Aspect Mining for Large Systems
— Demo —

Silvia Breu
University of Cambridge
Computer Laboratory
Cambridge, UK
silvia@ieee.org

Thomas Zimmermann
Saarland University
Dept. of Computer Science
Saarbrücken, Germany
tz@acm.org

Christian Lindig
Saarland University
Dept. of Computer Science
Saarbrücken, Germany
lindig@cs.uni-sb.de

Abstract

The Eclipse plugin HAM identifies potential aspects in large programs. It analyzes the program's history and obtains sets of function calls that are likely to be cross-cutting. Later during programming, HAM informs the programmer when she is about to extend or change such a problematic concern.

Categories and Subject Descriptors D.2.7 [*Software Engineering*]: Distribution, Maintenance, and Enhancement—Restructuring, reverse engineering, and reengineering

General Terms Algorithms, Measurement, Documentation, Performance, Design, Experimentation,

Keywords Aspect Mining, Aspect-Oriented Programming, CVS, Eclipse, Formal Concept Analysis, Java, Mining Version Archives

1. Motivation

As a program evolves it is easy to overlook that certain functionality is not or no longer properly encapsulated but scattered over many methods. We are working on an Eclipse plug-in called HAM that will identify such cross-cutting concerns and will inform the programmer unobtrusively when she is about to add more such functionality. She might now go on as planned, or think about introducing an abstraction to encapsulate this functionality properly.

Our plug-in will work as follows: Imagine that you are working on some program code in Eclipse, inserting calls to a method `lock()` as well as to a method `unlock()`. Once you inserted both calls, they are marked by a light bulb as a cross-cutting concern. Why? A click on the bulb will reveal a view of all code locations that call both methods. This indicates that calls to `lock()` and `unlock()` are quite frequent. They might be better encapsulated by an object that acquires the lock and receives the action to be performed. These locations are obtained from the current version of the source code; however, the information that `lock()` and `unlock()` are cross-cutting is mined from historic data.

For its decisions, HAM preprocesses the program's CVS archive. Next, it applies formal concept analysis to additions of method calls [2]. As a result we obtain sets of methods that were frequently added together (like `lock()` and `unlock()`) and are spread throughout the source code. The presentation will demonstrate a prototype of HAM on large open-source projects and will include an overview of the techniques that make it scale so well [1].

Copyright is held by the author/owner(s).
OOPSLA'06 October 22–26, 2006, Portland, Oregon, USA.
ACM 1-59593-491-X/06/0010.

2. HAM Plugin Description

So far we have implemented a prototype of HAM that identifies cross-cutting concerns from CVS archives. The integration into the user interface of Eclipse will be our future work.

Figure 1 shows a screenshot when analysing ArgoUML for cross-cutting concerns. In the left pane **(A)**, the view *"Aspect Candidates"* lists all transactions of the CVS repository for which we found aspect candidates. For the transaction on April 4 2004 HAM finds five candidates: `illegalArgument(1)` (for which a call was inserted into 45 locations), `illegalArgument(2)` (101 locations), `illegalArgumentObject(1)` (75 locations), `illegalArgumentBoolean(1)` (27 locations), and `illegal-ArgumentCollection(1)` (72 locations).

Double clicking a transaction opens the corresponding lattice in view *"Concept Lattice"* **(B)** on the lower right hand side. This view allows to explore the relationship between candidates. The middle layer of five nodes represent the five aspect candidates that we found in this particular transaction. In this case, there is no path from the top to the bottom node that visits two aspect candidates. Thus the locations of the candidates are disjoint and unlikely to interfer.

Double clicking an aspect candidate (in any view) opens the *"Search"* view **(C)** of Eclipse in the lower left pane. This view lists all locations where a candidate was inserted. In our example, `illegalArgumentBoolean` was called in 27 location—among them `equalsPseudostateKind`. We can now inspect the code in the editor on the upper right hand side **(D)** and verify that the candidate is actually cross-cutting.

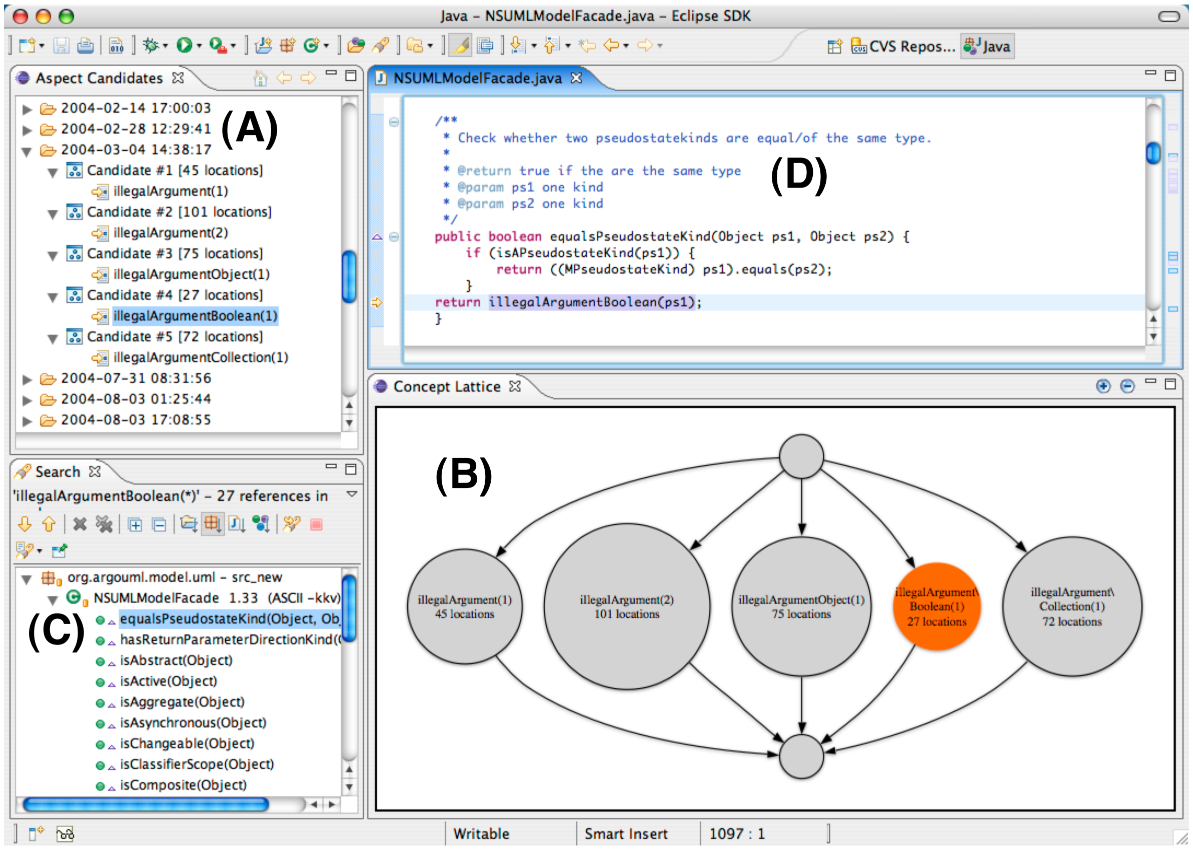
3. Contributions

We are the first to leverage version history to mine aspect candidates. The underlying hypothesis and motivation is that cross-cutting concerns may emerge over time. Our work shows that version archives are indeed useful for aspect mining.

HAM adds a new dimension to aspect mining. Previous approaches considered only a particular version of a program. Our approach uses project history as additional input. This enables a new view on the evolution of cross-cutting concerns.

HAM scales and is platform independent. HAM is the first aspect mining approach that scales to industrial-sized projects like Eclipse. Furthermore, it recognises cross-cutting concerns across code for different platforms.

HAM comes with high precision. We thoroughly evaluated 405 aspect candidates returned by HAM [1]. The precision increases with the project size and history, for Eclipse up to 90% for the top-10 candidates. For small projects, HAM suffers from the much fewer data available, resulting in lower precision (about 60%).


Java – NSUMLModelFacade.java – Eclipse SDK
CVS Repos...
Java
Aspect Candidates
(A)
(B)
(C)
(D)
Concept Lattice
Search
Writable
Smart Insert
1097 : 1
illegalArgument(1) 45 locations
illegalArgument(2) 101 locations
illegalArgumentObject(1) 75 locations
illegalArgument\ Boolean(1) 27 locations
illegalArgument\ Collection(1) 72 locations

```
▶ 2004-02-14 17:00:03
▶ 2004-02-28 12:29:41
▼ 2004-03-04 14:38:17
    ▼ Candidate #1 [45 locations]
        illegalArgument(1)
    ▼ Candidate #2 [101 locations]
        illegalArgument(2)
    ▼ Candidate #3 [75 locations]
        illegalArgumentObject(1)
    ▼ Candidate #4 [27 locations]
        illegalArgumentBoolean(1)
    ▼ Candidate #5 [72 locations]
        illegalArgumentCollection(1)
▶ 2004-07-31 08:31:56
▶ 2004-08-03 01:25:44
▶ 2004-08-03 17:08:55
```

```
'illegalArgumentBoolean(*)' - 27 references in
▼ org.argouml.model.uml - src_new
    ▼ NSUMLModelFacade 1.33 (ASCII -kkv)
        equalsPseudostateKind(Object, Ob
        hasReturnParameterDirectionKind(
        isAbstract(Object)
        isActive(Object)
        isAggregate(Object)
        isAsynchronous(Object)
        isChangeable(Object)
        isClassifierScope(Object)
        isComposite(Object)
```

```
/**
 * Check whether two pseudostatekinds are equal/of the same type.
 *
 * @return true if the are the same type
 * @param ps1 one kind
 * @param ps2 one kind
 */
public boolean equalsPseudostateKind(Object ps1, Object ps2) {
    if (isAPseudostateKind(ps1)) {
        return ((MPseudostateKind) ps1).equals(ps2);
    }
return illegalArgumentBoolean(ps1);
}
```

Figure 1. The HAM plugin. The *Aspect Candidates* view (A) lists cross-cutting concerns that can be investigated with other views (B,C,D).

4. Biographies

Silvia Breu graduated with a diploma degree in Computer Science, Math, and Applied Foreign Languages from the University of Passau in 2004, having done part of her studies at the University of Edinburgh. During the summer of 2004 she was a research intern at the NASA Ames Research Center within the Automated Software Engineering Group. With her work on dynamic aspect mining, she won the *Ernst Denert Software Engineering Award* in 2004, which is an award for the best MSc thesis in Germany. Before starting her PhD in 2006 at the University of Cambridge, supported by a scholarship from the Bill & Melinda Gates Foundation, Silvia worked as a researcher at the Software Engineering Chair at Saarland University. Her research interests include programming languages as well as program analysis such as slicing and aspect mining in order to support program understanding, validation and verification as well as debugging.

Tom Zimmermann received his diploma degree in Computer Science from the University of Passau in 2004. He is currently a PhD student at the Saarland University in Saarbrücken, supported by a scholarship from the DFG research training group on *Quality Guarantees for Computer Systems*. Tom was among the first to mine version archives: he built the eROSE tool, which helps developers navigating through source code. eROSE leverages the change history of projects and learns recommendations such as *Programmers who changed function f() also changed function g()*. Besides supporting developers, Tom's research interests are in software evolution, program analysis, and data mining.

Christian Lindig received a diploma degree in Computer Science from the Technical University of Braunschweig in 1993, followed by a PhD in Computer Science in 1999. After working at Harvard University he joined Saarland University in 2003 where he is currently a researcher at the Software Engineering Chair. His research interests cover the engineering of compilers and using compiler technology for bug localization. He also developed efficient algorithms and implementations for formal concept analysis and pioneered their application in software engineering. Christian is the co-author of the Quick C-- compiler for the portable assembly language C--, developed a Burg-style code generator for ML, and proposed a new architecture for implementing calling conventions in compilers. His automatic testing tool Quest routinely finds bugs in commercial C compilers. Recently he started to explore formal concept analysis as a tool to mine aspects and usage patterns from programs.

References

[1] S. Breu and T. Zimmermann. Mining Aspects from Version History. In *21st IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Tokyo, Japan, September 2006.

[2] S. Breu and T. Zimmermann and C. Lindig. Mining Eclipse for Cross-Cutting Concerns. In *Proc. Intl. Workshop on Mining Software Repositories (MSR)*, Shanghai, China, May 2006.