

# Aspectual Connectors: Supporting the Seamless Integration of Aspects and ADLs

Thaís Batista<sup>1</sup>, Christina Chavez<sup>2</sup>, Alessandro Garcia<sup>3</sup>,  
Uirá Kulesza<sup>4</sup>, Cláudio Sant'Anna<sup>4</sup>, Carlos Lucena<sup>4</sup>

<sup>1</sup>Computer Science Department, UFRN - Brazil

<sup>2</sup>Computer Science Department, UFBA - Brazil

<sup>3</sup>Computing Department, Lancaster University, United Kingdom

<sup>4</sup>Computer Science Department, PUC-Rio - Brazil

thais@ufrnet.br, flach@ufba.br, garciaa@comp.lancs.ac.uk

{uira,claudios,lucena}@inf.puc-rio.br

**Abstract.** *With the emergence of Aspect-Oriented Software Development (AOSD), there is a need to understand the adequacy of Architecture Description Languages (ADL) connection abstractions to capture the crosscutting nature of some architectural concerns. In this paper, we present Aspectual Connector (AC), a special kind of architectural connector, as the only necessary enhancement to an ADL in order to support a seamless integration of AOSD and Software Architecture. We also present AspectualACME, an extension to ACME that incorporates ACs and additional facilities to modularize architectural crosscutting concerns. We use a Web-based information system as the main case study.*

## 1. Introduction

Aspect-Oriented Software Development (AOSD) (Filman et al 2005) aims to provide systematic support for the identification, modularization, and composition of crosscutting concerns throughout the software lifecycle. At the architecture design level, a crosscutting concern could be any concern that cannot be effectively modularized using the given abstractions of an Architecture Description Language (ADL) (Shaw and Garlan 1996), leading to increased maintenance overhead, reduced reuse capability and generally resulting in architectural erosion over the lifetime of a system. Since the emergence of the Software Architecture discipline, the main focus of ADLs has been on the conception of architectural connection abstractions, such as interfaces, connectors, and configurations (Shaw and Garlan 1996). Hence, there is a pressing need for understanding to what extent they are able to capture the crosscutting interaction of certain architectural components. Ideally, ADL designers should promote a natural blending of conventional architecture abstractions and aspects.

Some Aspect-Oriented Architecture Description Languages (AO ADLs) (Navasa et al., 2002) (Pérez et al., 2003) (Pessemier et al., 2004) (Pinto et al., 2005) have been proposed, either as extensions of existing ADLs or developed from scratch employing AO abstractions commonly adopted in programming frameworks and languages, such as aspects, joinpoints, pointcuts, advice, and inter-type declarations. Though these AO ADLs provide interesting first contributions and viewpoints in the

field, there is little consensus on how AOSD and ADLs should be integrated, especially with respect to the interplay of aspects and architectural connection abstractions. The main problem is that existing proposals typically provide heavyweight solutions (Batista et al 2006), thereby making it difficult their adoption and the exploitation of the available tools for supporting ADLs.

In (Batista et al 2006) we have presented a reflection around seven issues that arise in this context. We have discussed how and why extensions are required or not to traditional notions of interconnection ADL elements, such as interfaces, connectors, and architectural configurations. Our conclusion was that Software Architecture promotes the principle of separation of concerns by explicitly separating connections from components. Hence a systematic integration of architectural abstractions and AOSD would even enhance the degree to what certain crosscutting concerns are modularly represented at the architectural level. The idea is to use the same abstractions used in the conventional ADL description, with minor adaptations to support effective modeling of crosscutting concerns without introducing additional complexity into the architecture specification.

This work presents *Aspectual Connector* (AC) as the only necessary enhancement to an ADL that supports the seamless integration of AOSD into Software Architecture. The central point is the composition mechanism built around the notion of an AC that is an extension of the traditional connectors to describe the crosscutting interaction involving certain architectural components. In order to instantiate this concept in a well-known ADL, we incorporated the AC abstraction in ACME (Garlan et al 1997). Therefore, we also present AspectualACME, an ADL that supports the seamless exploitation of AOSD composition mechanisms in architecture design. We evaluate and illustrate the AspectualACME concepts with a Web-based information system that exhibits some traditional crosscutting concerns in its architecture description, such as persistence and distribution. We also assess the simplicity and generality of our approach with respect to related work and according to an evaluation framework that is also proposed in this paper.

The remainder of this paper is organized as follows. Section 2 provides a background concepts involving AOSD and software architecture. Section 3 presents an evaluation framework for AO ADLs. This framework presents seven important issues related to aspects and architectural connection. Section 4 presents the notion of aspectual connectors. Section 5 illustrates how to incorporate ACs into ACME. Section 6 compares our proposal with related work. Finally, Section 7 presents the final remarks.

## **2. ADLs and Aspect-Oriented Software Development**

This section revisits the central concepts of the two techniques of interest in this paper that promote the principle of separation of concerns (SoC): Architecture Description Languages (Sections 2.1 and 2.2) and Aspect-Oriented Software Development (Section 2.4). It also motivates the investigation of how to integrate both techniques through an example (Section 2.3) that illustrates the manifestation of crosscutting concerns in ADL representations.

## 2.1 Architecture Description Languages

Architectural concerns are typically expressed by using abstractions supported by Architecture Description Languages (ADLs). According to a well-known conceptual framework (Medvidovic and Taylor, 2000), the building blocks of an architectural description are components, connectors, and architectural configurations. In fact, ADLs enforce the SoC principle by explicitly distinguishing architectural elements used to specify computation (components) from those used to express interconnection (connectors) between components. *Components* are the units of computing, while *connectors* are the locus of interaction. Components and connectors may have associated interfaces, types, semantics and constraints, but only explicit component interfaces are a required feature for ADLs. A *component's interface* is a set of interaction points between it and the external world. It specifies the *services* (messages, operations, and variables) a component provides and also the services it requires of other components. Component types are templates that encapsulate functionality into reusable blocks and can be instantiated many times.

Connectors model interactions among components and specify rules that govern those interactions. Similarly, connector types are templates that encapsulate component communication, coordination, and mediation decisions. A *connector's interface* specifies the interaction points between the connector and the components and other connectors attached to it. It enables proper connectivity of components by exporting as its interface those services it expects of its attached components. *Configurations* define architectural structure and how components and connectors are connected.

## 2.2 ACME

ACME (Garlan 2000) supports the definition of: (i) architectural structure, that is, the organization of a system into its constituent parts, (ii) properties of interest, information about a system or its parts that allow one to reason abstractly about overall behavior, both functional and nonfunctional, and (iii) types and styles, defining classes and families of architecture. Architectural structure is described in ACME with components, connectors, systems, attachments, ports, roles, and representations. *Components* are potentially composite computational encapsulations that support multiple interfaces known as *ports*. *Ports* are bound to ports on other components using first-class intermediaries called *connectors* which support so-called *roles* that attach directly to ports. *Systems* are the abstractions that represent configurations of components and connectors. A system includes a set of components, a set of connectors, and a set of attachments that describe the topology of the system. *Attachments* define a set of port/role associations. *Representations* are alternative decompositions of a given element (component, connector, port or role) to describe it in greater detail. Thus, the representation may be seen as a more refined depiction of an element. For instance, ports may have a representation to encapsulate a large set of API calls as a single port. Inside the representation, a set of ports is used to represent individual API calls.

The other three parts of ACME involve more sophisticated architectural features. *Properties* of interest are  $\langle name, type, value \rangle$  triples that can be attached to any of the above ACME elements as annotations. Properties are a mechanism for annotating designs and design elements with detailed, generally non-structural, information. Architectural *styles* define sets of types of components, connectors,

properties, and sets of rules that specify how elements of those types may be legally composed in a reusable architectural domain. The ACME type system provides an additional dimension of flexibility by allowing type extensions via the *extended with* construct. The ACME fragment in Figure 1 illustrates the main ACME concepts.

### 2.3. Crosscutting Concerns in ADL Representations: An Example

The architectural elements outlined above are sufficient for defining the structure of software architecture as a graph of components and connectors. However, they do not provide the adequate means to capture some architectural crosscutting concerns. Figure 1 illustrates the example that we will use through this paper. The HealthWatcher (HW) system is a Web-based information system developed by the Software Productivity research group from the Federal University of Pernambuco (Soares et al 2002). It supports the registration of complaints to the health public system. It is composed of the three main architectural components: (i) the *GUI* (*Graphical User Interface*) component provides a web interface for the system, (ii) the *Business* component defines the business rules, and (iii) the *Data* component stores the information manipulated by the system. Figure 1 depicts the ACME textual and graphical descriptions of this example.

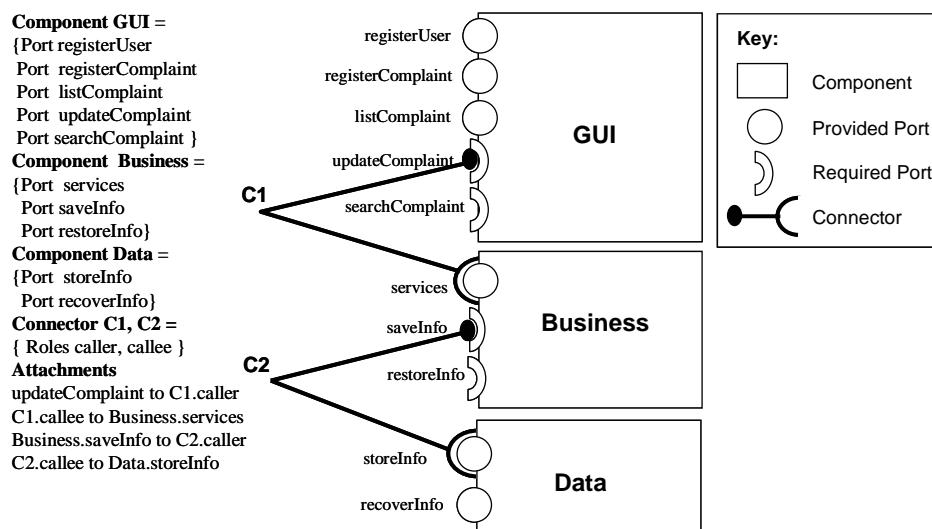


Figure 1. ACME Description of the HealthWatcher System

However, some architectural concerns cannot be modularly captured with traditional abstractions supported by ADLs, such as ACME. Some concerns are *crosscutting* even at the architectural design level, since they cannot be easily localized and specified with individual architectural units such as traditional interfaces, components, connectors, and configurations. Similar to the notion of aspect at the programming level (Kiczales et al, 1997), we say that these concerns crosscut the architectural units and denote the so-called *architectural aspects* (Araújo et al, 2005)(Baniassad et al, 2006)(Chitchyan et al, 2005)(Cuesta, 2005)(Krechetov et al, 2006).

Three crosscutting concerns affect the components of the HW system: (i) Persistence – supports issues related to the data management in web-based systems (transaction management, data update, repository configuration); (ii) Distribution – supports the distribution of the Business component services; (iii) Concurrency –

specifies mechanisms to apply different concurrency strategies to the functional components. The problem is that very often the crosscutting property of these architectural concerns (Figure 1) remains either implicit or is described in informal ways leading to reduced uniformity, impeding traceability and hindering detailed design and implementation decisions.

## 2.4 Aspect-Oriented Software Development

Aspect-Oriented Software Development (AOSD) (Filmann et al, 2005) provides new abstractions and *composition* mechanisms to support the explicit representation of aspects through software development stages, such as the software architecture design. The use of such new composition mechanisms allows for the encapsulation of crosscutting concerns into separated modular units, which are composed with other system modules at well-defined *join points*. Hence AOSD supports the *quantification* of structures and behaviors relative to a concern, which otherwise would be tangled and scattered through the representation of other concerns in software artifacts. Structural and behavioral enhancements can be typically applied *before*, *after* and *around* certain join points.

## 3. A Framework for Evaluation of Aspect-Oriented ADLs

This section presents a conceptual framework that subsumes a set of core issues that need to be considered while dealing with architectural aspects. Our goal is to use such a conceptual framework to support the systematic evaluation of existing aspect-oriented (AO) ADLs with respect to their proposed abstractions and extensions on the top of existing non-AO ADLs. The proposed framework is a result of a conceptual blending involving an AOSD glossary (van den Berg et al, 2005) and the well-known abstractions of ADLs (Section 2.1). We have aggregated and filtered the terms found in those sources, for marking out the most vital issues for the design of AO. In order to identify the more relevant architectural concepts, such aggregation and filtering processes were based on a widely-recognized terminology for software architecture descriptions (Medvidovic and Taylor, 2000).

The conceptual framework was also derived from our extensive experience on: (i) the design of aspect-oriented software architectures in different application domains (Garcia et al, 2004)(Kulesza et al, 2004)(Garcia et al, 2006)(Kulesza et al, 2006)(Kulesza et al, 2006b), (ii) the development of modeling approaches to handle different categories of crosscutting concerns at the architectural stage (Chavez et al, 2006)(Garcia et al, 2006a)(Krechetov et al, 2006)(Kulesza et al, 2004), and (iii) analysis of the suitability of existing ADLs to support architectural aspects (Chitchyan et al, 2005)(Batista et al, 2006).

As a result, our comparison framework is composed of seven main elements, which are described in Table 1. The first column lists the framework issues, while the second column defines the purpose of the respective issue and describes potential choices in the design of an AO ADL. The first issue is dedicated to understanding which architectural elements (e.g. components and interfaces) in an architectural description are typically affected by a crosscutting concern. The following six issues correlate AOSD concepts with conventional abstractions of ADLs (Section 2.1). For example, the fourth issue is related to how to specify interfaces of aspects. The last

issue is particularly concerned with the need of a new abstraction for aspects at the architectural level. We recommend that the interested readers explore the details of our extensive discussion on the issues that inspired the conception of our evaluation framework (Batista et al, 2006).

<b>Architectural Issue</b>	<b>Description</b>
Base Elements	An AO ADL must define which architectural building blocks may be affected by aspects. As discussed in Section 2, the main architectural building blocks are components, connectors, and interfaces. Hence, the design of an AO ADL can consider to define a subset or all of them as base elements.
Aspectual Composition	The ADL must support the composition involving base elements and aspects. The issues here are whether and where the aspectual composition should be defined.
Quantification	Quantification is the process of referring to affected structural and behavioral elements in a way that avoids the need to explicitly specify each architectural join point of interest. An AO ADL can support or not quantification mechanisms and, if so, decide for where and how quantification should be specified.
Aspect Interfaces	The ADL should allow the explicit description of aspect interfaces. The issue is whether the traditional notion of architectural interfaces should be changed or not to express the boundaries of aspects.
Join point Exposition	An AO ADL must support join point exposition. Architectural join points are the instances of base elements in an ADL-based specification that can be affected by a certain aspect. The issue is whether the base elements should have a different interface exposing the join points to the aspectual components.
Interface Enhancements	Interface enhancements are the ability of refining the component interfaces with new elements, such as services and attributes. An AO ADL needs to decide for supporting or not interface enhancements.
Aspect	An AO ADL must support the description of aspects. The issue is whether it should provide or not a new architectural abstraction for describing them.

**Table 1. An Evaluation Framework for Aspect-Oriented ADLs**

In a previous work (Batista et al, 2006), we have used our conceptual framework to evaluate several AO and non-AO ADLs. For each issue of the framework, we analyzed how the different ADLs offer support to address it. As a result of our analysis, we have concluded that no extra architectural abstractions are needed to represent aspects. We have proposed the extension of the connector and configuration architectural abstractions to support the modeling of the composition mechanism used in the crosscutting concern representation at the architectural level. These extensions are related with the need to support new ways of composition, as well as the quantification property supported by a number of AO approaches. Next section describes our proposal of aspectual connectors which is the core of our extension proposal.

#### **4. Aspectual Connectors**

As already stated, software architecture descriptions rely on a *connector* to express the interactions between components. This section discusses why crosscutting interactions (Section 4.1) involving architectural components can be localized through the use of an extended notion of traditional connectors, called *Aspectual Connectors* (Section 4.2).

From herein, we use the term *aspectual component* to refer to a component that implements a crosscutting concern (architectural aspect).

#### 4.1. Modularizing Crosscutting Interactions in ADL Representations

A connector is a fundamental building block to model simple or complex interaction protocols as discussed in the taxonomy of connectors (Mehta et al 2000). In addition, since ADLs (Section 2.1) explicitly distinguish components (units of computing) from connectors (units of interaction), this SoC approach should also play a key role in the integration of ADLs and AOSD. First of all, the component abstraction should be enough to model any kind of architectural concern independently from its crosscutting interaction with other components. In fact, a central goal of architecture specifications is to come up with a unifying abstraction – the component – to capture different types of computing units defined in specific architectural styles (Medvidovic and Taylor, 2000), such as objects, layers, meta-objects, and aspects. The key distinction between aspectual and regular components is in the way aspects compose with the rest of the system – the scope of the composition is broad and affects multiple components or multiple architectural elements.

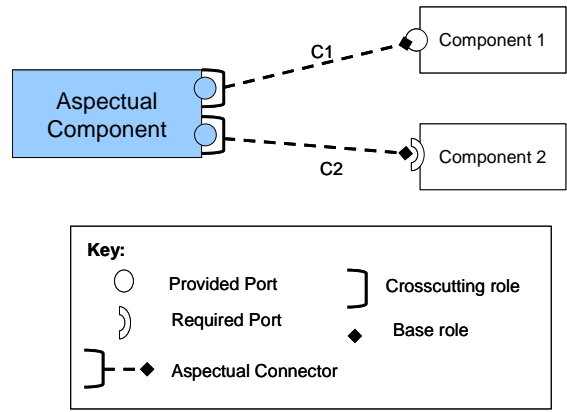
Second, as connectors are widely used for different interconnection purposes, they are enough to model the interaction between traditional components and components that represent a crosscutting concern. However, the way that an aspectual component composes with a regular component is slightly different from the composition between traditional components. A crosscutting concern is represented by a provided service of an aspectual component and it can affect provided or required services of other components. As in ADLs valid configurations are those that connect provided and required services, it is impossible to represent a connection between a provided service of an aspectual component and a provided service without extensions to the traditional notion of architectural connections.

#### 4.2. The Structure of Aspectual Connectors

In order to address the issues mentioned in Section 4.1, we propose an innovative abstraction, called *Aspectual Connector (AC)*, which is a **regular** connector with a new interface. The purpose of such a new interface is twofold: to make a distinction between the elements playing different roles in a crosscutting interaction – i.e. affected base components and aspectual components; and to capture the way both categories of components are interconnected. The AC interface contains: (i) a *base role*, (ii) a *crosscutting role*, and (iii) a *glue* clause. Figure 2 depicts a high-level view of the composition between an aspectual component and two components. C1 and C2 are examples of aspectual connectors. Note that we do not have a distinct abstraction to represent architectural aspects, which are similarly represented as regular components; the different colors in Figure 2 are only to emphasize which one is playing the role of aspectual component in the crosscutting collaborations.

The base role is specified to be connected to a port of the regular component and the crosscutting role is specified to be connected to a port of an aspectual component. The pair base-crosscutting roles do not impose the constraint of connecting provided and required ports. A crosscutting role defines the place at which an aspectual component joins a connector. In Figure 2 the aspectual connector C1 connects a provided port of the aspectual component with a provided port of Component 1. C2

connects another provided port of the aspectual component with a required port of Component 2. The glue clause specifies the details about a connection such as the place where the connector joins the component – after, before, around, and others.



**Figure 2. Aspectual Composition**

### 4.3 Aspectual Composition

In ADLs, the connections between components and connectors are defined in the *configuration* section. The configuration description picks up architectural join points at which an aspectual component acts. The join points of interest are certain elements of the component interfaces, which are captured and associated with a base role of a specific AC. Thus, such elements of component interfaces are the collection of join points where the regular components and aspectual connector are combined. In fact, the concept of configuration already defines the point where a component joins a connector. Thus, we are just taking advantage of this concept to identify the join points affected by a crosscutting interaction. In order to specify several join points in a single statement, wildcards, and logical expressions can be used in the configuration part.

## 5. AspectualACME: An Aspect-Oriented ADL

This Section presents the description of AspectualACME, an extension of ACME with the goal of supporting a seamless integration of aspects and ADLs. Section 5.2 we evaluate AspectualACME according to the framework presented in Section 3.

### 5.1. Extending ACME

We address the integration of aspects and ADLs to conform to the issues discussed in Sections 3 and 4, by extending ACME to introduce aspectual connectors and quantification support at the configuration level. Additionally, AspectualACME is expected to support simplicity, expressiveness, and to provide a conservative extension so that software architects can foster reuse of ACME libraries and tools. We have selected ACME as our base ADL because it presents a relatively simple core set of concepts for defining system structure and it captures the essential elements of architectural modeling (Medvidovic and Taylor 2000). In addition, unlike most ADLs, ACME is not domain-specific and provides generic structures to describe a wide range of systems. It comes with tools that provide a good basis for designing and manipulating architectural descriptions and generating code. The complete BNF of AspectualACME is available at (AspectualACME, 2006).

### 5.1.1. ACME extension for aspectual connectors

The first extension that we propose is a specialization of ACME's connector abstraction. This extension allows the expression of aspectual connectors and their inner constructs: base roles, crosscutting roles, and the composition between them denoted by *glue*. We extend the connector interface in order to support the specification of base and crosscutting roles. The base role may be connected to the port of a component (provided or required) and the crosscutting role may be connected to a port of an aspectual component. The distinction between base and crosscutting roles addresses the constraint typically imposed by many ADLs about the valid configurations between provided and required ports. An aspectual connector must have at least one base role and one crosscutting role. Figure 3a and 3b present examples of a regular connector and an aspectual connector in ACME.

<pre>Connector aConnector = {   Role aRole1;   Role aRole2; }</pre>	<pre>Connector aConnector = {   Base Role aBaseRole;   Crosscutting Role aCrosscuttingRole;   Glue glueType; }</pre>
<b>(a) regular connector in ACME</b>	<b>(b) aspectual connector in AspectualACME</b>

Fig. 3. Regular and Aspectual Connectors

We also introduce a new construct - the *glue* clause - to specify details about the composition between components and aspectual components, such as the place where the port from an aspectual component will affect the regular component. There are three types of aspectual glue: *after*, *before*, and *around*. The semantics are similar to that of advice composition from AspectJ (AspectJ Team, 2006). For binary aspectual connectors (only one crosscutting role and one base role), the glue clause is simply a declaration of the glue type (Figure 3b), but whenever more than one base role and one crosscutting role are declared inside an aspectual connector, the glue clause must be more elaborated (Figure 4).

```
Connector aConnector = {  
  Base Role aBaseRole1, aBaseRole2;  
  Crosscutting Role aCrosscuttingRole1,  
                    aCrosscuttingRole2;  
  Glue { aCrosscuttingRole1 before aBaseRole1;  
        aCrosscuttingRole2 after aBaseRole2;  
  }  
}
```

Fig. 4. Glue Clause

### 5.1.2. ACME extension for quantification

The second extension addresses quantification to avoid the need to refer explicitly to each join point in an architectural description. Since the Attachments part is the place where structural join points are identified in ACME, we have decided for defining the quantification mechanism by extending the configuration part. It is also possible to use wildcards in order to denote names or part of names of components and their ports. The quantification must be used in the attachment of a base role with target component(s).

In Figure 5, the star symbol ('\*') is used to specify that `aConnector.aBaseRole` is bound to all components that offer a port with a name that begins with `prefix`.

```
System Example = {  
  Component aspectualComponent = { Port aPort }  
  Connector aConnector = {  
    baseRole aBaseRole;  
    crosscuttingRole aCrosscuttingRole;  
    glue glueType;  
  }  
  Attachments {  
    aspectualComponent.aPort to aConnector.aCrosscuttingRole  
    aConnector.aBaseRole to *.prefix* }  
}
```

**Fig. 5 ACME Description of the Composition**

### 5.1.3. Example

In this section, we present the modeling of the Distribution and Persistence concerns in the context of the HealthWatcher (HW) system (Section 2.2). We discuss two different configurations of the HW system architecture. This allows us to illustrate the flexibility and expressivity of AspectualACME to represent different architectural decisions when modeling an architecture. Figure 7 and 8 show the modeling of the two HW configurations using AspectualACME.

In the first system configuration only the Persistence is modeled as a crosscutting concern, the Distribution concern is specified as a non-aspectual component which allows the GUI component to remotely access the services provided by the Business component. The Persistence aspectual component existing in this configuration addresses: (i) the modularization of an update protocol to persist information modified by the GUI component; and (ii) the transaction demarcation of the services provided by the Business component using a transaction service available in the Data component.

Figure 7 depicts the AspectualACME description of the HW system including the Persistence crosscutting concern. The Persistence component affects the GUI component and the Business component. The composition of the Persistence component with the GUI component is modeled by the *Persist* aspectual connector. In the attachments section, the *Persist* connector connects *UpdateStateControl* with *registerUser* and with *registerComplaint* (both are referred by the \* wildcard in the attachments description). The glue clause of *Persist* specifies that the element linked to the crosscutting role (source) acts after the execution of the element linked to the base role (sink), this means that whenever a user or a complaint is registered, a persistence function is activated by the *Persistence* component. The internal implementation of *updateStateControl* needs to invoke the service of the *Distribution* Component, modeled by the C3 connector. However, as in ACME and in a lot of other ADLs, the internal implementation details is not described at the architectural specification, this internal feature is not registered in the AspectualACME description. However, if the

architect desires to expose an internal feature, ACME properties can be used for this purpose.

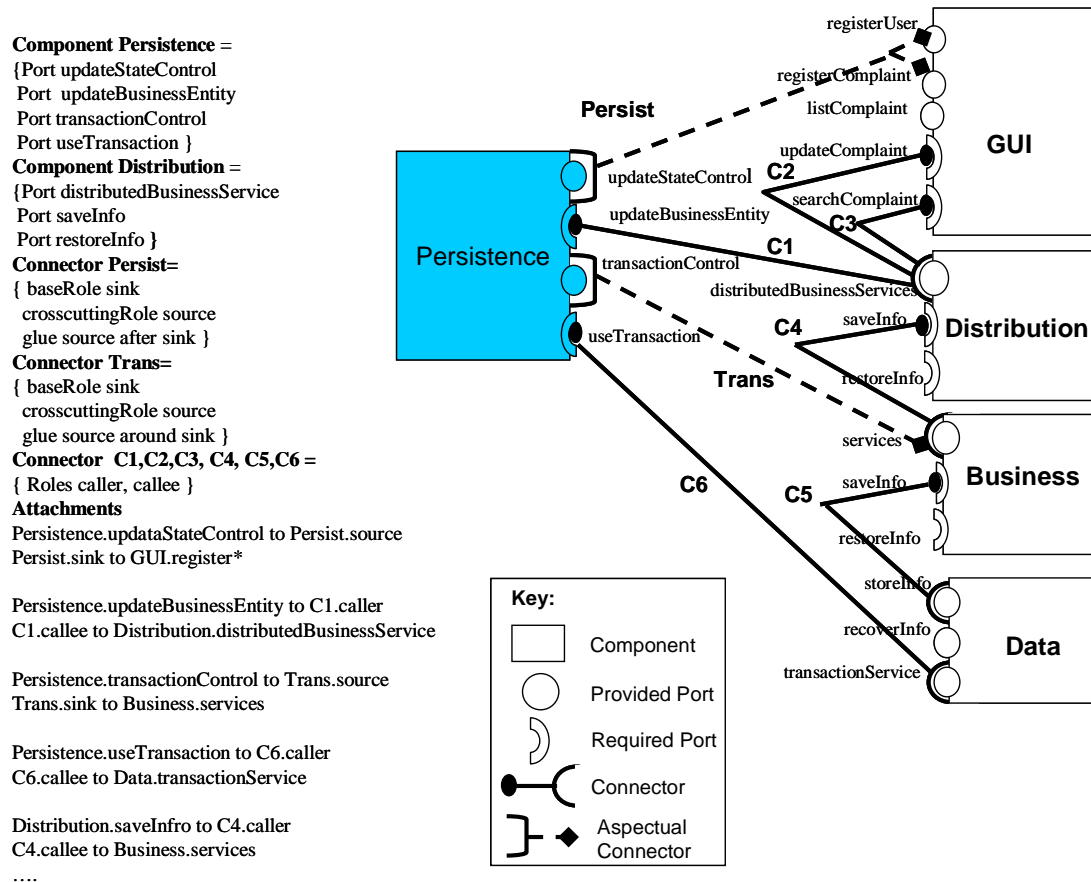


Fig. 7 HW AspectualACME Description with Persistence

The composition of the *Persistence* component with the *Business* component is modeled by the *Trans* aspectual connector. It connects the *services* with the *TransactionControl*. It defines that whenever a *service* is requested, a transaction control mechanism acts during this action. The idea is that the transaction control mechanism of the *Persistence* component uses the transactional operations (*begin\_transaction*, *comit\_transaction*, and *rollback*) provided by the *transactionService* of the *Data* component. However, again, as this information is internal of the *TransactionControl* implementation, it is not specified in the architectural description. This interaction is modeled by a traditional connector (C6) and it can be explicitly registered using ACME Properties.

The second configuration shows both *Persistence* and *Distribution* modeled as aspectual components addressing crosscutting concerns (Figure 8). This is the architectural modeling presented by an aspect-oriented implementation of the system (Soares et al 2002).. The *Persistence* is now responsible only for the transactional demarcation of the *Business* services. The *Distribution* aspectual component modularizes: (i) the transparent configuration of the calls from the *GUI* component to the *Business* to be realized through remote access; and (ii) the update protocol that persists information modified by the *GUI* component. This functionality is now

implemented by the Distribution component because it requires to invoke remotely the Business component.

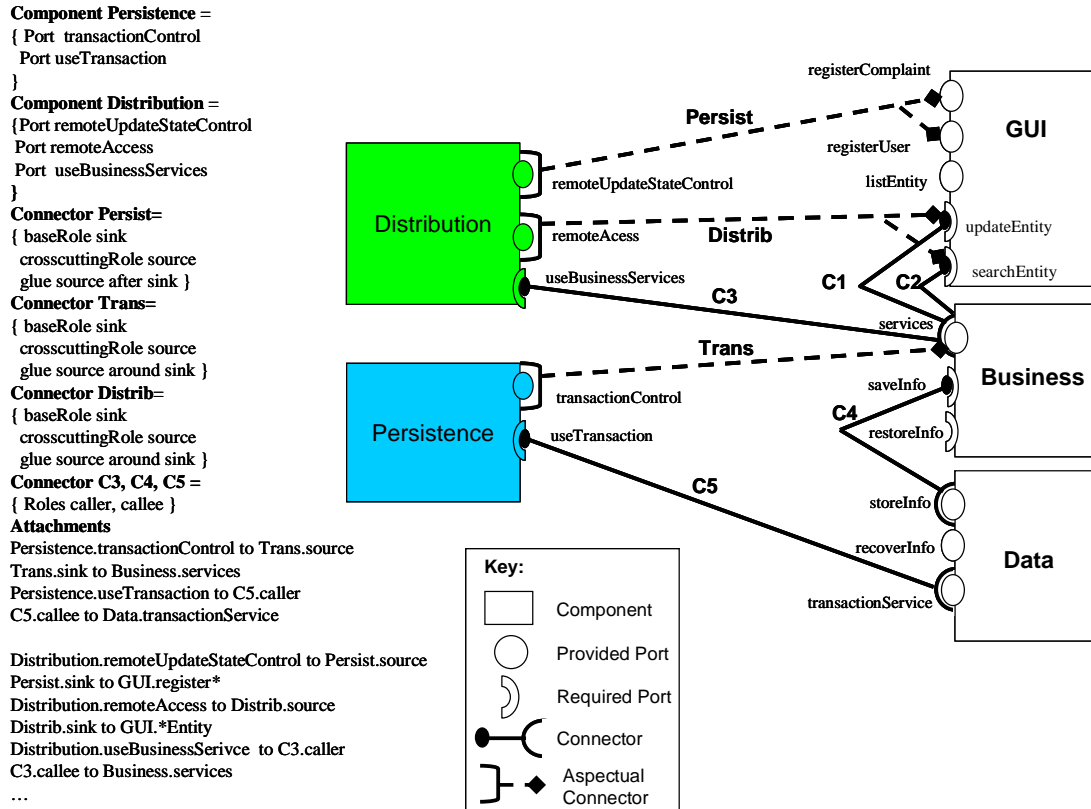


Fig. 8 HW AspectualACME Description with Persistence and Distribution

Figure 8 shows the AspectualACME description for the second configuration of HW. To model the update protocol, mentioned above, the Distribution aspectual component affects the *registerComplaint* and *registerUser* by quantifying them using wildcard expressions (*register\**). It is realized by specifying the *Persist* aspectual connector. The *Persist* glue clause guarantees the service attached to the crosscutting role is invoked after the execution of the services attached to the base role. The *Distribution* component also models the transparent distributed access of the Business component by the GUI component. The *Distrib* aspectual connector is responsible for this task. The attachments section defines that the *RemoteAccess* service affects *UpdateEntity* and *SearchEntity*. The idea is that internally, the *RemoteAccess* service redirects (using around) every invocation to the *BusinessServices* to be executed by means of the C3 connector. As this information represents implementation details of the *RemoteAccess* service, it is not described in the AspectualACME specification. Finally, the Persistence aspectual component models the transaction control in a similar way of that described in the first configuration.

## 5.2. Evaluation

We have evaluated the applicability of the proposed notion of Aspectual Connectors (Section 4) and the extensions provided in AspectualACME (Section 5.1) in the context of two case studies: the HealthWatcher system (Section 2.3), which has been partially

discussed in the previous section, and a context-sensitive tourist guide system described in (Batista et al, 2006). The second case study encompasses the manifestation of three architectural aspects: replication, security, and performance. In fact, the choice of such systems was driven by the heterogeneity of the aspects, and the way they affect regular components and each other.

Our approach has scaled up well in both case studies mainly by the fact that ACs and AspectualACME are following a symmetric approach, i.e. there is no explicit distinction between regular components and aspectual components. The modularization of the crosscutting interaction into connectors facilitated, for example, the reuse of the persistence component description from the first to the second case study. Persistence was a crosscutting concern only in the HealthWatcher architecture (Figure 7). Hence, we have not applied an aspectual connector in the second case. The definition of quantification mechanisms (Section 5.1.2) in attachments also has shown to be the right decision choice as it improves the reusability of connectors. The other reason was that it was easier to determine how multiple interacting aspects affect each other by looking in a single place in the architectural description – i.e. the attachments section.

Table 2 presents an evaluation of the proposed ADL according to the framework presented in Section 3. From this table, we can observe the simplicity of the proposed set of ACME extensions (Section 5.1). Our proposal advocates that no new architectural abstractions are needed to represent aspects. Regular components are used for this purpose. In addition, we have argued that no changes are required in components interfaces. AspectualACME defines a composition model that takes advantage of existing architectural connection abstractions – connectors and configuration – and extends them to support the definition of some composition facilities. In this way, it avoids introducing complexity in the architectural description and comparing with the existing solutions (Section 6), we identified a reduced set of required extensions to deal with architectural crosscutting concerns. As a result, the architects can model crosscutting concerns using the same abstractions, with minor adaptations, used in the conventional ADL description.

	<b>AspectualACME</b>
Base Elements	Aspectual components affect components through aspectual connectors.
Aspectual Composition	Modeled by aspectual connectors with base and crosscutting roles and by configurations.
Quantification	Supported by wildcards defined at the configuration section.
Aspect Interfaces	No extension required
Join Point Exposition	Components can expose their internal events
Interface Enhancement	Not supported
Aspects	Aspects are modeled by means of connectors, crosscutting roles, base roles, and components.

**Table 2. An Evaluation of the Proposed ADL**

## 6. Related Work

There is a diversity of viewpoints on how aspects (and generally concerns) should be modeled in ADLs. However, so far, the introduction of AO concepts into ADLs has

been experimental in that researchers have been trying to incorporate mainstream AOP concepts into ADLs. In contrast, we argue that most of existing ADLs abstractions are enough to model crosscutting concerns. For this purpose, is just necessary to extend the connector concept.

Most AO ADLs are different from AspectualACME because they introduce a lot of concepts to model AO abstractions (such as, aspects, joinpoints, and advices) in the ADL. DAOP-ADL (Pinto et al. 2005) defines components and aspects as first-order elements. Aspects can affect the components' interfaces by means of: (i) an *evaluated interface* which defines the messages that aspects are able to intercept; and (ii) a *target events interface* responsible for describing the events that aspects can capture. The composition between components and aspects is supported by a set of *aspect evaluation rules*. They define when and how the aspect behavior is executed. In the Prisma approach (Perez et al 2003), aspects are new ADL abstractions used to define the structure or behavior of architectural elements (component and connectors), according to specific system viewpoints. Components and connectors include a weaving specification that defines the execution of an aspect and contains weaving operators to describe the temporal order of the weaving process (after, before, around). Pessemier et al (Pessemier et al. 2004) extend the Fractal ADL with Aspect Components (ACs). ACs are responsible for specifying existing crosscutting concerns in software architecture. Each AC can affect components by means of a special interception interface. Two kinds of bindings between components and ACs are offered: (i) a direct crosscut binding by declaring the component references and (ii) a crosscut binding using pointcut expressions based on component names, interface names and service names.

Similarly to our proposal, FuseJ (Suvéé et al. 2005) defines a unified approach between aspects and components. It provides the concept of a gate interface that exposes the internal implementation of a component and offers access-point for the interactions with other components. In a similar way to our proposal, FuseJ concentrates the composition model in a special type of connector that extends regular connectors by including constructs to specify how the behaviour of one gate crosscuts the behaviour of another gate. However, differently from our work, our compositional model works in conjunction with the component traditional interface while FuseJ defines the gate interface that expose internal implementation details of a component. In addition, FuseJ does not work with the notion of configuration. It includes the definition of the connection inside the connector itself. This contrasts with the traditional way that ADLs works – that declares a connector and binds connectors' instances at the configuration section.

## **7. Conclusions**

In this paper we have proposed the concept of aspectual connector as a central element to support the integration of crosscutting concerns in an ADL description. We have also instantiated this concept in the context of a general-purpose ADL – ACME – and we have illustrated the concept with an example that presents two crosscutting concerns. Our proposal defines a composition model, centered on the concept of an aspectual connector, which takes advantage of traditional architectural connection abstractions – connectors and configuration – and extends them to support the definition of some

composition facilities such as a quantification mechanism. In this way, it avoids introducing complexity in the architectural description and comparing with the existing solutions, we identified a reduced set of required extensions to deal with architectural crosscutting concerns. As a result the architects can model crosscutting concerns using the same abstractions, with minor adaptations, used in the conventional ADL description. As such our proposal is based on enriching the composition semantics supported by architectural connectors instead of introducing new abstractions that elevate programming language concepts to the architecture level. Our proposal, therefore, supports effective modeling of crosscutting concerns without introducing additional complexity into the architecture specification. Planned future work includes evaluating our ADL by modeling a large-size system.

## References

- Aldrich, J. (2005) "Open modules: Modular reasoning about advice". In: Proc. of the European Conf. on Object-Oriented Programming (ECOOP'05), 144-168, July 2005.
- Araújo, J. et al. (2005) "Early Aspects: The Current Landscape." Technical Notes, CMU/SEI and Lancaster University, 2005.
- AspectJ Team (2006). "The AspectJ Programming Guide". <http://eclipse.org/aspectj/>.
- AspectualACME (2006). "AspectualACME". [www.teccomm.les.inf.puc-rio.br/aspectualacme](http://www.teccomm.les.inf.puc-rio.br/aspectualacme)
- Baniassad, E. et al. (2006) "Discovering Early Aspects". IEEE Software, Jan/Feb 2006.
- Batista, T. et al. (2006) "Reflections on Architectural Connection: Seven Issues on Aspects and ADLs." Workshop on Early Aspects, ICSE'06, May 2006, Shanghai, China.
- Chavez, C., Garcia, A., Kulesza, U., Sant'Anna, C., Lucena, C. (2006). "Taming Heterogeneous Aspects with Crosscutting Interfaces". Journal of the Brazilian Computer Society, Jun 2006.
- Chitchyan, R., Rashid, A., Sawyer, P., Garcia, A., Pinto, M., Tekinerdogan, B., Clarke, S., Jackson, A (2005) "Survey of Analysis and Design Approaches", AOSD-Europe Report, Deliverable D11, 18 May 2005.
- Cuesta, C. et al. (2005) "Architectural Aspects of Architectural Aspects". 2nd European Workshop on Software Architecture (EWSA), LNCS 3527, pp. 247-262, 2005.
- Filman, R. et al (2005). Aspect-Oriented Software Development. Addison-Wesley, 2005.
- Garcia, A., Kulesza, U., Lucena, C (2004). "Aspectizing Multi-Agent Systems: From Architecture to Implementation". In: Software Engineering for Multi-Agent Systems III, Springer-Verlag, LNCS 3390, December 2004, pp. 121-143.
- Garcia, A., Lucena, C. (2006) "Taming Heterogeneous Agent Architectures with Aspects". Communications of the ACM, March 2006. (submitted)
- Garcia, A., Batista, T., Rashid, A., Sant'Anna, C. (2006a) "Driving and Managing Architectural Decisions with Aspects". Submitted to IEEE Software, 2006.
- Garlan, D., Monroe, R., Wile, D (1997) "ACME: An Architecture Description Interchange Language". Proc. CASCON '97, Nov. 1997.
- Garlan, D. Moroe, R., Wile, D. (2000) .ACME: Architectural descriptions of component-based systems. :In Foundations of Component-based Systems. Gary T.Leavens and Murali Sitaraman(eds), Cambridge University Press, 2000, pp 47-68.
- van den Berg, K., Conejero, J., Chitchyan, R. (2005), "AOSD Ontology 1.0 – Public Ontology of Aspect-Oriented", AOSD-Europe Report, Deliverable D9, 27 May 2005.
- Kiczales, G. et al. (1997) "Aspect-Oriented Programming". European Conference on Object-Oriented Programming (ECOOP), LNCS 1241, Springer, Finland, June 1997.

- Krechetov, I., Tekinerdogan, B., Garcia, A., Chavez, C., Kulesza, U. (2006) Towards an Integrated Aspect-Oriented Modeling Approach for Software Architecture Design. 8th Workshop on Aspect-Oriented Modelling (AOM.06), AOSD.06, Bonn, Germany.
- Kulesza, U., Garcia, A., Lucena, C. (2004), "Towards a Method for the Development of Aspect-Oriented Generative Approaches." Workshop on Early Aspects, OOPSLA'04, November 2004, Vancouver, Canada.
- Kulesza, U., Garcia, A., Bleasby, F., Lucena, C. (2005) "Instantiating and Customizing Aspect-Oriented Architectures using Crosscutting Feature Models." Workshop on Early Aspects, OOPSLA'05, November 2005, San Diego, USA.
- Kulesza, U., Sant'Anna, C., Garcia, A., Coelho, R., Lucena, C. (2006) "Quantifying the Effects of Aspect-Oriented Programming: A Maintenance Study". Submitted to ICSM.06, 2006.
- Kulesza, U., Alves, V., Garcia, A., Lucena, C., Borba, P. (2006b). "Improving Extensibility of Object-Oriented Frameworks with Aspect-Oriented Programming". Proc. Intl. Conf. on Software Reuse, June, 2006.
- Medvidovic, N., Taylor, R. (2000). "A Classification and Comparison Framework for Software Architecture Description Languages". IEEE Trans. Soft. Eng., 26(1):70-93, Jan 2000.
- Mehta N., Medvidovic, N., Phadke, S. (2000) Towards a Taxonomy of Software Connectors. Proc. 22nd Intl Conf. on Software Engineering (ICSE), Limerick, Ireland, 2000.
- Navasa, A. et al. (2002) Aspect Oriented Software Architecture: a Structural Perspective. Workshop on Early Aspects, AOSD'2002, April 2002.
- Pérez, J., Ramos, I., Jaén, J., Letelier, P., Navarro, E. (2003) PRISMA: Towards Quality, Aspect-Oriented and Dynamic Software Architectures. In Proc. of 3rd IEEE Intl Conf. on Quality Software (QSIC 2003), Dallas, Texas, USA, November (2003).
- Pessemier, N., Seinturier, L., Duchien, L. (2004) Components, ADL and AOP: Towards a Common Approach. In Workshop ECOOP Reflection, AOP and Meta-Data for Software Evolution (RAM-SE04), June 2004.
- Pinto, M., Fuentes, L., Troya, J., (2005) A Dynamic Component and Aspect Platform, The Computer Journal, 48(4):401-420, 2005.
- Shaw, M. and Garlan, D. (1996): Software Architecture: Perspectives on an Emerging Discipline, Prentice Hall.
- Soares, S. et al (2002). "Implementing Distribution and Persistence Aspects with AspectJ". Proceedings of the OOPSLA'02, pp. 174-190.
- Suvéé, D., De Fraine, B. and Vanderperren, W. (2005) FuseJ: An architectural description language for unifying aspects and components. Software-engineering Properties of Languages and Aspect Technologies Workshop @ AOSD2005.