FINT: Tool Support for Aspect Mining

Marius Marin
Delft University of Technology
The Netherlands
A.M.Marin@tudelft.nl

Leon Moonen
Delft Univ. of Technology & CWI
The Netherlands
Leon.Moonen@computer.org

Arie van Deursen
Delft Univ. of Technology & CWI
The Netherlands
Arie.vanDeursen@tudelft.nl

Abstract

Aspect mining requires adequate tool support to locate source code elements implementing crosscutting concerns (aka seeds*), to explore and understand relations describing these elements, and to manage concerns and seeds during the project's life cycle.*

FINT *is a tool implemented as an Eclipse plug-in that presently supports a number of techniques for the automatic identification of crosscutting concern seeds in source code. Furthermore,* FINT *allows for combination of mining techniques (results), facilitates code navigation and comprehension to reason and decide about candidate-seeds, and supports seeds management and persistence.*

1. Problems addressed

Research in the area of *aspect mining* is aimed at providing (automatic) support for reverse-engineering crosscutting concerns in existing, non-aspect-oriented code. The results of the mining activities can further be used for program comprehension purposes or for improving the modularization of the code using aspect-oriented techniques.

The identification of a crosscutting concern typically involves the search or the existence of a so-called *seed*: a program element, such as a method or interface, part of the concern's implementation. Concern seeds provide first insights into a system and a starting point for concern exploration.

Few tools at the moment support the automatic identification of crosscutting concern seeds. FINT[1] is aimed at implementing analyses that capture symptoms specific to various crosscutting concerns described in literature. The analyses rely on the observation that crosscutting functionality is often implemented by following specific idioms [4]: for example, consistent logging or pre- and post-conditions checks [1] occur as scattered calls to specific functionality, while concerns like persistence or event and state listeners require classes to implement multiple roles.

FINT currently includes support for several techniques like:

- Fan-in analysis, which looks for candidate-seeds among methods invoked from a large number of scattered call-sites, and hence have a high fan-in value,
- Grouped calls analysis, which reports groups of methods that share a large number of their callers, and
- Redirections finder, which investigates classes whose methods consistently execute redirection of their callers to dedicated methods of another class.

FINT has been successfully applied to a number of case-studies, including PETSTORE, JHOTDRAW, and TOMCAT, for which we provided detailed results as part of an effort to establish a common benchmark for aspect mining [2, 3]. The systems analyzed are as large as JBOSS (around 350K non-comment LOC), and include the source code of FINT itself.

2. Tool description

FINT is a plug-in for Eclipse[2], an extensible, open-source platform that provides a reusable framework for source code analysis. To execute any of the analyses supported by FINT, the user selects a program element in the default *Package explorer* view of Eclipse; the view shows the Java element hierarchy of the Java projects in the active workbench. The default analysis of FINT, fan-in analysis, looks into the selected element, builds and stores the call graph for its methods and their callees, together with class hierarchy information.

The results of the analysis are displayed in the *Fan-in analysis view* as a tree structure of callee-callers elements, sorted by name or fan-in value. From this view, shown in Figure 1, the user can inspect the source code of each of the displayed elements and apply various filters to restrict the elements in the view to the most relevant candidates: (1) The *fan-in value* filter selects elements for which the number of callers is higher than a chosen threshold; (2) the *accessors* filter eliminates getter or setter methods by examining their signature and/or implementation; (3) the *utilities* filter allows the user to select sub-elements in the Java hierarchy of the analyzed element which are to be excluded from the set of results. The set of utilities consists of elements that could have a large number of callers but typically do not implement a crosscutting concern, like, for instance, collection classes.

The filters also allow to indicate sets of elements that should not contribute to the fan-in value of a candidate. For example, the user can indicate, similarly to selecting utilities, that calls from (JUnit) test packages have to be ignored when computing the fan-in metric.

To support the user in reasoning and deciding about a candidate, FINT includes *callers-analyzers* that check, for in-

[1] `http://swerl.tudelft.nl/view/AMR/FINT`

[2] `eclipse.org`

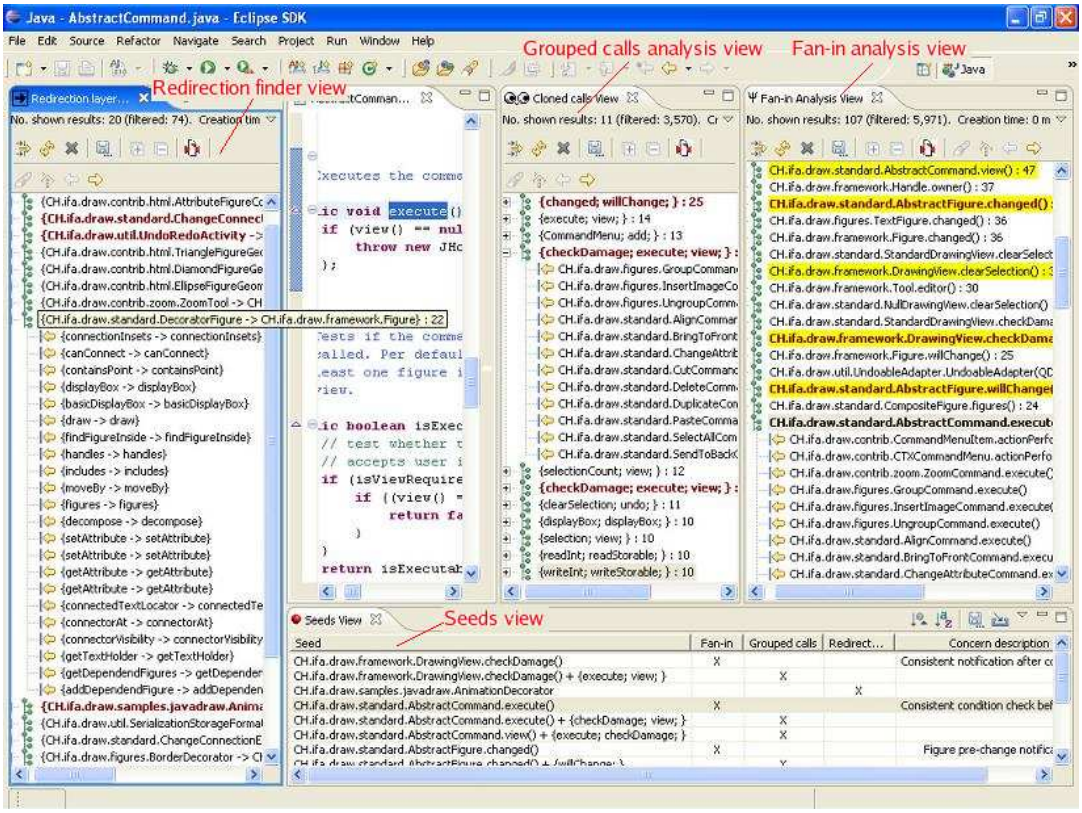

Figure 1. FINT **in action.**

stance, if the callers' declarations belong to the same interface, or if the calls to the investigated callee occur at the same position in the caller's body.

Similar filters are also applicable to the results of Grouped (/Cloned) calls analysis. The analysis groups methods by common callers by applying formal concept analysis to the call relations in the investigated element. The results, shown as callees-callers concepts in Figure 1, can also be filtered by the minimum number of callees in the concept.

Grouped calls analysis searches for concerns that are implemented by related method calls, like pre- and post-operation notifications, consistent initialization and clean-up of resources, and multi-step set-up operations. This analysis is more restrictive than fan-in analysis for a same value of the fan-in threshold, as it requires that the callers share more than one callee. However, most of the observations applicable to fan-in analysis to reason about a candidate apply to this technique as well.

Redirections finder is a technique that aims at identifying classes whose methods consistently redirect their callers to dedicated methods of a specific object. Implementations of the Decorator or Adapter pattern are typical examples of crosscutting concerns exhibiting the symptoms targeted by this technique. The technique checks for pair classes whose methods are in an exclusive one-to-one call relation, and reports these classes together with the methods executing and receiving the redirection, respectively. Filters for this analysis allow to set the minimum number and/or percentage of redirector-methods in a candidate.

Candidates from each analysis can be selected in the view and marked as seeds. The seeds are shown in Figure 1 as bold-colored elements. The seeds are also added to the *Seeds view*, where they can be assigned a description, sorted, and saved/loaded on/from file.

Two (compatible) techniques can be combined, for instance, by synchronizing their respective views: the intersection of the results is shown by the highlighted elements.

More details about the techniques supported by FINT are available in our work on an aspect mining framework [3].

References

[1] The AspectJ Team. *The AspectJ Programming Guide*. Palo Alto Research Center, 2003. Version 1.2.

[2] M. Marin, A. van Deursen, and L. Moonen. Identifying aspects using fan-in analysis. In *Proceedings of the 11th Working Conference on Reverse Engineering*, pages 132–141. IEEE, 2004.

[3] M. Marin, L.Moonen, and A. van Deursen. A common framework for aspect mining based on crosscutting concern sorts. In *Proceedings of the 13th Working Conference on Reverse Engineering*. IEEE, 2006.

[4] M. Marin, L. Moonen, and A. van Deursen. A classification of crosscutting concerns. In *Proceedings of the International Conference on Software Maintenance (ICSM'05)*, pages 673–676. IEEE, 2005.