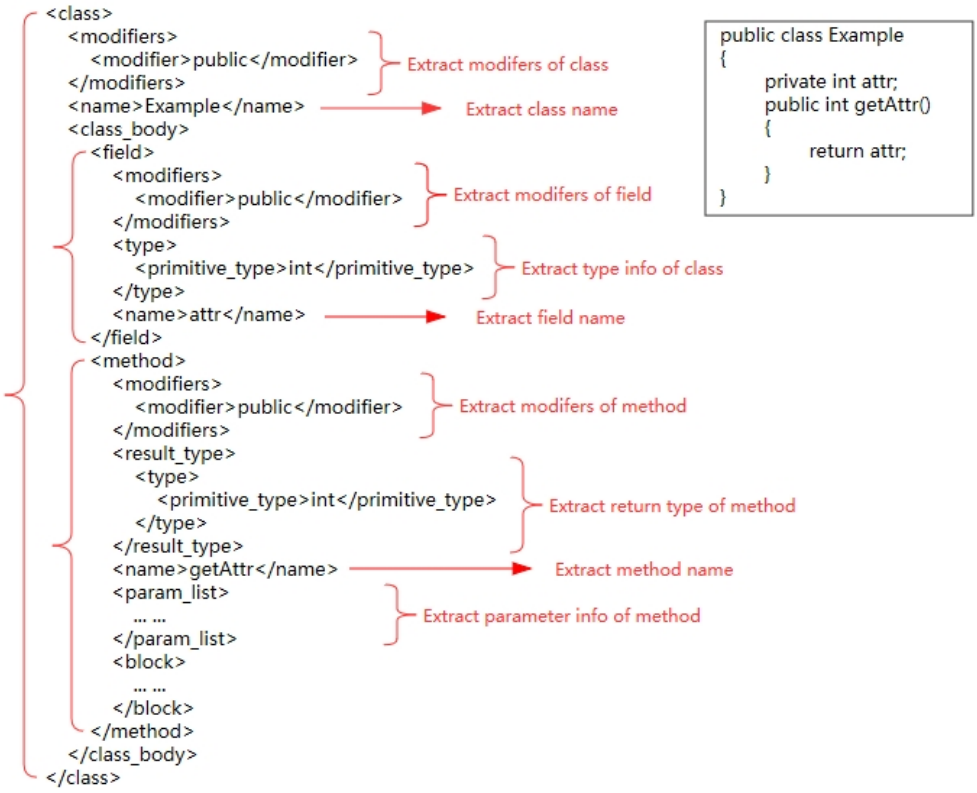


Fig.1 A example of J2X

Approach Overview. Based on J2X, our reverse engineering method framework is shown in Fig.2. There are four main steps in the method framework:

1.Transform source code to J2X representation, the way to transform was mentioned in the previous section;

2.Extract basic information such as class name, method name, etc. based on J2X;

3.Construct object flow model based on J2X, then combine object flow model and the information extracted in step 2 to construct UML class diagram;

4.Construct control flow model based on J2X, then combine control flow model and the object flow model extracted in step 3 to construct UML sequence diagram.

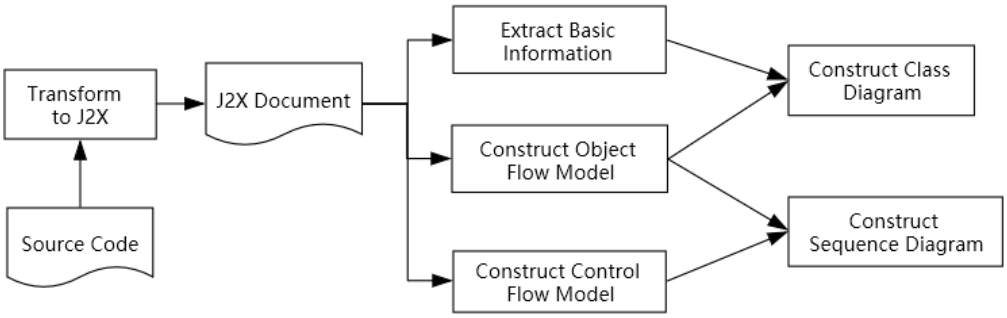
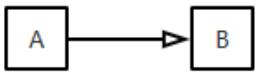


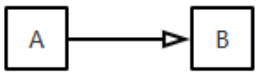
Fig.2 Method framework

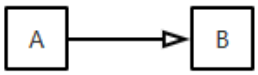
Table 2 Basic class information

Class	Package name	Class name	Modifier(visibility, abstract, etc.)		
Interface	Package name	Interface name	Modifier(visibility, final, etc.)		
Method	Class name	Method name	Return type	Parameter	Modifier
Parameter	Method name	name	Type	Modifier(final)	
Field	Class name	name	Type	Modifier(visibility, static)	

Table 3 A mapping between the type of relationship and J2X representation

Generalization
 	
class A extends B
{
}
	<class>
	<name>A</name>
	<extends>
	<name>B</name>
	</extends>
	<class_body/>
	</class>
		

Implementation
 	
class A implements B
{
}
	<class>
	<name>A</name>
	<implements>
	<name>B</name>
	</implements>
	<class_body/>
	</class>
		

Association
 	
class A
{
B b;
}
	<class>
	<name>A</name>
	<class_body>
	<field>
	<type>
	<name>B</name>
	</type>
	<name>b</name>
	</field>
	</class_body>
	</class>
		

Dependency
 	
class A
{
  void f1(B b){}
  void f2(){C c;}
}
	<class>
	<name>A</name>
	<class_body>
	<method>
	<return_type>
	<void/>
	</return_type>
	<name>f1</name>
	<param_list>
	<param>
	<type>
	<name>B</name>
	</type>
	<name>b</name>
	</param>
	<param_list>
	<block/>
	</method>
	<method>
	.....
	</method>
	</class_body>
	</class>
		

Class Diagram Extraction

Basic Class Diagram Extraction. UML class diagram models the static structure of Object-Orient system. To extract class diagram from J2X needs to get two kinds of information: basic class information and relationships among classes. Basic class information is shown in Table 2, includes class information, interface information, method and field information. It's easy to extract these information from J2X, our implementation tool can take different action in different XML elements during parse J2X document and save them in corresponding data structure for further processing. Fig.1 is a intuitive show of extraction process, it starts a extraction of class information when the parser encounters a "class" elements in J2X document.

There are four kinds of relationship between two classes: generalization, implementation, association and dependency. To extract them, we give a mapping between the type of relationship and J2X representation in Table 3. It's easy to detect generalization and implementation relationship by simply detecting "extends" and "implements" labels. Association is regard as "has-a" relationship, it can be detected by finding the class type filed. It's needed to identify local variable or method parameter of class type to extract dependency relationship.

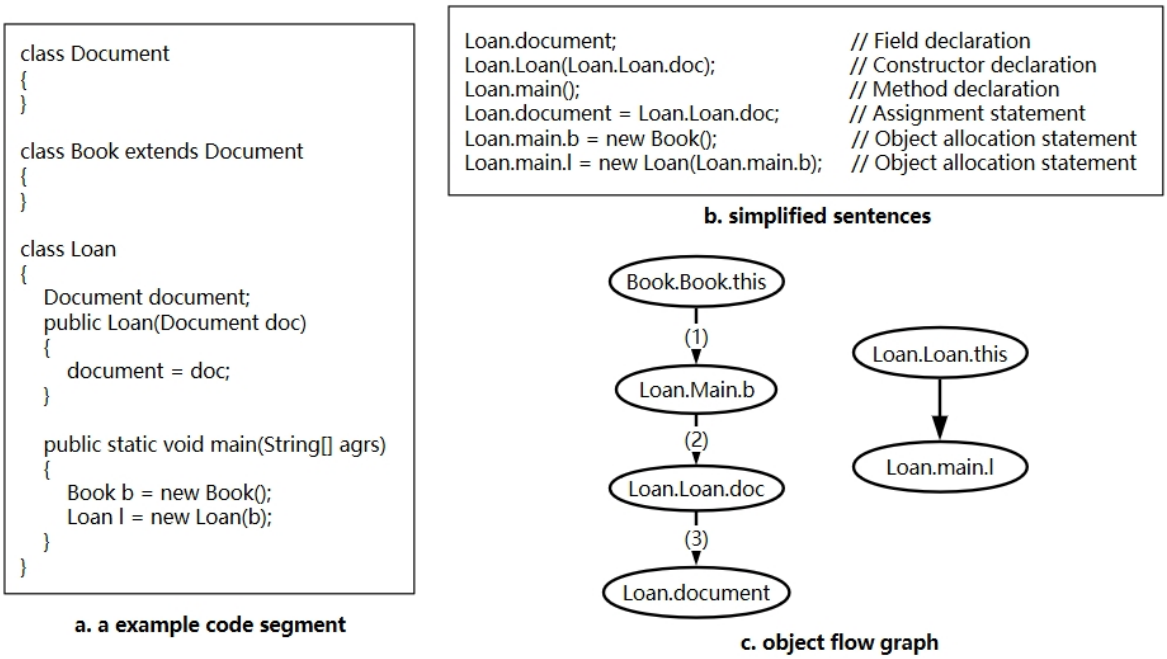


Fig.3 A example of OFG

Extract More Accurate Relationship. According to previous section, the detecting of generalization and implementation relationship is intuitive and has credible result. But to association and dependency relationship, using the mapping rules in Table 3 sometimes can't get accurate result. Considering the program in Fig.3(a), class Document is the super class of class Book and there is a Document type field in class Loan. We can get result that there is a association relationship between class Loan and Document according the mapping rules in Table 3. But carefully analyze the code, a instance b of class Book is assigned to the formal parameter doc, then doc is assigned to the field document in Class Loan. So in fact there is a association relationship between class Book and Loan, not between class Loan and Document.

We use OFG(Object Flow Graph) analysis method to solve this problem. OFG is a directed graph used to reflect the propagation of objects in object-oriented program.The OFG allows tracing the flow of information about objects from the object creation by allocation statements, through object assignment to variables, up until the storage of objects in class fields or their usage in method invocations[2]. So we can detect the real type of a object reference by tracing OFG. Two steps must be gone in order to generate OFG.

First, convert J2X representation to a kind of simplified sentences which eliminates control statements and the declaration statements associated with none-class type, and keeps only the declarations and statements associated with objects and objects transfer. The declarations to reserve include field declaration, method declaration, constructor declaration. The statements to reserve include method call statement, assignment statement and object allocation statement. Besides,every name in abstraction sentence should be written in global name (dotted the package name, class name and identifier). For example, the code in Fig.3 (a) can be convert to the simplified sentences shown in Fig.3 (b).

Second, construct OFG based on the simplified sentences. Local variables, class field, and method parameter in simplified sentences is called program location. The construction of OFG follows two rules:

1)Each program location are a node of OFG;

2)There is a directed edge between two node if there is a assignment, argument passing or object allocation relation between two program location.

Follow the rules, we can get the OFG (Fig.3 (c)) of simplified sentences shown in Fig.3 (b). Left object flow in Fig.3 (c) shows that a object instance b of class Book is allocated(1), then the b is assigned to parameter doc(2), finally parameter doc is assigned to field document(3). So the reference document is point to the object of class Book although it's declaration type is class Document.

Sequence Diagram Extraction

Sequence diagram models the behavior of Object-Orient System through describing objects and message sequence among objects. There are three kind of information needed to be obtained to reverse extract sequence diagram[3]: 1)the objects involved in interaction; 2)the messages exchanged between objects; 3)the control flows and control conditions associated with the interaction of objects. The objects involved in interaction can be obtained using object flow analysis. But for last two, it seems not so intuitive. In order to extract them, our method construct CFG(Control Flow Graph) model of program first.

CFG Model. CFG is a abstraction of method call and control statement (conditional statements, looping statements) that include method call in a method. Fig.4 is the class diagram of CFG model. As Fig.4 shows, a Class contains multiple methods, a Method consists of some CodeSection in order. The attribute signature of Method is in the formal of packageName.className.methodName. A CodeSection may be a MethodCall or a ConditionalSection.

There are three kinds of ConditionalSection Loop, Alt and Opt. Loop can be for, while or do loop statement in actual code, Alt is the if-else condition statement, Opt stands for branch conditional statement. ConditionalDescription in ConditionalSection is the description of control condition.

MethodCall is a method call, it's attribute signature indicates which method is invoked, attribute outerMethod is the name of the method that includes current method call, attribute outerClass is the name of the class that includes current method call.

To construct the CFG model from J2X representation, we totally parse J2X document two passes. First pass is a preprocessing aimed to remove the statement that is neither method call statement nor the statement not include method call. Then extract method call and control statement in second pass.

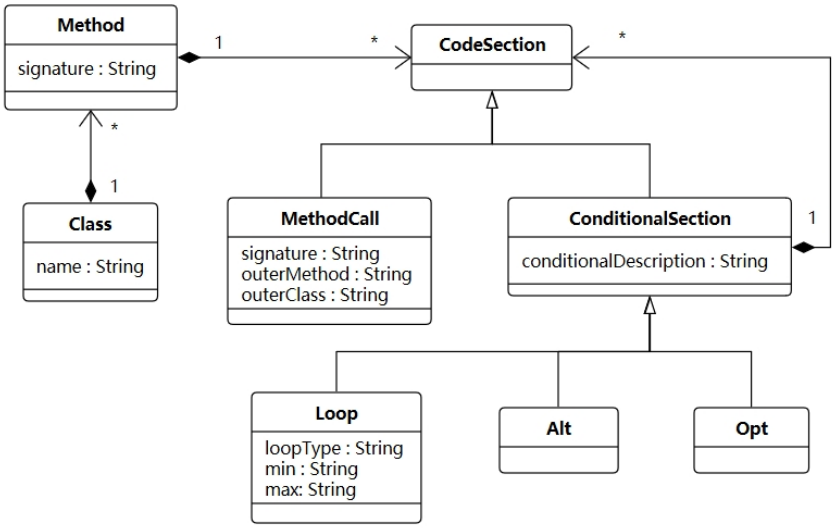


Fig.4 The class diagram of CFG model

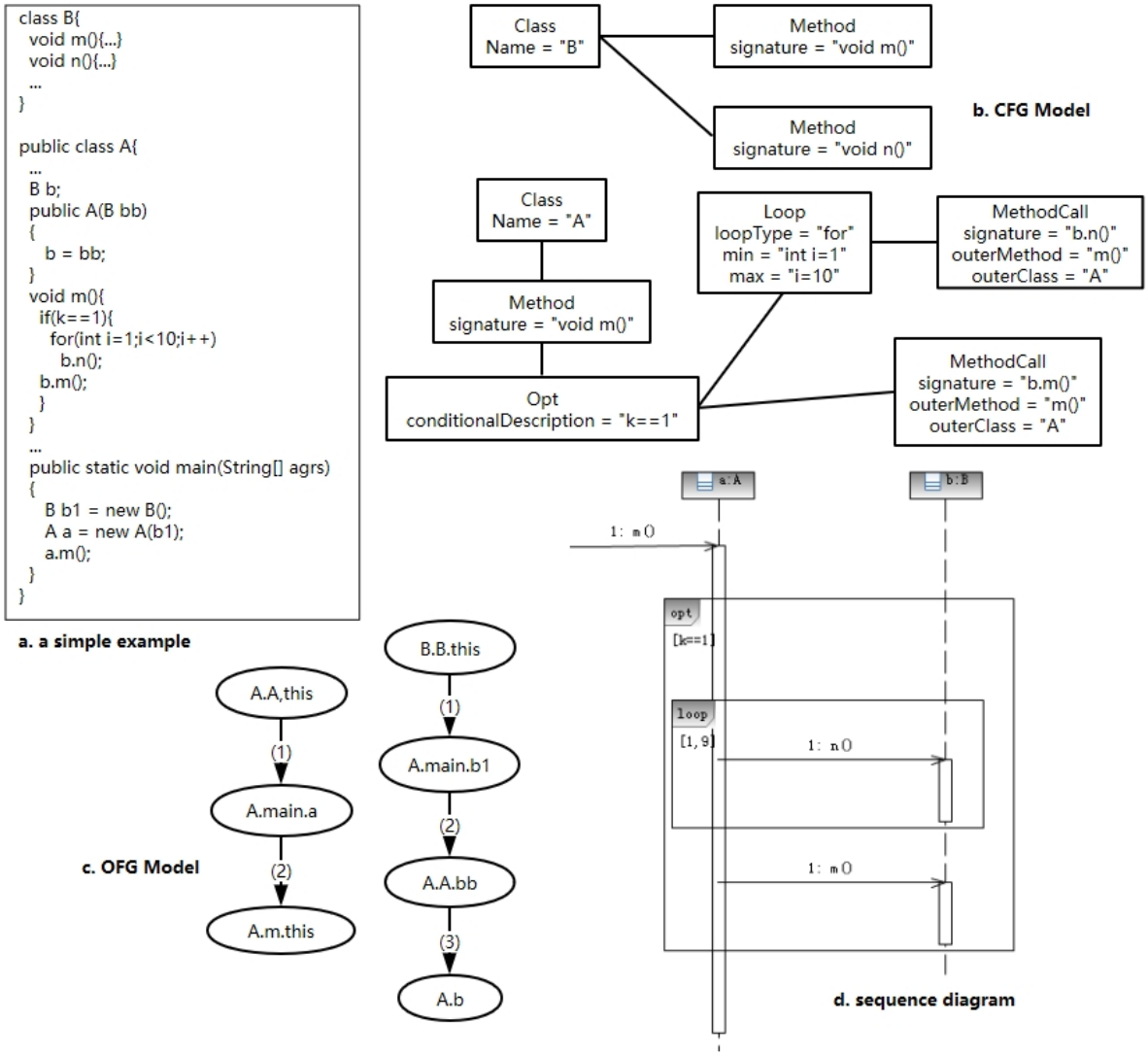


Fig.5 A example of reverse extracting sequence diagram

A Example. This section, we use a simple example to illustrate the process of reverse extracting of sequence diagram (Fig.5 (a)). There are two classes A and B in the example, class B includes two method m and n, class A holds a instance b of class B and sends messages to b through invocation of method m and n.

First, construct CFG model (Fig 5 (b)) from J2X representation of example program using the way proposed in section "CFG Model". There are two methods m and n in class B, more details about the CFG of class B are ignored. For class A, there is a Opt section in method m, the Opt section include a Loop and a MethodCall and the Loop section also include a MethodCall.

Second, use OFG analysis method to determine the objects involved in interaction namely the objects which send or receive message. Fig.5 (c) is the OFG of example program, according left part ,we know a points to a instance of class A, b points to a instance of class B similarly. On the other hand, there is a invocation to method m of object a in method main, and two invocations to method m and n of object b in class A's method m.

So we can get information as follows based on the analysis of the above:

1) a is a instance of class A namely a(A), b is a instance of class B namely b(B);

2) there is message m which is sent to a(A) from main thread;

3) there are two messages n and m which are sent to b(B) from a(A);

4) the message n is enclosed by a Loop(with range 1 to 9), the Loop and the message m are enclosed by a Opt(with boolean condition "k==1").

It's enough for us to construct sequence diagram use the information above, the result is shown in Fig.5(d).

Experiment and Evaluation

A reverse engineering tool called J2UML was implemented based on the method framework proposed in this paper. J2UML includes a parser which is used to get AST of source code and is automatic generated by JavaCC[7] which is a Java parser generator. J2UML can read Java code as input and then generate class diagram model corresponding. We use a set of small-scale test cases to test the accuracy and performance of reverse extracting UML class diagram(Table 4). The tool runs on a PC with Windows system (Windows 10 Professional edition 32bit, 3G Memory and 2.93Ghz Intel Core 2 Duo CPU). Column (a) in Table 4 is the number of class included in test case, column (b) is running time, column (c) and column (d) means the number of class and relation pairs extracted by tool J2UML respectively.

Table 4 Experiment result

Test Case	(a)	(b)	(c)	(d)	(e)	(f)
eLib	10	257ms	10	7	100.0%	90.4%
Minesweeper	16	403ms	16	12	100.0%	84.3%
Blog	25	1243ms	25	20	100.0%	66.7%
PayrollSys	33	1854ms	33	41	96.4%	65.0%
myAlgsLib	42	1069ms	42	9	100.0%	86.2%

The performance of the tool is acceptable according to the running time in column (b). The time generally increases with the number of class and the complexity of relations among class. So you can notice that PayrollSys has a higher running time than myAlgsLib although the former has less class. The myAlgsLib is a set of common algorithms and the class in myAlgsLib have little relations with each other, so it takes less time to reverse extract myAlgsLib compared to PayrollSys which includes too many relations among it's class.

Considering the accuracy, we reviewed the code of five test case manually and get the knowledge about class and their relations (we called these knowledge as reference knowledge). Then compared the class diagram generated by the tool with reference knowledge, calculated the accuracy rate of class extracting(column (e)) and relations extracting(column (f)). If use N_{cc} stand for the total number of the class extracted correctly and use N_{ct} stand for the total number of class in the reference knowledge, the value of accuracy rate of class extracting is simply equal to N_{cc}/N_{ct}. The value of accuracy rate of relation extracting is calculated in the same way.

There is a higher accuracy in extracting class than extracting relations according the data in column (e) and column(f). Some redundancy and errors exist in relation extracting result, but it's totally acceptable. Moreover, J2UML tool detected successfully more accurate relationship discussed in section "Extract More Accurate Relationship".

Conclusions

A method framework for reverse extracting UML class diagram and sequence diagram was proposed in this paper. In general, the contributions of this paper include:

1) Defined a intermediate markup language J2X which make it easier to storage, exchange and extract semantic and structure information of programs;

2) Proposed a method to extract class diagrams based on J2X and discussed about the aspect to extract more accurate class relationship using the object flow analysis;

3) Proposed a method to extract sequence diagram combining object flow analysis and control flow analysis;

4) Implemented a reverse engineering tool J2UML based on the method proposed previous, and analyzed the performance and accuracy of tool using a set of test case.

Compared with other reverse engineering method, our method is based on J2X which make it easier to storage, exchange and extract semantic and structure information of programs. So it's possible to use same reverse method such as OFG to extract program in different program language based on J2X. The method has a good accuracy and performance in extracting UML model according to the experiment in a set of test cases. The future work about reverse extracting UML model may combine static and dynamic analyses to get more accurate result.

References

[1] Chikofsky, Elliot J., and J. H. Cross Ii. IEEE Software 7.1(1990):13-17.

[2] Tonella Paolo, Alessandra Potrich. *Reverse engineering of object oriented code.* Springer New York, 2005.

[3] Briand, L. C., Y. Labiche, and Y. Miao. *Towards the reverse engineering of UML sequence diagrams.* Reverse Engineering, 2003. Wcre 2003. Proceedings. Working Conference on IEEE Xplore, 2003:57-66.

[4] Rountev, Atanas, O. Volgin, and M. Reddoch. Acm Sigsoft Software Engineering Notes 31.1(2005):96-102.

[5] Labiche, Yvan, B. Kolbah, and H. Mehrfard. *Combining Static and Dynamic Analyses to Reverse-Engineer Scenario Diagrams.* IEEE International Conference on Software Maintenance IEEE Computer Society, 2013:130-139.

[6] Xin Wang, and Yuan X. *Towards an AST-Based Approach to Reverse Engineering.* Electrical and Computer Engineering 2006 Canadian Conference on (2006):422-425.

[7] Information on https://javacc.java.net/

[8] Briand Lionel C, Y.Labiche, and J.Leduc. Software Engineering IEEE Transactions on 32.9(2006):642-663.

[9] Ziadi, Tewfik, et al. *A Fully Dynamic Approach to the Reverse Engineering of UML Sequence Diagrams.* IEEE International Conference on Engineering of Complex Computer Systems IEEE, 2011:107-116.

[10] Roubtsov, Serguei, et al. Iet Software 7.3(2011):155-164.

[11] Guo, X., J. R. Cordy, and T. R. Dean. *Unique renaming of Java using source transformation.* IEEE International Workshop on Source Code Analysis and Manipulation, 2003. Proceedings IEEE, 2003:151-151.

[12] Badros, Greg J. Computer Networks the International Journal of Computer & Telecommunications Networking 33.1-6(2000):159-177.

[13] Kodaganallur, Viswanathan. IEEE Software 21.4(2004):70-77.

[14] Jarzabek, S., and I. Woon. ***Towards a precise description of reverse engineering methods and tools.*** Euromicro Working Conference on Software Maintenance and Reengineering IEEE Computer Society, 1997:3-9.

[15] Aman, H., and R. Ibrahim. ***Reverse Engineering: From Xml to Uml for generation of software requirement specification.*** International Conference on Information Technology in Asia IEEE, 2013:1-6.