Reasoning about assessing and improving the seed quality of a generative aspect mining technique

Marius Marin
Software Evolution Research Lab
Delft University of Technology
The Netherlands
A.M.Marin@ewi.tudelft.nl

Abstract

We propose a new measure for assessing generative *aspect mining techniques: the* (candidate-)seed quality. *We show the relevance of this measure and investigate how it can be improved for* fan-in analysis, *a generative mining technique that we have proposed in a previous work. The investigation results in a number of properties aimed at improving the quality of the (candidate-)seeds reported by fan-in analysis.*

1. Introduction

Aspect mining is a research area aimed at developing techniques for automatic identification of crosscutting concerns in existing code. A number of approaches to aspect mining provide support for identifying and investigating the code pertaining to a crosscutting concern starting from a *seed*: a program element (method, interface or group of statements) that is part of the crosscutting concern implementation. We describe these approaches as *explorative* or *query-based* approaches.

A second category of aspect mining approaches focus on identifying crosscutting concern seeds by looking in the source code for symptoms of crosscutting behavior, like tangling and scattering. We group this techniques in the *generative* category.

The mining techniques, and the generative ones in particular, face serious challenges in describing, comparing and combining their results due to (1) the lack of a clear definition of the crosscutting concerns, and (2) their focus on generic symptoms (tangling and scattering) that can occur in various concerns, of different level of granularity. This poses further questions about how the results of a combination of aspect mining techniques can be assessed, how to show improvement of results, and what measures are relevant for assessing these results and a mining technique in general.

To address these problems, we have proposed a classification system of the crosscutting concerns based on *sorts* (atomic, generic crosscutting concerns) and a preliminary set of sorts[4, 5]. The crosscutting concern sorts are described by their specific symptoms (i.e., implementation idiom in a non-aspect-oriented language), and a (desired) aspect mechanism to refactor a sort's concrete instances to an aspect-based solution. Sorts are relevant for aspect mining because they associate symptoms to generic concerns and are able to provide a focus for mining techniques and criteria for comparing their findings.

In this work we further consider how the (generative) aspect mining techniques can be assessed, by proposing and discussing a new measure: the *(candidate-)seed quality* is an indicator of the human effort involved for analyzing the results of an aspect mining technique. We look in more detail at this measure and how it can be improved for a specific mining technique, namely fan-in analysis. Our investigation results in a number of properties to be considered for improving the (candidate-)seed quality for this technique.

2. Assessing generative mining techniques

The results of a generative aspect mining technique are *candidate-seeds*: program elements that might pertain to a crosscutting concern implementation, and which can be marked as either seeds or *false positives* by an human analyzer examining the output of the technique. The decision of the human analyzer is typically supported by a number of elements provided by the mining technique for reasoning about its output: grouping of results by some criteria (e.g., naming), (structural) relationships between properties describing the results (e.g., related method calls), etc.

For simplification, we will refer to candidate-seeds in the following sections simply as *candidates*. We will also use *seed quality* if we discuss about candidates that have been marked as seeds.

A serious limitation of the (generative) aspect mining techniques consists of lacking a clear definition of the crosscutting concerns and thus of those concerns they aim to identify. As a result, most of the techniques focus on generic symptoms of crosscutting behavior, like tangling and scattering. This makes the comparison and description of their results difficult:

- How is the reported candidate (program element) related to a crosscutting concern, and what is that concern?

- How relevant is this relation for the mining technique: Is this an expected result or just an accidental match?
- Are the reported candidates relevant for the crosscutting nature of the associated concern, or just part of its broader, more complex implementation?
- Is the technique able to find similar concerns and is it able to easily distinguish between concerns that are not similar?

To answer these questions we believe that the focus of a mining technique should not be (only) on general symptoms but (also) on expected results (and specific symptoms): generic crosscutting concerns (sorts) that the technique expects to identify. This would allow for a clear assessment of the results (whether it is a relevant element of an expected sort instance) and the technique.

The next example illustrates this point for fan-in analysis, a generative aspect mining technique[3].

2.1. Sorts of crosscutting concerns

A *crosscutting concern sort* is a generic, atomic crosscutting concern described by a number of properties common to all its (concrete) instances like: (1) a general intent, (2) an implementation idiom in a non-aspect-oriented language (i.e., a specific symptom), (3) and a (desired) aspect mechanism to support the modularization of the sort's concrete instances.

Table1 shows several sorts from a longer list of proposed canonical sorts[4]. These are described by the three elements defining them as well as several examples of concrete instances.

Complex crosscutting concerns described in literature, like design patterns[2], are compositions of sort instances: an Observer pattern implementation, for example, involves instances of the *role superimposition* sort for the two defined roles (Subject and Observer), as well as instances of *consistent behavior* for the consistent actions of listeners notification and registration.

2.2. Defining "Target" crosscutting concern sorts for fan-in analysis

Fan-in analysis generates candidates based on the fan-in metric of a method: if a method is called from many, scattered places, the method is considered a potential seed. Hence fan-in is essentially a metric for the scattering symptom of the crosscutting concerns.

To focus our analysis, we define the targeted crosscutting concerns of fan-in analysis as instances of the consistent behavior sort. The sort is shown in Table 1; it is described by an action and a number of (method) elements that consistently execute the action as part of their complete functionality. These elements are part of a context that can be formalized by a pointcut definition.

Concrete instances of the sort comprise credentials checks as part of the authorization mechanism, or logging of exception throwing events in a system.

2.3. (Candidate-)seed quality

We propose the (candidate-)seed quality as a measure for assessing an aspect mining technique. The measure is defined as the percentage of relevant elements to make a decision about a candidate in the set of elements that the technique provides for reasoning about its output.

In order to measure the (candidate-)seed quality, a technique has to define its primary elements to support reasoning about a candidate (i.e., its output). Fan-in analysis defines these elements as the callers of the methods with a high fan-in value: A fan-in candidate for consistent behavior is labeled as seed if its callers are part of a relationship that can be formalized in a pointcut definition.

3. (Candidate-)seed quality for fan-in analysis

We discuss the quality measure for the fan-in technique and investigate what properties/attributes could be relevant for improving it. The discussion uses for exemplification a number of selected results from the JHOTDRAW case-study.

JHOTDRAW [1] is an editor for 2D graphics developed as an open-source project and a show-case for how to apply design patterns[1]. We used the application as a case-study for fan-in analysis in a previous work, in which we also describe the concerns associated with the results of the analysis[3].

3.1. Seed quality for a number of concerns in JHOTDRAW

The selected concerns in JHOTDRAW (typically) involve several crosscutting elements (i.e., sorts instances), like, for instance, roles superimposition and consistent behavior instances for the Observer pattern implementation. We look at how the elements pertaining to consistent behavior instances in these complex concerns are identifiable by fan-in analysis, and what is the quality of the seeds for these instances.

The selection of the concerns is aimed at showing various relations between the elements (callers) provided by the technique for reasoning about its results.

3.1.1. Undo

The *Undo* concern involves around 30 classes like *commands*, *tools*, and *(figure) handles* elements. The changes spawned by the execution of the activities associated with these elements can be undone by specialized, dedicated *UndoActivites*. The

[1] www.jhotdraw.org

Sort	Intent	Object-oriented Idiom	Aspect mechanism	Instances
Consistent Behavior	Implement consistent behavior as a controlled step in the execution of a number of methods that can be captured by a natural pointcut.	Method calls to the desired functionality	Pointcut and advice	Log exception throwing events in a system; Wrap/Translate business service exceptions [3]; Notify and register listeners; Authorization;
Contract enforcement	Comply to design by contract rules, e.g., pre- and post-conditions checking	Method calls to method implementing the condition checking	Pointcut and advice	Contract enforcements specific to design by contract
Redirection Layer	Define an interfacing layer to an object (add functionality or change the context) and forward the calls to the object	Declare a routing layer (decorator/adapter), and have methods in this layer to forward the calls	Pointcut and around advice	Decorator pattern, Adapter pattern [2]; Local calls redirection to remote instances (RMI) [6];
Role superimposition	Implement a specific secondary role or responsibility	Interface implementation, or direct implementation of methods that could be abstracted into an interface definition	Introduction mechanisms	Roles specific to design patterns: Observer, Command, Visitor, etc.; Persistence [3]

Table 1. Sorts of crosscuttingness.

Seed	Composite concern	Targeted sort instance	Fan-in value
UndoableAdapter.undo()	Undo	YES	24
UndoableAdapter.UndoableAdapter(DrawingView)	Undo	YES	25
Undoable.isRedoable()	Undo	YES	24
Figure.addFigureChangeListener(FigureChangeListener)	Figure Change Observer	YES	11
Figure.changed()	Figure Change Observer	YES	36
Figure.listener()	Figure Change Observer	NO	21
Figure.removeFigureChangeListener(FigureChangeListener)	Figure Change Observer	YES	10
Figure.willChange()	Figure Change Observer	YES	25

Table 2. Selected fan-in seeds from JHOTDRAW

UndoActivity classes are nested within their associated activities and implement the *Undoable* interface.

Fan-in analysis reports three seeds for the *Undo* concern, all of them part of instances of the targeted sort(s). The seeds are shown in Table 2. The first seed corresponds to a method implementing undo functionality and which is called by 24 methods. Most of the callers (22) are implementations of the *undo* method in the nested (*UndoActivity*) classes. The 22 callers follow the same idiom to invoke the reported seed: the invocation is the first call in the caller to check if the specific activity can be undone. The other two callers (*UndoCommand.execute* and *UndoRedoActivity.redo*) do not follow this idiom and are not part of a consistent behavior.

In order to decide about this candidate, we have to be able to observe specific symptoms of consistent behavior, like the structural relation between the first 22 callers to define a context, or the similar positions of the calls. The other two callers, on the other hand, make the analysis of the candidate harder, because they have to be investigated although they turn out not to be part of the consistent behavior concern. The relevant elements in the analysis of the callers are the first 22 callers and hence the quality is 22/24 (92%).

Similar, the quality for the other two candidates is 22/25(88%) and 20/24(83%), respectively.

3.1.2. Figure Change Observer

The *Figure* elements in JHOTDRAW participate in an implementation of the Observer pattern (Figure Change) by implementing the Subject role. A number of elements listen for changes in *Figure*s by implementing the *FigureChangeListener* interface. The concrete figures implement or inherit the methods specific to the Subject role for allowing (de-)registration of listeners, and notification of changes in their state.

The (de-)registration action is a consistent behavior for the listeners of figure changes. This is implemented as a call to the method that adds/removes listener objects for a *Figure*: *Figure.add-/remove-FigureChangeListener*.

The notification of changes occurred in the *Figure*s' state is also a consistent behavior applying to elements changing this state. The concern is implemented as a call to the notification method: *Figure.willChange*, before making the changes, and *Figure.changed*, after the changes were made.

The consistent behavior for these instances of the sort is due to the relation between the callers and the callee in the context of the pattern: the listeners interested in changes have to register in order to be notified, and the actions changing the state of the Subject object have to notify about this change. Hence all the callers of the reported seeds are relevant because the calls occur due to the participation into the pattern implementation. This participation is a secondary concern for the callers. Moreover, the concern implemented by the callees (the reported high fan-in methods) implies calls only from participants in the pattern, in the context of the pattern. The quality of these seeds is thus 100%.

Fan-in analysis reports several implementations of the described candidates. Table 2 shows the results omitting the

Seed	Orig. Quality	Same hierarchy(same method)	Role relation	Same Call Position
UndoableAdapter.undo()	22/24 = 92%	22/23(22/22) = 96% (100%)	22/23 = 96%	22/22 = 100%
UndoableAdapter.UndoableAdapter(DrawingView)	22/25 = 88%	22/22(22/22) = 100%(100%)	22/22 = 100%	22/22 = 100%
Undoable.isRedoable()	20/24 = 83%	19/21(18/19) = 90%(95%)	19/21 = 90%	18/21 = 86%
Figure.addFigureChangeListener(FigureChangeListener)	11/11 = 100%	Not relevant	10/10 = 100%	Not relevant
Figure.changed()	36/36 = 100%	Not relevant	Not relevant	33/33 = 100%
Figure.listener()	-	-	-	-
Figure.removeFigureChangeListener(FigureChangeListener)	10/10 = 100%	Not relevant	9/9 = 100%	Not relevant
Figure.willChange()	25/25 = 100%	Not relevant	Not relevant	20/20 = 100%

Table 3. (Candidate-)seeds analysis for selected concerns

multiple implementations that occur due to polymorphism.

The last reported method, *Figure.listener*, provides access to the listener reference in the class implementing the Subject role. The method is part of this role, however, it does not belong to a consistent behavior instance in the analyzed implementation of the Observer pattern.

3.2. Properties/attributes to improve the quality of the fan-in candidates

To improve the quality of the fan-in candidates, we propose a number of properties to be considered for analyzing the callers of a candidate. These properties show possible relations between the callers of a method with a high fan-in, and are aimed at reducing the percentage of irrelevant elements for reasoning.

The list of proposed properties comprises:

- structural relations between the callers. These relations include:
  - same hierarchy: The methods (callers) are defined by the same interface (/super class). As a particular case, the callers could be implementations of the same method. The (callers of the) seed for the *Undo* concern previously discussed falls into this category.
  - common roles: A method is associated with all the roles implemented by its class, so that the methods can share common roles. A role is typically defined by an interface. Methods that belong to the same hierarchy will also share the role that defines the hierarchy: the callers of the discussed *Undo* seed are declared by the *Undoable* interface, which defines the main role of the classes implementing the concrete callers of the seed. Similarly, the callers of the registration method in the previous Observer example are associated with the *FigureChangeListener* role. The main role for these callers is, however, defined by other interfaces, like *Figure*, which is common to most of them.
  - same class: The callers belong to the same class, as for the case of a class level contract.
  - etc.

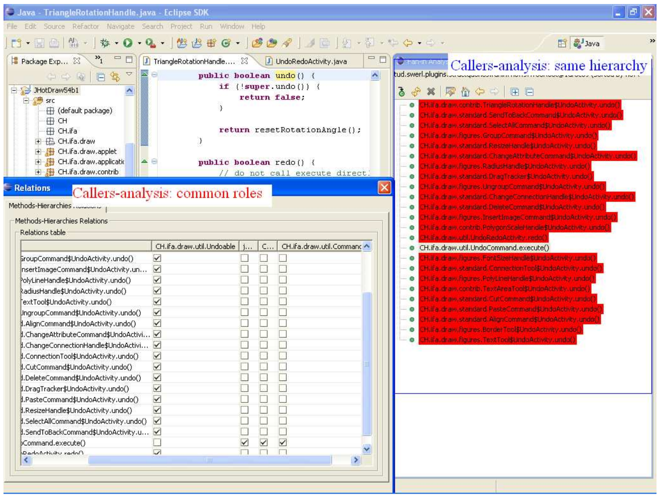

Figure 1. FINT support for callers-analysis

- consistent call position: The position of the call, relative to the caller's body, is consistent for the callers of the reported method with a high fan-in value. Such a case has been shown for the seeds discussed in Section3.1.
- naming-based relations: The callers have similar names. The naming-based and the structural relations can be expressed by an AspectJ-like pointcut definition, while the call position could be an indication of the advice type (before/after).
- relations based on the structure of the call: cloned/similar call sites. An example is the exception wrapping concern that consists of catching a specific type of exception and re-throwing an exception of a different type[3].
- "intentional" relations between callers, such as modifiers of Subject objects in the context of the Observer pattern. The relations between the callers are due to their participation in the pattern implementation.

3.3. Results after extending fan-in with a callers analysis

Table 3 shows the quality of the fan-in seeds after analyzing the callers based on some of the proposed properties.

The mined instances that are part of the Undo concern are representative for many (/most) of the concerns identified using fan-in analysis. In most cases, the quality of the seeds is improved by analyzing the proposed properties.

Although the original quality of the seeds for the Observer implementation is 100%, the analysis of the callers based on the proposed properties is meaningful due to the complex relation between the callers that has to be observed. Because the relation is intentional, it is expected that the investigated, mainly structural, properties to be irrelevant, offering little or no insight into the intentional relation. However, the structural properties can provide insight into the specific consistent behavior instances: Most of the callers (10 out of 11) of the registration method belong to classes that implement the listener role and that register themselves as listeners of a *Figure* object. The *common role* property groups and relates the 10 methods by this common, relevant concern.

A similar grouping and insight into the callers' relation can be obtained by considering the *call position* property for the callers of the notification method.

Part of the proposed callers-analyzes are implemented by FINT[2], the tool support for fan-in analysis. Figure 1 shows the user interfaces that allow for hierarchy and role relations inspection: The callers of a method-candidate are displayed in the same color if they are declared by the same interface. The relations between methods and roles are displayed in a relational matrix.

4. Discussion and conclusions

The proposed quality measure is useful to assess the relevance of the results of an aspect mining technique and the effort required to analyze these results. The proposed analysis of the callers to reason about a fan-in candidate also shows the potential relation between the elements describing a candidate. This relation is important for defining the context of the concern in a pointcut construct.

The measure asks for a clearly defined target as a sort of crosscutting concerns. We believe that sorts provide consistent criteria for defining the focus, assessing and improving the performance of a mining technique. They allow for consistency in reporting the results of aspect mining, combining (and comparing) mining techniques, and evaluating the manual effort to analyze the output of a technique. Sorts can also be used to define other, known measures that apply to aspect mining, like precision and recall.

Our investigation has focused on instances of the consistent behavior sort. However, the discussion and the results also apply to contract enforcement: The two sorts differ only in intent, sharing the idiom (specific symptom) and the aspect mechanism to modularize their instances. The distinction between the intent of the identified instances is based on the relation between the instance and design by contract contexts.

The situations described for the *Undo* concern where the callers are related by structural relationships are common to many of the concerns discovered by fan-in analysis. These include consistent behavior and contract enforcement instances in JHOTDRAW, for *Command* and *Tool* elements, for instance, as well as concerns in other analyzed case-studies, like PETSTORE and TOMCAT [3]. The intentional relationship, on the other hand, are typically harder to detect. The intention of a code developer could be captured by annotations in the code, which to be analyzed by aspect mining techniques.

Acknowledgments The author would like to thank Arie van Deursen (TU Delft) for his reviews and feedback.

References

[1] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, 1994.

[2] J. Hannemann and G. Kiczales. Design Pattern Implementation in Java and AspectJ. In *Proceedings of the 17th Annual ACM conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, pages 161–173, Boston, MA, 2002. ACM Press.

[3] 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).*, pages 132–141, Los Alamitos, CA, 2004. IEEE Computer Society Press.

[4] M. Marin, L.Moonen, and A. van Deursen. A classification of crosscutting concerns. In *Proceedings International Conference on Software Maintenance (ICSM 2005)*. IEEE Computer Society, 2005.

[5] M. Marin, L.Moonen, and A. van Deursen. An approach to aspect refactoring based on crosscutting concern types. In *Int. Workshop on the Modeling and Analysis of Concerns in Software, ICSE*. Software Engineering Notes (volume 30, issue 4), 2005.

[6] S. Soares, E. Laureano, and P. Borba. Implementing Distribution and Persistence Aspects with AspectJ. In *Proc. 17th Conf. on Object-oriented programming, systems, languages, and applications*. ACM Press, 2002.

[2] http://swerl.tudelft.nl/bin/view/AMR/FINT