On the Modular Representation of Architectural Aspects

Alessandro Garcia [1], Christina Chavez [2], Thais Batista[3], Claudio Sant'anna[4], Uirá Kulesza[4] , Awais Rashid [1], Carlos Lucena[4]

[1] Computing Department, Lancaster University, United Kingdom
a.garcia@lancaster.ac.uk, marash@comp.lancs.ac.uk
[2] Computer Science Department , Federal University of Bahia, Brazil
flach@dcc.ufba.br
[3]Computer Science Department, Federal University of Rio Grande do Norte, Brazil
thais@ufrnet.br
[4]Computer Science Department, Pontifical Catholic University of Rio de Janeiro, Brazil
claudios@les.inf.puc-rio.br, uira@les.inf.puc-rio.br, lucena@inf.puc-rio.br

Abstract. An architectural aspect is a concern that cuts across architecture modularity units and cannot be effectively modularized using the given abstractions of conventional Architecture Description Languages (ADLs). Dealing with crosscutting concerns is not a trivial task since they affect each other and the base architectural decomposition in multiple heterogeneous ways. The lack of ADL support for modularly representing such aspectual heterogeneous influences leads to a number of architectural breakdowns, such as increased maintenance overhead, reduced reuse capability, and architectural erosion over the lifetime of a system. On the other hand, software architects should not be burdened with a plethora of new ADL abstractions directly derived from aspect-oriented implementation techniques. However, most aspect-oriented ADLs rely on a heavyweight approach that mirrors programming languages concepts at the architectural level. In addition, they do not naturally support heterogeneous architectural aspects and proper resolution of aspect interactions. This paper presents AspectualACME, a simple and seamless extension of the ACME ADL to support the modular representation of architectural aspects and their multiple composition forms. AspectualACME promotes a natural blending of aspects and architectural abstractions by employing a special kind of architectural connector, called *Aspectual Connector*, to encapsulate aspect-component connection details. We have evaluated the applicability and scalability of the AspectualACME features in the context of three case studies from different application domains.

Keywords: Architecture Description Languages, Aspect-Oriented Software Development, Architectural Connection.

1. Introduction

Aspect-Oriented Software Development (AOSD) [8] is emerging as a promising technique to promote enhanced modularization and composition of crosscutting concerns through the software lifecycle. At the architectural level, aspects provide a new abstraction to represent concerns that naturally cut across modularity units in an architectural description, such as interfaces and layers [1, 6, 9, 15]. However, the representation of architectural aspects is not a straightforward task since they usually require explicit representation mechanisms to address the heterogeneous manifestation of some widely-scoped properties, such as error handling strategies, transaction policies, and security protocols [5, 6,,10, 11]. By heterogeneous manifestation of widely-scoped properties – or , simply, *heterogeneous crosscutting* –, we mean that some properties impact multiple points in a software system, but the behavior that is provided at each of those points is different. Such architectural crosscutting concerns may interact with the affected modules in a plethora of different ways. Moreover, aspects may interact with each other at well-defined points in an architectural description. Hence, it is imperative to provide software architects with effective means for enabling the modular representation of aspectual compositions.

Software Architecture Description Languages (ADLs) [16] have been playing a central role on the early systematic reasoning about system component compositions by defining explicit connection abstractions, such as interfaces, connectors, and configurations. Some Aspect-Oriented Architecture Description Languages (AO ADLs) [19-22] 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, join points, 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 [1, 6, 24, 17]. In addition, such existing proposals typically provide heavyweight solutions [1, 25], making it difficult their adoption and the exploitation of the available tools for supporting ADLs. More importantly, they have not provided mechanisms to support the proper modularization of heterogeneous architectural aspects and their compositions.

This paper present AspectualACME, a general-purpose aspect-oriented ADL that enhances the ACME ADL [14] in order to support improved composability of heterogeneous architectural aspects. The composition model is centered on the concept of aspectual connector, which takes advantage of traditional architectural connection abstractions – connectors and configuration – and extends them in a lightweight fashion to support the definition of some composition facilities such as: (i) heterogeneous crosscutting interfaces at the connector level, (ii) a minimum set of aspect interaction declarations at the attachment level, and (iii) a quantification mechanism for attachment descriptions. Our proposal does not create a new aspect abstraction and is strictly based on enriching the composition semantics supported by architectural connectors instead of

introducing elements that elevate programming language concepts to the architecture level. This paper also discusses the applicability and scalability of the proposed ADL enhancements in the context of three case studies from different domains, and the traceability of AspectualACME models to detailed aspect-oriented design models.

The remainder of this paper is organized as follows. Section 2 introduces the case study used through the paper, and illustrates some problems associated with the lack of explicit support for modularizing heterogeneous architectural aspects and their interactions. Section 3 presents AspectualACME. Section 4 describes the evaluation of our approach. Section 5 compares our proposal with related work. Finally, Section 6 presents the concluding remarks and directions for future work.

2. Health Watcher: A Case Study

In this section we present the basic concepts of the ACME ADL [14] (Section 2.1) and discuss the architecture design of the case study that we are going to use as running example through the paper (Section 2.2), with emphasis on the heterogeneous crosscutting nature of some architectural concerns (Section 2.3) and their interactions (Section 2.4).

2.1 ACME in a Nutshell

ACME is a general purpose ADL proposed as an architectural interchange language. 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 the 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. *Properties* of interest are <*name, type, value*> 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. These ACME concepts are illustrated through this paper.

2.2 Health Watcher Architecture

The HealthWatcher (HW) system is a Web-based information system developed by the Software Productivity research group from the Federal University of Pernambuco [27]. It supports the registration of complaints to the health public system. Figure 1 illustrates a partial, simplified ACME [14] textual and graphical representation of the HW architectural description, which combines a client-server style with a layered style [30]. It is composed of five main architectural concerns: (i) the GUI (Graphical User Interface) component provides a Web interface for the system, (ii) the Distribution component externalizes the system services at the server side and support their distribution to the clients, (iii) the Business component defines the business elements and rules, (iv) the TransactionManager and Data components address the persistency concern by storing the information manipulated by the system, and (v) the ErrorHandling component which is charge of supporting forward error recovery through exception handling.

Figure 1 also illustrates a set of provided/required ports and connectors which make explicit the interactions between the architectural components. The saveEntity required port from the GUI component, for example, is linked to the distributedInterface provided port from the Distribution component by means of a connector. Despite many of the interactions between the architectural components have been appropriately represented using the port and connector abstractions, it is not possible to use these common ADL abstractions to represent the crosscutting relationships between two component services. Consider, for example, the transactionService provided port of the Transaction Manager component. It affects the execution of the savingService provided port of the Business component, by delimiting the occurrence of a business transaction (operations of begin, end and rollback transaction) before and after the execution of every operation invoked on savingService port. There is no existing abstraction in current ADLs which explicitly captures this crosscutting semantic between architectural component services. Because of lack of support to represent such kinds of crosscutting interactions between components, Figure 1 alternatively models it by defining the useTransaction required port. This description, however, does not make explicit the existence of crosscutting relationships between the components.

2.3 Heterogeneous Architectural Crosscutting

Exception handling is considered a widely-scoped influencing concern in the HW architectural specification [28], which is mostly realized by the Error Handling component. This component consists of the system exception handlers, and it provides the services in charge of determining at runtime the proper handler for each of the exceptions exposed by the system components, such as Distribution, Persistence [24], and TransactionManager. In fact, Figure 1 shows that the Error Handling component has a crosscutting impact on the HW architecture since it affects the interfaces of several components in the layered decomposition. Almost all the architectural interfaces need to

expose erroneous conditions, which in turn need to be handled by the error handling strategy. Figure 1 gives some examples of exceptional interfaces in the component's ports savingService and distributedInterface. Hence, the broadly-scoped effect of this component denotes its crosscutting nature over the modular architecture structure of the HW system.

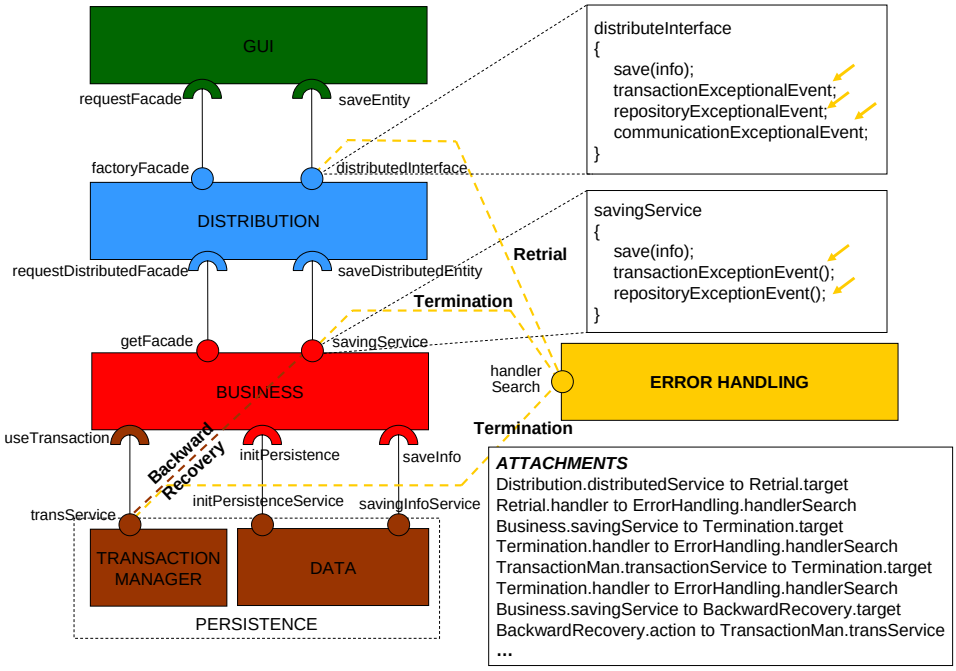

Figure 1. Error Handling in the HW Architecture: A Heterogeneous Crosscutting Concern

However, the influence of this crosscutting concern is not exactly the same over each affected HW component; it crosscuts a set of interfaces in heterogeneous ways, depending on the way the exception should be handled in the target component. In the HW system, there is at least two forms of interaction between a faulty component and the Error Handling component: the termination protocol (Termination connector), and the retry protocol (Retrial connector). However, the heterogeneous crosscutting composition of ErrorHandling and the affected architectural modules can not be expressed in a modular way. For instance, the connector Termination needs to be replicated according to the number of affected interfaces, and separated connectors for expressing the Retrial collaboration protocols need to be created. For simplification, Figure 1 only contains some examples of those connectors; the situation is much worse in the complete description of the HW architecture since almost all the interfaces expose exceptions. Also the attachment section contains a number of replicated, similar attachments created only for the sake of

combining the replicated error handling connectors (Figure 1). Finally, the "provided" interface handlingStrategy needs to be connected with the "provided" interfaces containing exceptional events, which is not allowed in conventional ADLs.

2.4 Aspect Interaction

In addition, there are other architectural breakdowns when using conventional ADLs to define interactions between crosscutting concerns. For example, the TransactionManager is another architectural aspect that crosscuts several elements in the Business layer in order to determine the interfaces that execute transactional operations. Most of these affected interfaces are also connected with the error handling connectors (Section 2.2). Figure 1 illustrates this situation for the savingService interface. The problem is that it is impossible to express some important architectural information and valid architectural configurations involving the interaction of the ErrorHandling and TransactionManager aspects. For example, although the attachments section allows the architect to identify that both aspects are actuating over the same architectural elements, it is not possible to declare which aspect has precedence over others affecting the same interfaces or whether only one or both of the backward and forward recovery strategies should be used.

3. AspectualACME

This Section presents the description of AspectualACME. We present the ACME extension to support the modeling of the crosscutting interactions (Section 3.1) and the definition of a quantification mechanism (Section 3.2). This section ends with a discussion about the AspectualACME support for modeling heterogeneous architectural aspects (Section 3.3) and aspect interaction (Section 3.4).

3.1 Aspectual Connector

As software architecture descriptions rely on a *connector* to express the interactions between components, an equivalent abstraction must be used to express the crosscutting interactions. We define an *aspectual component* as a component that represents a crosscutting concern in a crosscutting interaction. The traditional connector is not enough to model the crosscutting interaction because the way that an aspectual component composes with a regular component is slightly different from the composition between regular components only. A crosscutting concern is represented by provided services of an aspectual component and it can affect both provided and required services of other components which can be, in turn, regarded as structural join points [8] at the architectural level. As discussed in Sections 2.2 and 2.3, since ADL 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. Although ACME itself does not support a syntactic distinction between provided and required ports, this distinction can be expressed using properties or declaring port types.

In order to express the crosscutting interaction, we define the *Aspectual Connector (AC)*, an architectural connection element that is based on the connector element but with a new kind of 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) *base roles*, (ii) *crosscutting roles,* and (iii) a *glue* clause. Figure 2 depicts a high-level description of a traditional connector (Fig. 2a) and an aspectual connector (Fig. 2b).

```
Connector homConnector = {
   Role aRole1;
   Role aRole2;
}
```

(a) Regular connector in ACME

```
AspectualConnector homConnector = {
   BaseRole aBaseRole;
   CrosscuttingRole aCrosscuttingRole;
   Glue glueType;
}
```

(b) Aspectual connector in AspectualACME

Figure 2. Regular and Aspectual Connectors

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. The composition between components and aspectual components is expressed by the *glue* clause. The aspectual glue specifies the way an aspectual component affects one or more regular components. There are three types of aspectual glue: *after*, *before*, and *around*. The semantics is similar to that of advice composition from AspectJ [29].

```
AspectualConnector aConnector = {
  BaseRole aBaseRole1, aBaseRole2;
  CrosscuttingRole aCrosscuttingRole1,
                   aCrosscuttingRole2;
  Glue { aCrosscuttingRole1 before aBaseRole1;
         aCrosscuttingRole2 after aBaseRole2;   }
}
```

Figure 3. Heterogeneous aspectual connector.

The glue clause can be simply a declaration of the glue type (Figure 2b), or a block with multiple declarations, where each relates a crosscutting role, a base role and a specific glue type (Figure 3). The description of heterogeneous aspectual interactions (Section 3.3) requires more elaborated glue clauses.

Although the idea of the aspectual connector is derived from the traditional connector, it is not modeled as a subtype of the traditional connector, since the aspectual connector can be used in a connection between two provided ports. This would result in an invalid configuration (ill-formed connection) using the traditional connector and its subtypes.

Figure 4 contains a graphical notation that we propose to represent Aspectual Connectors. C1 is an aspectual connector that defines a crosscutting and heterogeneous interaction involving the Aspectual Component, Component 1, and Component 2.

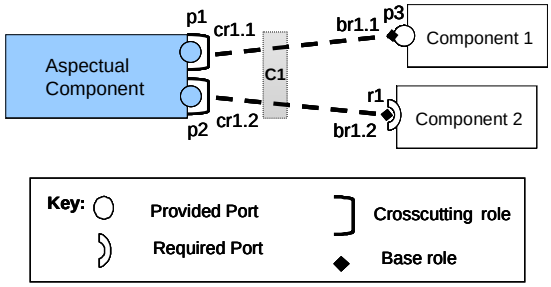

Figure 4. Graphical Notation to the Aspectual Connector

3.2 Quantification Mechanism

A base role of an aspectual connector may be bound to several ports of possibly different components. These ports represent *structural join points* that may be affected by aspectual components. To express these bindings, many attachments should be defined, where each one binds the *same* base role instance to a different component port. We propose an extension to the attachments part of an ACME configuration to allow the use of patterns. Wildcards such as '*', can be used in attachments to concisely describe sets of ports to be attached to the same base role.

```
System Example = {
Component aspectualComponent = { Port aPort }
AspectualConnector aConnector = {
   BaseRole aBaseRole;
   CrosscuttingRole aCrosscuttingRole;
   glue glueType;
}
Attachments {
  aspectualComponent.aPort to aConnector.aCrosscuttingRole
  aConnector.aBaseRole to *.prefix* }
}
```

Figure 5. ACME Description of the Composition .

The attachment "aConnector.aBaseRole **to** *.prefix*" (Figure 5) specifies the binding between aConnector.aBaseRole and ports from the "set of component ports where the port name begins with prefix". By avoiding explicit enumeration of ports and definition of multiple attachments, this extension promotes economy of expression and improves writability in architectural configurations.

3.3 Heterogeneous Architectural Aspects

Figure 2b presented a simple aspectual connector that has a homogeneous crosscutting impact on the architectural decomposition. Figure 3 shows how AspectualACME supports heterogeneous crosscutting. Multiple base and crosscutting roles can be used to define the different ways a crosscutting concern can affect the component interfaces. Different or similar glue types can be used in the definition of the pairs of base and crosscutting roles. Figure 6a is an example of heterogeneous aspectual connector for the error handling concern discussed in Section 2.2. Note that the two ways of interacting with the ErrorHandling component – i.e. retrial and termination – can now be modularized in a single architectural element. In addition, quantification mechanisms can be used in the attachments specification to describe in single statements which component ports are affected by those two crosscutting roles specified in the ForwardRecovery connector (Figure 6b).

```
AspectualConnector ForwardRecovery = {
   BaseRole toBeTerminatedTarget, toBeRetriedTarget;
   CrosscuttingRole termination, retrial;
   Glue {termination after toBeTerminatedTarget;
         retrial after toBeRetriedTarget;
   }
}
```

(a) an example of heterogeneous aspectual connector

```
Attachments {
  ForwardRecovery.toBeTerminatedTarget to *.*Service
  ForwardRecovery.termination to ErrorHandling.handlerSearch
  Distribution.distributedInterface to
      ForwardRecovery.toBeRetriedTarget
  ForwardRecovery.retrial to ErrorHandling.handlerSearch
  ...// to be continued in Figure 5b
}
```

(b) specification of join points using AspectualACME quantification mechanisms

Figure 6. Supporting Heterogeneous Crosscutting

Figure 7 presents a graphical notation for the HW example, where the ForwardRecovery is defined as a heterogeneous aspectual connector. The yellow vertical rectangle indicates that ForwardRecovery is a heterogeneous aspectual connector.

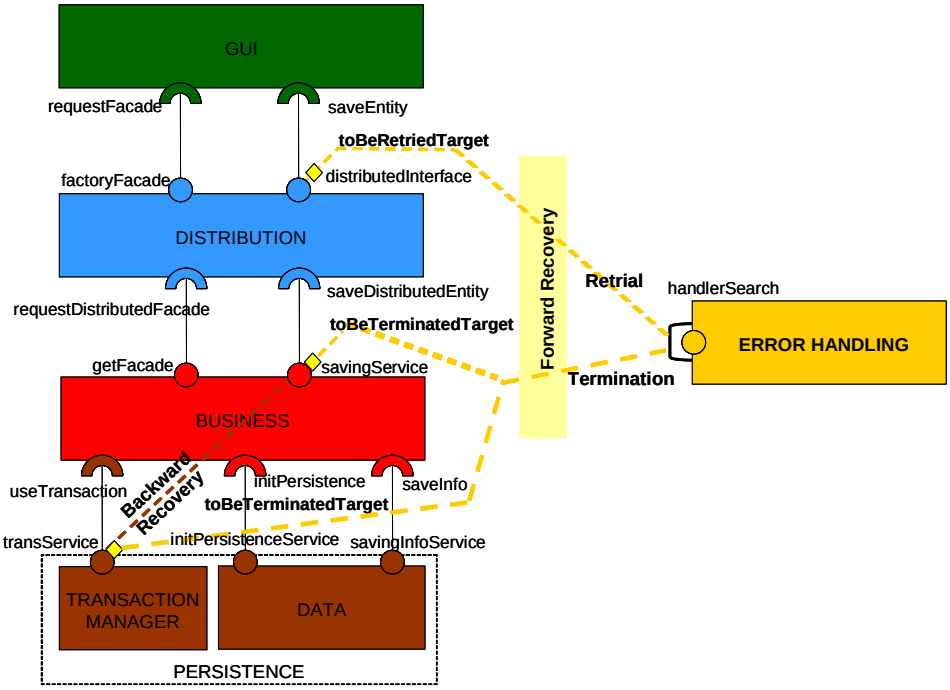

Figure 7. An Example of Aspectual Connector: Forward Recovery

3.4 Aspect Interaction

AspectualACME also allows the specification of aspectual architecture-level interaction between two or more aspectual connectors which have join points in common. Such interactions are declared in the configuration description since the attachments part is the place where join points are identified. The ADL supports two basic kinds of composition operators: precedence and XOR (Figure 8b). The architect can specify that the precedence is either valid for the whole architecture or only at specific join points. Figure 8b illustrates both situations: (i) in general, the Retrial connector has precedence over the Termination connector at all the join points they have in common, and (ii) at the port savingService, it is always tried first forward recovery through termination-based error handling and, second, the backward recovery with abort in case the exception was not successfully handled. When there is a precedence relation between two connectors X and Y, where the execution of Y depends on the satisfaction of a condition associated with X, the architect can explicitly document it using a condition statement together with an around glue in X. Figure 8b also illustrates the use of XOR: at a given join point, only one

of the either termination or retrial should be non-deterministically chosen. Finally, it is important to highlight that the elements participating in a precedence or XOR clause can be components instead of connectors: it means that the relationship applies to all the connectors involving the two components (see Section 4.1).

```
AspectualConnector BackwardRecovery = {
  BaseRole target;
  CrosscuttingRole transBegin, transAbort, transCommit;
  Glue {transBegin before target;
        transCommit after target;
        transAbort after target;
  }
}
```

(a) an example of aspectual connector

```
Attachments {
  //continued from Figure 4c
  Business.savingService to BackwardRecovery.target
  BackwardRecovery.transBegin to TransactionManager.transService
  BackwardRecovery.transCommit to TransactionManager.transService
  BackwardRecovery.transAbort to TransactionManager.transService
  Distribution.distributedInterface to ForwardRecovery.retriedTarget
  ForwardRecovery.Retrial to ErrorHandling.handlerSearch
  Precedence {
    ForwardRecovery.retrial, ForwardRecovery.termination;
    savingService:
         ForwardRecovery.termination, BackwardRecovery.transAbort;
  }
  XOR {
    ForwardRecovery.resumption, ForwardRecovery.termination;
  }
}
```

(b) specification aspectual interactions

Figure 8. Supporting Aspect Interaction Declarations

4. Evaluation

This Section presents the evaluation of AspectualACME in three case studies with respect to the usefulness of the proposed composition enhancements. We have evaluated the applicability and scalability of the notion of Aspectual Connectors (Section 3.1) and the extensions provided in AspectualACME (Sections 3.2 to 3.4) in the context of three case studies: the HealthWatcher system [28] (Section 2), a context-sensitive tourist information guide (TIG) system [9, 1], and AspectT – a multi-agent system framework [10, 11, 12]. As indicated in Table 1, the TIG architecture encompassed the manifestation of three heterogeneous architectural aspects: replication, security, and performance. The AspectT architecture included five main heterogeneous architectural aspects: autonomy,

adaptation, learning, code mobility, and interaction. The choice of such case studies was driven by the heterogeneity of the aspects, and the different ways they affect the dominant architectural decomposition and each other.

Our approach has scaled up well in all the case studies mainly by the fact that AspectualACME follows 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 TIG architectural specification. The definition of quantification mechanisms (Section 3.2) in attachments also has shown to be the right decision choice as it improves the reusability of connectors. The other reason was that is 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 specification.

Case Study	**Heterogeneous Aspects**	**Aspect Interactions**	
		# Total	**Some Examples**
Health Watcher	Error Handling, Transaction Management, Distribution	13	*Precedence*: Error Handling, Transaction Manag. *XOR*: ForwardRecovery.resumption, Forward Recovery.termination
TIG	Replication, Performance, Security	7	*Precedence*: Security, Performance *XOR*: Replication.passive, Replication.active
AspectT	Autonomy, Adaptation, Learning, Code Mobility, Interaction	15	*Precedence*: Interaction, Autonomy, Adaptation *Precedence*: Autonomy.execution, Autonomy.proactiveness *XOR*: Mobility, Collaboration

Table 1. Examples of Heterogeneous Architectural Aspects and their Interactions

Table 1 presents a summary on how AspectuaACME has been used through the three case studies to capture certain heterogeneous architectural aspects. It also describes how many aspectual interactions have been explicitly captured in those studies, followed by some examples of Precedence and XOR interactions. In our evaluation, we have noticed that two or more crosscutting roles of the same heterogeneous aspectual connector can naturally be linked to the same join point (a component port). Hence, the proposed aspect interaction mechanisms (Section 3.4) can be used to define their relationships. For example, Table 1 shows a XOR relationship in the HW architecture involving two crosscutting roles of the same connector: ForwardRecovery. Other interesting possibilities have been also explored in the case studies, such as declaring that all the connectors of Error Handling aspect have precedence over all the connectors of Transaction Management in the HW system. Also, we have observed that the explicit definition of such aspectual relationships in the architectural stage enhances the documentation of design choices that need to be observed later on the design of applications, and variation points in a certain product-line design [31].

5. 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, it is just necessary to define a new configuration element based on the traditional 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. Navasa et al 2005 [19] present a proposal to introduce the aspect modeling in the architecture design phase. Aspects are used to facilitate the architecture evolution by allowing easily either to modularize crosscutting concerns, or to incorporate new requirements in the system architecture. The composition between the architectural components and the aspects is based on an exogenous control-driven co-ordination model. The incorporation of the authors' model to existing ADLs, such as ACME, is still under investigation. Navasa et al 2002 [18] do not propose an AO ADL, but define a set of requirements which current ADLs need to address to allow the management of crosscutting concerns using architectural connection abstractions. The requirements are: (i) definition of primitives to specify joinpoints in functional components; (ii) definition of the aspect abstraction as a special kind of component; and (iii) specification of connectors between joinpoints and aspects. The authors suggest the use of existing coordination models to specify the connectors between functional components and aspects. Differently from our lightweight approach, they suggest the definition of AO specific ADL constructs. Furthermore, they do not mention in their proposal the need for supporting important AO properties such as quantification, interaction between aspects and heterogeneous aspects.

DAOP-ADL [22] 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. Besides, they also include a number of rules concerning with interaction between aspects. With regards to precedence, aspects can be evaluated in two ways: sequentially or concurrently. In addition, aspects can share information using a list of input and/or output global properties. Nevertheless, DAOP-ADL does not provide mechanisms to support quantification at the attachment level and explicit modularization of heterogeneous architectural aspects.

Similarly to our proposal, FuseJ [26] defines a unified approach between aspects and components, that is, FuseJ does not introduce a specialized aspect construct. It provides the concept of a gate interface that exposes the internal implementation functionality 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 exposes internal implementation details of a component. However, FuseJ provides explicit support neither for defining the interaction between aspects nor for modularizing heterogeneous aspects. Moreover, it only allows quantification over the same gate methods. 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.

Pessemier et al [21] defines the Fractal Aspect Component (FAC), a general model for mixing components and aspects. Their aim is to promote the integration between aspect-oriented programming (AOP) and component-based software engineering (CBSE) paradigms. FAC model proposes three new abstractions: (i) aspect components – that modularize a crosscutting concern by implementing the service of a regular component as a piece of an around advice; (ii) aspect bindings – which define bindings between regular and aspectual components; and (iii) aspect domains – that represents the reification of regular components affected by aspect components. FAC model is implemented under Fractal [2], an extensible and modular component model, and its respective ADL. There are similarities between the aspect component from the FAC model and our aspectual connector. Both are used to specify crosscutting concerns existing in the system architecture. The aspect bindings of FAC define a link between a regular and an aspect component. This latter can modify/extend the behavior of the former by affecting its exposed join points. In our approach, this is addressed by the definition of: (i) base and crosscutting roles – which allow specifying the binding between two components; and (ii) the glue clause – that define the semantic of crosscutting composition between them.

6. Conclusions and Future Work

This paper has addressed current issues related to aspect-oriented architecture modeling and design. The analysis of heavyweight solutions provided by some AO ADLs yielded to the design of AspectualACME, a general-purpose aspect-oriented ADL that supports improved composability of heterogeneous architectural aspects. The composition model is centered on the concept of aspectual connector, which takes advantage of traditional architectural connection abstractions – connectors and configuration – and, based on them, provides a lightweight support for the definition of some composition facilities such as: (i) heterogeneous crosscutting interfaces at the connector level, (ii) a minimum set of aspect interaction declarations at the attachment level, and (iii) a quantification mechanism for attachment descriptions. In this way, AspectualACME encompasses a reduced set of minor extensions, thereby avoiding the introduction of additional complexity in architectural descriptions. The paper also discussed the applicability and

scalability of the proposed ADL enhancements in the context of three case studies from different domains. Our approach has scaled up well in all the case studies mainly by the fact that AspectualACME follows a symmetric approach, i.e. there is no explicit distinction between regular components and aspectual components. Also, we have observed that explicit aspect interaction declarations in the architectural stage enhances the documentation of design choices that need to be observed later on the design of applications.

As future work, we plan to further elaborate on several issues related to the expressiveness of the AspectualACME language, as well as on traceability issues. Architectural descriptions in AspectualACME can be mapped to aspect-oriented design languages that support aspect-oriented modeling at the detailed design level, such as aSideML [5] and Theme/UML [8]. Tools need to be developed to support the creation of AspectualACME descriptions and their transformation to design level descriptions. Once these tools are available, designers may fully exploit the benefits from the aspect-oriented ADL and explicitly “design” aspectual connectors.

Acknowledgments

This work has been partially supported by CNPq-Brazil under grant No.479395/2004-7 for Christina. Alessandro is supported by European Commission as part of the grant IST-2-004349: European Network of Excellence on Aspect-Oriented Software Development (AOSD-Europe), 2004-2008. This work has been also partially supported by CNPq-Brazil under grant No.140252/03-7 for Uirá, and grant No.140214/04-6 for Cláudio. The authors are also supported by the ESSMA Project under grant 552068/02-0.

References

1. Batista, T., Chavez, C., Garcia, A., Sant’Anna, C., Kulesza, U., Rashid, A., Filho, F. Reflections on Architectural Connection: Seven Issues on Aspects and ADLs. Workshop on Early Aspects ICSE'06, pages 3-9, May 2006, Shanghai, China.
2. Bruneton, E., Coupaye, T., Leclercq, M., Quema, V., Stefani, J.-B. An open component model and its support in Java. In Proc. of the Intl Symposium on Component-based Software Engineering, Edinburgh, Scotland, May 2004.
3. Cacho, N., Sant'Anna, C., Figueiredo, E., Garcia, A., Batista, T., Lucena, C. Composing Design Patterns: A Scalability Study of Aspect-Oriented Programming. Proc. 5th Intl. Conference on Aspect-Oriented Software Development (AOSD'06), Bonn, Germany, 20-24 March 2006.
4. Chavez, C. A Model-Driven Approach for Aspect-Oriented Design. PhD thesis, Pontifícia Universidade Católica do Rio de Janeiro, April 2004.
5. Chavez, C., Garcia, A., Kulesza, U., Sant’Anna, C., Lucena, C. Taming Heterogeneous Aspects with Crosscutting Interfaces. Journal of the Brazilian Computer Society, vol.12, N.1, June 2006.
6. Chitchyan, R., et al. A Survey of Analysis and Design Approaches. AOSD-Europe Report D11, May 2005.
7. Clarke, S. and Walker, R. Generic aspect-oriented design with Theme/UML. In [8], pages 425-458.

8. Filman, R.,Tzilla E., Siobhan Clarke, and Mehmet Aksit, editors. Aspect-Oriented Software Development. Addison-Wesley, Boston, 2005.
9. Garcia, A., Batista, T., Rashid, A., Sant'Anna, C. Driving and Managing Architectural Decisions with Aspects. Proc. SHARK.06 Workshop at ICSR.06, Turin, June, 2006.
10. Garcia, A., Kulesza, U., Lucena, C. 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.
11. Garcia, A., Lucena, C. Taming Heterogeneous Agent Architectures with Aspects. Communications of the ACM, March 2006. (accepted)
12. Garcia, A., Lucena, C., Cowan, D. Agents in Object-Oriented Software Engineering. Software: Practice & Experience, Elsevier, Volume 34, Issue 5, April 2004, pp. 489-521.
13. Garcia, A., Sant'Anna, C., Figueiredo, E., Kulesza, U., Lucena, C., Staa, A. Modularizing Design Patterns with Aspects: A Quantitative Study. Transactions on Aspect-Oriented Software Development, Springer, LNCS, pp. 36 - 74, Vol. 1, No. 1, February 2006.
14. Garlan, D. et al. ACME: An Architecture Description Interchange Language, Proc. CASCON'97, Nov. 1997.
15. Krechetov, I., Tekinerdogan, B., Garcia, A., Chavez, C., Kulesza, U. Towards an Integrated Aspect-Oriented Modeling Approach for Software Architecture Design. 8th Workshop on Aspect-Oriented Modelling (AOM.06), AOSD.06, Bonn, Germany.
16. Medvidovic, N., Taylor, R. A Classification and Comparison Framework for Software Architecture Description Languages. IEEE Trans. Soft. Eng., 26(1):70-93, Jan 2000.
17. Mehta N., Medvidovic, N. and Phadke, S. Towards a Taxonomy of Software Connectors. Proc. of the 22nd Intl Conf. on Software Engineering (ICSE), Limerick, Ireland, pp. 178 – 187, 2000.
18. Navasa, A. et al. Aspect Oriented Software Architecture: a Structural Perspective. Workshop on Early Aspects, AOSD'2002, April 2002.
19. Navasa, A., Pérez, M. A., Murillo, J. M. Aspect Modelling at Architecture Design. EWSA 2005, pp. 41-58, LNCS 3527, Pisa, Italy, 2005.
20. Pérez, J., et al., E. 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).
21. Pessemier, N., Seinturier, L., Coupaye, T., Duchien, L. A Model for Developing Component-based and Aspect-oriented Systems. In 5th International Symposium on Software Composition (SC'06), Vienna, Austria, March 2006.
22. Pinto, M., Fuentes, L., Troya, J., "A Dynamic Component and Aspect Platform", The Computer Journal, 48(4):401-420, 2005.
23. Quintero, C., et al. Architectural Aspects of Architectural Aspects. Proc. of European Workshop on Software Architecture (EWSA2005)- Pisa, Italy, June 2005, LNCS 3527.
24. Rashid, A., Chitchyan, R. Persistence as an Aspect. Proc. of the 2nd Intl. Conf. on Aspect-Oriented Software Development (AOSD'03), USA, March 2003.
25. Rashid, A., Garcia, A., Moreira, A. Aspect-Oriented Software Development Beyond Programming. Proc. of ICSE.06, Tutorial Notes, May 2006, Shanghai, China.
26. Suvée, 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.
27. SPG – Software Productivity Group at UFPE. http://twiki.cin.ufpe.br/twiki/bin/view/SPG, 2006.
28. Soares, S., Laureano, E. and Borba, P.. Implementing Distribution and Persistence Aspects with AspectJ. In Proc. of OOPSLA'02, Seattle, WA, USA, 174-190, November 2002. ACM Press.
29. The AspectJ Team. "The AspectJ Programming Guide". http://eclipse.org/aspectj/
30. Kulesza, U., et al. Quantifying the Effects of Aspect-Oriented Programming: A Maintenance Study. Proc. of the Intl Conf. on Software Maintenance (ICSM'06), Philadelphia, USA, September 2006.
31. Kulesza, U., Alves, V., Garcia, A., Lucena, C., Borba, P. Improving Extensibility of Object-Oriented Frameworks with Aspect-Oriented Programming. Proc. of the 9th Intl Conf. on Software Reuse (ICSR'06), Turin, Italy, June 2006.