Representing Architectural Aspects with a Symmetric Approach

Alessandro Garcia[1], Eduardo Figueiredo[2], Claudio Sant'Anna[3], Monica Pinto[4], Lidia Fuentes[4]

[1]Computer Science Department, Pontifical Catholic University of Rio de Janeiro, Brazil
[2]Computing Department, Lancaster University, United Kingdom
[3]Computer Science Department, Federal University of Bahia (UFBA), Brazil
[4]Dpto. de Lenguajes y Ciencias de la Computacion, University of Malaga, Spain

afgarcia@inf.puc-rio.br, e.figueiredo@lancaster.ac.uk, santanna@dcc.ufba.br, {pinto, lff}@lcc.uma.es

ABSTRACT

Aspect-oriented (AO) techniques are emerging as promising approaches to enhance the representation of crosscutting concerns throughout the software lifecycle. This includes new AO specification mechanisms for the architectural design stage that is at the heart of the software process. However, existing modelling languages have failed short to provide simple and scalable notations for visually representing the so-called "architectural aspects". This paper reports our ongoing effort on the definition of a visual architecture representation for aspect-oriented systems. Our proposal follows a symmetric approach and provides a more expressive set of visual elements in order to: (i) provide a more intuitive notation for expressing aspectual compositions, (ii) facilitate a symbiotic transition of AO requirements specifications to AO architecture designs, (iii) make the transition of architectural descriptions to AO detailed designs more straightforward, and (iv) improve the early detection of modularity anomalies in aspect-oriented design. We discuss the advantages and drawbacks of our modelling proposal in terms of two applications from different domains.

Categories and Subject Descriptors

D.2.11 [**Software Architectures**]. Languages.

General Terms

Design, Languages.

Keywords

Architecture, symmetric visual notation, aspects.

1. INTRODUCTION

Aspect-Oriented Software Development (AOSD) is one of the most eminent post-OO software development paradigms. Existing modelling languages have been enriched with new modularization and composition forms in order to support modular representation of crosscutting concerns throughout the software lifecycle. Crosscutting concerns are features that affect several modularity units in a certain system representation. One of the main reasons for visual AOSD models not achieving maturity is that effective visual representations of AO software architectures have been clearly neglected [11].

Even though a number of AO design languages [14, 9] and requirements specification techniques [11, 14] have been consistently defined, researchers have not paid enough attention to visual notations for AO architectures. This leads to a number of methodological breakdowns as software architecture provides the link between the problem and solution models. Architectural models also allow the communication amongst a plethora of different stakeholders, including requirements engineers, detailed designers, and the quality assurance team.

However, all the existing approaches for representing AO software architectures are in a preliminary stage of research [19]. All of them try to provide visual means to express *aspectual compositions*, by defining how architectural aspects affect architecture elements in well-defined *join points*. Typically, they are *asymmetric* extensions of the component-and-connector model [1, 12, 13, 18], which is historically a core architecturally-relevant system representation mechanism. By asymmetric, we mean that all of them make an explicit distinction between aspects (i.e., "aspectual components") and non-aspectual components. The visual asymmetry in such approaches leads to a number of scalability and expressiveness problems. It also makes the transition of requirements to architecture specifications difficult.

This paper reports on our ongoing effort for the definition of an expressive and intuitive notation for AO software architectures. This work is partially funded by the AOSD-Europe project, and the contributions are twofold: (i) a discussion of some limitations in existing asymmetric languages for modelling AO architectures (Section 2), and (ii) the provision of an innovative symmetric notation to visually represent such designs (Section 3). We also discuss a preliminary evaluation we performed in the context of two case studies (Section 4). Section 5 concludes this paper.

2. ISSUES ON THE VISUAL NOTATION OF ARCHITECTURAL ASPECTS

This section reports on some problems we faced when applying existing asymmetric AO architectural notations in several projects [6]. The goal in these projects was to achieve visual representations of AO architectures that were: (i) easy to

EA'09, March 3, 2009, Charlottesville, VA, USA.

understand in terms of where architectural aspects occur and which forms of composition with other architectural components are used, (ii) straightforwardly translating aspect-oriented concepts commonly supported by multiple AO architecture description languages (ADLs) [2], (iii) supporting appropriate high-level modularity measurements, and (iv) smoothly mapping requirements-level aspects to architecture-level aspects. We should highlight that not all the problems discussed later in this section are necessarily intrinsic to asymmetric approaches.

The HW Architecture. The Health Watcher (HW) system is a real-life Web-based information system [5] that supports the registration of complaints to the health public system. It is used as our running example throughout this paper. Figure 1 illustrates a partial graphical representation of the HW architecture, which is based on a set of components mainly realizing an instance of the layered style. It is composed of seven architectural components; three of them are layers: (i) the GUI (Graphical User Interface) component provides a Web interface for the system, (ii) the Business component defines the business elements and rules, and (iii) the Data component addresses the data management by storing the information manipulated by the system.

The aspect-oriented HW architecture also contains four architectural aspects: Persistence, Distribution, Concurrency and Error Handling (EH). For instance, the Distribution aspectual component externalizes the system services at the server side and supports their distribution to the clients. Figure 1 shows an asymmetric representation of the HW architecture, based on the AOGA language, which we chose to illustrate the limitations of existing visual notations and respective meta-models [19]. In AOGA, aspects are aspectual components that are represented by UML components with a diamond in the top.

Problem 1: Expressiveness Impairments. Since AOGA and other asymmetric notations create a specific symbol to represent an aspect, it is not possible to smoothly use the same notation for a component that does not play the role of an aspect in a context, but not in others. This expressiveness bottleneck also hinders reuse of component representations across different projects, when the target component is an aspect in one architectural design. Such a dichotomised notation gives the wrong impression that a certain aspectual component cannot assume different roles defined by, for example, different styles. For instance, in one of the HW releases [16], the Distribution component played the role of being both an aspect and a layer. With an asymmetric architectural notation, it would not be obvious to notice that Distribution was free to take part in other collaborations and play different architecture stylistic roles, i.e. "being a layer".

Problem 2: Inability to Represent Heterogeneous Aspectual Compositions. We also observed that in existing approaches [19], there are not many visual ways to graphically specify and distinguish different forms of collaborations between non-aspectual and aspectual components. For example, there is no possibility of clearly communicating the sequencing of a crosscutting composition; i.e. the order (e.g. before, after, or around) in which the aspect computation will affect the base computation. In general, architects cannot easily check the composition sequencing at a glance, because they are only supported in the expanded view of component interfaces [19] and, even worse, it is textually declared together with the crosscutting service. Fig. 1 illustrates this problem in the context of the TransactionControl interface. Also, in asymmetric notations the sequencing is typically associated with a service in a certain aspect interface, which in turn also reduces the component specification reusability. This also might cause problems when the same service is involved in different aspectual compositions, thereby affecting the target join points in distinct orders. Finally, some of these notations (AOGA is an example) typically use the same symbol to represent different composition mechanisms, even though they have different architecturally-relevant semantics. For example, crosscutting interfaces and relationships are used to denote both behaviour-based (pointcut-advice-like) and structural compositions (such as inter-type declarations or structure merges).

Problem 3: Limited Scalability. In the projects [6, 16] where we used existing asymmetric architecture notations, a number of scalability issues were detected. Some examples are discussed in the following. First, they do not scale when a crosscutting interface affects several join points in the architecture, even via the same aspect interface. Fig. 1 illustrates this problem for the ConcurrencyManager interface that affects all interfaces of the Data component. Actually, this is a generic problem in AO design notations as crosscutting is a kind of relationship that often implies a plethora of links between the aspect and the affected elements. In other words, such notations suffer from not having visual resources to quantify such links. Second, they neither support graphical capabilities for representing certain inter-aspect dependencies. Also, they are typically textual and alternatively based on the use of stereotypes [19]. Even though AOGA has a stereotype dedicated for annotating aspect precedence, the granularity is aspect-aspect level and cannot be tailored to certain compositions or particular architectural join points, such as a particular service.

Problem 4: Hindering Architecture Modularity Assessment. A direct consequence of problems 2 and 3 is that architects are not

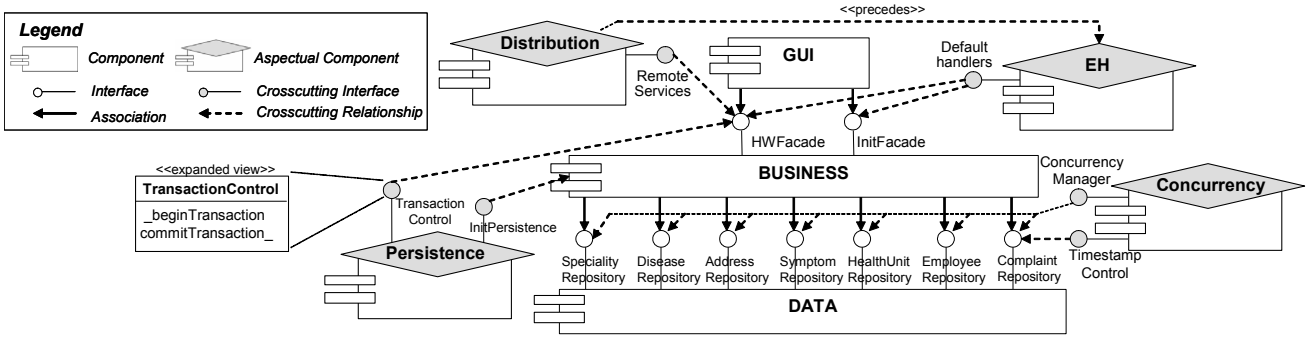

Figure 1. Health Watcher architectural design with AOGA

able to effectively assess modularity properties of an AO architecture design. Differently from current practice in UML 2 [17], where different kinds of connectors (delegators, dependencies, or assemblies) are supported by the notation and the underpinning meta-model, existing AO architecture notations are not yet mature to serve as expressive artefacts to support early modularity assessment. We experienced this problem in architectural assessment of 3 case studies [6, 16]. For instance, because the differences in the representation of certain architectural aspect compositions are not made explicit (problem 3), computation of specific architectural metrics such as afferent and efferent couplings [6] is impaired.

Problem 5: From Requirements to Architecture. From our experience defining a mapping process and guidelines [7, 8] to relate AO requirements (specified using RDL [10]) and AO architecture (specified using AO-ADL [3]), we learned that a 1-to-1 mapping is not possible. Instead, the same crosscutting concern can be mapped either to a non-aspectual or to an aspectual component, or even to an architectural decision depending on the application context. An example is Distribution that, as mentioned before, can play the role of being an aspect or a layer depending on the HW release. The problem is that asymmetric visual notations provide different abstractions to represent components and aspects and, as a consequence, force us to make the decision of mapping requirements to a component or to an aspect as part of the mapping process itself. This is neither necessary, as the decision can be postponed until a refined version of the mapped architecture, nor desirable, as emerging AO requirement proposals are symmetric and do not make such a distinction.

3. A SYMMETRIC VISUAL NOTATION

This section presents our visual notation in terms of: (i) its meta-model (Section 3.1) with the key architectural abstractions supported, and (ii) a set of graphical elements to allow the representation of aspectual compositions in component-and-connector models. Both meta-model and graphical elements were defined to address the limitations discussed in Section 2.

The proposed notation is an evolution of our previous work [12, 19], rather than a totally new approach. It has being systematically derived from: (i) a previous systematic analysis of four modelling approaches, namely TranSAT [1], PCS Framework [18], AOGA [12], and CAM of DAOP-ADL [13], (ii) a primitive visual notation defined for an AO extension to the ACME language [15], (iii) an analysis of abstractions consistently appearing across existing ADLs, such as AO-ADL [3], AspectualACME [15], DAOP-ADL [13], and others [2]. The derivation of our current approach involved the "transformation" of a previous asymmetric notation [19], unified from the 4 approaches mentioned above in (i), into a new symmetric notation. Hence, Section 4 evaluates the benefits and drawbacks obtained in this transformation process.

3.1. Meta-Model

Figure 2 presents our notation meta-model. Our visual notation extends the set of architecturally-relevant abstractions and respective graphical elements of UML 2 [17], such as services, components, interfaces, and connectors. In fact, we use UML 2 as the basis without modifications to its existing visual elements. As a result, existing UML architectural models can be straightforwardly refactored to accommodate architectural aspects. It is not the goal of this paper to discuss the integration of our notation's meta-model and UML 2 meta-model. Also, for the sake of simplicity we omitted from the meta-model (Fig. 2) some conventional architectural concepts in UML 2, such as ports. From this integration perspective, the discussion here is limited to evaluate on why specific UML connectors, with associated graphical representations, are not appropriate to represent a crosscutting composition (Section 3.3).

The meta-model focuses on the definition of new aspect-oriented concepts and their relations. The meta-model (Fig. 2) subsumes 3 main categories of elements: (i) components and interfaces (Section 3.2); (ii) aspectual connectors (Section 3.3); and (iii) crosscutting relationships (Section 3.4).

3.2. No Specialized Components and Interfaces

A *component* is considered a modularity unit within a system architecture that has one or more *provided and/or required interfaces* (potentially exposed via ports). A component specifies a formal contract of the services that it provides to its clients and those that it requires from other components or services in the system in terms of its provided and required interfaces. Its internals are hidden and inaccessible other than as provided by its interfaces. Such access constraints also apply to components playing the role of architectural aspects, i.e. those ones involved in a crosscutting collaboration with other components (Section 3.3). If an architectural aspect needs to know any internal detail of a certain component, such a detail needs to be made available at one of its interfaces.

The meta-model is symmetric in the sense that it does not define an explicit abstraction for an aspect. Both crosscutting and non-crosscutting concerns are represented by components. The distinction is made at the connector level (Section 3.4), i.e. it is the way two or more components are composed that denote that a crosscutting composition is taking place. No new "aspectual"

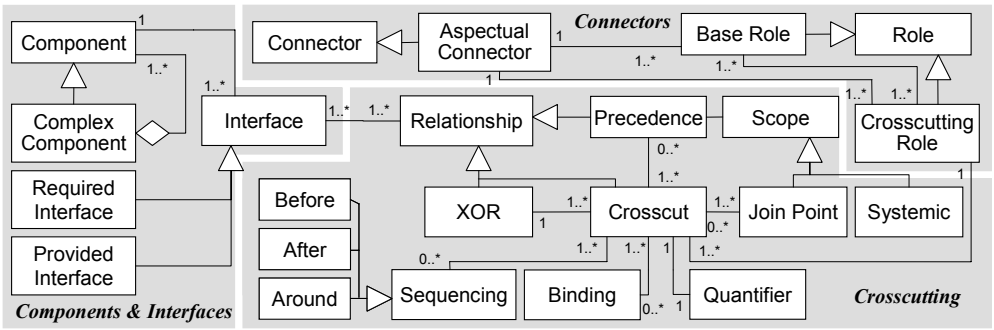

Figure 2. Meta-Model of the Symmetric Visual Notation

component interface is defined in the meta-model as we believe that "aspectual components" also offer services and expose events or attributes, like any other component.

3.3. Aspectual Connectors

Our position is that a minimum of new abstractions and respective graphical elements should be supported by the visual notation. The reason is that architectural description languages, whether textual or visual, were conceived with the goal of being agnostic to specific architectural styles, such as layered and pub-sub architectures. Hence, architecture design languages should be kept as small as possible, while accommodating support for representing architecture-relevant aspects.

In fact, UML 2 and other component-and-connector notations do not create specific graphical elements to denote that a certain component is a layer, a publisher, or an aspect. This visual distinction would be very counter-productive in large architecture designs since it is common to find single components playing multiple roles defined by different architectural styles. As discussed in Section 2, the Distribution component (Fig. 1) is an example of this case in the HW architecture. Also, based on our experience, it is becoming increasingly clear that the key difference of an aspect-oriented architecting style is the composition semantics [2].

Hence, the visual representation of AO software architectures should provide support for the possible architecture-level crosscutting compositions observed in our case studies (Section 2). This should be rooted at the traditional notion of connectors (and attachments) of the software architecture discipline. The reason is that connectors are the locus of composition in architectural design [2]. As a result, our visual notation supports the notion of aspectual connectors (Figure 3). This emphasis on aspectual connectors is not currently supported by the investigated visual notations for aspect-oriented software architectures [1, 12, 13, 18]. However, it is consistently becoming a common practice in recent AO textual description languages [15, 3]. Before describing how we represent aspectual connectors, we discuss first why conventional connector types, available in UML 2, are not appropriate to capture the notion of crosscutting compositions.

Connectors can define a wide range of composition styles, ranging from simple dependencies to complex collaboration protocols. For example, UML 2 defines three specialized connectors for interlinking components, namely dependencies, assemblies, and delegators. Different visual elements are associated with each of them. Crosscutting compositions cannot obviously be represented by dependencies; hence, we concentrate our discussions on assemblies and delegators.

Assembly vs. Aspectual Connectors. We cannot rely on assembly connectors to represent crosscutting compositions because they imply a simple relation between required and provided ports [17]. In addition, an assembly connector must only be defined from a required interface (or port) to a provided interface (or port), which violates a typical composition property of crosscutting collaborations [2]: an aspectual component and affected components can be linked through both their provided interfaces.

Delegation vs. Aspectual Connectors. In addition, we cannot reuse the notion of delegation connectors. They have a number of modelling constraints that do not match the requirements for aspectual compositions at the architectural level. The main problem is that they subsume a "forwarding" semantic. A delegation connector is a connector that links the external contract of a component (as specified by its ports) to the internal realization of that behaviour by the component's parts [17]. Besides, a delegation connector must only be defined between used Interfaces or Ports of the same kind (e.g., between two provided interfaces or between two required interfaces). Aspectual compositions involve the identification of several join points (e.g. affected interfaces or services) to be connected to the component encapsulating a "crosscutting concern". They also specify composition operators on when or how those points are being connected with other services provided by components encapsulating a "crosscutting concern".

Finally, architecture-level crosscutting compositions require that aspectual connectors might actuate directly over other connectors. Most architecture representation languages do not grant this property to connectors [2], which reinforces the need for a specialized type of connector. Fig. 3 shows our visual representation for aspectual connectors. The use of the stereotype is optional and not motivated. The aspectual connector is a component-like graphical notation with elements to specify the "crosscutting collaboration" amongst involved architectural elements. A simpler notation (cf. Fig. 5) is available in case connector internals are not relevant.

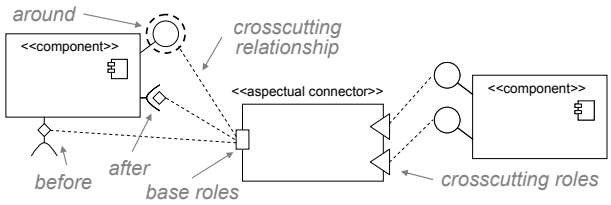

Figure 3. Notation for Aspectual Connectors

3.4. Base and Crosscutting Roles

Aspectual connectors (Figure 2) are basically formed by base and crosscutting roles (Figure 3). These roles consist of two types of connector's interfaces, and define the roles the connected components are playing in a crosscutting composition. A crosscutting role defines which component is playing the role of an "aspect" in the architectural model, i.e. which component is encapsulating a crosscutting concern and needs to affect other interfaces. Crosscutting roles are represented by triangles "cutting across" the connector boundaries. Base roles are associated with different join points affected by the components attached to the crosscutting roles. They are represented by small rectangles in the opposite extreme of an aspectual connector (Figure 3).

Crosscutting relationships define how the connectors and components are attached. In another words, they are equivalent to attachments in ADLs (ACME and xADL), and their visual representation is a dashed arrow. The arrows associate crosscutting or base roles with component interfaces. In the presence of multiple base and crosscutting roles, the dashed arrows can also cut across the aspectual connector representation in order to show how base and crosscutting roles are interlinked. This situation is illustrated in Fig. 4, which is a symmetric visual representation of the HW system shown in Fig. 1 (Section 2). The DataControl connector has multiple roles which are bound through the dashed arrows.

3.5. Pointcut and Sequencing Specifications

The set of join points of interest (i.e., pointcuts) in a certain crosscutting composition are conventionally indicated by visual

(and sometimes, textual) elements associated with a crosscutting relationship. When a component interface is touched by an arrow, it means that one or more of the interface services are affected by an aspectual connector. If a precise indication of which service(s) are being connected, the name of the service(s) is attached to the arrow using stereotypes. Fig. 4 illustrates an example of specific services being bound through the Synchronization connector. In addition, whenever it is required, a sequencing operator can be associated with a crosscutting relationship. It specifies when or how the connector is affecting the service(s). By now, the notation includes graphical elements for three sequencing operators: before, after, and around (Figure 3). However, other operators could be used. Some concrete examples for the HW architecture are presented in Figure 4.

3.6. Quantification and Aspect Interaction

Our visual notation provides support for specifying quantifications, i.e. describing in a single place which elements a certain aspectual connector is affecting. The goal is to overcome the problem 3 discussed in Section 2, i.e. visually support quantification and reduce the number of arrows for crosscutting relationships. An example is presented in Figure 4: the Synchronization connector affects all the Data interfaces.

The notation used is: (i) a set of multiple grouped dashed arrows pointing to the direction of the affected elements, plus (ii) a label with an expression indicating more precisely a property that matches the affected elements. Fig. 4 shows that the Synchronization connector is affecting all interfaces of the Data component. We defined specific visual elements to represent certain recurring quantifications that we observed in our study (Section 2), such as: "all the provided interfaces in…" and "all the required interfaces in…". Due to space limitation, we cannot present the visual notations for all of them here.

The visual notation also provides elements for addressing aspect interactions (problem 3). Fig. 4 illustrates a scenario where we specify that the same aspectual connector is affecting (in an after fashion) the same join point, i.e. the interface HWFacade. As a result, two diamonds are on the top of this interface. However, priority is given to the element that is associated with the diamond closer to the interface circle. It means that TransactionControl has precedence over InitPersistence. The same semantics applies to before and around operators; in the case of two or more around operators actuating over the same join points, inner circles have priority over the enclosing ones. Graphical elements are also used to represent XOR and OR relationships.

4. EVALUATION

This section summarizes the evaluation of the proposed visual notation (Section 3) using two case studies: (i) the complete specification of the HW architecture (Section 2), and (ii) the definition of an auction system's architecture based on AO requirements. In particular, we tried to observe to what extent the visual notation addressed the challenges discussed in Section 2.

First, the expressiveness problems were solved since we do not have a separate visual element for representing aspects (problem 1). Moreover, it is still straightforward to identify the set of components playing the role of aspects in the architectural design: it consists of all elements bound to crosscutting roles (the triangles in the aspectual connectors). Our visual notation also allows more intuitive and clear representations of different kinds of aspectual compositions (problem 2). For instance, the symbols used for before, after, and around have demonstrated to be a nice addition for both communication and measurement purposes. The sequencing operators can be often inferred from use cases and/or AO requirements documents. They are useful to distinguish different forms of coupling early in the design process, thereby facilitating application of AO architecture metrics [6].

It is true that some points in the architecture model might aggregate a number of visual elements, such as join points that are shared by multiple aspectual connectors. For example, like the two interfaces between GUI and Business layers (Fig. 4). However, it causes also a desirable effect: the architects and programmers should pay special attention to this part of the architecture since this is a point where multiple "aspects" interact. Visual means to express quantification were very useful in the HW architecture, where four cases of broadly-scoped aspectual connectors were identified. They were associated with synchronization, persistence, and distribution, and generic error handling issues. Hence, scalability-related impairments have been substantially reduced (problem 3). When more complex aspectual connectors were required, we exploited the resource of internal representations available in UML.

Even though the resulting visual language is much richer than the original asymmetric notation [19], a number of simplifications were also achieved. The notation meta-model no longer has abstractions and visual elements dedicated for aspects, aspect interfaces (i.e. crosscutting interfaces in the aspectual components), advice, and inter-type declarations. These elements are also present in almost all the asymmetric notations [1, 12, 13, 18] we analyzed. In addition, because we support the specification of multiple forms of crosscutting compositions, it also facilitates

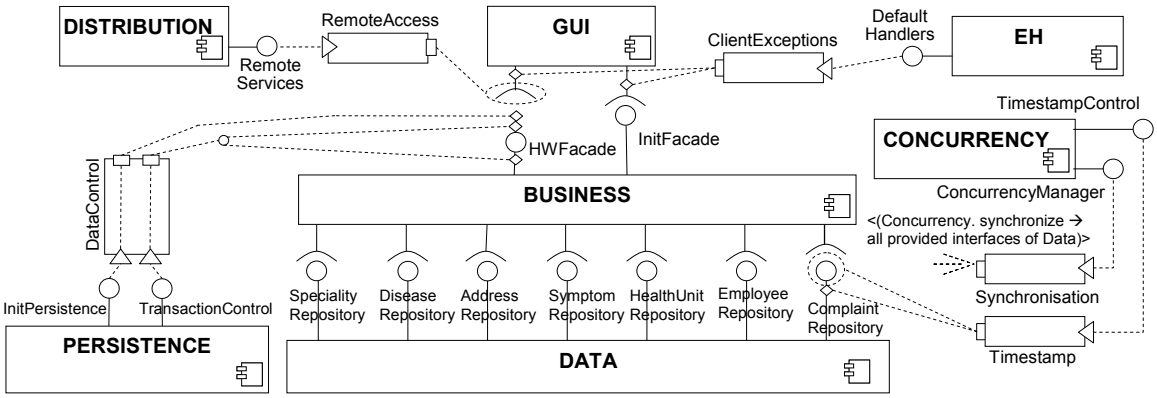

Figure 4. Symmetric Representation of the Health Watcher Architecture.

the transition of architectural design to detailed design. It is easier to identify which slice of the component boundaries is likely to be translated to a design or programming aspect.

Finally, we also tried to analyze whether benefits or drawbacks were obtained from the viewpoint of requirements-architecture transitions (problem 5). The symmetric visual notation was used in the context of an end-to-end methodology for AOSD [8, 9]. It has the objective of defining a single approach that, starting from aspect-oriented requirements, results in an aspect-oriented architecture specification [3]. The Auction System case study is used in [8] in order to illustrate the integrated approach. We omitted requirements and architectural specifications due to lack of space. More details can be found in [7-8].

A typical concern in the auction system case study is security, with a requirement specifying that "*Users have to log on to the auction system for each session*". Following a symmetric decomposition model, security, user and auction are modelled at the requirements level using the same element (concern, viewpoint, goal). The requirement used as example states that the security concern is related to the interactions among the user and the auction concerns, modelling a user that needs to be authenticated before buying and selling in an auction. During the mapping from requirements to architecture these concerns are mapped to components in the visual notation. Figure 5 shows the User, the Auction and the Security components. The 'log-on' verb in the requirement of the security concern is mapped to an operation of a provided interface of the Security component.

In the Auction System analysis, a crosscutting influence has been identified between security and the interaction among users and auctions, both at the requirements and at the architecture levels. Notice, however, that using the symmetric visual notation, there is no impediment to use Security as a non-aspectual component in other architectures. Thus, the Security component can be composed either as a non-aspectual or as an aspectual component with no difference in its component specification.

The decision of the role played by a component is taken during the specification of the connectors. Concretely, in the UserAuctionSecurity connector in Figure 5, the User and Auction components are connected to base roles of the connector, participating in the interaction as non-aspectual components, and the Security component is connected to a crosscutting role of the connector, participating in the interaction as an aspectual component. Notice that the crosscutting behaviour modelled by the Security component can be any operation defined as part of its provided interface (log-on interface in Figure 5). The kind of binding (sequencing in section 3.5), 'before' in this example, is also represented in the visual notation, as shown in Figure 5.

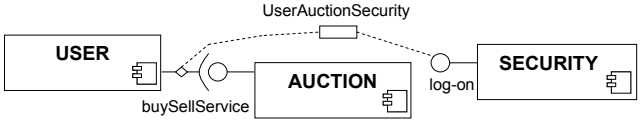

Figure 5. Mapping of the security non-functional requirement to the symmetric visual notation

5. FINAL REMARKS

Even though some ADLs (e.g. DAOP-ADL, AspectualACME, Fractal) have been proposed, they focus on the provision of a textual description. This paper presented a symmetric visual notation for representing AO software architectures. In our case studies, we observed that most of the expressiveness and scalability problems identified in existing asymmetric notations were addressed by our symmetric modelling approach.

As a next step, we are planning to enrich the visual notation with elements to express structural aspect-oriented compositions. In fact, this is a major limitation that we identified in the current visual notation. In an industrial-strength case study [4], we observed that more structural composition operators, such as merge and unification are also required in architecture specifications. We have been working on the definition of new operators as extensions to the xADL language [4], but have not reflected much about visual representation for such operators. However, we learned that the connector abstraction is potentially not the best abstraction to capture such structural compositions, as connectors have been historically explored for behavior-dependent architecture compositions.

6. REFERENCES

[1] O. Barais *et al*. TranSAT: A Framework for the Specifcation of Software Architecture Evolution. Ws on Coordination and Adaptation Techniques for Software Entities, ECOOP, 2004.
[2] T. Batista et al. Reflections on Architectural Connection: Seven Issues on Aspects and ADLs. Early Aspects at ICSE, 2006.
[3] M. Pinto, L. Fuentes. AO-ADL: An ADL for describing Aspect-Oriented Architectures. Early Aspects at AOSD, 2007.
[4] N. Boucke, A. Garcia, T. Holvoet. Composing Architectural Structures in xADL. Early Aspects at AOSD, 2006.
[5] S. Soares, E. Laureano, P. Borba. Implementing Distribution and Persistence Aspects with AspectJ. Proc. of OOPSLA, 2002.
[6] C. Sant'Anna, C. Lobato, C. Chavez, A, Garcia, C. Lucena. On the Quantitative Assessment of Modular Multi-Agent Architectures. NetObjectDays, 2006, Germany.
[7] R. Chitchyan, M. Pinto, A. Rashid, L. Fuentes. COMPASS: Composition-Centric Mapping of Aspectual Requirements to Architecture. Trans. on AOSD, vol. 4, pp. 3-53, 2007.
[8] R. Chitchyan et al. From Aspectual Requirements to Design. AOSD-Europe Newsletter, 2nd edition, Jan 2007.
[9] R. Pawlak et al. A UML Notation for Aspect-Oriented Software Design. Aspect Oriented Modelling at AOSD, 2002.
[10] R. Chitchyan et al. Semantics-based Composition for Aspect-Oriented Requirements Engineering. Proc. of AOSD 2007.
[11] R. Chitchyan et al. Survey of Analysis and Design Approaches, AOSD-Europe, Deliverable D11, 2005.
[12] U. Kulesza, A. Garcia, C. Lucena. Towards a Method for the Development of Aspect-Oriented Generative Approaches. Early Aspects at OOPSLA, 2004, Vancouver, Canada.
[13] M. Pinto, L. Fuentes, J. Troya. DAOP-ADL: An Architecture Description Language for Dynamic Component and Aspect-Based Development. LNCS 2830, 118-137, 2003
[14] S. Clarke and E. Baniassad. Aspect-Oriented Analysis and Design: the Theme Approach. Addison-Wesley, 2005.
[15] A. Garcia et al. On the Modular Representation of Architectural Aspects. European Ws. on Software Architecture, EWSA, 2006.
[16] P. Greenwood et al. On the Impact of Aspectual Decompositions on Design Stability: An Empirical Study. Proc. of ECOOP 2007.
[17]UML www.omg.org/technology/documents/formal/uml.htm
[18] M. Kande. A Concern-Oriented Approach to Software Architecture. PhD Thesis, Swiss Fed. Inst. Tech. (EPFL), 2003.
[19] I. Krechetov, B. et al. Towards an Integrated Aspect-Oriented Modeling Approach for Architecture Design. AOM at AOSD'06.