Application of FOP and AOP Methodologies in Concert for Developing Insurance Software Using Eclipse-Based Open Source Environment

Amita Sharma¹ and S.S. Sarangdevot²

¹ Tulsi Shree, B-16, Kanta Khaturia Colony, Bikaner-334003, India
   amita214@rediffmail.com
² Director, Deptt. of Computer Science & I.T., J.R.N. Rajasthan Vidyapeeth (Deemed)
   University, Udaipur-313001, India
   drsssarangdevat@yahoo.com

Abstract. Feature-Oriented Programming (FOP) and Aspect-Oriented Programming (AOP) are complementary methodologies that can be combined to overcome their individual limitations. In the present study, usefulness of this approach has been investigated in developing Insurance Software: TG_LifeInsurancesoft, using Eclipse-FeatureIDE-AJDT open source environment with Feature composition framework and enhanced tool chain: FeatureHouse. It is observed that integration of AOP concepts into stepwise refinement method of FOP enhances its capability of expressing and handling homogeneous as well as advanced dynamic crosscutting, thus reducing code redundancy, complexity, development time, maintainability and cost of the software system. The study concludes that combination of FOP and AOP methodologies using Eclipse-FeatureIDE-AJDT environment offers a powerful support for modular design and implementation of comprehensible, reusable, consistent, maintainable and cost effective insurance software system with user selected features.


1 Introduction

Feature-Oriented Programming (FOP) [1, 2, 3, 4] is a programming paradigm that allows decomposition of a program into its constituent features and thus extends the principle of separation of concerns [5, 6] to features. A feature [7, 8] is a logically cohesive piece of functionality or end-user characteristic or requirement that is relevant to a stakeholder. It is used to express the commonalities and variabilities of the domain specific programs and software systems [9]. FOP aims at the modularity of features and models a program as sets of user selected features that in aggregate represent the final product.
One of the strengths of FOP is its ability to produce many similar but functionally different programs simply by selecting the desired features. With the same set of features, a developer can generate several different software systems that share common features and differ in other features. Features refine other features incrementally. This stepwise refinement [10] leads to a layered stack of features. This helps in constructing well-structured software that can be tailored to the specific needs of the user and the application scenario.

AHEAD (Algebraic Hierarchical Equations for Application Design) [11, 12] is an architectural model for FOP. It is a basis for large-scale compositional programming [3]. It extends the concept of FOP to all software artifacts. However, command line operation is its big limitation. FeatureIDE [13, 14] is an Eclipse-based IDE that supports evolution of program families following architecture model. It provides tools for the feature-oriented design and implementation process.

FeatureHouse [15] is a descendent of AHEAD program generator. It provides facilities for feature composition based on a language independent model of software artifacts. It also provides an automatic plug-in mechanism for the integration of new artifact languages. It relies on a general model of the structure of software artifacts, called feature structure tree (FST), which represents the essential modular structure of a software artifact and abstracts from language-specific details. FeatureHouse can be used with the Eclipse-based open source visual development environment FeatureIDE.

Designing software using FOP provides significant advantages. However, some issues emerge which reveal shortcomings of FOP approach and require further consideration. It is observed that lack of crosscutting modularity, scalability and feature interactions are the main drawbacks of FOP. During software evolution several modifications and extensions are made to fit the unanticipated requirements. These crosscut many existing implementation units in numerous ways and cause code scattering [16] and tangling [17], thereby increasing the complexity and impairing software quality. This highlights the need for the improvement in the approach so that efficient software can be evolved to suit the tailor made requirements of the user.

Aspect-Oriented Programming (AOP) [18, 19, 20] focuses on the separation and modularization of crosscutting concerns [21]. Invented by Kiczales [18], AOP defines a new program construct — ‘aspect’ [8, 22], which is a software entity that implements crosscutting functionality in a modular way and provides most promising solution for elimination of code scattering and tangling. This reduces software complexity and improves quality. AspectJ [23, 24] is the most popular general purpose AOP extension to Java. It adds to Java a few new constructs: pointcuts, advice, intertype declarations and aspects. AspectJ Development Tools (AJDT) [25, 26] provides most popular and commercially successful Eclipse-based IDE support for AspectJ implementations with a rich set of features like Aspect Visualizer, Outline View, Editor Support and Debugger. Thus Eclipse-AJDT [27, 28] provides the most useful environment for AOP implementations.

However, in many cases aspects of AOP are not adequate to implement features stand alone. This is because features are mostly implemented by collaborations, and AOP technique is not very suitable to express and encapsulate collaborations. Another drawback is that aspect cannot be bounded to a certain scope. Also AOP technique does not support incremental software development process.