Adapting Aspect-Oriented Applications: A Trial Experience

Claudia Marcos and Jane Pryor

E-mail: {cmarcos,jpryor}@exa.unicen.edu.ar
ISISTAN Research Institute, Facultad de Ciencias Exactas, UNICEN
Paraje Arroyo Seco, B7001BBO Tandil, Argentina
Tel/Fax: + 54–2293–440362/3 http://www.exa.unicen.edu.ar/~isistan/

Abstract. During a system's life cycle, new requirements or changes in the existing ones imply modifying the system. Aspect-oriented software development is a new approach to the modularization of systems, yet it does not provide mechanisms to aid the evolution of software. The effort required to support the evolution greatly depends on the tool used for its construction. For this reason, the selection of a tool should also take into account its support for implementing evolving requirements. In this paper we present a comparison of two different tools, AspectJ and Alpheus, to support the construction and evolution of aspect-oriented applications. AspectJ is an aspect-oriented programming language based on Java. Alpheus is an aspect-oriented development tool based on a reflective framework.

Keyword. System evolution, unanticipated system evolution, aspect-oriented applications, aosd evolution, reflective architecture for aspects.

1 Introduction

All systems evolve during their life cycle due to new requirements or to changes in their functionality [1]. A system´s evolution may be anticipated or unanticipated in its development. When the evolution has been anticipated, the changes to a system can be carried out without major problems. However, unanticipated evolution usually produces deterioration of a system. For this reason it is very important to have tools which support unanticipated system evolution.

The aspect-oriented paradigm provides constructors which encapsulate the elements whose code tends to be disseminated throughout many functional components. These constructors are called *aspects* [2] [3]. The goals of this paradigm are the encapsulation of these aspects and the minimization of the

dependency among them and the basic functional components. In general terms, the system qualities obtained through the separation of concerns also have an impact on the ease of a system´s evolution, due to independent and well encapsulated code.

This work presents an evaluation and documentation of different techniques, tools and programming languages, for the development and evolution of aspect-oriented software. In order to carry out this evaluation, a case study was developed. To study the impact of evolution, requirements were modified and also added at different stages of the life cycle, using these tools. The example was developed with AspectJ, a language for aspect-oriented programming, and with Alpheus, a visual tool for the construction of aspect-oriented applications. Then it is evaluated how the tools supported changes in requirements, both during the development of the application and once completed, and the incorporation of new requirements.

The following two sections introduce AspectJ and Alpheus, respectively. Section 4 describes the example used to compare both tools. Section 5 shows how the example is developed with Alpheus, and how it supports the system's evolution and the evaluation of this support. In Section 6 the example and evaluation is developed using AspectJ. The remaining section presents the conclusions.

2. AspectJ: an Aspect-Oriented Programming Language

AspectJ extends Java with new kind of classes called *aspects* [2]. These aspects crosscut the classes, interfaces and other aspects. In AspectJ, an aspect is a Java class, but it adds five new entities: join-points, point-cuts, introductions, advices and aspects themselves.

A *join-point* is a well-defined point in the execution of a program, such as method calls, method executions, access to attributes, exception handling, etc. A *point-cut* captures a collection of events in the program execution. It is a structure which has been designed to identify and select join-points in an AspectJ program. When a join-point is reached in the primary application code, the corresponding point-cut is activated and the aspect code is executed. The *advices* define the implementation code of the aspect, which is to be executed in the places defined by the point-cuts. *Introductions* and *declarations* are used to change the original structure of a program by adding or extending interfaces and classes. They may introduce new elements such as methods, constructors, or attributes.

3. Alpheus: A Tool for Aspect-Oriented Applications

Alpheus is a tool based on a reflective framework [4] that supports the development of aspect-oriented applications of different domains, enhancing desired software qualities such as adaptability and reuse [5][6]. The support for aspects that Alpheus provides has the following characteristics:

- **Flexible strategies for the runtime association and activation of aspects**: that is at what point the thread of control to the aspect [7]. When all methods and objects of a specified class are associated to an aspect: we call

this strategy *class association*. When some methods are associated to an aspect: *method association*. When some objects are associated to an aspect: *object association*. When a particular method of an object is associated to an aspect: *object-method association*. Additionally, the activation of the aspect can take place *before* and/or *after* the intercepted method.

- **Reuse of planes**: The concept of *planes* has been introduced in order to obtain a clear separation and encapsulation of concerns. A *plane* is a collection of aspects which carry out similar or related functionality.
- **Definition and solving of conflicts between competing aspects**: *Conflicts* may occur if two or more aspects compete for activation. Different categories of conflict activation policies and different levels of granularity between conflicts are defined [6].

The tool allows developers to define the components of the application and then generates the Java code of the application. Alpheus also provides the visualization of the components of an application, plus some UML diagrams [8].

4 An Example – Personal Web Server

A Personal Web Server (PWS) is a server application which receives petitions for documents from a web client, locates and then sends the document. The HyperText Transfer Protocol (HTTP) is used to establish the connection. HTTP is a simple protocol implemented in TCP/IP. The HTTP client sends a document identifier to the server and the server replies by sending HTML documents or common text. A firewall is a filter mechanism that applies security policies to the network traffic. The firewall has some access policies applied from and to the external network.

This example will evolve in two different ways. During its development, the new requirement is the necessity to register the access of the HTML documents stored in the PWS. When it is working it is necessary to store other types of documents (gif, jpg) not only HTML. It is also necessary to introduce a new firewall at night time for some statistics.

As the PWS has a server which offers services to clients, the natural architecture for this system is a client-server one [9]. Clients have to know which servers are available but they do not know anything about the other clients [1].

5. Personal Web Server with Alpheus

Three planes are defined in Alpheus: *PlanoFirewall*, containing the policies related to the access from and to the network; *PlanoIncidencias*, containing the actions log; and *Base*, containing the functional application. The aspects and objects are then defined for each plane.

The composition (called association) between the aspects and objects can be defined. For each association it is necessary to specify when and how the aspect is

activated (before, after, etc.), plus the strategy to follow (class, method, class-method, etc). For the PWS two associations have been defined.

The first association is in order to control the access to the network. It is defined between the OFirewall object and the ASP_Firewall aspect of the PlanoFirewall plane. The association has some characteristics: *before*, because the aspect is to be activated before the base element; *method-reflection*, as the reglas_red(id) method of the OFirewall class will be modified by the aspect's functionality. The second association is created to store the access to the HTML documents. It is defined between the MC_Firewall aspect of the PlanoFirewall plane and the ASP_Incidences of the PlanoIncidencias plane.

6.2.1 Evolution during Development

A new requirement is introduced the access to HTML documents is to be registered by the system because statistics. In order to support this new requirement a new plane is defined, *PlanoEstadistica*. Secondly, the aspects in this plane are specified (ASP_Estadistica) (Figure 2). Lastly, an association is established between the OConnectionThread of the base plane and the newly created ASP_Estadistica of the PlanoEstadistica plane. As a result, whenever the OConnectionThread is invoked the ASP_Estadistica oversees the access to the HTML documents.

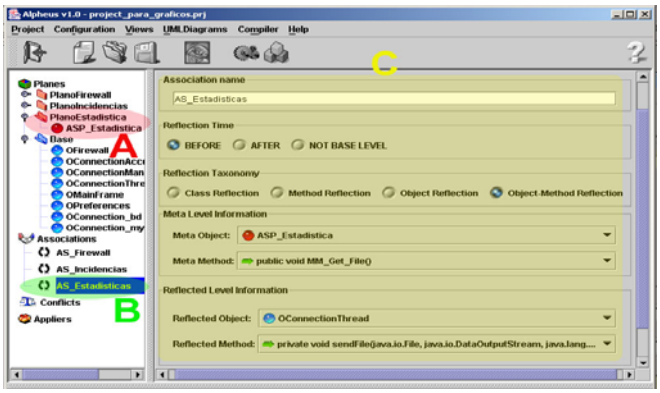

Figure 2. Evolution during development

6.2.2 Evolution When the System is Working

Once the system components were defined Alpheus uses this specification of the application and generates the corresponding Java code. Once the system is working, it is necessary to register the access to all documents, not only HTML. To support this, the add_statistic method of the ASP_Estadistica aspect has to be modified and the aspect has to be recompiled.

The system continues to evolve when it is necessary to introduce a new firewall for night-time. The new plane PlanoFirewallNoche and the aspect ASP_Firewall_noche are specified (Figure 4). The association between the Firewall class of the base plane and the ASP_Firewall_noche aspect is specified.

This new composition causes a conflict between aspects, because when an OFirewall object receives a message, two aspects compete for activation:

ASP_Firewall and ASP_Firewall_noche. The activation of the firewalls depends on the time of day, therefore it is not possible to determine before-hand which of the aspects has to be activated (context-dependent conflict). To solve this type of conflict, the designer specifies the conflict and the programmer inserts the corresponding code. For the rest of conflicts the tool generates automatically the solution.

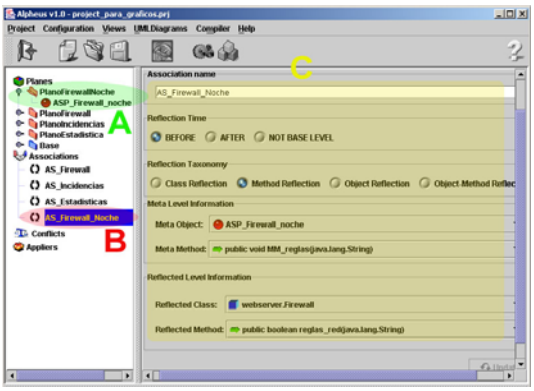

Figure 4. FirewallNoche association

6.3 Personal Web Server with AspectJ

The PWS application was also developed with Java (IDE) Borland JBuilder6 Enterprise and AspectJ. For the PWS example, two aspects have to be coded. The ASP_Firewall aspect (Figure 5 A) implements the policies of the firewall, crosscutting the Firewall class as it is in charge of supervising the access to the HTML document. The second aspect is called ASP_Incidences, and it registers the events of the application by storing them in a data base.

The ASP_Firewall aspect defines a point-cut for the invocation of the reglas_red method of the Firewall class (Figure 5 B). The advice (Figure 5 C) has been declared as *before*. Before analyzing whether the access to the document is allowed, the ASP_Firewall modifies the result variable of the firewall object according to the information retrieved from the database.

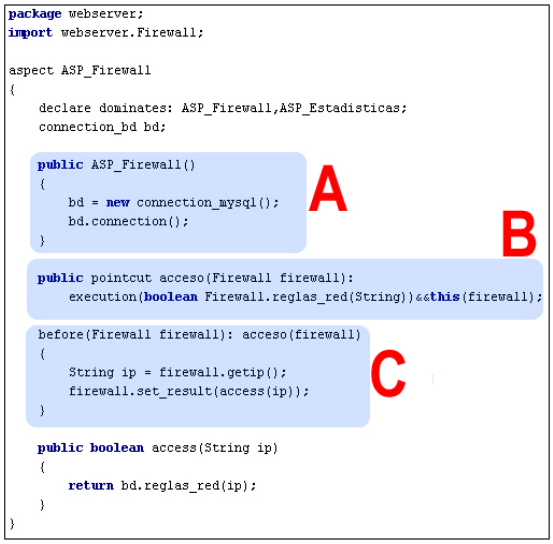

Figure 5. Aspect definition with AspectJ

The application code is generated in two steps: firstly the weaver converts the aspect code to Java code, and secondly, the Java compiler generates the Java object code (.class), where the application and aspect code are mixed together.

6.3.1 Evolution During Development

Because AspectJ is a programming language and is therefore used during the implementation phase, it is not really possible to evaluate evolution during development. However, it is possible to introduce the new requirement by creating a new aspect called ASP_Estadistica (Figure 6). This aspect will register the access to the HTML documents when the sendFile of the ConnectionThread class is invoked. Section A of Figure 6 describes the definition of the aspect, and section B shows the point-cut and its *estadistica* advice.

The joint point for the point-cut *estadistica* is related to the sendFile method of the ConnectionThread class and the ct instance. This aspect registers the access to the HTML documents in the system database.

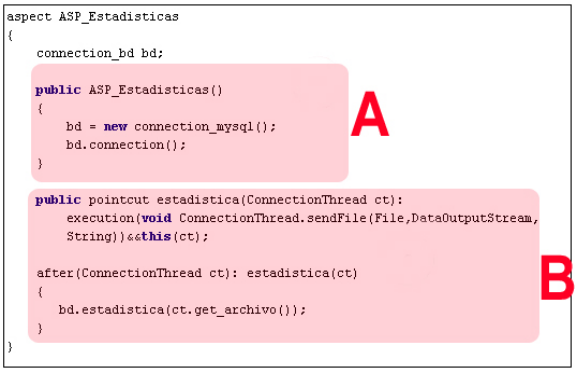

Figure 6. Definition of the ASP_Estadisticas aspect

6.3.2 Evolution When the System is Working

Once the system is working it is necessary to register the access to all kinds of documents and not only the HTML ones. The system is then extended in order to introduce a new firewall for night-time analysis, so the ASP_FirewallNoche is added. The ASP_FirewallNoche aspect is activated when the documents are requested from 00:00 hrs to 8:00 hrs. To support the activation of the aspect during the night a conditional sentence has to be implemented. The ASP_Firewall aspect also has to be modified introducing the conditional sentence to decide when this aspect has to be activated (during the day).

The ASP_Firewall and ASP_FirewallNoche aspects have a conflictive situation which is not of precedence but dependent on the context, because their activation depends on the time. AspectJ does not support this kind of conflict so the solution has to be coded into the aspects. The only mechanism supported by AspectJ for the resolution of conflicts is of precedence. Both aspects have to be modified in order to introduce the sentences needed to verify the hour.

3. Comparison of the Tools

These tools were evaluated in their support for the system evolution in two ways: their flexibility to support changes in the requirements, and their extensibility for introducing new functionality. Moreover, they also were evaluated during the system development and once the system is working.

As Alpheus is a research tool it is free and open-source (http://www.exa.unicen.edu.ar/catedras/reflex/). The available documentation may be found in the form of papers describing Alpheus, the reflective framework it instantiates, and how it works by means of examples. The user interface of Alpheus is very intuitive and friendly. To aid the designer, Alpheus also provides consistency validation and visualization of the application by means of different diagrams and also provides some UML diagrams.

AspectJ provides the means to code aspect-oriented applications using a well-known development environment. Java developers therefore have all the support necessary to begin with the development of aspect-oriented applications. AspectJ does not support the resolution of different kinds of conflicts that an application may have. AspectJ has very good documentation and it is widely-used for the development of aspect-oriented applications. The development environment used with AspectJ provides some extra benefits, such as code generation, different kinds of reports, code documentation, etc. Table 1 shows the results of the evaluation of the example evolution using AspectJ (AJ) and Alpheus (A).

Table 4. Tools Evaluation

Evolution	Statistics for HTML Documents		Statistics Extension		Night-time Firewall	
Elements to Evaluate	**AJ**	**A**	**AJ**	**A**	**AJ**	**A**
Viability of implementing new requirements	YES	YES	YES	YES	YES	YES
Number of classes to be implemented	0	0	0	0	0	0
Number of classes to be modified	0	1	0	1	0	1
Number of aspects to be created	1	1	1	0	1	1
Classes to be compiled	13	2	13	1	13	2
Implementation time	20 min	30 min	20 min	30 min	30 min	45 min

7 Conclusion

This paper presents an evaluation of two different tools, Alpheus and AspectJ, which support the development of aspect-oriented applications. They were evaluated analyzing their support for the evolution by the development of an example.

Alpheus is a visual development tool which instantiates a reflective framework. With this tool it is possible to specify all the components of an aspect-oriented application and then automatically generate the corresponding code. It also provides different levels of visualization of the application and automatic detection of conflicts. AspectJ is an aspect-oriented programming language based on Java which introduces some new concepts in order to code aspects and their characteristics.

One of the main differences in the tools is the way in which the aspect weaving process is carried out. In Alpheus, the weaving is done at run-time and AspectJ has a static weaver, but on the other hand, the performance is better. In both tools it was possible to support the evolution of the Personal Web Server application, and the amount of aspects and classes needed for this evolution were almost the same. In AspectJ it was always necessary to recompile all the classes and in Alpheus only the affected classes are compiled again.

8 Bibliography

[1] I. Sommerville. *Ingeniería de Software*. Sexta edición 2002.

[2] Aspect-Oriented Programming Home Page. At Http://aosd.net

[3]AOSD 2002, 1st. International Conference on Aspect-Oriented Software Development. Enschede. Gregor Kiczales, ed., (ACM Press, The Netherlands, 2002).

[4] P. Maes. Concepts and Experiments in Computational Reflection. In Proceedings of OOPSLA ’87.

[5] J. Pryor, and C. Marcos. Constructing Aspect-Oriented Applications using a Reflective Framework. Technical Report TR-28-02, ISISTAN Research Institute, Universidad Nacional del Centro de la Provincia de Buenos Aires (UNICEN), 2002.

[6] F. Valentino, A. Ramos, C. Marcos, and J. Pryor, A Framework for the Development of Multi-Level Reflective Applications. Proc. of the Second Argentine Symposium on Software Engineering (ASSE), Argentina, 2001.

[7] C. Marcos, Patrones de Diseño como Entidades de Primera Clase, PhD. Thesis, Facultad de Ciencias Exactas, ISISTAN Research Institute, Universidad Nacional del Centro de la Provincia de Buenos Aires (UNICEN), April 2001.

[8] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language*. User Guide (Addison-Wesley, 1999).

[9] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-Oriented Software Architecture - A System of Patterns*. John Wiley & Sons, 1996.