

Towards an Integrated Aspect-Oriented Modeling Approach for Software Architecture Design

Ivan Krechetov¹

Bedir Tekinerdogan¹

Alessandro Garcia²

Christina Chavez³

Uirá Kulesza⁴

¹Computer Science Department, University of Twente, The Netherlands

²Computing Department, Lancaster University, UK

³Computing Department, Federal University of Bahia – UFBA, Brazil

⁴Computer Science Department, Pontifical Catholic University of Rio de Janeiro – PUC-Rio, Brazil

{i.krechetov, bedir}@cs.utwente.nl, garciaa@comp.lancs.ac.uk

flach@dcc.ufba.br, uira@les.inf.puc-rio.br

ABSTRACT

There is now an increasing agreement that aspects are not only an issue at the programming level, but also tend to arise at the requirements analysis and software architecture design. Unfortunately conventional software modeling approaches lack abstractions to support the modular representation of crosscutting concerns in such development stages. Hence, several researchers have proposed modeling techniques that primarily aim for modeling aspects at the early development phases. Each of these approaches seems to focus on specific issues in modeling aspects and have their own benefits and drawbacks. This paper focuses on the modular representation of aspects at the software architecture design, and provides an attempt to integrate the best practices of the existing aspect-oriented (AO) architectural approaches in a single general-purpose modeling language. The integration is based on a systematic comparison of four modeling techniques. The aspect-oriented architecture modeling approach is based on the Unified Modeling Language (UML) and presented using an illustrative example on concurrent versioning systems.

Keywords

Aspect-oriented software architecture, visual modeling, UML.

1. INTRODUCTION

The software architecture of a computing system is a high-level design representation of the structure of the system, which comprises software components, the externally visible properties of those components captured by their interfaces, and the inter-component relationships [5, 25, 21, 20]. It is generally accepted that software architecture design should support the required software system qualities. For ensuring the quality factors the common assumption is that identifying the fundamental concerns for architecture design is necessary. A number of specification approaches have been introduced to derive the fundamental architectural abstractions. Although these approaches vary in deriving architectural abstractions they share the common idea that such abstractions should represent the relevant concerns of the system. However, some concerns, even at the architectural design level, can not be easily localized and specified with individual architectural units such as traditional interfaces and components. Similar to the notion of aspect at the programming level, we say that these concerns are crosscutting the architectural unit and denote the so-called *architectural aspects* [10, 11, 3, 2].

A crosscutting concern at the architecture design level could be any concern that cannot be captured in the conventional architectural units. Architectural aspects can be adopted to define a common behavior or structure of architectural elements. Example of a crosscutting architectural concern is to ensure recovery for every architectural unit. Another example is the platform concern for the architectural units. These concerns can not be easily mapped to a single unit in conventional architectural modeling. Very often, the crosscutting property of the architectural concerns remains either implicit or is described in informal ways leading to reduced uniformity, impeding traceability and hindering detailed design and implementation decisions.

Unfortunately, conventional architecture modeling approaches do not appropriately support the modular representation of architectural aspects. Hence, several researchers [13, 14, 8, 4, 23] have proposed UML-based modeling approaches that primarily aim at providing proper abstractions and corresponding visual notations to modularly capture architectural crosscutting concerns. Each of these approaches seems to focus on specific issues in modeling aspects and have their own benefits and drawbacks. This paper provides a first attempt to integrate the best practices of the aspect-oriented software architecture design approaches in a single general purpose modeling language. The integrated modeling approach has been mostly carried out in the context of the AOSD-Europe project. After having defined a survey on aspect-oriented architecture design approaches [10], we have selected the key aspect-oriented architecture design approaches, and compared and analyzed them with respect to their ability to model the required aspect-oriented concepts. The paper presents the main important results of this work. The aspect-oriented architecture modeling approach is presented using an illustrative example on concurrent versioning systems.

This paper is structured as follows. Section 2 presents the concepts that should be addressed by an AO modeling approach, and the desired traits of the approach itself. Section 3 describes the aspect-oriented architecture modeling approaches that have been considered as the objects for the integration effort. Section 4 presents the integrated UML-based approach for aspect-oriented architecture modeling. Section 5 adopts the integrated modeling technique to represent architectural aspects of a concurrent versioning system (CVS). Section 6 contains a general discussion and an assessment of the unified modeling approach in the

presented case study. Finally, Section 7 presents our conclusions and final remarks.

2. MODELING ASPECT-ORIENTED SOFTWARE ARCHITECTURES

Section 2.1 will present the key aspect-oriented concepts that we think need to be captured at the architecture design stage. Then, we'll formulate the requirements for a general-purpose AO modeling notation.

2.1 AO Modeling Concepts

We have defined a comparison framework that depicts a core set of concepts that, according to our point of view, should be supported by an aspect-oriented architecture description. Such concepts were derived from a joint analysis of an AOSD glossary [6] and the conceptual frameworks defined in [15, 20, 18]. We have aggregated and filtered the terms found in those sources, for marking out the most vital issues for the architectural stage. 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 [25, 21]. As a result, our comparison framework is composed of the following AOSD concepts.

- C1. Aspect.
- C2. Component.
- C3. Point-cut.
- C4. Advice.
- C5. Static and dynamic crosscutting.
- C6. Aspect-component relation.
- C7. Aspect-aspect relation.

The concept *component* (C2) is considered, as it is an important conventional (non-aspect-oriented) construction element, which is typically a container of crosscutting concerns at the architectural level. The importance of the *component* concept is also confirmed by [15], where it plays a key role in aspect-oriented middleware reference architecture. Aspect (C1), point-cut (C3), and advice (C4) are included because they are often considered the primary abstractions to capture crosscutting concerns [6, 15, 20, 18]. Note that considering C5 we shall concentrate on the *crosscutting* itself, leaving the *weaving* issue [6, 18] out of the focus. The two categories of relations – aspect-component (C6) and aspect-aspect (C7) are important because they represent which and how architectural building blocks are collaborating to realize the system requirements.

2.2 Desired Properties for an Aspect-Oriented Modeling Notation

There is a need in general purpose visual notation for the aspect-oriented architecture modeling. Our integration guidelines for the proposed unified approach are comprised of the following requirements:

- R1. Visual modeling language should be general-purpose and UML-based.
- R2. It should be complete, which means having a supporting abstraction for each of the commonly accepted AOSD concepts (C1-C7). Besides, different concepts should be implicitly or explicitly designated into different existing or new first-class UML elements.

- R3. It should be implementation language independent, until the lowest level of detail is involved. In this way, the resulting aspect-oriented architectural models should be easily mapped to elements of distinct aspect-oriented programming languages/frameworks and detailed design notations.
- R4. Finally, the integrated UML-based notation should promote simplicity and avoid unnecessary extensions.

3. COMPARING THE AO MODELING APPROACHES

In section 3.1 we shall present the selected source approaches. Then, section 3.2 provides the comparative analysis.

3.1 Analyzed Approaches

Taking a comprehensive survey [10] as a reference, we have chosen the four source architecture modeling approaches, because each of them: (a) focuses on the architecture design stage; (b) provides an aspect-oriented visual notation and, optionally, a meta-model. The selected techniques are:


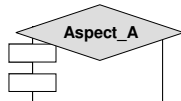
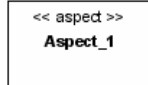

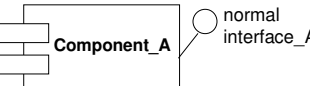
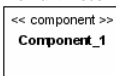
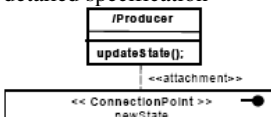
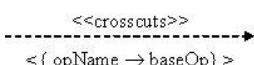
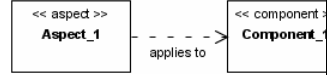
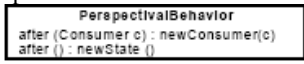
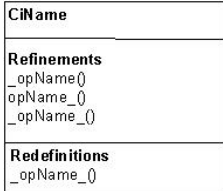
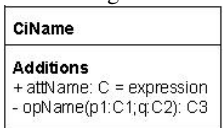
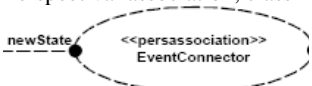
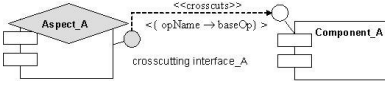
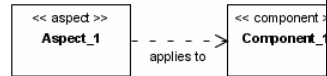
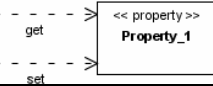
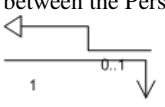
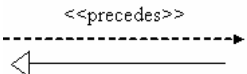
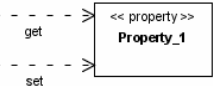
1. PCS Framework [13].
2. AOGA [14].
3. TranSAT [4].
4. CAM of DAOP-ADL [23].

PCS Framework, AOGA, TranSAT and CAM provide visual notations for representing architectures. In general these visual notations extend UML [22] to represent aspect-oriented concepts at the architecture design level. PCS Framework notation intensively utilizes the AspectJ syntax for the design details specification. The AOGA approach extends the aSideML notation [8] to handle architectural aspects. At the moment of writing this paper the TranSAT visual meta-model was not completely expressed by the corresponding notation. The existing TranSAT notation elements are just the high level components and connectors; but we shall anyway benefit from taking into consideration the TranSAT meta-model concepts. As for CAM notation, it is a visualization of the XML-based DAOP-ADL architecture description language.

3.2 Comparative Analysis

Table 1 summarizes the modeling of commonly accepted AOSD concepts in PCS Framework, AOGA and CAM. It presents a comparison of the investigated approaches according to our classification framework (section 2.1). Our goal here is not extensively describe each concept of such existing approaches; for further details the reader should refer to the original papers [13, 14, 4, 23]. Due to the space limitations, we do not show the TranSAT notation elements in Table 1, as they would only appear in the *Component* row (as UML component), and the *Aspect-component relation* row (as UML connector). TranSAT is the most concise notation, albeit lacking the direct visual support for several architecturally-relevant AOSD concepts, such as aspect (C1), point-cut (C3), static and dynamic crosscutting (C5), and aspect-aspect relations (C7). We can observe that all the approaches present different abstractions and associated graphical elements to support the modeling of the AOSD concepts. The only clear agreement is the use of UML modules (components or classes) to capture the component concept (C2). Section 4 discusses the differences of the chosen techniques and identifies the abstractions for the integrated approach according to the integration guidelines (section 2.2).

Table 1. Source approaches and their support to the AOSD concepts

	PCS Framework	AOGA	CAM of DAOP ADL
Aspect	<p>Perspectival association</p> 	<p>UML Component with diamond decoration.</p> 	<p>UML class with <<aspect>> stereotype</p> 
Component	<p>UML component</p> 	<p>UML Component with interfaces.</p> 	<p>UML class with <<component>> stereotype</p> 
Point-cut	<p>Connection point, join-point, attachment. AspectJ syntax is used for a detailed specification</p> 	<p><<crosscuts>> relation with bindings. Each binding relates a placeholder from the aspect's crosscutting interface to a component element</p> 	<p><<applies to>> association with constraints</p> 
Advice	<p>PerspectivalBehavior section of a PerspectivalAssociation element. AspectJ syntax is used for a detailed specification</p> 	<p>The Refinements and Redefinitions compartments of a crosscutting interface list operations to be combined before, after, before/after or to override base operations</p> 	<p>None</p>
Static and dynamic crosscutting	<p>Introduction section of a PerspectivalAssociation element. AspectJ syntax is used for a detailed specification.</p>	<p>The Additions compartment of a crosscutting interface</p> 	<p>No static crosscutting supported</p>
Aspect-component relation	<p>Perspectival association, classifier roles</p> 	<p>The crosscutting relationship relates aspects to base components. It may include a list of bindings..</p> 	<p>1. <<applies to>> association with constraints.</p>  <p>2. <<get>> and <<set>> associations between a components, aspects properties.</p> 
Aspect-aspect relation	<p>Inheritance and precedence relation between the PerspectivalAssociations.</p> 	<p>Crosscutting relationship and inheritance between aspects. Several kinds of dependency: <<precedes>>, <<requires>>, <<xor>>.</p> 	<p><<get>> and <<set>> associations between aspects and properties.</p> 

4. THE INTEGRATED AO MODELING APPROACH

Guided by the requirements R1-R4, in the sections below we motivate and present our choices based on the source approaches the expressive elements for the integrated UML-based visual modeling approach. Table 2 gives short examples for all the visual elements proposed in the integrated aspect-oriented modeling approach.

4.1 Aspect

We prefer to designate a separate architecture element for an aspect (R2), not mixing it with the aspect relations – unlike the way it is done in PCS Framework approach. We keep the separate elements for aspects, point-cuts, introductions and binding. Thus we take a stereotyped component for a visual aspect representation, just as in AOGA (equivalently) and TranSAT approaches. In contrast with CAM of DAOP-ADL we choose the UML component, not the UML class as a base to make aspects the more coarse-grained elements. This suites better for the architecture design stage, and contributes to both R1 (general-purpose-ness) and R2.

4.2 Component

Naturally, just as all of the evaluated visual notations do, we choose UML *Component* for (C2) concept representation.

4.3 Crosscutting Interface

We propose the use of aspect interfaces to capture the crosscutting influence of the aspectual components. It captures the notion of pointcuts and advice at the architectural models. For the representation of aspect interfaces we combine the features of three different approaches: *crosscutting interfaces* from AOGA, *evaluated interface* from DAOP-ADL (CAM), and the *point-cut labels* from TranSAT. This decision was taken to satisfy R1. The notion of crosscutting interface is a seamless extension to the notion of interfaces in UML (R1, R2, and R4), in which point-cut labels capture the crosscutting nature on how an aspect affects the other architectural components (R2). It also allows for abstracting the different ways in which different programming languages and detailed design modeling languages implement composition. The *crosscutting interface* with *point-cut labels* is presented as an UML stereotyped interface with method names being the point-cut labels. Thus, if an aspect has a *crosscutting interface* attached, this means it operates on certain points in the execution flow bound to the *point-cut labels* of the *crosscutting interface*. It is also possible to leave the *crosscutting interface* empty (no *point-cut labels* bound), and postpone the point-cut specification until later development stages. This strategy contributes to R3.

The members of a crosscutting interface are not exactly the class methods in the sense of object-oriented programming. They are actually instances of a generalization of the method concept that must be bound to certain points in the evaluated component's code, in a way for the component to adhere to the *crosscutting interface* contract [7, 9]. Note that this *crosscutting interface* with *point-cut labels* facility resembles the connection point – attachment combination in the PCS Framework approach. Only the composition rules are omitted. As far as PCS Framework utilizes AspectJ syntax for the composition specification, due to (R3) we incline to choosing other means for a detailed point-cut

definition. How exactly it will be done is now an open question. As an option, this may be a sub-component of the aspect, *realizing* the composition rules (or point-cuts) for the *crosscutting interface*, basing on the *point-cut labels* – just as the *point-cut mask* does in the TranSAT meta-model.

None of the evaluated approaches provide a notation for advice which conforms well to (R1-R3) requirements. The closest is the *PerspectivalBehavior* from PCS Framework, but it does not align with R3. Consequently, we raise another open question here – the advice representation. Like the point-cut representation, an advice may be modeled as a sub-component, which gets a delegation from the enclosing aspect (UML *delegation* connector) to *realize* the *advice* for the *crosscutting interface*. It may be convenient to introduce commonly used stereotypes of advices, like transformation of message signature, conditional filtering, and others.

4.4 Static and Dynamic Crosscutting

For the dynamic crosscutting modeling the integrated approach supports *point-cut* and *advice* concepts. To model the static crosscutting we combine the following available concepts from the observed approaches: *Introduction* element from PCS Framework and the *crosscutting interface to component* relation from AOGA.

Namely, we state the “crosscuts” unidirectional association between a *crosscutting interface* of an *aspect* and a *component* to indicate that there is a static crosscutting within the component structure. For a detailed specification of the static crosscutting an *introduction* sub-component of the aspect can be used. It abstracts the structural changes that are necessary for a target component to hold the *crosscutting interface* contract. Again, the *delegation* UML connector may be used to link the *introduction* and the served *crosscutting interface*.

PCS Framework utilizes the AspectJ syntax to particularize the inter-type declarations brought by an *Introduction*. We have to find an implementation language-independent representation (R3).

4.5 Aspect-Component Relation

We propose to reuse aspect-component relation from AOGA approach, formulating it as follows. A *crosscutting interface* crosscuts either internal elements of a *component* or other interfaces. The first case means that the aspect directly affects the internal structure or dynamic behavior of the target component. The second case means that the aspect affects the contract defined by other interfaces.

Aspect-component relation is effectively a binding of an aspect to a component. Apparently, each individual binding needs a detailed specification at the design stage. That's what *ConnectionPoint's Composition* in PCS Framework, *integration rules* in TranSAT and the list of bindings from AOGA stand for. To keep the aspect implementation and aspect binding decoupled (R2), it may be reasonable to provide an association class for the aspect-component relation, or introduce association tagged values for this purpose. This will be an encapsulation of binding details.

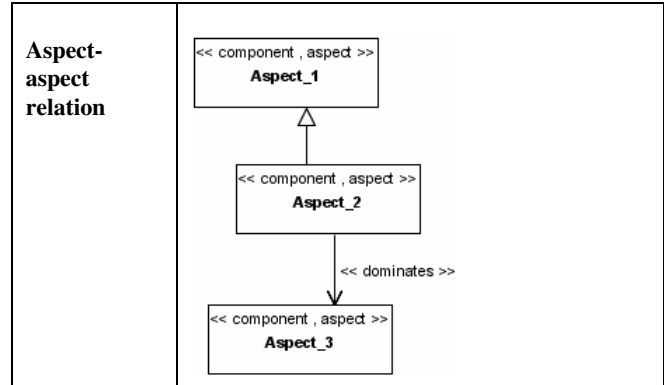
4.6 Aspect-Aspect Relation

It is proposed to take the inheritance and precedence relation from the PCS Framework. This gives the means for defining the

“abstract” aspects that may be detailed and refined by successors, and for setting aspect priorities. In UML we’ll model this as *generalization* and “dominates” navigable binary association respectively.

Table 2. Integrated approach relation to the AOSD concepts

Concepts	Means for visual modeling
Aspect	
Component	
Point-cut	
Advice	
Static and dynamic crosscutting	
Aspect-component relation	



5. EXAMPLE: CVS

To illustrate the proposed modeling approach, we will take a case study based on a Concurrent Versioning System. Due to space limitations, we will show the architectural models based on the more condensed views of our proposed integrated approach.

5.1 Problem Statement

Software Configuration Management (SCM) deals with control of software changes, proper documentation of changes, the issuing of new software versions and releases, the registration and recording of approved software versions. An important functionality in SCM forms the concurrent version control system (CVS). A concurrent version control system keeps a history of the changes made to a set of files that can be concurrently accessed. The CVS provides functions for checking in and checking out files. When files are checked out they can be edited and compiled. Afterwards it is possible to check in the modifications to the file. Checking out a file does not give a developer exclusive rights to that file. Other developers can also check it out, make their own modifications, and check it back in. The CVS should automatically detect when multiple developers make changes to the same file and merge those changes. To support the versioning the CVS maintains a history of a source tree, in terms of a series of changes. It stamps each change with the time it was made and the user name of the person who made it. The developers can provide a textual description describing why they made the change as well. Given that information, CVS can help developers to inspect who made the given change, why they did it, when they did it. *Developers* can check files, change files, merge the changes, commit the changes and check out files. The *Administrator* can initiate the repository in which the shared files are stored, set profiles for the developers, monitor activities of developers and set the authentication policies.

5.2 The Model

The first diagram (Figure 1) gives the CVS architecture overview.

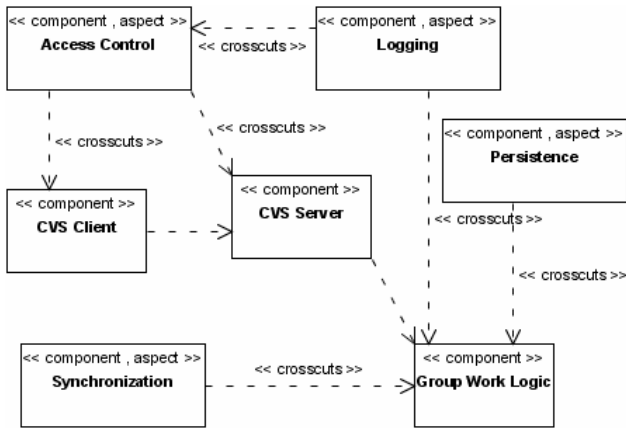


Figure 1. CVS architecture expressed in the Integrated modeling language.

Let's take a little closer look at the *Access Control* aspect. The Integrated modeling language diagram at Figure 2 shows the crosscutting interfaces this aspect provides, and the components and interfaces that are the subjects to the crosscutting functionality.

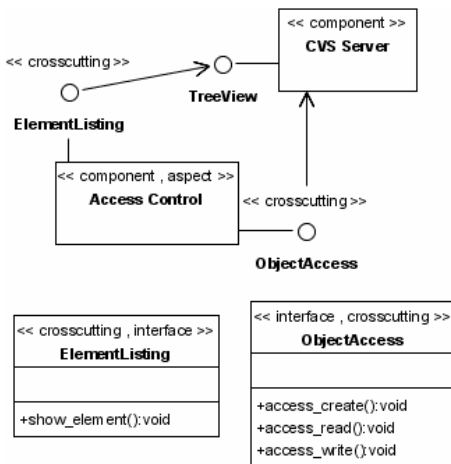


Figure 2. Crosscutting of the Access Control aspect.

ElementListing crosscutting interface defines a single *show_element()* point-cut label, which is bound to the join-points within the *TreeView* interface of *CVS Server* component. Another crosscutting interface defined by the *Access Control* aspect is *ObjectAccess*. Point-cut labels of this crosscutting interface are bound to the business objects' create, read and write operations. On the diagram we generally show this by a relation of the crosscutting interface to the *CVS Server* component. These relations are to be detailed on the later development stages.

6. DISCUSSION

The architecture design activity should be integrated into the complete software development life-cycle. This means the smooth transition of the connected stages and traceability of the involved artifacts. It must be clear how the elements of the requirements model (concerns, goals, constraints, guidelines, and others) are reflected in the architecture model. Moreover, it must be possible to validate and evaluate the architecture against

the stated requirements. On the other hand, the software architecture model must be an appropriate input for the software design stage – a good architecture modeling facility should permit the further refinement and detailing. The adoption of UML based aspect-oriented approaches strongly benefits a seamless integration between architectural artifacts and design-level classes and aspects. Our proposed integrated architectural approach takes it into consideration by inheriting from the studied approaches several UML based elements. By reusing many concepts from the aSideML approach, we can also obtain the benefit of supporting traceability by explicitly linking elements of our architectural model to corresponding elements in aSideML detailed design models, such as class and sequence diagrams [7].

In the example presented in section 5, we can observe the advantages brought by the integrated approach as a consequence of the integration guidelines defined (section 2.2). The resulted visual modeling language (R1) makes explicit the architectural aspects which address important crosscutting concerns of the CVS, such as, Access Control, Logging, Persistence and Synchronization. Each aspect is specified by presenting their interaction with the CVS components and the other aspects. Both component and aspect interfaces help to detail the way these elements interact. Finally, the aspect interfaces specify clearly which component join points a specific architectural aspect is interested to observe and possibly affect the functionality. All the elements adopted by the integrated approach are UML based (R2). Most of them derived from the aSideML approach.

The CVS AO architecture presented in Figure 1 and Figure 2 is also programming language independent (R3). The integrated modeling language allows specifying the high-level components and aspects of the system by omitting implementation details related to specific AO programming languages. Finally, our integrated modeling language is well-aligned with the new elements from UML 2.0, the standard adopted by the industry. It promotes simplicity in the sense that it does not introduce new elements (R4). Therefore, it may be easier learnt by the UML users and supported by the existing tools.

7. CONCLUSION

In this paper, we investigated and compared three aspect-oriented architecture design approaches that had been published before: PCS Framework, AOGA, TranSAT and CAM. Our aim was to specify a unified general-purpose aspect-oriented architecture modeling approach.

In order to analyze the proposed approaches we defined a comparison framework (section 2.1) that depicts a core set of concepts which we believe should be supported by an aspect-oriented architecture description. Our analysis of the investigated approaches (section 3.2) showed that these concepts are not completely fulfilled by each of the approaches. PCS Framework is too close to implementation and hard-linked to AspectJ. The AOGA approach encompasses means for elaborating the point-cuts, advice and static crosscutting through an expanded view. Many of the CAM features serve the specifics of DAOP platform, focusing on the implementation details, and thus weakening universality. At the moment, the means of visual modeling in TranSAT are limited to the

expressive power of UML 2.0 component diagrams. Even though an aspect-oriented meta-model is provided, it is not realized in the visual modeling facilities yet, ending up in the XML back-end only.

We established a set of requirements (section 2.2) to work as integration guidelines in the definition of a unified general-purpose aspect-oriented architecture modeling approach. Based on these requirements, we selected from the investigated approaches their respective expressive elements (section 4). The integration between the approaches was more difficult than we expected, basically because of the different conceptual views on (aspect-oriented) software architecture. There are very useful elements in each approach (like *point-cut labels* in TranSAT, *crosscutting interfaces* in AOGA, *connection points* and *introductions* in PCS Framework, *evaluated interface* in CAM), but directly combining these in a single approach was more difficult and we had sometimes to provide radical decisions to integrate the elements in one framework. We have to note that even after our integration efforts there are still some aspect-oriented concepts that are not addressed by the studied approaches and as such are not yet directly supported by the integrated approach. This will provide us a good basis for our further research activities in aspect-oriented architecture design.

Obviously, this study shows that we can benefit from the existing aspect-oriented architecture design approaches that have been published so far. We think that we have also provided a first important integrated approach that integrates the best practices. However, another important conclusion is that still more research is required on aspect-oriented architecture design. We hope and think that our work will provide an important input for these activities.

8. ACKNOWLEDGEMENTS

This work is supported by European Commission Grant IST-2-004349: European Network of Excellence on AOSD (AOSD-Europe).

9. REFERENCES

- [1] R. J. Allen, "A Formal Approach to Software Architecture", Ph.D. Thesis, Carnegie Mellon University, CMU Technical Report CMU-CS-97-144, May 1997.
- [2] Araújo, et al. "Early Aspects: The Current Landscape," technical note, CMU/SEI and Lancaster University.
- [3] Baniassad, et al. "Discovering Early Aspects," IEEE Software, Jan/Feb 2006.
- [4] O. Barais et al, "TranSAT: A Framework for the Specification of Software Architecture Evolution," Workshop on Coordination and Adaptation Techniques for Software Entities, ECOOP 2004, Oslo, Norway, 2004.
- [5] L. Bass, P. Clements, and R. Kazman, Software Architecture in Practice: Addison-Wesley, 1998.
- [6] K. van den Berg, J. M. Conejero and R. Chitchyan, "AOSD Ontology 1.0 – Public Ontology of Aspect-Oriented", AOSD-Europe, 2005.
- [7] C. Chavez, and C. Lucena. A Theory of Aspects for Aspect-oriented Software Development. Proc. of the XVII ACM Sigsoft Brazilian Symp. on Soft. Engineering, October 2003.
- [8] C. Chavez "A Model-Driven Approach to Aspect-Oriented Design", PhD Thesis, Computer Science Department, PUC-Rio, April 2004.
- [9] Chavez, C., Garcia, A., Kulesza, U., Sant'Anna, C., Lucena. Taming Heterogeneous Aspects with Crosscutting Interfaces. Journal of the Brazilian Computer Society, SBC, Jan 2006.
- [10] R. Chitchyan, A. Rashid, P. Sawyer, A. Garcia, M. Pinto, J. Bakker, B. Tekinerdogan, S. Clarke, and A. Jackson, "Survey of Analysis and Design Approaches", AOSD-Europe, 2005.
- [11] C. Cuesta; et al. Architectural Aspects of Architectural Aspects. 2nd European Workshop on Software Architecture (EWSA), LNCS 3527, pp. 247-262, 2005.
- [12] D. Garlan, R. T. Monroe, and D. Wile, "Acme: Architectural Description of Component-Based Systems", *Foundations of Component-Based Systems*, G. T. Leavens and M. Sitaraman (eds), Cambridge University Press, 2000, pp. 47-68.
- [13] M. Kande, "A concern-oriented approach to software architecture," PhD. Lausanne, Switzerland: Swiss Federal Institute of Technology (EPFL), 2003.
- [14] U. Kulesza, A. Garcia, and C. Lucena, Towards a Method for the Development of Aspect-Oriented Generative Approaches. Workshop on Early Aspects, OOPSLA'04, November 2004, Vancouver, Canada.
- [15] N. Loughran et al, "Requirements and Definition of AO Middleware Reference Architecture", AOSD-Europe, 2005.
- [16] D. Luckham, "Rapide: A Language and Toolset for Simulation of Distributed Systems by Partial Orderings of Events", Paper presented at DIMACS Partial Order Methods Workshop IV, Princeton University, July 1996.
- [17] J. Magee, N. Dulay, S. Eisenbach and J. Kramer, "Specifying Distributed Software Architectures", Proc. of 5th European Software Engineering Conference (ESEC '95), Sitges, September 1995, LNCS 989, 1995, pp. 137-153.
- [18] H. Masuhara, G. Kiczales: Modeling Crosscutting in Aspect-Oriented Mechanisms. Proc. of ECOOP 2003: 2-28
- [19] N. Medvidovic et al. "Using Object-Oriented Typing to Support Architectural Design in the C2 Style", Proc. of the FSE'96, p.24-32, ACM, San Francisco, CA, October, 1996.
- [20] N. Medvidovic et al. Modeling software architectures in the UML. ACM TOSEM, 11(1):2--57, Jan. 2002.
- [21] N. Medvidovic, R. Taylor. A Classification and Comparison Framework for Software Architecture Description Languages. IEEE Trans. Soft. Eng., 26(1):70-93, Jan 2000.
- [22] Object Management Group, "Unified Modeling Language: Superstructure. Version 2.0", available at <http://www.omg.org>, 2005.

- [23] M. Pinto, L. Fuentes, and J. M. Troya, "A Dynamic Component and Aspect Platform", *The Computer Journal*, 48(4):401-420, 2005.
- [24] P. Schmidt et al., "A Systems Engineering Perspective of Aspect-oriented Software Architectural Analysis using UML", 3rd aspect-oriented modeling workshop at AOSD 2003, Boston, MA, March 2003.
- [25] M. Shaw and D. Garlan, *Software Architectures: Perspectives on an Emerging Discipline*. Englewood Cliffs, NJ: Prentice-Hall, 1996.