Chapter 30
Aspect-Oriented Programming (AOP) Support on RIAs Development

Giner Alor-Hernández
Instituto Tecnológico de Orizaba, Mexico

Viviana Yarel Rosales-Morales
Instituto Tecnológico de Orizaba, Mexico

Luis Omar Colombo-Mendoza
Instituto Tecnológico de Orizaba, Mexico

ABSTRACT

This chapter emphasizes the importance of employing Aspect-Oriented Programming (AOP) on software development, especially in software engineering. Some advantages in the development of RIAs by using AOP are Maintainability, Extensibility, and Reusability. This chapter presents a review of several success stories of AOP implementation in real world development projects and discusses the lessons learned in these projects. The works analyzed in the state-of-the-art are classified by Web development, Usability Engineering, and other related perspectives. Finally, the chapter also addresses AOP support between JavaScript-based RIA frameworks and non-JavaScript-based RIA frameworks providing either native or third-party AOP facilities. Some code snippets depicting the use of these facilities for implementing AOP concepts are also presented.

1. INTRODUCTION

Nowadays, AOP (Aspect-Oriented Programming) is one of the concepts on computer programming primarily used in research and industry. Its use is an evolutionary way of developing software that improves upon OOP (Object-Oriented Programming), in the same way that OOP improved upon procedural programming. OOP introduced the concepts of encapsulation, inheritance, and polymorphism for creating a hierarchy of objects that model a common set of behaviors. Although OOP has become

DOI: 10.4018/978-1-5225-3422-8.ch030

relevant, it has failed in handling common behaviors that extend across unrelated objects. This means that OOP enhances vertical relationships but not horizontal relationships. As an example, logging code is often horizontally scattered across object hierarchies, but it has nothing to do with core functions of the objects scattered across. This situation occurs with other types of code, such as security and exception handling AOP provides a solution for abstracting crosscutting code that spans object hierarchies without functional relevance to the code it spans. AOP is a tool that enables to abstract the crosscutting code into a separate module, known as an aspect, rather than embedding crosscutting code in classes and then dynamically applying the code where it is needed. The application of the crosscutting code is achieved by defining specific places, known as pointcuts, in the object model where the crosscutting code should be applied (Ekabua, 2012). Depending on the intended AOP framework, crosscutting code is injected at the specified pointcuts at runtime or compile-time. Ideally, AOP introduces a very powerful concept, which allows the introduction of new functionalities into objects without the objects needing to have any knowledge of that introduction (Holmes, 2012). Defects and deterioration of software are caused by changes in source code, and a lot of these changes cannot be avoided; however, they can be minimized. In most cases, when changes are made to software, the entire program is reengineered (Fayad & Adam, 2001).

Changes are inseparable part of software evolution. Changes take place in the process of development as well as during software maintenance. Huge costs and low speed of implementation are characteristic to change implementation. Often, change implementation implies a redesign of the whole application. The necessity of improving the software adaptability is fairly evident. Changes are usually specified as alterations of the base application behavior. Sometimes, it is needed to revert a change, which would be best done if it were expressed in a pluggable way. Another benefit of change pluggability is apparent if the change has to be reapplied. However, it is impossible to have a change implemented to fit any context, but it would be sufficiently helpful if a change could be extracted and applied to another version of the same base application. Such a pluggability can be achieved by representing changes as aspects (Dolog, Vrani´c & Bielikov´a, 2001). Some changes appear as real crosscutting concerns in the sense of affecting many places in the code, which is yet another reason for expressing them as aspects. This would be especially useful in the customization of web applications.

Typically, a general Web application is adapted to a certain context by a series of changes. With the arrival of a new version of the base application, all these changes have to be applied to it. In many occasions, the difference between the new and the old application does not affect the structure of changes.

A successful application of AOP requires a structured base application. Well-structured Web applications are usually based on the MVC (Model-View-Controller) pattern with three distinguishable layers: model layer, presentation layer, and persistence layer (Bebjak, Vranic & Dolog, 2007).

AOP can be implemented in RIAs. This kind of programming is capable of providing many benefits to RIAs, such as adding new levels of security and functionality without modifying the original code application. As it was mentioned above the advantages of applying AOP development of RIAs are varied and very important. Some of the most important advantages in the development of RIAs by using AOP are mentioned below:

- *Maintainability* is very important for RIAs development, since it enables to make changes as effectively and efficiently as it is possible. Moreover, the AOP provides a high level for maintainability.