Eighth IEEE International Working Conference on Source Code Analysis and Manipulation

Rejuvenate Pointcut: A Tool for Pointcut Expression Recovery in Evolving Aspect-Oriented Software*

Raffi T. Khatchadourian†
Ohio State University
khatchad@cse.ohio-state.edu

Awais Rashid
Lancaster University
awais@comp.lancs.ac.uk

Abstract

Aspect-Oriented Programming (AOP) strives to localize the scattered and tangled implementations of crosscutting concerns (CCCs) by allowing developers to declare that certain actions (advice) should be taken at specific points (join points) during the execution of software where a CCC (an aspect) is applicable. However, it is non-trivial to construct optimal pointcut expressions (a collection of join points) that capture the true intentions of the programmer and, upon evolution, maintain these intentions. We demonstrate an AspectJ source-level inferencing tool called REJUVENATE POINTCUT *which helps developers maintain pointcut expressions over the lifetime of a software product. A key insight into the tool's construction is that the problem of maintaining pointcut expressions bears strong similarity to the requirements traceability problem in software engineering; hence, the underlying algorithm was devised by adapting existing approaches for requirements traceability to pointcut maintenance. The Eclipse IDE-based tool identifies intention graph patterns pertaining to a pointcut and, based on these patterns, uncovers other potential join points that may fall within the scope of the pointcut with a given confidence. This work represents a significant step towards providing tool-supported maintainability for evolving aspect-oriented software.*

1 Introduction

Aspect-Oriented Programming (AOP) has emerged to localize the scattered and tangled implementations of crosscutting concerns (CCCs) in source code, allowing developers to characterize that certain actions (advice) should be taken at specific points (join points) during the execution of software. The applicability of advice is denoted via the use of a pointcut expression (PCE) which specifies where a CCC (an aspect) is applicable. A PCE does so by logically connecting various predicate expressions over static and dynamic program characteristics like method and field naming conventions, lexical scope, and execution control-flow.

The true power of AOP lies not only in the ability to express the applicability of existing join points captured by a PCE but also in the applicability of join points that will be added in the *future*. An effective PCE is one that truly conveys the *essence* of where a CCC applies in the underlying system (base-code) so that not only are the current join points that correspond to where a CCC applies are selected but future ones as well. Thus, creating effective PCEs requires developers to anticipate structural (static) and behavioral (dynamic) properties that the base-code will exhibit as the software evolves. Constructing such optimal PCEs, however, can be a daunting task, especially considering that these expressions should remain valid as the software evolves. The problem is that as new requirements surface and corresponding program elements emerge to realize a particular requirement, the PCEs which were once intended to capture *core* (base) concerns in which a certain CCC was deemed applicable may no longer be valid.

It has been shown that the cause of this problem is rooted at the inherent *fragility* of the typical PCE language available in popular AO languages such as AspectJ Fortunately, maintaining PCEs as software evolves seems to share some resemblances to the requirements traceability problem. Software is said to be *requirement traceable* if a developer has the ability to follow the life of the realization of a particular requirement in both a forwards and backwards direction. Unfortunately, the luxury of possessing such an ability can necessitate a significant amount of costly developer time; therefore, it is essential to cache such information for later consultation. Nevertheless, even seemingly innocuous changes to any traced element are likely to invalidate the corresponding trace information. Consequently, effort must be made to maintain the trace information with each artifact alteration. This situation is similar to one that occurs in AOP, specifically, that *PCEs* should also be maintained in a analogous fashion as the base-code evolves.

In an effort to combat this problem, we demonstrate an

*This material is based upon work supported by the European Commission grant IST-33710 (AMPLE) and grant IST-2-004349 (AOSD-Europe).

†This work was administered during this author's visit to the Computing Department, Lancaster University, United Kingdom.

978-0-7695-3353-7/08 $25.00 © 2008 IEEE
DOI 10.1109/SCAM.2008.32

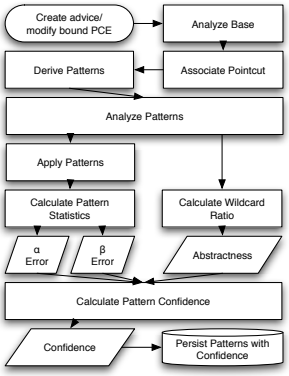

Figure 1. Phase I: Pointcut analysis.

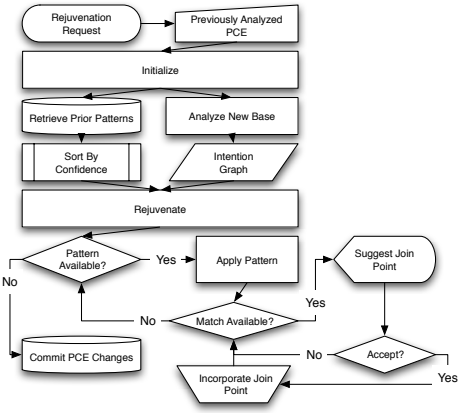

Figure 2. Phase II: Pointcut rejuvenation.

AspectJ source-level, inferencing tool named REJUVENATE POINTCUT. The Eclipse IDE plug-in alleviates the burden associated with maintaining PCEs over the lifetime of software by enabling tool-assisted augmentation of a PCE upon changes to the base-code. In the following sections, we provide a brief overview of our tool, including a high-level architecture, and summarize its innovative features.

2 Rejuvenation Approach

To assist developers maintain PCEs over the lifetime of the base-code, our tool is composed of two phases: *analysis* and *rejuvenation*.

Phase I: Analysis. The analysis phase, whose flow diagram is depicted in Figure 1, is triggered upon modifications to advice-bound pointcuts. The phase consists of analyzing underlying patterns in the base-code prior to its modification and then associating the pointcut with these patterns. At the core of Phase I is an interprocedural inferencing algorithm that uses an *intention graph* to capture the developer's intentions in creating certain program elements. The graph is constructed with the aid of the JayFX fact extractor which generates facts using class hierarchal analysis (CHA) pertaining to structural properties and relationships, e.g., field accesses, method calls, residing in the base-code. The phase then proceeds to associate the PCE with both vertices and labeled arcs by leveraging the AspectJ Development Tools (AJDT). These associated graph elements are referred to as *enabled* with respect to a PCE. Next, *intention patterns* expressing general *shapes* of acyclic paths that include enabled elements are derived. The patterns are then themselves analyzed in order to produce and associate a *confidence*. This metric symbolizes the degree of credence in using the pattern to identify join points that should be augmented to a given PCE upon base-code evolution. Consequently, results produced by the pattern are correlated with and ranked by this confidence when presented to the developer. Finally, the patterns are associated with the PCE and persisted for later use in Phase II.

Phase II: Rejuvenation. We envision our tool to be most helpful in scenarios where a developer changes the base-code and then proceeds to update PCEs to reflect those changes so that new join points are captured correctly. As depicted in Figure 2, prior to manually altering the PCE, the developer selects the PCE in the Eclipse IDE and executes our tool for assistance in performing the updates correctly. At this juncture, the intention patterns previously associated with the PCE are retrieved from storage and ran against the new base-code version in order to unveil the suggested join points, which are those that share intentional similarities with the join points previously selected by the PCE in the original version of the base-code. Each suggestion is presented with the derived patterns' confidence. The developer then selects the desired join points to be incorporated into a new version of the PCE.

3 Conclusion and Future Work

By adapting existing approaches for requirement traceability maintenance, the REJUVENATE POINTCUT Eclipse IDE plug-in helps developers maintain pointcut expressions in evolving aspect-oriented software. Additional join points whose associated program elements exhibit common patterns in the intentional structure of the base-code are suggested for incorporation to existing PCEs. In its current state, the tool presents the user with the new suggested join points for manual integration. In future versions of the tool, once the selection is final, the pointcut will be rewritten using existing refactoring support adapted for AspectJ constructs. Furthermore, our current refactoring module prototype is able to handle simple augmentations, e.g., via disjunction of join points, however, we plan to utilize existing refactoring tool-support in order to perform more complex pointcut rewriting via join point clustering and string analysis of program element names.