

# Aspect Oriented Software Architecture: a Structural Perspective<sup>1</sup>

A. Navasa, M.A. Pérez, J.M. Murillo, J. Hernández

Computer Science Department. University of Extremadura  
Escuela Politécnica. Avda. de la Universidad s/n 10071 Cáceres. Spain  
{amparonm, toledano, juanmamu, juanher}@unex.es

**Abstract.** The positive results obtained by researchers on aspect-oriented programming during the last few years are promoting the aim to export their ideas to the whole software development process. One of the stages in which Aspect Oriented Software Development (AOSD) techniques can be introduced is software architectural design. This would make design of complex systems an easier task whilst cost development, cost maintenance, reuse, etc., would be improved. However, integrating both architectural design and aspect orientation is a non-trivial task. The different nature of the aspects that can be involved in an application and the different requirements that applications impose on the treatment of aspects make it difficult to handle aspects in a uniform way while at the same time preserving the simplicity of the design process. In this study the structure of the problem of separation of crosscutting concerns in the architectural design based on the aspect oriented approach is analysed. The analysis will identify the kind of changes that must be made in the current technology to manage this sort of integration. In particular, this paper focuses on how aspects separation can be handled by Architecture Description Languages and architectural styles. The aim of this article is not to propose a particular solution but to propose some general guidelines on which solutions can be based.

## 1. Introduction

During the last few years, a great amount of work has been done to propose AOP techniques [Kic+96]. Time to harvest results has arrived and the results have shown the real benefits of using aspect orientation in the application development process. These include: increase in productivity, re-usability and adaptability. Thus, Aspect Orientation has been shown to be so relevant that there is an increasing demand for it to be extended to the whole software development process [Cla+99, Nak99, ChLu01]. So, concepts from AOP are being extended to early stages in the software lifecycle, generating new disciplines like Aspect Oriented Software Development (AOSD) and Aspect Oriented Design (AOD).

One of the stages on which AOD is focused is the architectural design of applications [Gru00]. It seems reasonable to observe software architecture from the aspect-oriented point of view due to several reasons. On the one hand, there are aspects that are inherent to the systems themselves and, consequently, they could be treated in their architectural design. On the other hand, aspects separation in the systems architecture would make design an easier task, improving cost development, making it easy to reuse designs, reducing maintenance cost and so on. However, aspect orientation and architectural design are disciplines that have evolved separately and their integration is not as trivial as it may seem.

In this paper some methodological considerations about how to face the integration of the aspects separation in the architectural design are presented. Thus, this work examines the separation of crosscutting concerns during software architecture design from a structural point

---

<sup>1</sup> This work has been supported by CICYT, project TIC 99-1083-C2-02

of view instead of a morphological one by studying the structure of the problem and extracting general conclusions more than proposing a particular solution. This reasoning will conclude that aspects separation at architectural level have some similarities with typical coordination problems solved using coordination models and languages [Car92]. Leaning on this, the kind of changes that must be introduced in current Architecture Description Languages to manage the aspect separation when designing software systems is discussed.

In section 2, the difficulties that appear when integrating aspects separation in the current state of software architecture will be shown. Next, in section 3, the process of how an application can be designed by handling separately the non-functional aspects intervening on it will be analysed. This analysis will show how separations of crosscutting concerns at architectural design can be handled as a coordination problem. Finally, section 4 shows the conclusions and outlines future works.

## **2. Software architecture and Aspect Orientation**

During the last few years, software architecture design techniques have been acquiring more and more relevance as the complexity of systems has been growing. The experience has demonstrated the benefits of using these techniques when designing such systems. Software architecture allows the software designer to specify the systems structure in terms of components and connectors. Components specify the functionality of the system whilst connectors determine the interaction between components. In these terms, software architects can concentrate on the structural properties of the systems avoiding the implementation details. This makes possible to face complex systems reducing cost and developing time.

Two of the most useful tools for software architects are both architectural styles and Architectural Description Languages (ADL). Architectural styles give rules to build system families with similar characteristics. A number of architectural styles that direct the work of the software architects has been catalogued [ShGa96]. ADL are languages that provide primitives to specify components and connectors. Several ADL have been developed with different features: Rapide [Luc95], Darwin [MaKr96], Wrigh [All97], Acme [GaMoWi97], etc..

In order to introduce the concepts from aspect orientation in the architectural design of applications, it would be necessary to adapt the current software architect s tools. However, some previous works of our research group [NaPeMu01, PeNaMu01, NaPeMu02] has revealed that this is a non-trivial task. With respect to the architectural styles, one may consider proposing an architectural style for systems in which aspects will be handled in a separated way. Nonetheless, our experience tells us that it is not easy for the following two reasons:

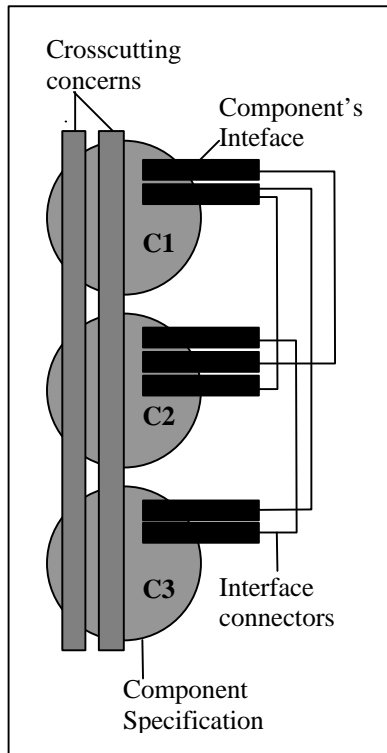
1. On the one hand, the different nature of the aspects intervening in a system makes it difficult to handle them in a simple and uniform way using a single style.
2. On the other hand, the same aspects in different systems could require different treatment and this makes it difficult to solve the problem in a simple way. For example, an application with real time and distribution constraints may require treating the real time constraints before, and then the distribution constraints. A second application with the same real time and distribution constraints may require treating the aspects in the opposite order. Therefore, it would be difficult again to use a single architectural style for the two applications.

With respect to ADL, current languages as the ones mentioned before do not provide primitives to specify the aspects separation. ADL are designed to specify components and the connections between their interfaces. Now, it would be necessary to specify connections not only between components but also between components and aspects and aspects with others aspects. However the idea of having interfaces on the aspects is not clear. It is not even clear how to specify aspects.

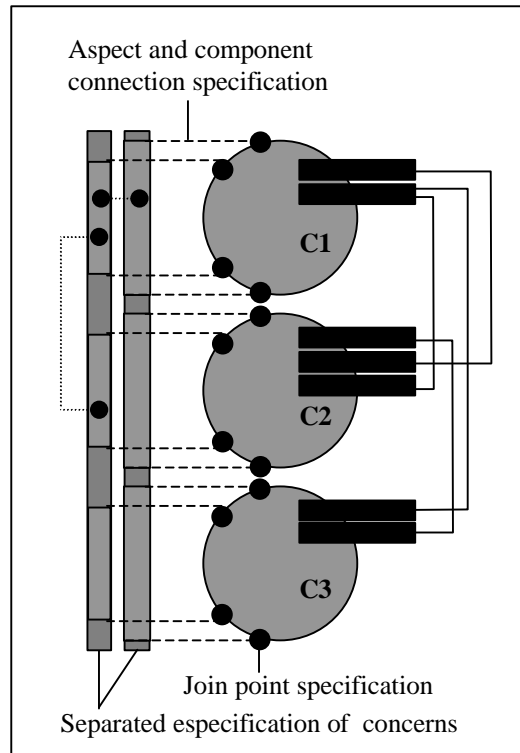
All the above allows us to conclude that new mechanisms are required to introduce aspects separation during architectural design. The next question is, what must be the nature of such a mechanism?. To answer this question a structural point of view was adopted.

### 3. Aspect Separation at the Architectural Level: a Coordination Problem

In this section it is analyzed how separation of crosscutting concerns can be handled in architectural design of software systems. The analysis will allow us to extract some general conclusions. The structure of the faced problem is depicted in figures 1a and 1b



**Fig 1a.** Current specification of software architecture



**Fig 1b.** Separated specification of crosscutting concerns at the software architecture level

Let us suppose that figures 1a and 1b specify the software architecture of the same system. Figure 1a shows how software architectures are currently specified. The software architects have to specify the functionality of the system using components and the interaction between components using connectors. However the specification of the concerns intervening in the system is crosscutting the specification of components. On the other hand, introducing concepts from aspect orientation in architectural design must lead to a situation as the one described in figure 1b.

In figure 1b the specification of aspects has been separated. Now the software architect has to specify components with their interfaces, connections between interfaces and the separated aspects. However, in order to produce an equivalent design to the one showed in figure 1a software architects must carry out the following tasks:

1. To specify in components the points from which the specification of the aspects has been extracted. That is, the software architect have to specify the **join points** in the specification of components.
2. To specify the connections between aspects and join points. Such connections have a different nature that connection between interfaces. The purpose of such connectors is to sort

out the specifications of the whole system in order to maintain the coherence of the original system preserving its semantic. So, such connectors have to specify when and how each aspect must be treated. This is a typical problem of coordination that has been solved with coordination models and languages [Gpap98].

Having all the above in mind it can be concluded that to manage the separation of concerns at architectural design of software systems the current ADL must be extended in order to provide the following functionality:

1. To specify functional components with interfaces and connection between interfaces. This functionality is already provided by current ADL. However, new ADL should make possible to specify join points in functional components. So new primitives must be provided in order to manage the specification of the joint points. These primitives should support the specification of every kinds of joint points not restricted to a predefined set of them.
2. To specify aspects. Aspects have a different morphology that functional components in the sense that they do not provide services and interfaces. However, in order to modularize aspects, they would be specified as a special kind of component described by new primitives.
3. Finally, to specify connectors between join points and aspects. As it has been introduced before the purpose of such connectors is to sort out the specification of the whole system. This problem is already solved by coordination models and languages. So it must be studied the possibility to import solutions from this area. In particular, exogenous coordination models [Arb98, Frø96, Mur99 to name a few] specify the code to determine how functional components coordinate in separated entities from those to be coordinated. Different approaches support different kinds of functional and coordination components. However, in all proposals the coordination components sort out the global execution of the tasks implemented by the system. The same schema could be used here to specify connectors between aspects and components. In this case there are two different kinds of components: functional and aspect components, and connectors could be coordination components. The mission of such components would be to specify when and how the aspect components must be treated.

The sort of aspect components and connectors mentioned before could be reused from one system specification to another. Patterns of usual aspects and connectors could be deduced and, in this way, aspect-oriented architecture styles could be proposed.

#### 4. Conclusions and future works

In this paper some guidelines to integrate concepts from aspect orientation in architectural design of software systems have been presented. In particular the work is focused on the way in which aspect separation can be handled by means of architectural styles and Architecture Description Languages. Based on the previous experience of our research group it is concluded that it is very difficult to deal with such integration preserving the simplicity during the design process. Then the structure of the problem of separation of concerns at the architectural level has been analysed. This analysis has allowed us to extract two important conclusions. First, aspect separation at the architectural level can be reduced to a coordination problem. Second, based on the above, a proposal has been put forward regarding the kind of changes that must be made in the current tools for architectural design in order to support the separation of concerns.

Currently we are working along different lines. In particular, in order to be able to specify the places where aspects must be treated, the way in which joint points can be characterised in components is being analysed. Also, to specify the order and the treatment that aspects must receive, we are studying the nature of the connectors. To specify this sort of connectors we are studying the possibility of using **Rew** [Arb02] from Farhad Arbab. Farhad Arbab has a recognised experience as a researcher in the area of coordination models and languages. He is

the father of the exogenous coordination model IWIM. **Rew** is a channel-based exogenous coordination model wherein complex coordinators, called "connectors" are compositionally built out of simpler ones. These features make connectors a simple and powerful mechanism suitable to be used here. Finally, work is being done to determine the way in which the specification of aspects can be reused from one system's architecture to another. In particular it is being studied how the trading of aspect components could be done.

## Acknowledgements

We would like to thank the anonymous referees for their helpful comments.

## References

- [AlChNo01] J. Aldrich, C. Chambers, D. Notkin. ArchJava: Connecting Software to Implementation. Workshop on Language Mechanism for Software Component in OOPSLA 2001 Octubre 2001 Tampa Bay, Florida, USA
- [All97] R. Allen. *A Formal Approach to Software Architecture*. Tesis Doctoral. School of Computer Science. Carnegie Mellon University, USA CMU-CS-97-144. 1997.
- [AlDoGa98] R. Allen, R. Douence, D. Garlan. CMU / JRJSA-JNRJA /CMU. Proceeding of 1998 Conference on Fundamental Approaches to Software Engineering. Lisbon, Portugal Marzo 1998
- [AkTe98] M. Aksit, B. Tekinerdogan. *Solving the Modeling Problems of Object-Oriented Languages by Composing Multiple Aspects Using Composition Filters*. Workshop AOP. Bruselas. 1998.
- [Arb98] Farhad Arbab. What Do You Mean Coordination? Bulletin of the Dutch Association for Theoretical Computer Science (NVTI). March 1998.
- [Arb02] Farhad Arbab and Farhad Mavaddat. *Coordination Through Channel Composition*. F. Arbab and C. Talcott (Eds.). 5th International Conference, COORDINATION 2002, YORK, UK. LNCS 2315, Springer-Verlag. April 8-11, 2002.
- [Car92] N. Carreiro, D. Gelernter. Coordination languages and their significance. *Communications of the ACM*, 35 (2):97-107, February 1992
- [ChLu01] Christina von Flach G. Chávez and Carlos J. P. De Lucena. Design-level Support for Aspect Oriented Software Development. Doctoral Symposium OOPSLA 2001 Octubre 2001 Tampa Bay, Florida, USA
- [Cla+99] Siobhán Clarke, William Harrison, Harold Ossher, Peri Tarr. Separating Concerns Throughout the Development Lifecycle. ECOOP'99. Workshop on Aspect-Oriented Programming. Lisbon. Portugal. 1999.
- [Frø96] S. Frølund. Coordinating Distributed Objects. An Actor-Based Approach to Synchronization. *The MIT Press*. 1996.
- [GaMoWi97] D. Garlan, R.T. Monroe, D. Wie. *Acme: An Architecture Description Interchange Language*. In proceeding of CASCON'97, Ontario. Canada. 1997.
- [Gpap98] G.A. Papadopoulos, F. Arbab. Coordination Models and Languages. *Advances in Computers*, 48. Academic-Press, 1998.
- [Gru00] J. Grundy. A Method and Support Environment for Distributed Software Component Engineering. Proceeding of 2000 Conference on Software- Methods and Tools. Wollongong, Australia 2000
- [Ki+96] G. Kiczales et al. *Aspect-Oriented Programming*. In Max Mühlhäuser ed, Special Issues in Object-Oriented Programming, Workshop Reader of the 10th. European Conference on Object-Oriented Programming, ECOOP 96, Dpunkt-Verlag. 1997.
- [Ki+01] G. Kiczales, E. Hilsdale, J. Hugunim, M. Kersten, J. Palm, W. G. Griswold. Getting Started with AspectJ. *Communications of the ACM*, October 2001, vol 44, num 10.
- [LiLoMi00] K. Lieberherr, D. Lorenz, M. Mezini. *Programming with Aspectual Component*. Northeastern University, EEUU. 2000.
- [Luc95] D.C. Luckman et al. *Specification and Analysis of Systems Architecture using Rapide*. IEEE Transactions on software Engineering. San Francisco. EEUU. 1995.

- [MaKr96] J. Magee, J. Kramer. *Dynamic Structure in Software Architectures*, en ACM Foundations of Software Engineering. San Francisco. EEUU. 1996.
- [Mur99] J.M. Murillo, J. Hernández, F. Sánchez, L.A. Álvarez. Coordinated Roles: Promoting Reusability of Coordinated Active Objects Using Events Notification Protocols. P. Ciancarini and A. L. Wolf (Eds.). *Third International Conference COORDINATION 99*. Amsterdam, The Netherlands. Springer-Verlag, LNCS 1594. April 1999.
- [Nak99] Shin Nakajima. Separation of Concerns in Early Stage of Framework Development. Workshop on Multidimensional Separation of Concerns. OOPSLA 2001 Octubre 2001 Tampa Bay, Florida, USA
- [NaPeMu01] A. Navasa, M.A. Perez, J.M. Murillo, Developing Components Based Systems using AOP Concepts. Workshop on Advanced Separation of Concerns in ECOOP 2001. Budapest Hungría
- [NaPeMu02] A. Navasa, M. A. Pérez, J.M. Murillo Definición de un estilo arquitectónico para desarrollos software de sistemas complejos, basado en separación de aspectos. Cáceres, Informe Técnico de la Universidad TR-13/2002. Febrero 2002.
- [OsTa01] H. Ossher, P. Tarr. Using Multidimensional Separation of Concerns to (Re)Shape Evolving Software. Communications of the ACM, October 2001, vol 44, num 10.
- [PeNaMu01] M.A. Perez, A. Navasa, J.M. Murillo. An Architectural Style to Integrate Components and Aspects. Workshop on Feature Interaction in Complex Systems in ECOOP 2001. Budapest Hungary
- [ShGa96] M. Shaw, D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall. 1996.