Towards the architectural definition of the Health Watcher system with AO-ADL

Mónica Pinto, Nádia Gámez, Lidia Fuentes
Dpto. Lenguajes y Ciencias de la Computación, University of Málaga, Málaga, SPAIN
{pinto,nadia,lff}@lcc.uma.es (http://caosd.lcc.uma.es/)

Abstract

AO-ADL is an aspect-oriented architecture description language. The main contributions of AO-ADL are two. First contribution is the definition of a symmetric composition model, where functional and non-functional concerns are modeled by the same architectural block. Second contribution is the extension of the semantic of connectors with aspectual composition information. In this paper we describe the software architecture of the Health Watcher system using AO-ADL. We present a subset of the functional and extra-functional concerns identified for this system, as well as the compositions among the components modeling them. We specially focus on aspectual compositions to illustrate how the extensions that AO-ADL introduces into connectors provide appropriate support for representing separation of concerns at the architectural level.

1. Introduction

AO-ADL [8, 15] is a new XML-based architecture description language specifically well-suited for describing aspect-oriented architectures, solving the tangled and the scattered behavior problem earlier, at the architectural level. In this paper we use AO-ADL to describe an AO software architecture of the Health Watcher (HW) system [20].

The structural organization of AO-ADL is based on the fact that the main difference between crosscutting and non-crosscutting concerns is merely in the role they play in a particular composition binding and not in the internal behavior itself. Therefore, instead of inventing a new structural element to model aspects, as other aspect-oriented ADLs did [14, 16], components in AO-ADL model either crosscutting (named *aspectual component*) or non-crosscutting behavior (named *base component*) exhibiting a symmetric decomposition model. Thus, a component is considered an aspect when it participates in an aspectual interaction. This approach increases the reusability of components, which may play an aspectual or non-aspectual role depending on the interactions in which they participate.

Following a symmetric approach, the crosscutting nature of a component only depends on the connections with other components, which in ADLs are specified in connectors. Thus, another contribution of AO-ADL is the extension of the semantic of traditional connectors to represent the crosscutting effect of 'aspectual' components. This means that AO-ADL connectors provide support to describe different kinds of interactions among components – not only typical communication as in traditional ADLs, but also crosscutting influences among them. That is, AO-ADL connectors specify how 'aspectual' components are weaved with 'base' components during components' communication.

From the requirements of the HW system, we specify and model as separated components those functional and non-functional concerns that are scattered and tangled with other concerns. Then, we compose these 'aspectual' components with the rest of 'base' components in the system, using the aspectual roles and aspectual bindings provided by AO-ADL connectors.

Along the paper we show the advantages of the novel characteristics of AO-ADL using examples from the HW case study. Concretely, we show that: (1) With the symmetric decomposition model of AO-ADL a component can play an aspectual or non-aspectual role depending on the interactions in which that component participates (section 3.1); (2) AO-ADL explicitly and homogenously represents the dependencies of both 'aspectual' and 'base' components, by means of its provided and required interfaces (section 3.1); (3) The modularity of components is improved, solving the tangled and the scattered behavior problem earlier, at the architectural level (section 3.2); (4) In AO-ADL the pointcut specification is not part of the 'aspectual' component definition, but part of the connector aspectual binding specification (sections 3.3 and 4); (5) The extended specification of AO-ADL connectors helps to localize in the same architectural block all aspect-oriented information (section 4), and (6) The use of quantifications[1] help to model more generic and reusable connectors (section 4).

After this introduction the identified non-crosscutting and crosscutting concerns and their modeling in AO-ADL

[1] We understand quantifications as wildcards and binary operators

Early Aspects at ICSE: Workshop in Aspect-Oriented
Requirements Engineering and Architecture Design (EARLYASPECTS'07)
0-7695-2957-7/07 $20.00 © 2007 IEEE

are described in section 2 and 3 respectively. The composition of these concerns using AO-ADL connectors is specified in section 4. Then, section 5 describes the related work section and section 6 our conclusions and future work.

2. Identified concerns

This section describes the architecture of the Health Watcher (HW) system. The purpose of this system is to collect then manage public health related complaints and notifications [20]. Our primary aim is to show the advantages of modeling an aspect-oriented version of the HW system using AO-ADL. Thus we emphasize the separation of crosscutting concerns more than the detailed description of the components that constitute the architecture. These components are graphically represented in Fig. 1[2]. Since the symmetric decomposition model defined by AO-ADL makes no difference in the specification of 'base' and 'aspectual' components, all of them can be used either as base or as aspectual components, or even simultaneously playing both roles, depending on the kind of collaboration with other components (sections 3.1 and 4).

For the sake of simplicity, we consider only a high-level abstraction of the core behavior of the system, modeling three main 'base' components: User GUI, HW System and Data Store. Other aspect-oriented approaches describing the architecture of this system follows the same approach in order to focus on crosscutting behavior [2, 6].

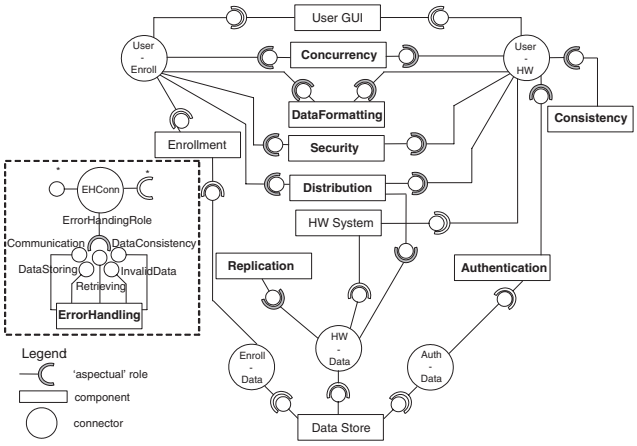

Figure 1. The HW System Architecture

In traditional ADLs components are the locus of computation and state [9], where the services they provide and require from other components in the environment are specified by means of their provided and required interfaces (i.e. their ports). As shown in Fig. 2, where the AO-ADL description of the User GUI component is provided, AO-ADL follows the same approach. The User GUI component represents the Graphical User Interface for the two kinds of users in the HW system, the citizens and the employees. In Fig. 2 we show (lines 1 to 6) two required interfaces of this component. Each interface has an associated *role name* that identifies the role provided or required by the component in their interactions with other components, and a *uri* that links such role name with the specification of the corresponding interface. The required interface ServicesHW (lines 12-18) requires methods to enable the citizen to make a query or complaint and to allow the employee doing some operations, such as update a complaint, update an employee and so on. On the other hand, the required interface Registration (lines 19-21) enables an employee to enroll other employees. Notice that a refined version of this architecture, out of the scope of this paper, may decompose this component in subcomponents such as citizen GUI and employee GUI, decomposing the above interfaces in citizen specific and employee specific interfaces. Other features of the AO-ADL language such as state attributes, semantic and composite components are out of the scope of this paper. A detailed description of AO-ADL can be found in [15].

```
1  <component name="UserGUI">
2     <required-interface role="ServicesHW"
3        uri="// interface[@name="ServicesHW"]/>
4     <required-interface  role="Registration"
5        uri="// interface[@name="Registration"]/>
6  </component>
7  <component  name="HWSystem">
8     <provided-interface role="Query&Complaint"
9        uri="// interface[@name="ServicesHW"]/>
10  …
11 </component >
```

```
12 <interface  name="ServicesHW">
13    <operation name="Query"/>
14    <operation name="Complaint">
15    <operation name="UpdateComplaint">
16    <operation name="UpdateEmployee">
17       …
18 </interface>
19 <interface  name="Registration">
20    <operation  name="enrollEmployee" >
21 </interface>
```

Figure 2. AO-ADL description of UserGUI

The HW System component (lines 7-11) is responsible for all business operations, such as register or update a complaint, and update a health unit. It has one provided interface to provide method to the User GUI and one required interface to save the information in the Data Store (see Fig. 1).

In the Data Store component all the data about complaints, employees, health units, specialities and so on is saved. This component has a provided interface to set and get information from the HW System, and two provided interfaces to provide services to the Enrollment 'base' component and to the Authentication 'aspectual' component. Notice that this means that the Authentication component (see Section 3.1), which plays the role of an 'aspectual' component in its interaction with the UserGUI component, is playing the role of a 'base' component in this interaction with the Data Store component. This duality is possible due to the symmetry of the decomposition model of AO-ADL.

[2] AO-ADL is a textual language. The graphical notation is used only to provide a clear understanding of the identified concerns

0-7695-2957-7/07 $20.00

The classical non-functional requirements have been modeled as 'aspectual' components: Concurrency, Security, Distribution, Authentication, and Replication. Apart from this, other crosscutting concerns have been identified: Enrollment, Error Handling, Consistency and DataFormatting.

3. Identified crosscutting concerns

In this section we describe the details of the identified crosscutting concerns. We have grouped the components that model them in different subsections, according to the contribution of AO-ADL that they help to illustrate. Notice that it is not the focus of this section showing their AO-ADL description, which is structurally identical to the description of the UserGUI component in Fig. 2. Instead, we focus on a description of these aspectual components that help to highlight the most important contributions of AO-ADL.

3.1. Authentication and Enrollment

As mentioned before, an important benefit of AO-ADL is that its symmetric decomposition model increases the possibility of reusing a component, which may play an aspectual or non-aspectual role depending on the particular interactions in which that component participates. That means that the same component can be (re)used either as a 'base' or as an 'aspectual' component in different software architecture specifications. Examples of this are the Authentication and the Enrollment components.

The Authentication component is responsible for intercepting the communication between a employee and the HW system and checking the user credentials when the operation requires it. Moreover, as mentioned before, in order to do the authentication, this component (playing now the role of a 'base' component) requires some methods from the DataStore component. This is another important contribution of the symmetric decomposition model of AO-ADL that explicitly represents the dependencies of architectural aspects (i.e. 'aspectual' components) with other components. This information is very useful for the study of the interactions among aspects [18]. In AO-ADL the 'aspectual' component specification explicitly shows all the dependencies that the 'aspectual' component has with other components by means of its provided and required interfaces. Moreover, AO-ADL provides an homogeneous way of representing such dependencies, since it does not make distinction between 'base' and ' aspectual' components.

On the other hand, the Enrollment component enables a employee to be registered. It provides methods to enable a employee to register other employees and requires methods to store the information about the new employee. Though enrollment has been identified as a crosscutting behavior, and has been modeled separately from the HW System component, in the architecture shown in Fig. 1 the Enrollment component has been modeled as a 'base' component. This is demanded by the system requirements that state that 'only the users with role of 'employee' can enrol other employees in the HW system'. However, in other applications, this behavior is often modeled as an 'aspectual' component. An example is a web application in which the user is enrolled when he/she tries to authenticate into the system and it is detected that it is not a registered user.

3.2. Consistency and DataFormatting

With AO-ADL the modularity of components is improved, for both 'base' and 'aspectual' components. This is achieved by modeling any kind of crosscutting concerns as separated components (i.e. 'aspectual' components), not only non-functional but also functional concerns. In this sense the tangled and the scattered behavior problem is solved earlier, at the architectural level.

Examples of these are the Consistency and the DataFormatting aspects shown in Fig. 1, which were not identified among the non-functional requirements of the system. Instead, they were scattered repeatedly among other functional requirements, and in consequence they have been encapsulated like 'aspectual' components. The Consistency 'aspectual' component is applied, during the interaction among the UserGUI and the HW System components, every time the HW system has to check the consistency of the received data. The join points affected by this aspectual component are shown in the composition section (section 4). In similar way, the DataFormatting component has to be applied every time the data have to be shown to the user.

3.3. Error Handling

Another advantage of AO-ADL is that the pointcut specification is not part of the 'aspectual' component definition. Instead, as shown in section 4, the pointcut definition is part of the specification of the connector.

An example of an 'aspectual' component that benefits from this approach is the ErrorHandling component. Error handling was not identified as a non-functional concern at the requirements level, though it is clearly tangled and scattered with the rest of concerns in the software architecture. Concretely, there are different exceptional situations in the HW system that require different solutions, such as communication problems, data storing and retrieving problems, invalid data problems and data consistency problems. All these problem categories can be represented as provided interfaces of the ErrorHandling component as shown in Fig. 1. The operations in these interfaces model the solutions to

these problems and will be attached to the core behavior during component composition (section 4)[3].

Moreover, these problems can occur in different interactions, such as 'when citizens query the HW system', 'when citizens register complaints with the HW system', 'when employees interact with the HW system either to login, or to register tables, or to update complaints, etc.'. These are the join points affected by the 'aspectual' component. However, the component behavior is independent of the context in which the problems occur and, in consequence, by including the definition of pointcuts as part of the component (as it is usually done in asymmetric approaches) the reusability of aspectual behavior is reduced. In AO-ADL this information is not represented as part of the component itself. Instead, it is specified inside the connector (see the example of the User-HW connector in Fig. 3), improving the reusability and evolution of 'aspectual' components. In AO-ADL only the execution context of the intercepted join points [7] is available to the 'aspectual' component. The context needs to be explicitly modeled as part of the provided and/or required interfaces of the 'base' components and it is represented by the parameters of the advice operation – i.e. the parameters of the operations specified in the provided interfaces of an 'aspectual component'.

Finally, notice that the error handling management is also an example of an aspectual behavior that can affect not only 'base' components but also other 'aspectual' components in the architecture. Concretely, since all the functionality related to 'data consistency' has been separated in the Consistency component, the 'data consistency problems' previously mentioned will be generated in the Consistency component, and not in the UserGUI or in the HealthWatcher components. Taking into account that the Consistency component has been identified as an 'aspectual' component, the symmetric decomposition model of AO-ADL helps to clearly represent both behaviors.

3.4. Replication, Security, Concurrency and Distribution

Other 'aspectual' components identified from the non-functional requirements of the HW system are Replication, Security, Concurrency and Distribution. In order to provide availability, the Replication component stores a copy of the data that the HW system sends to the data store. The Security component encapsulates a protocol to make secure the data sent over the Internet. The Concurrency component provides the ability to handle simultaneous users. Finally the Distribution component represents the remote location of the three principal components in the architecture (UserGUI, HW System and DataStore).

[3] When the component is composed as an 'aspectual' component, these operations play the role of 'advice' as it is understood in AOSD

4. Composition

Composition is the most relevant issue in AO-ADL. The architectural element for composition in AO-ADL is the connector, but extended with additional features to support interactions with 'aspectual' components. An example of a AO-ADL connector is shown in Fig. 3. The purpose of this connector is the specification of the connections between the Consistency component (which has been identified as a crosscutting concern in section 3.2) with the rest of components in the system.

AO-ADL connectors are structured in three main parts, which are shown in the specification of the User-HW connector in Fig. 3 and are detailed in the following subsections. These parts are: the specification of the connector's roles; the specification of the *component bindings* (interactions among 'base' components), and the specification of the *aspectual bindings* (interactions between 'aspectual' and 'base' components). The distinction between *component bindings* and *aspectual bindings* is the main difference of AO-ADL connectors with respect to traditional ones.

4.1. Role Specification

The User-HW connector in Fig. 3 has three different roles (lines 2 to 16): one provided role named UserQuery; one required role named HWuserQuery, and one aspectual role named Consistency. A novelty of AO-ADL is the definition of 'aspectual' roles. An 'aspectual' role (lines 12-16) indicates that the component attached to that role behaves as a crosscutting or 'aspectual' component.

Another important novelty of AO-ADL connectors is the possibility of using queries to specify the component(s) that may be connected to a particular role. This means that instead of always associating a particular interface to a connector role as described above, this role may be described by a query matching 'any component playing a particular role', or 'any component containing a set of particular operations in one of its provided interfaces', or 'any component having a particular attribute as part of its public state'.

This feature of AO-ADL is particularly useful for the specification of aspectual bindings (section 4.3). Since 'aspectual' components encapsulate crosscutting concerns, this means that the usual situation would be one in which the same 'aspectual' component affects different connections between 'base' components. However, without using queries to specify roles, it is not possible to avoid that the same aspectual binding rules appear replicated through different connectors. For instance, in Fig. 1, where we have not used queries to specify the roles of the User-Enroll and the User-HW connectors, the aspectual bindings associated to the DataFormatting, Security and Distribution 'aspectual' components is inevitably replicated in both con-

0-7695-2957-7/07 $20.00

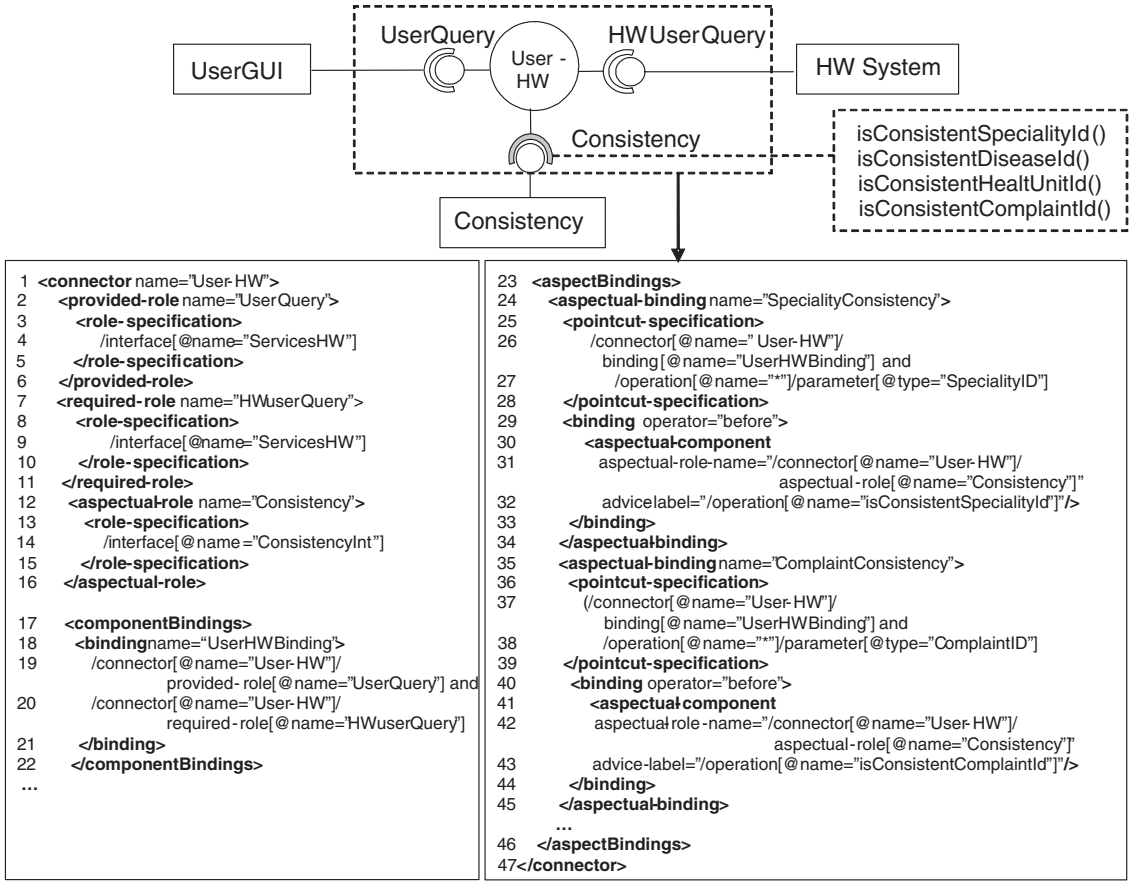

Figure 3. Part of the User-HW Connector: AO-ADL representation of composition

nectors. Contrarily, we used quantifications to specify the EH-Conn connector in Fig. 1. Concretely, the use of the "*" wildcard indicates that the ErrorHandling 'aspectual' component affects all the interactions in the HW system. More specific queries can also be specified. An example is '/component/provided-interface[@role="retrieve*"]' indicating that any component with a provided interface with a role name begining with "retrieve" can be attached to that connector's role.

Furthermore, the connectors' roles establish the scope of the join points that can be referenced by the connector's pointcuts (section 4.3). The idea is that we can easily represent different weaving scenarios with connectors by using queries to define their roles. For instance, at the architectural level typical join points would be: the 'reception of an operation defined in the provided interface of the component', where the source component is omitted from the pointcut specification; the 'sending of an operation defined in the required interface of a component', where the target component is omitted from the pointcut specification; the 'interaction among two components', where both the source and the target component are identified; the 'occurrence of a particular component's state or application's state', what is called a state-based join point, etc. The only requirement is that we need to expose, in the specification of the connector roles, all the information that we will then refer to during the specification of the pointcuts.

4.2. Component Bindings

As mentioned before, in the HW system, data consistency has to be checked for all the interactions between the UserGUI component (that will be attached to the UserQuery role of the connector) and the HW System component (that will be attached to the HWuserQuery role of the connector), in which there is a reference to the identifier of a health unit, complaint, disease, etc. The interaction among the 'base' components is specified in the component bindings section of the connector (lines 17 to 22), where the role UserQuery is connected to the role HWuserQuery. Note that we do not

discuss the kind of interactions or communication mechanisms among components that may be considered in AO-ADL, i.e. the type of connector (message-oriented infrastructures, pipes, filters, etc.). Interested readers are referred to [10, 19] for such a discussion.

4.3. Aspectual Bindings

Finally, the interaction of 'aspectual' components with 'base' components is specified in the aspectual binding section of the connector. The aspectual bindings are the main novelty of AO-ADL with respect to traditional ADLs. In this section, different aspectual bindings scenarios can be specified, each one including: (1) the description of the pointcuts identifying the join points that will be intercepted; (2) the kind of sequencing operator (before, after, around); (3) the list of 'aspectual' components to be composed for each sequencing operator; (4) the advice execution ordering, and (5) constraints restricting the injection of aspectual components under certain circumstances, which can be expressed as pre-conditions, post-conditions or invariants.

Concretely, as shown in Fig. 3, in the HW system the Consistency 'aspectual' component is attached to the Consistency role of the connector. This connector specifies one aspectual binding for each kind of data consistency problem that the 'aspectual' component may check. Only two of them are shown in Fig. 3. The first aspectual binding section, specified in lines 24 to 34, specifies that the isConsistenceSpecialityID operation (advice-label in line 32), of the provided interface of the component attached to the Consistency role of the connector, is attached before (binding operator in line 29) the interaction between the UserGUI and the HW System components takes place. Notice the use of wildcards ("*") in the definition of the pointcuts in lines 25 to 28 (... and /operation[@name="*"]/parameter[@type="SpecialityID"]) to specify that "the 'aspectual' component intercepts any operation of the interaction between the 'base' components that have a parameter of type SpecialityID".

Similarly, the other aspectual binding section is specified in lines 35 to 45 and describes that 'the isConsistenceComplaintID operation of the Consistency role is attached before the interaction between the UserGUI and the HW System components takes place; only if the operation has a parameter of type ComplaintID. The pointcut specification is defined in lines 36 to 39.

Notice that the definition of aspectual bindings inside connectors does not necessarily reduce the reusability of connectors in different contexts. The reason is that a connector can inherit the definition of roles, component bindings and aspectual bindings from other connectors by using XPath constructions. This issue has not been included in the paper due to the lack of space.

5. Related Work

In this paper we focus on comparing our approach with other aspect-oriented ADLs describing the HW system. We discuss two solutions, one specified in AspectualACME [6] and another one specified in aSideML [2].

In [6] the architecture of the system is described firstly using ACME. Then some crosscutting concerns are separated and the architecture is defined with AspectualACME [6], which extends ACME connectors to support crosscutting roles. Composition is modeled by defining base and crosscutting roles and configurations, being very similar to AO-ADL connectors. The main difference with respect to our approach is on the definition of pointcuts. While in AspectualACME pointcuts are defined as part of ACME's attachments, in AO-ADL pointcuts are defined inside connectors. Other difference is that, though both languages allow the use of quantifications in the specification of pointcuts, in AO-ADL the roles of connectors can also be specified using quantifications. As shown along the paper, this feature of AO-ADL improves the adaptability and reusability of connectors. Finally, in AspectualACME the aspectual connector is a special kind of connector and in AO-ADL the same connector is responsible of describing both the component and the aspectual bindings. This allows the specification of pointcuts in which the referenced join point is the binding between 'base' components (as specified in the components binding section of the connector).

Another work presents the architecture of the HW using aSideML [2], an aspect-oriented modeling language that provides notation, semantics and rules for specifying aspects at the design level. The main difference with our approach is that aSideML is asymmetric, making a distinction between the aspects and the components. Moreover, aspects only have crosscutting interfaces, but do not expose their dependencies with other components in the architecture. This means that the modeling of authentication in aSideML could not represent the relationship as a 'base' component of authentication with the data store component (section 3.1). Also, the asymmetric decomposition model does not allow the (re)use of aspects as components. Thus, the enrollment component may not be (re)used as an aspect in other software architectures (section 3.1).

Other related works are aspect-oriented ADLs such as DAOP-ADL [16, 17] (our previous work), Fractal [14], PRISMA [13] and Navasa's language [12]. They are not discussed in detail due to space limitations. A detailed comparison of them can be found in [3, 1].

6. Conclusions and Future Work

In this paper we have specified the software architecture of the HW system using AO-ADL. This language is

the aspect-oriented integrated ADL of the AOSD-Europe Network Of Excellence on AOSD. Concretely, in this project we are now collaborating on the definition of COMPASS [4], an approach that offers a systematic means to derive an aspect-oriented architecture, described in AO-ADL, from a given aspect-oriented requirements specification, described in RDL (Requirements Description Language) [5]. Also as part of the AOSD-Europe project we plan to use AO-ADL to describe a real system as part of an 'Architecture Design Method Case Study', as specified in [11].

Using examples from the case study we have discussed the main benefits of the symmetric decomposition model and the extension of AO-ADL connector with aspectual binding information. Using AO-ADL: (1) crosscutting behaviors (modeled by 'aspectual' components) can be separated and reused at the architectural level, and (2) compositions between 'aspectual' and 'base' components (modeled by 'aspectual bindings' inside connectors) are clearly represented at the architectural level.

As part of our ongoing work, we are defining a library of aspect-oriented architectural patterns. The idea behind connector templates is to avoid the definition of the connections between 'aspectual' and 'base' components from scratch. Instead, the software architect might use a set of aspect-oriented architectural patterns or templates for well-known crosscutting concerns. These templates are then instantiated in particular software architectures. We are mainly focusing on the definition of connector templates for modeling well-known crosscutting concerns. This will allow reusing these concerns in different software architectures.

Finally, the use of XML has important advantages, such as using the built-in XML tool support to define and manipulate architectural descriptions, or the possibility for a runtime execution environment of using this information during the application execution [17]. However, it also has an important drawback regarding the readability of the software architecture in the communication with stakeholders. In order to cope with this shortcoming we are now defining a tool suite that will provide support to define, manipulate and reuse AO-ADL specifications.

7. Acknowledgment

This work is supported by EC FP6 Grant AOSD-Europe: IST-2-004349, EC STREP Project AMPLE IST-033710, and Spanish Science and Technology grant: TIN2005-09405-C02-01.

References

[1] T. Batista and et. al. Reflections on architectural connection: Seven issues on aspects and adls. In *Early Aspect Workshop*, May 2006.

[2] C. Chavez and et. al. Crosscutting interfaces for aspect-oriented modeling. *Journal of the Brazilian Computer Society*, 12(1), June 2006.

[3] R. Chitchyan and et. al. Survey of AO analysis and design approaches. AOSD-Europe NoE Public Documents (AOSD-Europe-ULANC-9), 2005.

[4] R. Chitchyan, M. Pinto, A. Rashid, and L. Fuentes. COMPASS: Composition-centric mapping of aspectual requirements to architecture. *TAOSD: Special Issue on Early Aspects (accepted for publication)*, 2007.

[5] R. Chitchyan, A. Sampaio, P. Sawyer, and S. Khan. Initial version of aspect-oriented requirements engineering model. AOSD-Europe NoE Public Documents (AOSD-Europe-ULANC-17), 2006.

[6] A. Garcia and et. al. On the modular representation of architectural aspects. In *3rd. European Workshop on Software Architecture (EWSA'06)*, September 2006.

[7] G. Kiczales and et. al. An overview of AspectJ. In *15th ECOOP*, number 2072 in Lecture Notes in Computer Science, pages 327–355. Springer-Verlag, 18-22 June 2001.

[8] I. Krechetov, B. Tekinerdogan, M. Pinto, and L. Fuentes. Initial version of aspect-oriented architecture design. AOSD-Europe NoE Public Documents (AOSD-Europe-UT-D37), February 2006.

[9] N. Medvidovic and R. Taylor. A classification and comparison framework for software architecture description languages. *IEEE Transaction on Software Engineering*, 26(1):70 – 93, January 2000.

[10] N. Mehta, N. Medvidovic, and S. Phadke. Towards a taxonomy of software connectors. In *22nd ICSE'00*, pages 178–187, Ireland, 18-22 June 2000. ACM Press.

[11] R. Meunier and et. al. Study plan for empirical studies. AOSD-Europe NoE Public Documents (AOSD-Europe-Siemens-5), 2007.

[12] A. Navasa and et al. Aspect oriented software architecture: a structural perspective. In *Early Aspects Workshop at AOSD2002*, April 2002.

[13] J. Pérez and et. al. PRISMA:towards quality, aspect-oriented and dynamic software architectures. In *3rd IEEE Intl Conf. on Quality Software*, November 2003.

[14] N. Pessemier, L. Seinturier, and L. Duchien. Components, ADL and AOP: Towards a Common Approach. In *Workshop RAMSE'04 at ECOOP*, June 2004.

[15] M. Pinto and L. Fuentes. AO-ADL: An ADL for describing aspect-oriented architectures. In *Early Aspect Workshop at AOSD 2007*, 2007.

[16] M. Pinto, L. Fuentes, and J. M. Troya. DAOP-ADL: An Architecture Description Language for Dynamic Component and Aspect-Based Development. In *2nd International Conference on GPCE*, pages 118–137, Sept. 2003.

[17] M. Pinto, L. Fuentes, and J. M. Troya. A dynamic component and aspect-oriented platform. *The Computer Journal*, 48(4):401–420, March 2005.

[18] F. Sanen and et.al. Study on interaction issues. AOSD-Europe NoE Public Documents (AOSD-Europe-KUL-7), February 2006.

[19] M. Shaw, R. DeLine, and G. Zelesnik. Abstractions and implementations for architectural connections. In *ICCDS'96*, 1996.

[20] S. Soares, E. Laureano, and P. Borba. Implementing distribution and persistence aspects with AspectJ. In ACM Press, editor, *17th ACM conference OOPSLA'02*, pages 174 – 190, November 2002.