Retrieving Formal Concepts from Class-Method Data Table

Hnin Pwint Phyu, Thi Thi Soe Nyunt
University of Computer Studies, Yangon
ahpwint@gmail.com,thithisn@gmail.com

Abstract

In maintenance of object-oriented software, one of the most important concepts is inheritance, which organizes classes into a hierarchy. The relationships among packages, classes, access modifiers, inherited classes and methods can affect on modification of the software. The proposed system uses Formal Concept Analysis (FCA) to discover groups of closely related classes. The groups of classes can assist programmers to know the high level structure of large software systems without prior knowledge, and programmers can learn the classes in the same group together. The class-method data table is constructed for retrieving formal concepts (group of classes). Then, the formal concepts are retrieved from the class-method data table by using the proposed Concept_Computing algorithm. After applying FCA and forming a lattice of given class specifications, one can easily analyze the structure, change effects of the methods and dependencies of the classes. Seven static java source projects are analyzed in this system.

Keywords-- ***Formal Concept Analysis, Software Maintenance, Concept Computing Algorithm***

1. Introduction

When an object- oriented software or model becomes bigger and bigger, duplicated elements start to appear, decreasing the readability and the maintainability of the software. In object-oriented software system, understanding the relation of the classes with each other and dependencies of the methods are also important. The presence of the inheritance increases the number of potential dependencies within a program.

A well designed class hierarchy makes the software easier to understand, maintain and reuse [6]. Formal Concept Analysis (FCA) is proposed to analyze software for ease of maintenance by developers. In software maintenance, it is difficult to make modifications without understanding all relevant codes in a system. FCA is a well-established technique for identifying groups of elements with common sets of properties. It can be successfully applied to complex software systems in order to automatically discover a variety of different kinds of implicit, recurring sets of dependencies amongst design artifacts. FCA provides a conceptual tool for the analysis of data and for the formal representation of knowledge. FCA is especially well suited when it is necessary to deal with a big number of entities (or objects) that can be described using a rich set of properties (or attributes).

The various applications of FCA to software maintenance vary on their inputs, the concept lattices they create, and the use to which they put the concept lattices. The proposed system focuses on the problem how to provide an understanding of the software in terms of packages, classes, methods and inheritance relationship. The proposed system intends to provide the dependencies of classes and methods in the java project. Multi level inheritance is also considered in this system.

The organization of the paper starts with introduction of the problem domain and some other related work with overview (Section 2). Next, the theory background of the proposed system is discussed (Section 3), followed by the proposed class-method data table construction (Section 4). Retrieving formal concepts from the class-method data table with the concept computing algorithm is presented (Section 5). Running example is explained in Section 6. The evaluation of the system is described in Section 7. Finally, the paper is concluded in Section 8.

2. Related Work

Formal Concept Analysis has been applied in various problem domains of software engineering. Among them, some applications of FCA in software engineering are described. Dekel [3] investigated the application of FCA to the tasks of reverse engineering and code inspection of individual JAVA classes. The technique partitions the methods of the class according to their use of fields, and then presents those in the form of a concept lattice (or Galois lattice) are also discussed. It is shown how this lattice can be used to perform version comparisons and to select an order in which methods can be read effectively. This work also investigates the possibility of applying formal concept analysis to the problem of understanding a single JAVA class. However, the solution is limited (at this stage) in that it ignores the interactions of the class under inspection with other classes. Tilley et al. [5] presented a broader overview

by describing and classifying academic papers that reports the application of FCA to software engineering. The papers are classified using a framework based on the activities defined in the ISO12207 Software Engineering standard. Two alternate classification schemes based on the programming language under analysis and target application size are also discussed.

Algorithmic complexity of the algorithms is studied both theoretically (in the worst case) and experimentally [4]. Conditions of preferable use of some algorithms are given in terms of density/sparseness of underlying formal contexts. Principles of comparing practical performance of algorithms are discussed. This attempted to compare, both theoretically and experimentally, some well-known algorithms for constructing concept lattices.

3. Overview of the Proposed System

In software maintenance, it is difficult to make modifications without understanding all relevant codes in a system. A change in one location of the source code may be simple, but different places might be difficult. The proposed system is intended to present a framework of the source code comprehension for software maintenance. The framework of the system consists of three phases as shown in Figure 1. Each phase contains two portions. Three main phases are Extraction phase, Identifying binary relation phase, and applying FCA phase with binary relation.

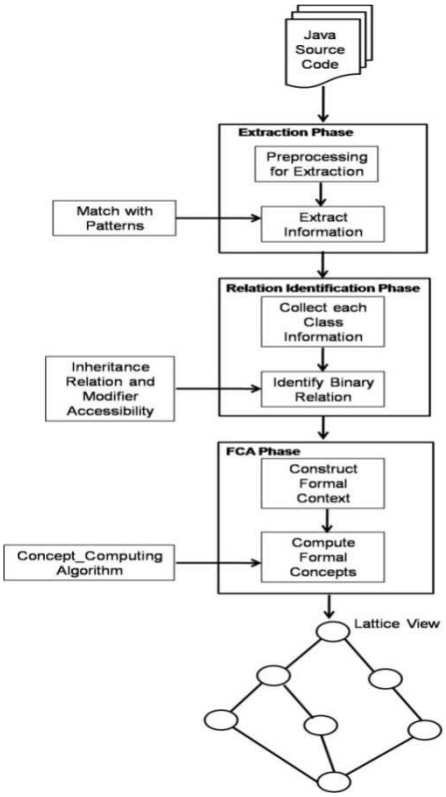


Figure 1. Framework of the Proposed System

Extraction phase consists of preprocessing step and extraction using pattern matching method. Identifying phase consists of collecting each class' information and identifying binary relation of classes and methods for constructing formal context. Specific formal context is needed in order to compute formal concepts and concept lattice. The last phase is applying FCA in the proposed system. FCA is composed of formal context, formal concept and concept lattice. In applying FCA, the first step is construction formal context according to the relation of the extraction phase. The next step is computing formal concepts from the formal context. The formal concepts are computed by the proposed Concept_Computing algorithm. Finally, the resulting output concepts are shown in concept lattice view.

The overview of Formal Concept Analysis and the theory background for constructing formal context are described in this section.

3.1. Formal Concept Analysis

Formal Concept Analysis (FCA) is a method of exploratory data analysis that aims at the extraction of natural clusters from object–attribute data tables. It forms the clusters of objects having common attributes. FCA can automatically classify and structure all the information around the "formal concepts" of the domain, which are natural pairs of objects and attributes sets.

FCA aims to formulate the philosophical understanding of a concept as a unit of two parts: its extension and its intension. The extension of a concept covers all the objects that belong to the concept, while the intension comprises all the attributes shared by all the objects under consideration. FCA is composed of formal context, formal concepts and concept lattice.

In order to apply FCA, the formal context or incidence table of objects and their respective attributes is necessary. The formal context consists of a set of objects G, a set of attributes M, and a binary relation $I \subseteq G \times M$ between objects and attributes, indicating which attributes are possessed by each object. Formally, it can be defined as K = (G, M, I).

From the formal context, FCA generates a set of concepts where every concept is a maximal collection of objects that possess common attributes. More formally, a concept is a pair of sets (A, B) such that:

$$A= \{g \in G \mid \forall g \in A: (g,m) \in I\} \quad (1)$$

$$B= \{m \in M \mid \forall m \in B: (g,m) \in I\} \quad (2)$$

where A is considered to be the extension of the concept and B is the intension of the concept. This set of concepts form a complete partial order where some concepts are superconcepts or subconcepts with respect to others.

A concept lattice can be represented graphically using line (or Hasse) diagrams. These structures are composed of nodes and links. Each node represents a concept with its associated intentional description. The links connecting nodes represent the subconcept-superconcept relation among them. The resulting lattices structure the knowledge hidden in the raw data, i.e. formal contexts, in a way which appeals to human experts, and allows them to navigate the data in a new way in order to understand relationships or create new hypotheses.

3.2. Access Control Modifiers and Inheritance Relation

Classes are created in hierarchies, and inheritance lets the structure and methods in one class pass down the hierarchy. That means less programming is required when adding functions to complex systems. Inheritance enables to add new features and functionality to an existing class without modifying the existing class.

Java provides a number of access modifiers to set access levels for classes, variables, methods and constructors. The following rules for inherited methods are enforced:

- Methods declared public in a superclass also must be public in all subclasses.
- Methods declared protected in a superclass must either be protected or public in subclasses; they cannot be private.
- Methods declared without access control (no modifier was used) can be declared more private in subclasses.
- Methods declared private are not inherited at all, so there is no rule for them.

4. Constructing Class-Method Data Table

In constructing class-method data table (formal context), inheritance relation, package level and access modifiers control are considered. A formal context is represented as a cross-table in which the rows are represented by the class names, the columns are represented by the method names. So, a cross ("×") in row ***g*** and column ***m*** means that the class g has the method m and a blank symbol ("0") indicates that the class does not have the method.

```
Begin
  for each class in project
    if (exist inheritance relation)
      if (check super class in project's classes )
        if (check super class in same package)
          if (check modifiers)
            process add methods except private modifier;
        else if (check modifiers)
          process add methods of public and protected modifiers;
      else show methods of each class;
    else  show methods of each class;
  end for
End
```

Figure 2. Binary Relation Identification Procedure

To form the class-method formal context, the intended information extraction from the source code is needed. The class names, super class names, method names, package names, and modifiers are extracted from the source code using pattern matching. From the extracted result, the relation of the formal context is identified by the binary relation procedure. The binary relation procedure is shown in Figure 2.

5. Retrieving Formal Concepts

The formal concepts are retrieved from the class-method data table using proposed concept computing algorithm in Figure 3.

Input: Formal context K= (G, M, I) where
G - set of objects
M- set of attributes
I – relation between G and M
Output: Concept {A, B} where
A- object set
B- attribute set

Step 1: for all A ∈ G do
compute INDEPENDENCE (I).
C ⟵ I {A, B};
return C ;
end.

Step 2: i ⟵ 1;
for all c_i ∈ C do
for all A, B ∈ c_i do
while ($c_i \neq x_i$ or $n(c_A) \neq 1$)
x_i ⟵ CLOSURE (A,B)$_i$;
C ⟵ x_i ;
return C ;
end while
i++;
end
end.

Figure 3. Comcept_Computing Algorithm

Notation: B- partial attribute set in attribute set
A- partial object set in object set
C- Concept set
c_i- each concept in initial Concept set
x_i- further growing concepts

The first step of the algorithm finds independence of object sets (I) as initial concepts. The second step of the algorithm computes iteratively the next concepts under each concept the previous step. The algorithm terminates each concept' object set is not further divided or the number of object set is equal to one. For every formal context, the algorithm finds independence of object sets (I) as initial decomposing concepts by using f($\mathbf{A_k} \subseteq \mathbf{A_j}$). Then, the algorithm computes iteratively the next closure concepts under each concept the previous step by using object set union and attribute set intersection function from the formal context.

6. Running Example

The proposed system implements FCA by using Galicia platform. In the running example, the net.sourceforge.javahmi.demo package is illustrated as example. The Demo package consists of 4 subpackages, 18 classes and 103 methods. From the class-method data table, the system computes the formal concepts by the proposed Concept_Computing algorithm (CCAlgo). First initial concepts from class-method table of the demo package are presented in Figure 4.

Independent Concepts:

{[CollectorRefTableModel] , [getColumnCount, getColumnName, getRowCount, getValueAt, updateData] }

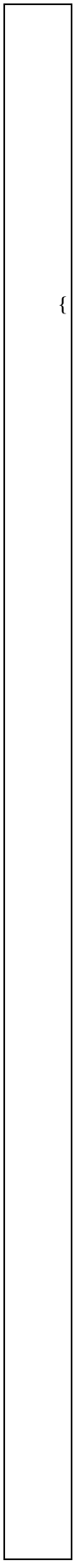
{[Demo1, Demo2, Demo3, NetworkingDemoController, NetworkingDemoView, SimpleMachineView] , [actionPerformed] }

{[Demo1, Demo2, Demo3, NetworkingDemoMain, SimpleMachineMain] , [main] }

{[Demo3, NetworkingDemoController, SimpleMachineController] , [changeObserved] }

{[DemoAnalogInput, DemoAnalogOutput, DemoDigitalInput, DemoDigitalOutput, DemoFlag, DemoRegister] , [read] }

{[NetworkingDemoController,



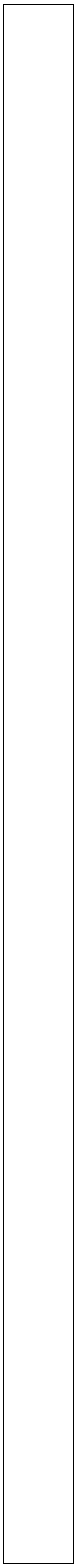


Figure 4. Initial Independent Concepts of the demo package

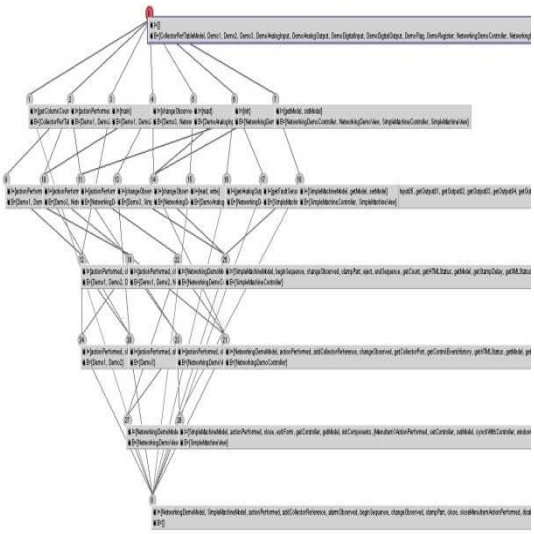


Figure 5. Concept Lattice of the demo package

Each initial concept is iteratively computed closure concepts. The resulted lattice view is presented in Figure 5. In Figures, '**I**' means intents (methods) and '**E**' means extents (classes). Demo 1, Demo 2 and so on is the classes included in **E**. The methods are changeObserved, read, init and so on is the methods included in **I**. By seeing the lattice view, the developer can analyze the dependencies of the classes and methods and estimate the change effects. In the resulted concept lattice, the top concept consists of all class names in a demo package and includes no methods (see Figure 6). Figure 7, Figure 8, Figure 9 and Figure 10 are the concepts in each layer of Figure 5.

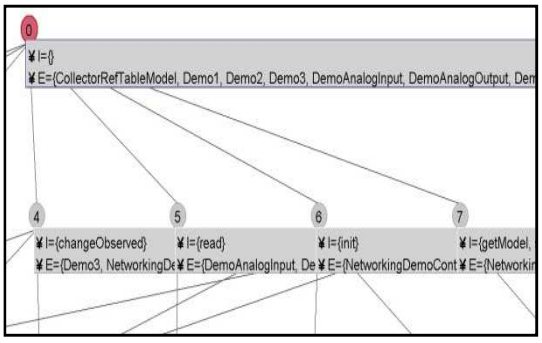


Figure 6. Top Concept

The layer concepts contain common methods in the classes. The uncommon methods of the classes decompose into other concepts. The first layer concepts are shown in Figure 7 and they are decomposed concepts of the top concept. The more goes down the concept lattice, the more specific concepts of common methods in the classes get.

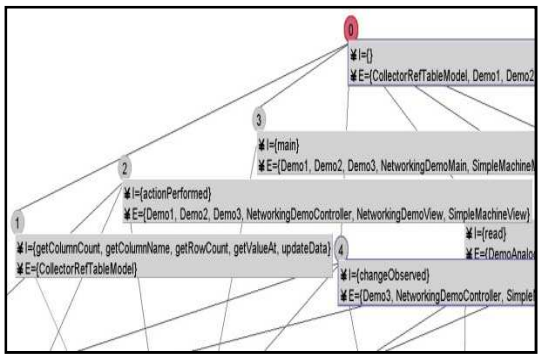


Figure 7. First Layer Concepts

The next layer concepts are decomposed by the more specific common methods in the classes. As the concept lattice goes down, the classes are fewer and the methods are more contain. The methods in the previous layer concepts also contain the methods of the next layer concepts.

The classes in the next layer concepts contain the classes in the previous layer concepts. The lower layer concepts contain more specific classes and methods. The more specific concepts are shown in Figure 8.

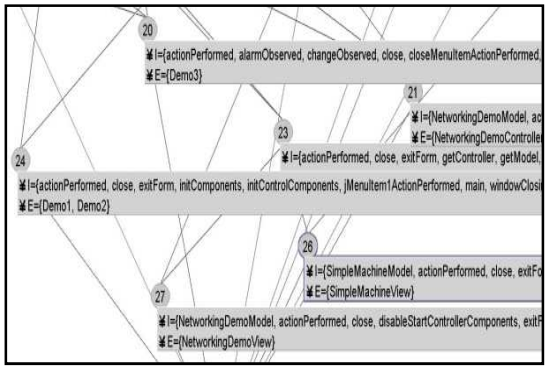


Figure 8. More Specific Concepts

Finally, the bottom concept is reached with all methods and no classes in a demo package (see Figure 9). By seeing the lattice, the developer can analyze which method changes affect which class, which methods common in which classes and can make modifications without affecting other classes.

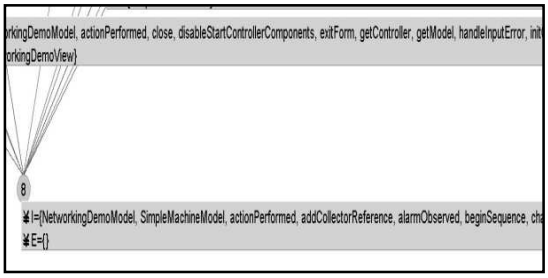


Figure 9. Bottom Concept

7. Analysis Results

The open source software systems utilized in the case study are micro simulator Antlr, JavaHMI, ArgoUML, JMeter, iBatis and Rhino. Micro simulator is the simulation system. The analysis results for these systems are presented. Antlr is a powerful parser generator for reading, processing, executing, or translating structured text or binary files. So, the parser is a complex software type. JavaHMI is a Java API for developing human machine interfaces. It is based on an observable/event driven control pattern. ArgoUML is an open source java implementation of a UML diagramming tool. JMeter is an open-source Java desktop application developed to allow users to load test functional behavior of web applications and other functions. iBatis is an object-relational mapping tool that facilitates the mapping of SQL databases to objects in a variety of programming languages. Rhino is an open-source software system that provides a Java implementation of JavaScript. The proposed system extracts the intended information from these seven projects. The precision and recall is calculated using the following equations.

$$Precision = \frac{|correct \cap retrieved|}{|retrieved|} \qquad Recall = \frac{|correct \cap retrieved|}{|correct|}$$

Table 1 shows precision and recall result of extraction for constructing class- method data table.

Table 1. Precision and Recall

Projects Names	Precision (Exactness)	Recall (Completeness)
Argo UML	0.95	0.96
ANTLR	0.90	1
Jmeter	0.96	0.96
JavaHMI	0.96	1
Micro Simulator	1	1
iBatis	0.96	0.94
Rhino-testsrc	0.94	0.95

The exactness and completeness of extraction over seven projects is shown in Figure 10.

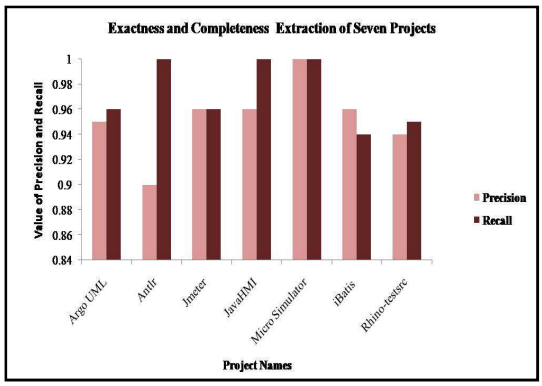


Figure 10. Exactness and completeness of Extraction

The proposed algorithm is compared with Bordat algorithm in Galicia platform. The Bordat algorithm is the best processing time in large context size and this algorithm contains drawing concept lattice. This algorithm is similar with the proposed algorithm except independent concept computing.

The Bordat algorithm finds cover (independent) concepts with the intersection of attribute set and uses attribute cache. The proposed CCAlgo finds independent concepts using subset function over object set and uses object list. The worst-case complexity of each algorithm is:

Bordat Algorithm- $O(|G||M|^2|L|)$

CCAlgo - $O(|G|^2|M||L|)$

In principal of concept computing, the complexity of each concept is |G| and |M|. G is the size of object set and M is the size of attribute set in a lattice L. The proposed algorithm is searched the independent concept based on object set. In the domain of software project, the size of object set is

generally less than the size of attribute set. In other words, the number of classes is smaller than the number of methods in most software projects. So, the worst-case complexity of the proposed CCAlgo is better than Bordat algorithm because independent concepts are computed over object set.

8. Conclusion

The main benefits of incorporating FCA to the design and construction of intelligent assistants as follows: FCA provides a formal, well founded and easy-to-use approach that can improve the interaction between user and system.

FCA is applied to discover groups of closely related classes. The groups of classes can assist programmers to know the high level structure of large software systems**.** The main purpose of this paper is to give concepts of common properties in classes and methods for dependencies of the source code. From the proposed comprehension framework, the change effects of the methods can be analyzed by the developers. By seeing the output concept lattice view, the developers can analyze the common methods and the decomposed methods within classes based on modifiers.

Besides, analyzing of modification for which class is closely related with other classes, which method changes can affect other parts, which methods should not be changed and so on can be made.

References

[1] K. Bertet, S. Guillas and J. –M. Ogier, “Extensions of Bordat's algorithm for attributes”, Proceedings of the Fifth International Conference on Concept Lattices and Their Applications, CLA 2007, Montpellier, France, October 24-26, 2007.

[2] B. Ganter, “Formal Concept Analysis: Theory and Applications”, Journal of Universal Computer Science (J.UCS), vol. 10, no. 8, pp. 926, 2004.

[3] U. Dekel, “Applications of Concept Lattices to Code Inspection and Review”, in Proceedings of the Insreli Workshop on Programming Languages and developmeny Environment, July 2002.

[4] S. O. Kuznetsov and S. A. Ob’’edkov, “Comparing performance of algorithms of generating concept lattices”, Journal of Experimental and Theorical Artificial Intelligence 14(2002) ,189–216.

[5] T. Tilley, R. Cole, P. Becker, P. Eklund, **“**A Survey of Formal Concept Analysis Support for Software Engineering Activities**”,** Proceedings of the First International Conference on Formal Concept Analysis - ICFCA’03 , 2003.

[6] G. Arevolo, S. Ducasse and O. Nierstrasz, “Lessons Learned in Applying Formal Concept Analysis to Reverse Engineering”, in Proceedings of the 3rd International Conference on Formal Concept Analysis (ICFCA'05), Lens, France, February 14-18, 2005, pp. 95-112.