# A Study on Online Aspect Mining

Yusuke Sakamoto[*], Haruhiko Sato[†], Masahito Kurihara[‡]

*Abstract*— **In aspect-oriented programming, we can encapsulate cross-cutting concerns scattered over many modules in a system. There are many studies for detecting candidates of cross-cutting concerns called aspect mining, and automation of it. However, existing all aspect mining tools are batch processing system aiming for large systems. Compared with existing batch-type tools, this paper propose an online-type aspect mining tool. The online-type aspect mining tool enables us to make cross-cutting concerns aspect modules during a coding stage. Therefore, it is possible to use it also for a small-scale project which has just started.**

***Keywords: cross-cutting concerns, aspect oriented programming, aspect mining, program analysis***

## 1 Introduction

Aspect mining is a technique derived from the software development technology called aspect oriented programming. Since aspect mining is closely related to aspect oriented programming in this chapter we briefly explain aspect oriented programming and aspect mining.

## 1.1 Aspect Oriented Programming

Aspect-oriented programming is a technology that strengthens the ability for modularization in object-oriented programming. Object-oriented programming has the ability to modularize concerns, and a minimum unit of the module is an object. However, it is known that there exists some concern which are distributed to two or more objects. Such distributed concerns are called cross-cutting concerns. Aspect-oriented programming is a technology to modularize these crosscutting concerns. Additionally, cross-cutting concerns might be simply called aspects. We show a case for logging as an example of cross-cutting concerns. Let us consider to log the method calls as follows:

```
class Dog {
        static void bark(){
                logger.log("start␣-␣Dog.bark");
                ......
                logger.log("end␣-␣Dog.bark");
        }
}
```

However, if we need to log the all method calls in every class, such logging codes are distributed all over the source codes. To avoid this problem, in aspect-oriented programming we can generate the logging codes by writing aspects as follows:

```
aspect Logging {
        around: execution(static void Dog.bark()) {
                logger.log("start␣-␣" + thisJoinPoint
                    .getSignature());
                proceed();
                logger.log("end␣-␣" + thisJoinPoint.
                    getSignature());
        }
}
```

## 1.2 Aspect Mining

Aspect mining is a technique to detect candidates of aspects from existing non-aspect oriented systems. The modularity of the codes can be improved by extracting some aspects obtained by aspect mining. There are various approaches for aspect mining and have been studied so far. Here, we introduce some techniques of typical aspect mining.

### 1.2.1 Clone Detection

Clone detection is a technique for making code clones an candidates of the aspect. Code clone is some parts in a source code which resembles or corresponds each other, and is often made by the processing such as "Copy and Paste". Because of the definition, there is a very strong possibility that code clone is the candidate of aspects.

### 1.2.2 Technique using Fan-in

Fan-in is numerical information that is the number of calls of a certain method in the source code. In this technique we calculate Fan-In of all the methods, and examine whether each method is a candidate of the aspect in descending order. It is one of the advantages of Fan-in that it can be calculated with a simple algorithm.

[*]Graduate School of Information Science and Technology, Hokkaido University. sayuu@complex.eng.hokudai.ac.jp
[†]haru@complex.eng.hokudai.ac.jp
[‡]kurihara@complex.eng.hokudai.ac.jp

#### 1.2.3 Pattern of Method Calls

This is a technique of regarding a pattern of similar method calls as a candidate of aspect. Even if other codes are included between the method calls, the pattern is regarded as the same if the order of method calls is the same. It is thought that we can extract more aspects using this technique compared to the case using code clone because the pattern of this technique is more flexible than that of the technique using the code clone.

#### 1.2.4 Technique using Execution Traces

This technique regards the main functions of a program as use-cases (for instance, if the program executes binary search, the use-cases might be "Insertion", "Search", etc.), and specifies all the methods used in some use-cases. If there exists some methods which are called from two or more use-cases, in this technique we adopt the set of methods as a candidate of an aspect.

#### 1.2.5 EA-Miner

EA-Miner is a technique for applying aspect mining to specifications, instead of source codes. It is useful for preventing big rework by rewriting the content of the specification that is appropriate for aspect-oriented programming before it is implemented as a source code. The specification is written by not programming language but natural language such as English or Japanese that we usually use. However, since this technique only supports English, it cannot be used for the specification written in other languages.

## 2 Online Aspect Mining Tool

In this section explains the online aspect mining tool proposed in this paper.

### 2.1 Making Tool Online

As an related study, there is a tool that supports online refactoring of the object-oriented programs, proposed by Hayashi et al. [2] They developed the tool in Eclipse IDE. A similar idea is applied to the tool of aspect mining in this study.

#### 2.1.1 Advantages of Online-type Tool

Existing tools are studied and developed under the assumption that some huge of source code group already exists. On the other hand, the tool we propose comparatively targets a small-scale source code under development. Compared with the batch-type tools, online-type tool enables us to find candidates of aspects in the early stage of development. Therefore, the online-type tool have an advantage of preventing the situation that causes big rework of the development beforehand because we can notice the place where the refactoring should be done ahead of time.

### 2.2 Design of Tool

We decided to adopt the technique using Fan-In as an aspect mining technique. There are two reasons to choose Fan-in. One reason is that Fan-in is treated in many papers which compares various methods of aspect mining techniques. The reason for another is that it is thought that implementation is easier than other techniques. However, because it is the technique designed for batch processing, it is not possible to use it for online aspect mining as proposed. Therefore, we need to slightly modify the Fan-in technique for online use.

#### 2.2.1 Fan-In for Online

In online processing, it is appropriate to apply the mining method only for the code written just now. Therefore, in online Fan-In technique, we only focus the method written just now In other words, online Fan-in determines the method last written, and calculates how many times the method was called. The Fan-In techniques only reports the number of times a method is called. This means that it is work for programmers to check adequacy of each methods whose Fan-In is high.

## 3 Simulation

In order to examine what effect is achieved when software was developed with a tool proposed in this paper, we performed experimentation explained in the following sections.

### 3.1 Description of experiments

The experimentation was executed as following steps:

1. First, we prepared a open source project written in Java, and we remove a file sample.java from the project.
2. Next, we re-construct sample.java from scratch, that is, we type the content original file by hand (not copy-and-paste).

### 3.2 Intention

The work explains by 3.1 is a simulation of the situation in which a new class is added to a certain project. It is thought that two or more classes are concurrently coded in practical software development. Therefore, this simulation that concentrates and develops only one class from first to last may not necessarily so realistic. However, we decided to simulate it in the way explained above since such simulation is reasonable and actual enough to obtain the useful experimental results. We adopted JHotDraw [15] that is an open-source project and is widely used in the research of aspect mining.

### 3.3 Results

The simulation explained by 3.1 was executed for class CommandMenu.java included in package *CH.ifa.draw.util* of JHotDraw5.3.

#### 3.3.1 Example 1

When you input *addMenuItem(command, new JMenuItem(command. name()));* to one line in the method definition *public synchronized void add(Command command)* The tool reacts to call of *name()* method. The identifier of the method (name, belonging package, and class) and the frequency where the method is called in other places are displayed in the view as follows.

```
[CH.ifa.draw.util.Command.name]:5
```

In addition, the identifiers of the method in which *name()* is called are displayed. Since it is called from the other 5 places, the 5 identifiers of the methods are displayed. When a method calls the target method *name()* two or more times such as *CH.ifa.draw.util.CommandButton.actionPerformed*, the caller method is displayed repeatedly.

```
CH.ifa.draw.util.UndoableCommand.name
CH.ifa.draw.util.CommandChoice.addItem
CH.ifa.draw.util.CommandButton.CommandButton
CH.ifa.draw.util.CommandButton.actionPerformed
CH.ifa.draw.util.CommandButton.actionPerformed
```

#### 3.3.2 Example 2

When you input *m.addActionListener(t)* to one line in the method definition *protected void addMenuItem(Command command, JMenuItem m)* The tool reacts to call of *addActionListener()* method. The display of the view was as follows.

```
[javax.swing.AbstractButton.addActionListener]:5
CH.ifa.draw.applet.DrawApplet.createButtons
CH.ifa.draw.applet.DrawApplet.createButtons
CH.ifa.draw.contrib.WindowMenu.buildChildMenus
CH.ifa.draw.samples.javadraw.JavaDrawApplet.
createButtons
CH.ifa.draw.util.CommandButton.CommandButton
```

For instance, you will be able here to determine that a candidate of the aspect might be included in these method definitions if you notice that a method is called in the methods of the same name (*createButtons*) belonging the different class.

### 3.4 Discussion

The frequency where the method was called ranged from 0 to 100 times. However, the methods called many times are often basic methods such as in java.lang package. From the results of experiments with our tool, we thought that it seems very possible that a method which satisfies the following conditions can be aspects:

- the method is self-made or occasional use
- the method is called with some frequency
- the method is called from some different places

## 4 Conclusion and Future Work

In this paper, we designed online-aspect mining tool based on the existing Fan-In technique, and implemented it as an Eclipse plug-in. The details are as follows.

### 4.1 Advantages of Online Aspect Mining

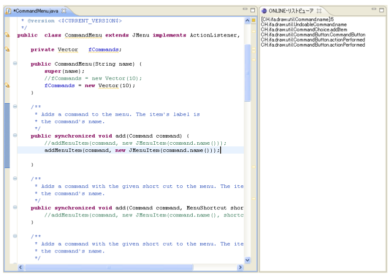
Online aspect mining is useful to discover aspects at the early stage of software development. In addition, there is an advantage that it becomes easy to verify the validity of the aspect, too.

### 4.2 Aspect Mining Technique

The technique of the tool made in this paper was made referring to Fan-in. We investigated existing techniques other than Fan-in to develop a new technique. However, because existing techniques were absolutely appropriate for the batch-type for a large-scale project, it turned out that it was unsuitable for the online-type that needs high-speed response. Therefore, it is necessary to develop the new technique that is different from existing tehniques, and appropriate for online aspect mining.

## References

[1] M. Marin, L. Moonen, and A. van Deursen. FINT: Tool Support for Aspect Mining. In Proceedings of the 13th Working Conference on Reverse Engineering (WCRE). IEEE, 2006.



```
 * @version <$CURRENT_VERSION$>
 */
public  class CommandMenu extends JMenu implements ActionListener,

    private Vector   fCommands;

    public CommandMenu(String name) {
        super(name);
        //fCommands = new Vector(10);
        fCommands = new Vector(10);
    }

    /**
     * Adds a command to the menu. The item's label is
     * the command's name.
     */
    public synchronized void add(Command command) {
        //addMenuItem(command, new JMenuItem(command.name()));
        addMenuItem(command, new JMenuItem(command.name()));

    }

    /**
     * Adds a command with the given short cut to the menu. The ite
     * the command's name.
     */
    public synchronized void add(Command command, MenuShortcut shor
        //addMenuItem(command, new JMenuItem(command.name(), shortc
    }

    /**
     * Adds a command with the given short cut to the menu. The ite
     * the command's name.
     */
```

```
[CH.ifa.draw.util.Command.name]:5
CH.ifa.draw.util.UndoableCommand.name
CH.ifa.draw.util.CommandChoice.addItem
CH.ifa.draw.util.CommandButton.CommandButton
CH.ifa.draw.util.CommandButton.actionPerformed
CH.ifa.draw.util.CommandButton.actionPerformed
```

Figure 1: A snapshot of the simulation 3.3.1 using tool made as Eclipse plug-in.

[2] S. Hayashi, M. Saeki, and M. Kurihara. Supporting Refactoring Activities Using Histories of Program Modification. IEICE Transactions on Information and Systems, vol.E89-D, no.4, pp.1403-1412. 2006.

[3] Thomas Kuhn, Olivier Thomann.
Abstract Syntax Tree. Eclipse Corner Articles, http://www.eclipse.org/articles/.

[4] org.eclipse.jdt.astview - AST View.
http://www.eclipse.org/jdt/ui/astview/index.php

[5] M. Marin, A. van Deursen, and L. Moonen. Identifying Aspects using Fan-In Analysis. In Proceedings of the 11th Working Conference on Reverse Engineering (WCRE2004), pp.132-141, IEEE Computer Society Press, 2004.

[6] M. Bruntink, A. van Deursen, R. van Engelen, and T. Tourwe. On the Use of Clone Detection for Identifying Crosscutting Concern Code. IEEE Transactions on Software Engineering, Vol.31, No.10, pp.804-818, 2005.

[7] T. Miyake, T. Ishio, K. Taniguchi, and K. Inoue. Detection of Crosscutting Concerns Using Method Call Patterns. Technical report of IEICE, SS, Vol.107, No.99, pp.1-6, 2007.

[8] Grigoreta Sofia Cojocar, Gabriela Serbian. On some criteria for comparing aspect mining techniques. In Proceedings of the 3rd workshop on Linking aspect technology and evolution (LATE), ACM, 2007.

[9] M. Ceccato, M. Marin, K. Mens, L. Moonen, P. Tonella, and T. Tourwé. A Qualitative Comparison of Three Aspect Mining Techniques. In IWPC '05: Proceedings of the 13th International Workshop on Program Comprehension, pp.13-22, IEEE Computer Society, 2005.

[10] M. Ceccato, M. Marin, K. Mens, L. Moonen, P. Tonella, and T. Tourwé. Applying and Combining Three Different Aspect Mining Techniques. Software Quality Control, 14(3):209-231, 2006.

[11] Dynamo - Dynamic Aspect Mining Tool.
http://star.itc.it/dynamo/.

[12] Wmatrix corpus analysis and comparison tool.
http://ucrel.lancs.ac.uk/wmatrix/.

[13] P. Tonella and M. Ceccato. Aspect Mining through the Formal Concept Analysis of Execution Traces. In Proceedings of the 11th Working Conference on Reverse Engineering (WCRE '04), pp.112-121, Washington, DC, USA, IEEE Computer Society, 2004.

[14] A. Sampaio, R. Chitchyan, A. Rashid, and P. Rayson. EA-Miner: a Tool for Automating Aspect-Oriented Requirements Identification. In ASE '05: Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering, pp.352-355, New York, NY, USA, ACM Press, 2005.

[15] JHotDraw. http://www.jhotdraw.org/.

[16] JavaTM Platform, Standard Edition 6 API Specifications.
http://java.sun.com/javase/ja/6/docs/ja/api/.