

Representing Aspect-Based Architecture of Software Engineering Environments *

Elisa Yumi Nakagawa[†] and José Carlos Maldonado

Dept. of Computer Systems
USP - University of São Paulo
São Carlos/SP, Brazil
{elisa, jcmaldon}@icmc.usp.br

Abstract

The Aspect-Oriented Programming (AOP) has been lately explored as an approach that aims at achieving a better separation of concerns, one of the key principles in Software Engineering area. In this context, an aspect-based architecture for SEEs (Software Engineering Environments), named RefASSET (Reference Architecture for Software Engineering Tools), has been established, considering the lack of reference architectures for supporting the development of tools and SEEs. In particular, it has explored the use of aspects as a communication mechanism among software engineering tools. This paper presents our experience in documenting RefASSET through architectural views. These views have been selected, adapted, and developed to support the representation of aspects. Besides, we compare our approach with other recent works.

Keywords software architecture, aspect-based software architecture, architectural documentation, architectural view, software engineering environment

1. Introduction

The Aspect-Oriented Programming (AOP) has arisen as a new technology to support a better separation of concerns and to more adequately reflect the way developers reason about the system [9], contributing to maintainability, reusability, and easiness to write software. The use of aspects in the software development requires that analysis, design techniques and methods, as well as techniques to document software architecture, be updated and established. It is important to highlight that there are not notations, techniques, and methods broadly accepted and consolidated to represent AO (Aspect-Oriented) systems. However, some initiatives can be found in this direction [1, 2, 3, 14], mainly based on UML notation. In the software architecture context, little support is available to architects who want to capture in the software architecture the elements and

relations introduced by AOP. A recent work has investigated how aspects affect the diverse architectural views [10].

Considering the lack of reference architectures that support the development of tools and SEEs, we have established RefASSET (Reference Architecture for Software Engineering Tools) [11], an aspect-based architecture that aggregates concepts, approaches, and technologies to develop evolvable and reusable software. We have investigated mechanisms to document adequately this architecture. The goal on this paper is to present our experience in documenting software architectures of systems that use aspects. We also compare our proposal with recent works in the same direction, such as Merson's work [10].

The remainder of this paper is organized as follows. At first, we briefly present the context of our research. Next, we discuss the architectural views developed and compare them with other works. Finally, in the last section, we present our conclusions.

2. Context of Our Research

RefASSET is not the focus of this work, but it is briefly described here aiming at contextualizing the issues discussed in this paper. In short, RefASSET is inspired on architectures sufficiently investigated, such as ECMA Reference Model [4]. It is based on several well-established researches from Software Engineering area: architectures of interactive systems and Web systems (architectural pattern MVC (Model-View-Controller) and architecture of interactive applications, specifically the 3-tiers architecture¹). For the purpose of achieving easier integration, maintainability, quality, and reusability of the environments built based on RefASSET, this architecture is strongly based on separation of concerns [12] and on international standard ISO/IEC 12207 (Information Technology - Software Life Cycle Processes) [7].

An important point that must be highlighted is how RefASSET treats the supporting and organizational activities. Based on ISO/IEC 12207, supporting and organizational activities have been seen as crosscutting concerns [12], since those activities occur through all software life cycle, from requirement specification to maintenance. For instance, documentation is a supporting activity that must be conducted from requirements specification to maintenance and, therefore, it is a crosscutting concern. In order to design and implement modules that automate supporting and organizational activities — named supporting crosscutting modules and organizational crosscutting modules — we have used aspects in their internal structure. Besides, we have been the first to adopt aspects as a mechanism to integrate supporting and organizational

* This work is supported by Brazilian funding agencies FAPESP and CNPq.

[†] She is also professor of the Dept. of Administrative Science and Technology, Uniara - Araraquara University Center, Araraquara/SP, Brazil.

¹ <http://www.sei.cmu.edu/str/descriptions/threetier.html>

crosscutting modules to tools that automated software engineering primary activities (requirement specification to maintenance).

RefASSET was specialized to software testing domain, resulting in RefTEST (Reference Architecture for Software Testing Tools) [11]. RefTEST establishes the core elements, for example, test cases, test requirements, and testing criteria, that must be managed by testing tools.

3. Representing Architectures of SEEs

In order to document and communicate adequately the reference architectures in the context of our work, we have investigated and developed the main architectural views: module, runtime, and deployment views. Other views — implementation and data model views — have not been developed yet, since these views require low level information which has not been developed for RefASSET yet. Also, for describing the views we have used UML 2.0 (Unified Modelling Language) [13], in the same vein of [8] and [5]. Among the views that represent RefASSET, in this paper we highlight those views whose notation used to represent them had to be extended to represent the aspects in their composition. These views — runtime and module views — are discussed below.

The module view shows the structure of the software in terms of code units. Packages and classes can be used to represent this view, as well as containment, specialization/generalization, and dependency relations. In UML 2.0, the dependency is shown as a dashed arrow between two model elements (from client to supplier) labelled with an optional stereotype; the stereotypes `<<use>>`, `<<instantiate>>`, and `<<refine>>` are often used. When aspects are used as a module or inserted into the module and these aspects crosscut or affect other modules, we have used the stereotype `<<crosscut>>` to represent this dependency relation, as in [1] and [14]. However, to avoid tangling in this view, in [10], it is proposed not to draw dashed arrows for crosscut relations, suggesting to follow the same simple representation used in AOP (when it is created a pointcut specification in an aspect-oriented language, it is not indicated all classes that contain the target join points; pointcuts are specified using signature patterns with wildcards or metadata annotations). We think that the module view becomes more expressive if the crosscut relations are indicated with a graphical representation (dashed arrows), mainly in our case (the representation of a higher level architecture).

Figure 1 illustrates the module view of RefASSET. For instance, the package `supporting_crosscutting_modules` contains modules, such as the `documentation`, that are implemented by classes and aspects. These aspects crosscut other modules, for example, the software engineering tools (contained in `primary_activities_tools` package), and change the execution flow of the tools inserting functionalities related to crosscutting concerns automated by these modules. Thus, there are dependency relations, labelled with `<<crosscut>>`, among the package `supporting_crosscutting_modules` and other modules. Therefore, for the purpose of representing this view, we have used UML elements and the stereotype `<<crosscut>>` in use in several works from AOP research area.

To represent the runtime view — also known as component and connector (C&C) view — that shows the structure of the system when it is executing, we have used UML elements: components, provided and required interfaces, packages, ports, and connectors. According to [10], the runtime view will not show the aspects individually, because their logic is dispersed in the binary code of the runtime components; therefore, runtime views of the architecture do not change due to the use of aspects in the implementation. We have agreed that in the runtime of a system the aspects do not exist, however, we think that it is relevant to highlight how the aspects or modules that use aspects communicate with other modules, i.e.

which are their interfaces. Obviously, these interfaces are different from the provided and required interfaces of modules composed only by objects. For that reason, we believe that these interfaces must be represented in runtime views.

As known, in UML 2.0, the interface realization dependency from a classifier (set of objects) to an interface is shown by representing the interface by a circle, labelled with the name of the interface, attached by a solid line to the classifier that realizes this interface. The usage dependency from a classifier to an interface is shown by representing the interface by a half-circle attached to the classifier that requires this interface.

UML 2.0 does not provide any notation to interfaces of modules that contain aspects in their structure and use these aspects to affect or crosscut other modules. We named these modules as Aspect-based Modules (AModule). We have proposed a filled circle, labelled with the name of the interface, attached by a solid line to the AModules, to represent the interface of these modules. We named this kind of interface as Interface Made by Aspects (IMA). We have also proposed a half-square, attached by a solid line to the module that is crosscutted or affected by aspects into AModules. Figure 2 illustrates this notation. Thus, modules that exclusively use aspects to communicate with other modules have only this kind of interface. Differently from provided interface, IMA represents a declaration of a set of coherent features and obligations, specifying a contract. In practice, it represents the characteristics that classifiers must have to use the AModules, such as classes and their operations. Our experience points out that the establishment of the IMAs provides facilities to reuse AModules.

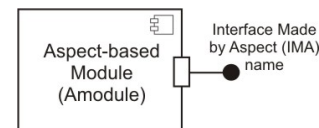


Figure 2. Notation for Interface Made by Aspects (IMA)

Figure 3 presents a runtime view of RefASSET. In order for this view become readable, the connections of the documentation module with other components are focussed. The documentation module has only IMAs and affects the behavior of the tools contained into `primary_activity_tools`. It is also affected by the `persistence` module that stores data in a database.

In short, the module and runtime views change due to the use of aspects in the implementation. The deployment view, a high level view of the architecture, does not change, since it describes the machines and network connections that are used by the software systems. We do not develop other views as already justified; although, we believe that implementation view is affected when aspects are used in the implementation. This view shows the structure of the software in terms of files and folders in the production or development environment, and aspects must appear as an unit. The last view — data model view — can change, since aspects can change the elements that are persisted in the database. For instance, the persistence — a well-known functionality that can be coded through aspects — can affect other modules and persist data of these modules according to the aspects.

3.1 Use of the Architectural Representation

Our research group has investigated several issues related to AOP, such as modelling and testing, as well as development of tools and SEEs. In this context, we are using RefASSET and RefTEST to develop and reengineer software testing tools. These tools have been

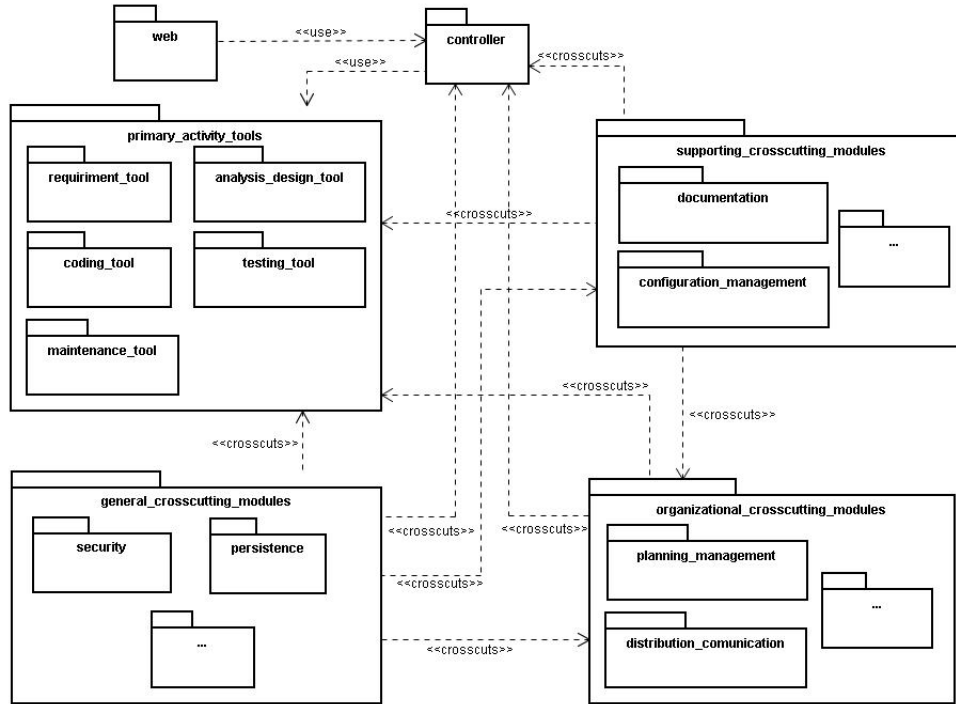


Figure 1. Module View of RefASSET

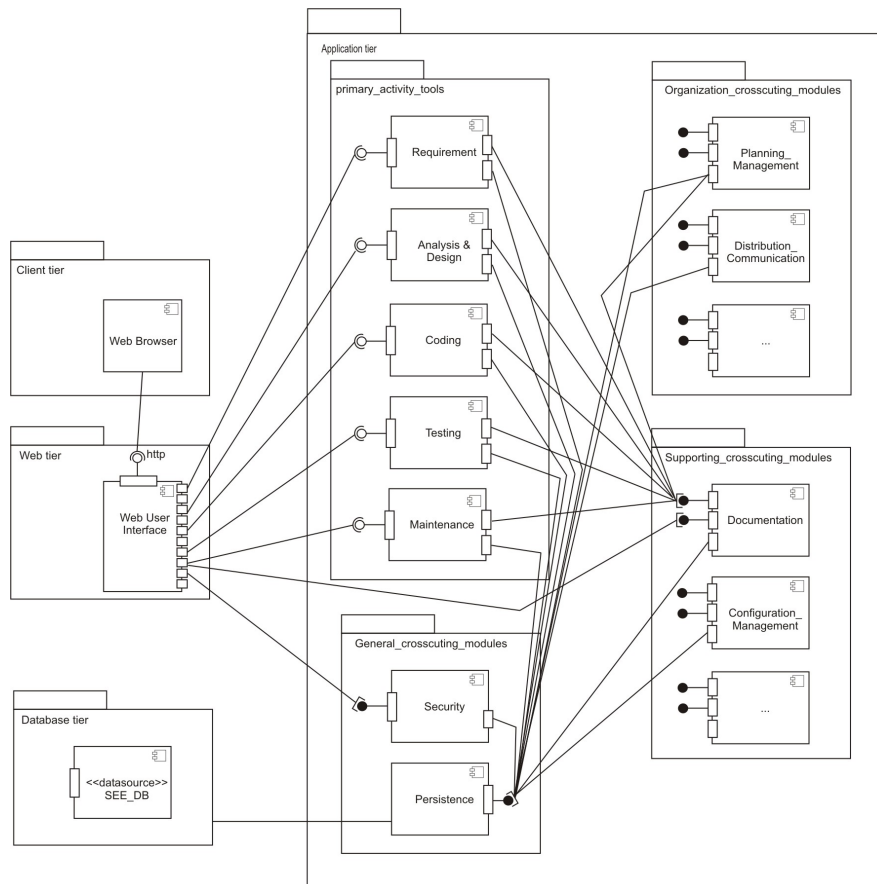


Figure 3. Runtime View of RefASSET

implemented in Java² with AspectJ³ and deployed in web platform. We can point out that the use of a standardized, well-understood notation — i.e. UML — has contributed to communicate more easily the knowledge contained in the reference architectures (RefASSET and RefTEST). Besides, the UML extensions used or proposed here have been easily understood by developers and other researchers and accepted as mechanisms to represent aspects into architectural views.

4. Conclusion

In spite of the relevance of software architecture and its documentation, as well as the recent use of aspects in the software development, it can be noticed that there is a lack of works that address the documentation of architecture of systems that use aspects. In this context, this paper presented our experience in documenting architecture using architectural views, based on our need to document reference architectures of aspect-based SEEs.

We have observed that not all architectural view needs to be extended to represent aspects. This observation can contribute to the revision of [6]. In general, higher level architectural views (i.e. the deployment view) are not changed due to the use of aspects in the implementation. Otherwise, the lower level views or more detailed views (for example, module, runtime, and implementation views) must be adapted and require changes in their notations in order to represent adequately the aspects. Thus, we have proposed new UML elements to document explicitly the use of aspects.

Finally, our experience pointed out that the use of a standardized and well-understood notation as UML to represent architectures is relevant. Thus, the proposal of an UML profile to software architecture, as well as to aspect-based software architecture, is a need.

References

- [1] O. Aldawud, T. Elrad, and A. Bader. A UML profile for aspect oriented modeling. In *Proc. of Third International Workshop on Aspect-Oriented Modeling*, Boston, USA, Mar. 2003.
- [2] C. V. F. G. Chavez. *A Model-Driven Approach to Aspect-Oriented Design*. PhD thesis, Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, RJ, Apr. 2004.
- [3] S. Clarke and E. Baniassad. *Aspect-Oriented Analysis and Design: The Theme Approach*. Object Technology. Addison-Wesley Publishing Company, 2005.
- [4] ECMA and NIST. Ecma and nist. reference model for frameworks of software engineering environments, dec 1991. Special Publication Report No. ECMA TR/55, 2nd Ed.
- [5] R. Hilliard. Using the UML for architectural description. In *Proc. of UML'99 - The Unified Modeling Language: Beyond the Standard, Second International Conference*, Fort Collins, CO, USA, Oct. 1999.
- [6] IEEE. 1471-2000 – IEEE recommended practice for architectural description of software-intensive systems, Sept. 2000.
- [7] ISO. ISO/IEC 12207. Information technology – software life-cycle processes, 1995.
- [8] J. Ivers, P. Clements, D. Garlan, R. Nord, B. Schmerl, and J. R. O. Silva. Documenting component and connector views with UML 2.0. Technical report, SEI/Carnegie Mellon University, 2004. CMU/SEI-2004-TR-008.
- [9] G. Kiczales, J. Irwin, J. Lamping, J. Loingtier, C. Lopes, C. Maeda, and A. Menhdhekar. Aspect-oriented programming. In M. Akşit and S. Matsuoka, editors, *Proc. of the European Conference on Object-*

Oriented Programming, volume 1241, pages 220–242. Springer-Verlag, 1997.

- [10] P. Merson. Representing aspects in the software architecture - practical considerations. In *Early Aspects Workshop, OOPSLA 2005*, San Diego, California, Oct. 2005.
- [11] E. Y. Nakagawa. *A Contribution to the Architectural Design of Software Engineering Environments*. Phd thesis, Instituto de Ciências Matemáticas e de Computação, University of São Paulo, São Carlos, SP, Brazil, Mar. 2006. (in Portuguese).
- [12] E. Y. Nakagawa, A. S. Simão, and J. C. Maldonado. Addressing separation of concerns in software engineering environments. In *IASTED International Conference on Software Engineering, 25th IASTED International Multi-Conference on Applied Informatics*, Innsbruck, Austria, Feb. 2007.
- [13] OMG. Unified modeling language. [*On-line*], *World Wide Web*, 2005. Available in <http://www.uml.org/> (Access in 01/17/2007).
- [14] D. Stein, S. Hanenberg, and R. Unland. A UML-based aspect-oriented design notation for AspectJ. In *Proc. of the 1st International Conference on Aspect-Oriented Software Development*, Enschede, The Netherlands, 2002.

² <http://java.sun.com/>

³ <http://www.eclipse.org/aspectj/>