PRISM is Research In aSpect Mining

Charles Zhang and Hans-Arno Jacobsen
Department of Electrical and Computer Engineering &
Department of Computer Science
University of Toronto
10 King's College Circle
Toronto, Ontario, Canada
{czhang,jacobsen}@eecg.toronto.edu

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques—*Modules and interfaces*

General Terms

Design, Measurement

Keywords

Aspect Oriented Programming, Aspect mining, Concern Identification

1. INTRODUCTION

Aspect oriented programming (AOP) provides programmers with a new level of modularization capability. Although improving the aspect oriented languages seems to be the most eminent task, we recognize that it is equally important for (professional) software developers to properly leverage the power of AOP and to apply AOP at large scales. As many researchers have realized, one of the best ways to assess the practicality of AOP is to retrofit AOP onto legacy software systems [6, 7, 1]. One of the important activities in doing so is *aspect mining*. Aspect mining aims at finding non-localized programming concerns of very large software systems, which in turn provide opportunities for aspect oriented modularization and aspect oriented refactoring.

Existing work of aspect mining focuses on aspect visualization and aspect browsing [3, 2]. These tools often involve tedious human interactions for complex pattern definition over large code base. Inspired by AMT [3], AMTEX [5] is a step towards building an aspect mining environment which facilitates either autonomous or assisted aspect discovery. Prism is a major extension of AMTEX, which is designed for IDE integration, cross-language mining support, and multi-modal analysis. Prism is also an extensible mining platform, which is open for supporting new mining techniques. The current implementation of Prism is built as an Eclipse[1] plugin. It supports Java systems and partially supports C# programs. Through Prism, users can define complex lexical, semantic, and syntactical patterns. The Prism engine searches these patterns efficiently and provides matches that are mapped to their locations in the source code.

To date, Prism has been successfully used in several research projects that revolve around discovering crosscutting concerns in legacy code bases. The objective is to refactor these concerns with aspect oriented techniques to increase the softwares customizability, configurability, and adaptability [6, 7].

[1]Eclipse. `http://www.eclipse.org`

OOPSLA'04, Oct. 24-28, 2004, Vancouver, British Columbia, Canada.
ACM 1-58113-833-4/04/0010.

2. ASPECT MINING IN PRISM

The aspect mining activities in Prism are centered around three main concepts: Prism *fingerprints*, Prism *footprints*, and Prism *advisors*. A Prism fingerprint represents a characterization, i.e., a pattern-like description, of a certain property of a crosscutting concern in the code. Prism provides a large variety of ways for users to characterize an aspect through Prism fingerprint definitions. These fingerprints can also be thought of as queries over the code base in search for coding patterns and (crosscutting) concerns. The definition of Prism fingerprints can be assisted by Prism *advisors*. A Prism advisor is a tool which autonomously computes a certain crosscutting property of the mining target. Prism advisors expose significant insights of the structure of complex systems and, hence, are helpful in bootstrapping mining procedures and improving the accuracy of fingerprint definitions. The Prism footprints are concrete instances of the fingerprints sought in the actual code. They represent either a specific line or a region in the target's source code.

2.1 Fingerprints

In Prism, a fingerprint is a representation of a certain trait of an aspect or a particular coding concern. A basic fingerprint provides a direct description of the coding pattern. A *composite fingerprint* provides an abstract pattern definition which is a Boolean combination of any other fingerprints. Composite fingerprints express more complex traits through the reuse of already defined fingerprints. The current Prism implementation supports binary AND and OR expressions through operators `&&` and `||`. Full Boolean expressions will be supported in future releases.

Currently, Prism supports three different categories of coding patterns. The simplest patterns are *lexical patterns* in the program texts using regular expressions. Prism also supports lexical patterns on type names and method names as well as patterns of inheritance relationships. Moreover, Prism supports *AnyPhrase* which can be any valid Java code fragment for representing call flows of aspects.

Each Prism fingerprint is associated with two types of filters in making search results more specific. *Scope filters* use either namespace information, i.e., package names in Java systems, or regular expressions on type names to cover the entire code base or any of its subsets. *Lexical filters* can be used to specify the lexical patterns of the actual text of the code. Lexical filters are used in conjunction with fingerprints specified using type patterns so that patterns of both type names and their instance names can be captured.

Prism provides GUI based fingerprint builders and facilitates the lifecycle management of fingerprints.

2.2 Advisors

Prism advisors are tools, each of which autonomously computes an independent characteristic of the code base in order to assist precise definitions of fingerprints for aspects. While the most desired feature of an advisor is the automatic discovery of convoluted concerns [7], a powerful advisor can make good suggestions of possible convolutions and their possible locations in the code. Based on this information, a fingerprint can be defined to accurately capture the code level representation of these properties. Developing powerful advisors is one of our major research engagements. Currently, Prism provides a ranking advisor which reports most frequently-used types across methods. This advisor has proven effective in discovering new aspects in our previous research projects [6, 7].

2.3 Footprints

Footprints are matches of fingerprints in the code base. They are the results of the queries represented as Prism fingerprints. The current implementation of footprints is able to represent matches at the granularity of lines. Matches of lexical patterns and call patterns are individual lines in the source code. Matches of *AnyPhrase* patterns are typically expressed in ranges of lines, i.e., regions in the code base.

3. RELATED WORK

In recent years, there has been a proliferation of tools aiming at a better reflective understanding of the source code bases of complex systems. Feat[2] implements the concept of concern graphs [4], "which allows a developer to manipulate a concern representation extracted from a Java system and to analyze the relationships of that concern to the code base." Feat captures navigational patterns during the source code browsing process. Prism aims at capturing crosscutting properties in code through the definition of pattern specifications analogous database querrying. The concern or aspect locating approaches embodied in Prism and Feat are fundamentally different and complementary. The concern manipulation environment (CME)[3] provides query-language-style search facilities to help AOSD adopters manage concerns in legacy code bases. Prism aims at supporting crosscutting concern identification. CME and Prism share certain functionalities, but differ in overall objectives.

Aspect browser (AB) [2] constitutes an earlier effort in using visual maps to represent aspects. Lexical patterns are primarily used in locating concerns in code. AMT [3], leveraging the concept in AB, supports additional type based queries and proposes the multi-modal analysis using both lexical patterns and type specifications. We see both AB and AMT as predecessors of Prism. Visual representation is less effective in dealing with large code bases. The lexical and type based pattern specifications are greatly extended in Prism.

[2]FEAT Eclipse Plugin. URL: `http://www.cs.ubc.ca/labs/spl/projects/feat/`

[3]The CME Eclipse Plugin. URL: `http://www.eclipse.org/cme`

4. CONCLUSION

Prism is implemented as an Eclipse plugin. In the software demonstration, we first discuss aspect mining in general, then illustrate the functionality of Prism, including the definition and composition of Prism fingerprints and the type ranking advisor. A concrete usage scenario is presented to show Prism in action for analyzing complex software system to reveal possible candidate aspects. More information about Prism can be found at `http://www.msrg.utoronto.ca/prism`.

5. REFERENCES

[1] Adrian Colyer and Andrew Clement. Large-scale AOSD for middleware. In *3rd International Conference on Aspect-oriented Software Development (AOSD'04)*, pages 56 – 65, Lancaster, UK, 2004.

[2] William G. Griswold, Jimmy J. Yuan, and Yoshikiyo Kato. Exploiting the map metaphor in a tool for software evolution. In *Proceedings of the 23rd international conference on Software engineering*, pages 265–274. IEEE Computer Society, 2001.

[3] Jan Hannemann and Gregor Kiczales. Overcoming the Prevalent Decomposition of Legacy Code. In *Workshop on Advanced Separation of Concerns at the International Conference on Software Engineering (ICSE)*, Toronto, Ontario, Canada, 2001. URL: `http://www.cs.ubc.ca/~jan/amt/`.

[4] Martin P. Robillard and Gail C. Murphy. Concern graphs: Finding and describing concerns using structural program dependencies. In *International Conference on Software Engineering*, Orlando, Florida, USA, May 19-25, 2002 2002.

[5] Charles Zhang, Dapeng Gao, and Hans-Arno Jacobsen. Extended Aspect Mining Tool. URL:`http://www.eecg.utoronto.ca/~czhang/amtex`, October 2002.

[6] Charles Zhang and Hans-Arno Jacobsen. Refactoring Middleware with Aspects. *IEEE Transactions on Parallel and Distributed Systems*, 14(11):1058–1073, November 2003.

[7] Charles Zhang and Hans-Arno Jacobsen. Resovling Feature Convolution in Middleware Systems. In *Proceedings of the 19th ACM SIGPLAN conference on Object-oriented Programming, Systems, Languages, and Applications*, September 2004.

Acknowledgments

The following people have contributed to the development of AMTEX, the predecessor of Prism, and to Prism itself: Dapeng Gao, Crystal Wang, and Helen Shi. The Prism Eclipse plugin was developed under the Eclipse Innovations Grant 2003 from IBM.