International Journal of Computer Applications (0975 – 8887)
Volume 38– No.4, January 2012

Application of Program Slicing for Aspect Mining and Extraction – A Discussion

Amogh Katti
Asst. Prof., Dept of CSE
WIT, Solapur.
Maharashtra, India.

Vyankatesh Bingi
Asst. Prof., Dept of IT
WIT, Solapur.
Maharashtra, India.

Vishwanath Chavan
Asst. Prof., Dept of CSE
WIT, Solapur.
Maharashtra, India.

ABSTRACT

Aspect Orientation removed the code scattering and tangling drawback of Object Orientation by encapsulating the cross cutting concerns into their own modules called Aspects. It is gaining popularity these days as lot of languages, frameworks, programming and modeling tools already support aspects and developers have started to embrace these. But there exists lot of legacy object oriented code that needs to be moved to aspects as this makes cross cutting concerns easy to change (localized changes would be enough), test, extend, more comprehensible, etc. Converting it manually is tedious and there exist different techniques that semi automate the process making the maintenance engineer's job easier. Another approach to the automation process using program slicing is also possible. In this paper, we discuss aspect mining and extraction from program slicing point of view.

General Terms

Software Engineering, Software Evolution, Aspect Orientation, Program Analysis, Refactoring.

Keywords

Aspect Mining using Program Slicing, Refactoring to Aspects, Program Slicing for Refactoring, Untangling.

1. INTRODUCTION

Cross-cutting concerns are aspects of a program which affect (crosscut) the main concerns. These concerns often cannot be cleanly decomposed from the rest of the system in both the design and implementation and result in either scattering or tangling of the program or both. Here the code is scattered or tangled, making it harder to understand and maintain. It is scattered when one concern (like logging) is spread over a number of modules (e.g., classes and methods). That means to change logging can require modifying all affected modules. Modules end up tangled with multiple concerns (e.g., account processing, logging, and security). That means changing one module entails understanding all the tangled concerns. This increases the system complexity and makes evolution considerably more difficult.

Aspect Oriented Programming (AOP) [8] attempts to solve this problem by allowing the programmer to express cross-cutting concerns in stand-alone modules called aspects. Aspects can contain advice (code joined to specified points in the program) and inter-type declarations (structural members added to other classes). For example, a security module can include advice that performs a security check before accessing a bank account. The pointcut defines the times (join points) that a bank account can be accessed, and the code in the advice body defines how the security check is implemented. That way, both the check and the places can be maintained in one place. Further, a good pointcut can anticipate later program changes, so if another developer creates a new method to access the bank account, the advice will apply to the new method when it executes.

The industrial adoption of the aspect-oriented paradigm needs migration of legacy software systems (object-oriented) to aspect oriented ones and a subsequent boost in research on software refactoring [2]. The reasons to migrate a legacy system to an aspect-oriented solution are multiple. Using aspect-oriented technology, the cross-cutting concerns can be modularized using language features like pointcuts and advices inside an aspect. In the resulting system, the different concerns are cleanly separated making the system easier to maintain and extend.

A program slice is a subset of program statements concerning a subset of program variables. It separates a subset of program behavior. Weiser [18] says it as the mental abstraction people make when debugging a program. Program slicing is used in applications like debugging, parallelization, program differencing and integration, software maintenance, testing, reverse engineering, and compiler tuning [7].

Existing aspect mining techniques [1,5,9-14,16,17,20,23-27] are based on pattern matching, formal concept analysis, natural language processing on source code, software metrics and heuristics, clone detection and fan-in analysis. All the techniques follow specific assumptions for deciding a concern to be cross cutting (shown in the table below). We can say that the knowledge inference regarding when is a concern said to be cross-cutting is poorly developed. We think an intelligent analysis technique along with program slicing can accomplish this. There exists no mining technique that uses program slicing.

Table 1. Various approaches to aspect mining and their assumptions

Aspect Mining Approach	Assumption
Analyzing recurring patterns of execution traces	Cross cutting concerns follow certain execution patterns
Natural language processing on source code	cross-cutting concerns are often implemented by rigorously using naming and coding conventions

Detecting unique methods using heuristics	cross cutting concerns are implemented as "unique methods" - a method without a return value which implements a message implemented by no other method
Detecting clones as indicators of crosscutting concerns	because the cross-cutting concerns could not be cleanly modularized, certain parts of the implementation show high levels of duplicated code.
Fan-in analysis	many of the well-known cross-cutting concerns are implemented using a technique which exhibits a high fan-in. fan-in of a method 'm' is the number of distinct method bodies which can invoke 'm'.

The slicing process consists of identifying the statements that form the slice, slice identification, and isolating these statements into an independent program, slice extraction. Thus slicing seems very much similar to the Aspect Mining and Refactoring process. Our broad goal is to apply and test slicing for aspect mining and refactoring in the following directions:

1. A program slice isolates a subset of program statement concerning a subset of program variables and aspect extraction deals with isolating a tangled concern into its own abstraction. Both seem to have considerable similarity. We wish to analyze all the similarities between slice identification/extraction and aspect identification/extraction in detail. The similarities will decide if the slicing techniques be used as aspect extraction techniques and would minimize the work towards aspect extraction as slicing techniques can be customized to suit aspects.
2. There are different types of cross-cutting concerns like logging, exception handling, authentication and authorization, etc. Do the similarities remain the same across different types of cross-cutting concerns?
3. In the slice extraction literature there exist many different slicing techniques based on both static and dynamic analysis [3,4,6,15,18,19,21,22]. Can these techniques be used for aspect identification/extraction or new one's need to be designed?
4. Do Different types of cross-cutting concerns require different types of slice identification/extraction approaches?
5. Characterizing the slicing technique that suits aspect isolation/extraction.
6. Developing new slicing techniques for aspect mining/extraction, if required, considering different cross-cutting concerns or customizing the existing techniques.
7. Comparison of slicing based aspect mining/extraction technique with other approaches in detail.

In this paper we discuss the above questions and use the decisions we make at various stages to outline an algorithm for aspect mining and refactoring using slicing.

2. QUESTIONS AND DISCUSSION

Can slicing be used for aspect extraction?

Aspect mining involves identification of cross-cutting concerns. It can be split into two steps namely identifying different concerns and finding which of them are cross cutting, which are potential aspects. Slicing can be used to identify different concerns. Once different concerns are identified next step is to determine if the concerns are cross-cutting. We can use some program analysis techniques like Java Debugger Architecture, Aspects, Byte Code Instrumentation, Source Code Instrumentation, etc. [28] and the analysis results can be used to find the cross cutting concerns. And finally, aspect extraction is syntax and semantic preserving program transformation which is same as slice extraction.

In a programming language like java, which is purely object oriented (i.e. no code is present outside a class), aspect extraction involves extracting methods as advice and instance or static variables as type declarations. That means it is predominantly method extraction. Since Program Slicing has been used for method extraction [3,4,6,19,21,22] and we can say that it can be used for aspect extraction as well.

Do the similarities between program slicing and aspect mining/extraction remain the same across different cross cutting concerns (and application specific cross cutting concerns)? If there is any difference, how does it affect the slicing technique required?

Aspect Mining and Slice Identification seem very much similar but there exists some difference between Slice Extraction and Aspect Extraction. If we extract a method using slicing, the entire slice will become a new method whereas if we extract an aspect the slice will be distributed among multiple advices, which will be triggered by different joinpoints. For example, if we extract logging messages from a method, some message will become a before advice, something else will become an after advice, etc.

And during aspect extraction, the number of advices the slice gets distributed to is dependent on the type of cross cutting concern involved. For example, If the cross cutting concern involved is Logging the slice needs to be divided among a number of advices equivalent to the different levels of logging supported in the application. And for the Exceptions cross cutting concern, it is dependent on the different types of exceptions used in the application, application as well as general exceptions. So we can say that the cross cutting concern involved influences the slice extraction algorithm. And while designing a slice extraction algorithm we need to take care that it is generic enough to handle the variability of different cross cutting concerns and it can handle application specific cross cutting concerns as well.

What are the characteristics of the slicing technique required for aspect mining and refactoring?

Properties of Program Slicing are:

- There are two varieties of program slicing: Static and Dynamic. In static slicing, static analysis of the source code is carried out to compute the slice. Whereas in dynamic slicing, we execute the source code for a particular input and compute the slice that is valid only for this particular execution. Slicing criterion for dynamic slicing includes a specific input for which the slice is valid along with other components in static slicing criterion.
- Slice computation can be accomplished in two ways: using Backward Slicing or using Forward Slicing. Backward slicing is the one which was introduced originally by Weiser: a slice contains all statements and control predicates that may have influenced a given variable at a given point of interest. By contrast, forward slices contain all statements and control predicates that may be influenced by the variable. Nevertheless, backward and forward slices are computed in a similar way. The only difference is the way the flow is traversed.
- Slices can span a single procedure or multiple procedures and are called Intra-procedural and Inter-procedural slices respectively. Intra-procedural slicing computes slices within one procedure. On the other hand, inter-procedural slicing can compute slices which span procedures and even different classes and packages when slicing object-oriented programs.

Characteristics of a program slicing technique required for aspect mining and refactoring can be identified to be the following:

1. If we want to use a static slicing technique, we may need to represent the entire software system in the form of a graph. This would be tedious and very expensive in terms of memory and time. And it is very expensive to handle polymorphism using static slicing. We can use dynamic analysis as it overcomes the above mentioned problems.

   Static slicing if used involves analyzing the entire code whereas if dynamic slicing is used only the concern related code will be executed for analysis.

   If we want inter procedural, inter modular slices (which are very much necessary for concern identification) dynamic slicing is the best option as there is no need to analyze the entire system.
2. If we wish to use dynamic slicing, forward slicing is the option in contrast to backward slicing.
3. The slicing algorithm should be inter-procedural as it has to extract concerns spread across different modules in the application and not just procedures.
4. The slice obtained should fit into a number of appropriate advices. So a slice splitting technique is also required.

Can extant slicing techniques be used?

"Program Slicing for Refactoring: Static Slicer Using Dynamic Analyzer" [3] describes a slicing algorithm for method extraction using dynamic analysis. It does not use any alternative program representation, exploits testing performed during unit testing and hence suits refactoring. The unit testing performed for method extraction can be replaced by functionality testing so that we can test the working of a particular concern. This functionality testing would help in identifying concerns.

3. ALGORITHM

In this section we present a crude algorithm for aspect mining and extraction using program slicing. Tasks involved in aspect mining/extraction using slicing and ways to achieve them are enumerated below:

1. Identify all the concerns in the system using slicing. After this step, each concern is a slice.
2. We can use some program analysis techniques like Java Debugger Architecture, Aspects, Byte Code Instrumentation, Source Code Instrumentation, etc. The analysis tool's reflection mechanism enables access to the exact location where an interest is cross cutting with others. So, we can find out cross cutting concerns by some analysis (This analysis algorithm has to be designed).
3. If we want to use dynamic analysis for slicing, the analysis tools have to be strengthened to include powerful reflection capabilities and extensive pointcuts. For example, if we want to use AspectJ we need local variable get and set pointcuts, reflection to identify statements instead of line numbers, etc.
4. Design and execute functionality test cases for each particular concern so that data analysis required for slicing happens. Automate this step also using some test case generation/execution tools.
5. Detect the cross cutting concerns.

   Use data collected during testing and analyze this data to determine cross cutting concerns.
6. Extract the cross cutting concerns, slices, into their own aspects.

   > The slice obtained cannot be placed into an aspect directly. It has to be split into different advices. Distribute the slice identified, if it is cross cutting, into different advices and decide the joinpoints for the execution of these advices. Encapsulate the advice and pointcuts for the joinpoints into an aspect.
   >
   > The slice obtained should make sense syntactically and should fit into aspect correctly. Encapsulate all the concern related data into aspect in the form of inter type declarations, only if it is related to this concern alone.

4. CONCLUSION AND FUTURE WORK

We have analyzed the applicability of slicing to aspect mining and extraction. The similarities found show that the success possibility of applying slicing principles to aspect mining is huge. At the same time the dissimilarities demand customization of the slicing algorithms, especially slice extraction, for aspect extraction.

Our endeavor is still in its infancy and we are working on building a working aspect mining and extraction algorithm, according to the previously mentioned, in the algorithm section, steps/directions. And once the algorithm is ready we will implement it and test it on some cross cutting concern rich legacy applications to show its accuracy as an aspect mining technique.

5. REFERENCES

[1] A. Budanitski. Semantic distance in wordnet: an experimental, application-oriented evaluation of five measures., 2001.

[2] A. van Deursen, M. Marin, and L. Moonen. Aspect Mining and Refactoring. In Proceedings First International Workshop on REFactoring: Achievements, Challenges, Effects (REFACE). University of Waterloo, 2003.

[3] Amogh Katti and Sujatha Terdal. Article: Program Slicing for Refactoring: Static Slicer using Dynamic Analyzer. International Journal of Computer Applications 9(6):36–43, November 2010. Published By Foundation of Computer Science. BibTeX

[4] Arun Lakhotia, Jean-Christophe Deprez. Restructuring programs by tucking statements into functions. Information & Software Technology 40(11-12): 677-689 (1998).

[5] D. Shepherd, T. Tourw´e, and L. Pollock. Using language clues to discover crosscutting concerns. In Workshop on the Modeling and Analysis of Concerns, 2005.

[6] Filippo Lanubile, Giuseppe Visaggio. Extracting Reusable Functions by Flow Graph-Based Program Slicing. IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 23, NO. 4, APRIL 1997.

[7] Frank Tip. A survey of program slicing techniques. Journal of programming languages, 3:121-189, 1995.

[8] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J. Loingtier and J. Irwin. Aspect Oriented Programming. Proceedings of the 11th annual European Conference for Object-Oriented Programming, vol.1241 of LNCS, pp.220-242(1997).

[9] J. Krinke and S. Breu. Control-flow-graph-based aspect mining. In 1st Workshop on Aspect Reverse Engineering, 2004.

[10] J. Krinke. Identifying similar code with program dependence graphs. In 8th Working Conference on Reverse Engineering, pages 301–309. IEEE Computer Society Press, 2001.

[11] J. Morris and G. Hirst. Lexical cohesion computed by thesaural relations as an indicator of the structure of text. Comput. Linguist., 17(1):21–48, 1991.

[12] K. Gybels and A. Kellens. An experiment in using inductive logic programming to uncover pointcuts an experiment in using inductive logic programming to uncover pointcuts. In First European Interactive Workshop on Aspects in Software, 2004.

[13] K. Gybels and A. Kellens. Experiences with identifying aspects in smalltalk using 'unique methods'. In Workshop on Linking Aspect Technology and Evolution, 2005.

[14] K. Mens and T. Tourw´e. Delving source-code with formal concept analysis. Elsevier Journal on Computer Languages, Systems & Structures, 2005. To appear.

[15] Karl J. Ottenstein, Linda M. Ottenstein. The program dependence graph in a software development environment. Software Development Environments (SDE), pages 177-184, 1984.

[16] M. Marin, A. van Deursen, and L. Moonen. Identifying aspects using fan-in analysis. In Working Converence on Reverse Engineering (WCRE), 2004.

[17] M. P. Robillard and G. C. Murphy. Concern graphs: Finding and describing concerns. In Proc. Int. Conf. on Software Engineering (ICSE). IEEE, 2002.

[18] Mark Weiser. Program slicing. IEEE Transactions on Software Engineering, 10(4):352-357, 1984.

[19] Mathieu Verbaere. Program Slicing for Refactoring. Master's thesis, 2003.

[20] P. Tonella and M. Ceccato. Aspect mining through the formal concept analysis of execution traces. In 11th IEEE Working Conference on Reverse Engineering, 2004.

[21] Raghavan Komondoor, Susan Horwitz. Semantics-Preserving Procedure Extraction. In Proc. of 27th ACM Symp. on Principles of Programming Languages (POPL), (Boston, Massachusetts, January 2000).

[22] Ran Ettinger. Refactoring via Program Slicing and Sliding. Ph D thesis 2006.

[23] S. Breu and J. Krinke. Aspect mining using dynamic analysis. In Workshop on Software- Reengineering, Bad Honnef, 2003.

[24] S. Breu and J. Krinke. Aspect mining using event traces. In Conference on Automated Software Engineering (ASE), September 2004.

[25] S. Breu. Towards hybrid aspect mining: Static extensions to dynamic aspect mining. In 1st Workshop on Aspect Reverse Engineering, 2004.

[26] T. Tourw´e and K. Mens. Mining aspectual views using formal concept analysis. In Source Code Analysis and Manipulation Workshop (SCAM), 2004.

[27] Tonella, P., Ceccato, M., Migrating Interface Implementation to Aspects, ICSM'04, Chicago, USA, September 2004.

[28] A. Cain, J.-G. Schneider, D. Grant, and T. Y. Chen. Runtime data analysis for Java programs. In Proceedings of ECOOP 2003 Workshop on Advancing the State-of-the-Art in Runtime Inspection (ASARTI 2003), July 2003.