Modeling and Evolving Crosscutting Concerns in ADORA

Silvio Meier, Tobias Reinhard, Reinhard Stoiber and Martin Glinz
Department of Informatics, University of Zurich, Switzerland
{ smeier | reinhard | stoiber | glinz }@ifi.unizh.ch

Abstract

For an effective handling of crosscutting concerns during the software process, adequate support is required not only in design and coding, but also in requirements engineering. For this purpose, we have developed an aspect-oriented extension of the requirements modeling language ADORA. *In this paper, we present an extension of our approach which makes it capable of supporting the evolution of aspect-oriented requirements models for both functional and non-functional aspects.*

1 Introduction

In the past few years, early aspects and their identification and separation have gained more and more attention in the research field of Aspect-Oriented Software Development (AOSD). However, the current approaches have several problems: some support functional crosscutting concerns only, others do not support evolution or can handle textually represented requirements only (c.f. Section 6).

We have developed a technique for visually modeling crosscutting concerns (CCCs) as aspects at the requirements and architectural stage for the modeling language ADORA [7]. However, this approach does not support the evolution of aspects and is restricted to functionally represented requirements. In this paper, we present an extension of our approach which makes it capable of modeling *high-level non-functional requirements* and supporting the *evolution* of both functional and non-functional CCCs from a very coarse level to precise requirements. The main contribution is the ability to model *partial aspects* and *abstract join relationships*. With these features, we support (i) modeling of non-functional requirements as aspects, (ii) early separation of crosscutting concerns in the requirements engineering process and (iii) systematic evolution of aspects in the software process. The remainder of the paper is organized as follows: Section 2 briefly presents our terminology, discusses the connection between non-functional requirements and CCCs and introduces the basics of the ADORA modeling approach. Section 3 summarizes the aspect-oriented extension for modeling functional aspects in ADORA. Section 4 presents an extension of aspect-oriented ADORA with partial aspects and abstract join relationships. Section 5 exemplifies the approach by presenting an extended example. Section 6 discusses related work. Section 7 concludes the paper with a short discussion of achievements, limitations, and future work.

2 Background

2.1 Terminology

We define the term *concern* according to [17]: *"a concern is any matter of interest in a software system"*.

Crosscutting is defined as a relationship between two artifacts *A* and *B* where *A* constrains *B* and *B* has no influence on the way how it is constrained by *A*. This definition conforms to those used by others, e.g. [12, 9, 13]. In our approach, crosscutting relationships are called *join relationships*. A concern is called *crosscutting* if it manifests in artifacts that crosscut other artifacts. Any non-crosscutting concern is called a *conventional concern*.

2.2 Non-functional Requirements vs. Crosscutting Concerns

There is numerous work in the field (e.g. [8, 12, 10]) which supports the hypothesis that non-functional requirements (NFRs) are a major source for aspects. Basically, there are two arguments for treating non-functional requirements as crosscutting concerns. Firstly, most NFRs constrain the system to be built in a crosscutting manner. Secondly, it is desirable to state every single requirement as a separate entity [12, 14]. For NFRs, this can be achieved by treating NFRs as crosscutting concerns and stating them as aspects.

We distinguish three different types of NFRs: NFRs of type (i) eventually evolve into a concrete piece of code and contribute directly to the functionality of the system. NFRs of type (ii) result in decisions at the architectural, design

or implementation stage of the software process. They do not trace to any specific piece of code, but influence how a system is built and executed. Hence, NFRs of type (ii) are observable when a system is executed. Finally, NFRs of type (iii) evolve to decisions which do not contribute directly to *what* or *how* the final system performs, but rather influence the software process. For example, a security requirement that evolves into concrete functionality such as authentication or encrypted transmission is of type (i). Performance requirements are of type (ii). Maintainability is a typical NFR of type (iii).

2.3 ADORA– An Object-Oriented Requirements and Architecture Language

ADORA is a language for modeling requirements specifications and software architectures [4]. Fig. 1 shows a library system described as a typical ADORA model, which is used as an example throughout this paper.

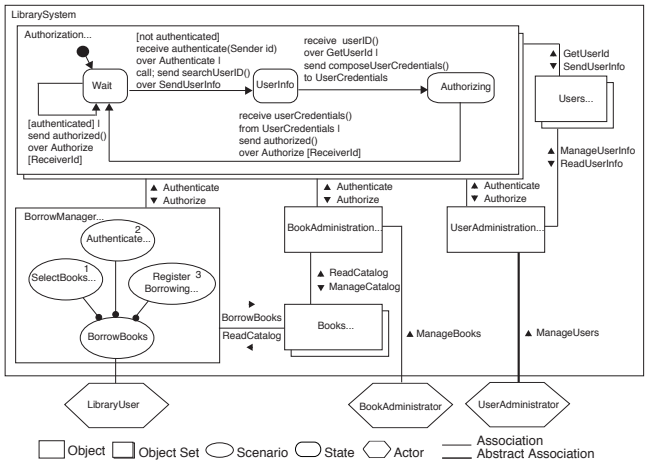

Figure 1. A part of a library system modeled in ADORA

A major difference between ADORA and other object-oriented modeling languages is that ADORA uses abstract objects (i.e. prototypical objects that have a name, but are not instantiated with attribute values) instead of classes as the basic modeling elements [4]. Using abstract objects allows hierarchical decomposition of models in a straightforward way with a simple and clear semantics. Decomposition, in turn, yields abstraction and the possibility of visualizing components in their context, thus making models easier to understand and evolve.

In Fig. 1, all rectangles represent abstract objects. Hierarchical structure is modeled by nesting objects, which in turn implicitly describes part-of-relationships between objects. Shadowed rectangles are object sets, i.e. they model collections of objects. For example, the library system contains an abstract object *BookAdministration* and an object set *Books*. Abstract objects and object sets are also called components.

For visualization, ADORA employs *views*. All views are generated from a common underlying model so that multiple views are always consistent among each other. The so-called *base view* consists solely of the hierarchical structure built of the objects and object sets. Beside this base view, additional views describe other facets of the system. In Fig. 1, all views are visible in combination. The *structural view* comprises the associations between objects and/or object sets, and the associations between actors and scenarios. Associations model information flow. Hence, associations may model directed structural relationships as well as the communication between components. The *behavior view* integrates states (drawn as rounded rectangles) and state transitions into the object hierarchy at all places where the internal behavior of the system has to be modeled. Together with the decomposition hierarchy, we can define the semantics of the behavior view with a simplified version of a statechart semantics. The *context view* shows the external actors (drawn as hexagons) of the system which are connected by associations to type scenarios [4]. Type scenarios are equivalent to use cases in UML. The scenarios (drawn as ovals) build the *user view* of an ADORA model. As interaction is frequently local, the scenarios are embedded in the object hierarchy at the position where they apply. Scenarios can also be decomposed into sub-scenarios, using a modified form of Jackson's JSP diagrams [5] as notation. In our modified version of JSP diagrams, iteration is a property of all possible node types. We also added parallel decomposition and made the notation layout-independent by numbering sequences of actions. The *functional view* describes the properties (i.e. the attributes and operations) of components. This view is not combined with other views and therefore not visualized with the rest of the model.

The ADORA language allows both semi-formal and formal modeling. Semi-formal means that some elements of a model don't abide by the formal semantics of the modeling language. A typical example is a state transition (a formal concept) for which the triggering condition is given in natural language.

Furthermore, ADORA supports partial models, i.e. models that are intentionally incomplete [15, 18]. In a partial model, some parts have not been modeled yet or will not be modeled at all. The difference to unintentional incompleteness is that the incomplete elements are known to be incomplete and therefore marked as such. Partial modeling is particularly useful in an evolutionary requirements modeling process, where we want to evolve a model in a controlled way through a series of iterations. In ADORA, we have two constructs for describing partial models: the first

one is the so-called *is-partial* property which indicates that a component is incomplete. This is especially useful if a system part is incomplete, imprecisely defined or at a coarse level, i.e. the system part is intended to be evolved further. The second construct is the so-called abstract association, which is represented as a bold line (see, for example, the association from *UserAdministrator* to *UserAdministration* in Fig. 1). *Abstract associations* can be used if the modeler knows that there is some communication between components, but at the time of modeling it is not clear how the concrete communication will look like. Note that ADORA supports not only partial *models*, but also partial *views* [4]. Partial views are an abstraction mechanism that is used for deliberately hiding certain model elements or levels of detail from a diagram (e.g. when a high-level abstract view of a system is desired). Partially viewed or modeled elements are indicated by names with three trailing dots (e.g. the object *BorrowManager*). In [15] we have sketched a process for evolving ADORA models.

3 Modeling Functional Crosscutting Concerns as Aspects in ADORA

In this section, we briefly present our aspect-oriented extension of ADORA [7]. This extension allows to model crosscutting functionality as aspects, thus yielding a better modularization of ADORA models. A modeler can switch between the aspect-oriented view of an ADORA model and a normal view where all aspects are woven into the model. This feature helps modelers to understand models with crosscutting parts better.

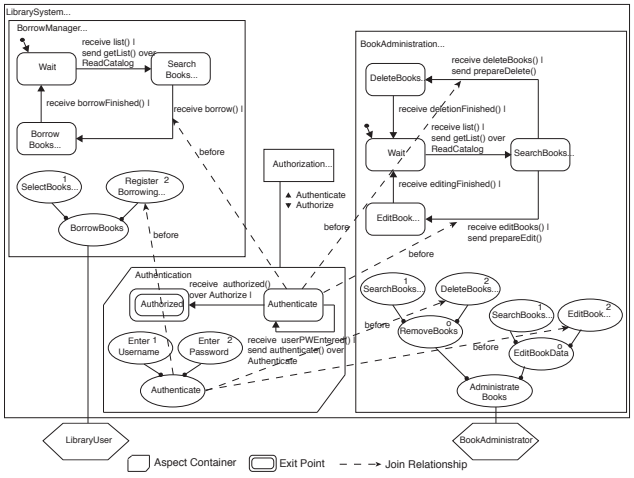

Figure 2. The library system containing an authentication aspect.

The aspect-oriented extension of ADORA is based on three concepts: (i) *Aspect containers* represent modules of crosscutting concerns (CCCs)[1] and comprise a description of crosscutting elements, such as behavior and scenarios. (ii) *Join relationships* denote explicitly where the CCCs affect other concerns. (iii) View mechanisms provide abstraction for aspect-oriented ADORA models.

Aspect Container. In an ADORA model, an aspect container is represented as a beveled rectangle that encapsulates the system's internal behavior, type scenarios and other elements of a CCC. In Fig. 2, a partial view of the Library System (see Fig. 1) is visualized, containing the *Authentication* concern as an aspect.

Behavior Chunks. The internal behavior is described by a statechart chunk, i.e. a fragmentary statechart. As a behavior chunk is not a self-contained description of the behavior, it does not have a start state. A *behavior chunk* contains one unique exit point for the crosscutting behavior. The exit point is denoted by a rounded rectangle with a double outline. In Fig. 2, the crosscutting behavior is described by the *Authenticate* state, its out-going transitions and the exit point. The actions that might be taken from this state are described by a reflexive transition triggered by the *userPWEntered()* event and another transition triggered by the *authorized()* event. The latter transition ends at the exit point of the crosscutting behavior, which denotes the point where the crosscutting behavior is left.

Scenario Chunks. So-called *scenario chunks* model the crosscutting type scenarios of a CCC. A scenario chunk is a fragmentary scenario model and consists of nodes and scenario connections as described in Section 2.3. In Fig. 2, the authentication, as it is seen from the actors, is described by the three scenarios *Authenticate*, *EnterUsername* and *EnterPassword*. The latter two are sub-scenarios of *Authenticate*.

Join Relationships. *Join relationships* indicate where an aspect impacts on other concerns, i.e. they denote the target points of weaving in scenario or behavior chunks. They either connect a state of a behavior chunk with a transition or a root scenario node of a scenario chunk with a scenario node. Join relationships are graphically represented as dashed arrows, as shown in Fig. 2. They can be attributed with an order and a priority (see the section about weaving semantics below).

View Concept. The *view* concept of the original ADORA language (see Section 2.3) is extended for handling aspects [7]. Aspects are represented primarily in their own view comprising aspect containers, behavior chunks, scenario chunks and join relationships. The aspect view can be hidden, i.e. the focus can be set only on those model parts with the conventional concerns. Additionally, any aspect construct, such as join relationships, aspect containers, behavior and scenario elements, can be hidden individually, allowing to create a partial aspect view. Furthermore, as ev-

[1] In contrast, components are modules of conventional concerns in ADORA.

ery aspect-oriented ADORA model can be transformed into a conventional one by applying our weaving semantics (see below), the modeler also has the possibility to switch between the aspect-oriented and the conventional view of a model. This feature helps modelers to understand aspects both as separate concerns and in the context where they apply.

Weaving Semantics. The weaving semantics [7] of aspect-oriented ADORA defines the mapping of aspect-oriented constructs to conventional ones. Join relationships indicate the targets of the weaving process. They can be attributed with an *ordering keyword* indicating the weaving order of the crosscutting behavior or scenario in relation with the crosscut transition or scenario. The ordering keyword must be either *before*, *after* or *instead*. The default value is *before*.

Furthermore, a priority defines the precedence of competing aspects. An aspect is competing with another aspect if both of them crosscut the same target element with the same ordering keyword. The priority is specified by a number between *1* (lowest) and *10* (highest). If no priority is given, the lowest priority is taken as default value. The aspect with the higher priority is woven first. If both aspects have the same priority, one of them is chosen non-deterministically to be the first one in the weaving order.

The weaving semantics for *crosscutting behavior* is illustrated in Fig. 3. On the left hand side, the aspect-oriented model is given; the woven version is on the right hand side. Fig. 3a, b, and c describe the semantics of the *before*, *instead*, and *after* ordering, respectively. The weaving of the crosscutting behavior is related to the action in the transition description $a \mid b$ of the crosscut transition, where a is the triggering event and b the executed action. Hence, for the weaving, the transition and its description are split up and the crosscutting behavior is inserted *before*, *after* or *instead* the action of the crosscut transition. In the *before* case, an additional state is required for ensuring that action d is executed before action b.

The nodes in ADORA scenario trees are typed as *sequence*, *alternative*, *parallel*, or *root*. Hence, weaving of a *crosscutting scenario chunk* is only defined when its root node has the same type as the target node. Scenario chunks with root node types *parallel* or *alternative* have rather trivial weaving semantics. They are just added as siblings of the target node, no matter what ordering or priority is given. However, the weaving semantics for sequential scenario chunks (i.e. those with root node type *sequence*) needs to consider the order of nodes in the target scenario. A sequential scenario chunk is inserted either *before*, *after* or *instead* of the target scenario node. The sequence numbers of the target node and its siblings are adapted accordingly to preserve the order of scenario execution. Fig. 4 illustrates the weaving semantics for sequential scenario chunks.

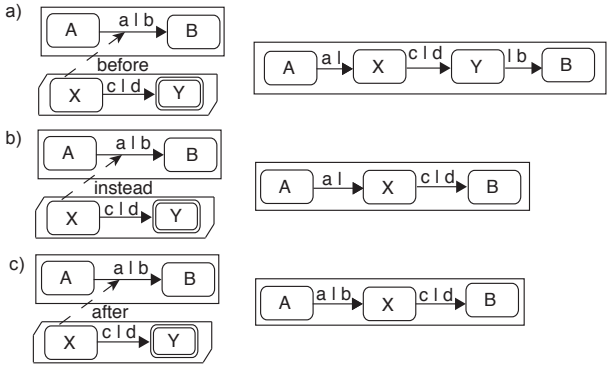

Figure 3. The weaving semantics for behavior chunks. Left side: aspect-oriented models, right side: woven models

The weaving of an aspect-oriented ADORA model also involves the functional description and may cause further changes in the structure of the model. However, for this paper, the discussion of these issues is out of scope. A more thorough discussion of the weaving semantics can be found in [7].

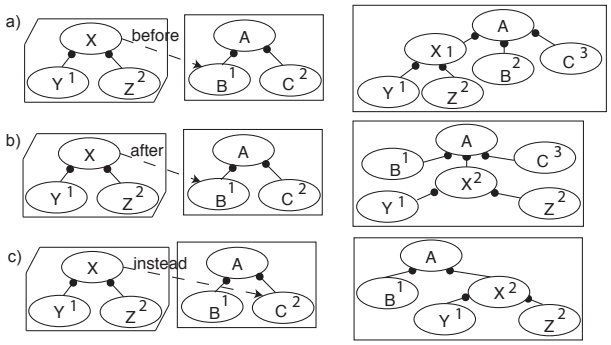

Figure 4. The weaving semantics for sequential scenario chunks. Left side: aspect-oriented models, right side: woven models

4 Evolution Support for Aspects in ADORA

As discussed in Section 2.3, the conventional ADORA language supports the evolution of conventional concerns in requirements models by two constructs [18, 15]: the partial indicator and the abstract association. Both elements indicate an intentional incompleteness and a future evolution of the corresponding elements.

The aspect-oriented extension of the ADORA language in [7] provides the possibility for modeling functional aspects and operationalized NFRs of type (i). However, a comprehensive approach for evolving aspect-oriented

ADORA models is missing yet. In this section, we extend our approach with *language support for the evolution of aspects*. Both functional and non-functional concerns are evolvable. However, as NFRs can be regarded as high-level CCCs which are a major source of aspects (see Section 2.1), we focus on the modeling and evolution of NFRs. For the support of aspect evolution, additional language concepts have to be introduced: partial aspects, abstract join relationships and the corresponding weaving semantics for the new elements.

Partial Aspects. Partial aspects are intentionally incompletely modularized CCCs. In ADORA, a partial aspect is denoted by three trailing dots after the aspect's name, i.e. in the same way as we denote the incompleteness of components, states and scenarios (cf. Section 2.3). The incompleteness of an aspect manifests in different ways. In an early stage of model development, an aspect may be coarsely specified only, i.e. there is missing or unclear information about the aspect or its formality is on an inappropriate level.

At the beginning of its evolution, an aspect might be expressed very informally by natural language. Each element in ADORA, including aspects, can contain textual information. For example, a *security* NFR might be stated as "*The system must be secure*", which can be stored as text in the aspect. In subsequent steps, this requirement in natural language may be re-stated in a more precise, or more formal way. In particular, aspects which represent NFRs can be realized by more specific sub-concerns. For example, in the case of the security NFR there may be the sub-concerns *Protect Confidentiality* and *Preserve Integrity*. Sub-concerns are supported by nesting aspect constructs as exemplified in Fig. 5a) where a concern consists of the two sub-concerns *SubAspect1* and *SubAspect2*. This language extension helps to decompose high-level non-functional requirements into more concrete NFRs and thereby enables a better traceability of the NFRs.

Abstract Join Relationships. Abstract Join Relationships indicate an intentional incompleteness of a crosscutting relationship. They are visualized as bold dashed arrows. There are three cases where abstract join relationships might occur. Case (a) results from not fully evolved functional aspects, such as not yet operationalized type (i) NFRs or functional CCCs. Join relationships out-going from these aspects have to be abstract, because the originating behavior chunk or scenario cannot be determined yet. Case (b) yields abstract join relationships because the target element is still under evolution and therefore is not modeled yet. Instead, a substitute element has to be targeted. Case (c) happens when NFRs of the type (ii) or (iii) are not yet operationalized in the model. However, they will never be functionally operationalized. Instead, they will result in decisions during the architectural or implementation phase which is, beside the partial indicator of the aspect, expressed by the abstract join relationship.

Several situations are illustrated in Fig. 5b-h. Figure 5b shows a situation where the aspect which is the origin of the crosscutting relationship is fully evolved. The final target does not yet exist, because it will be a direct or indirect child element of the given partial abstract object. Hence, the abstract join relationship points at the partial abstract object as a substitute for the final element. Fig. 5c and 5d describe a similar situation, where both the originating aspect and the target are partial. Hence, both the final originator and the final target are not clear yet and the join relationship has to be abstract. In Fig. 5e, the aspect is fully operationalized and the join relationship targets at the given transition $g \mid h$. Nevertheless, the crosscutting join relationship is abstract, which means that it is not yet clear whether the transition is the final target of the join relationship. Fig. 5f is an analogous situation to 5b where the target is a partial scenario. Figure 5g shows the case where an NFR constrains the communication channel of the targeted association. For example, this might happen in the case of a performance requirement which stipulates a minimum bandwidth for the communication. Such an NFR ends in a decision and is realized at a later stage. The meaning of Figure 5h is obvious: a partial aspect targets another partial one.

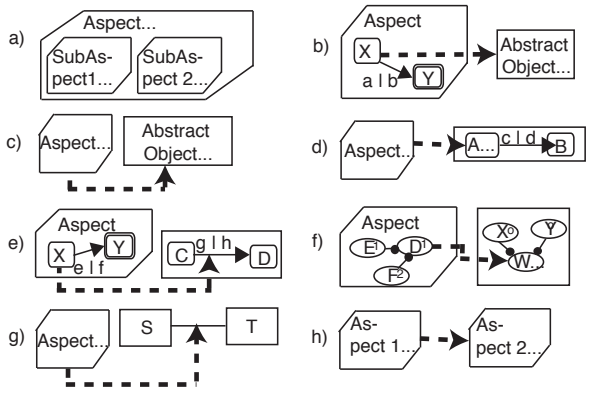

Figure 5. Usage of partial aspects and abstract join relationships.

Weaving Semantics. As argued in Section 3, it is crucial for the understanding of a model to have the possibility of switching between the woven and the aspect-oriented model. Therefore it is necessary to extend the weaving semantics in Section 3, which is done in a straightforward manner. Owing to the different degree of formality, the content of a partial aspect is handled as text. Semi-formal or formal elements (e.g. the behavior chunk in Fig. 5e) are transformed to text and embedded in the generated text chunk. After the weaving, the textual description of the partial aspect is inserted into the text chunk of each tar-

get element. Thereby, no information is lost in the woven model. Abstract join relationships may also contain an ordering keyword which indicates whether the generated text is inserted *before*, *after* or *instead* the text of the target element. A priority number finally indicates the arrangement of the text chunks from competing aspects. Both the ordering and the priority of the generated text may give a hint to the model reader for the conflict resolution between competing aspects.

Scalability. The scalability of the ADORA approach and the presented extension is supported by two concepts: firstly, by the ability to generate partial views, i.e. to hide elements which are not in the focus of interest (c.f. Section 2.3) and secondly, by the ability to decompose models hierarchically.

5 Extended Example

In this section, we present an example illustrating how the language features presented above can be used in an evolutionary requirements process, e.g. the one presented in [15]. The example concentrates on the evolution of non-functional aspects. The process may start with an initial analysis of very coarse functional and non-functional requirements. This initial elicitation is done with state of the art techniques. After that, a first ADORA model can be created which will be evolved in subsequent steps.

In Fig. 6a, the initial model for the library system is shown at an early stage in the requirements process. It includes a set of high-level NFRs, i.e. a *Security*, *Maintainability* and *Performance* requirement. The abstract join relationships indicate that the library system is crosscut by these NFRs. The document symbols in the aspect containers represent text chunks containing further information. During the evolution, the model is evolved by refining the conventional and the crosscutting concerns. Figure 6b and Figure 6c show subsequent steps in the evolution of the requirements model.

In the course of their evolution, NFRs are rendered more precisely by eliciting the NFRs in more detail and by elaborating their natural language description. Furthermore, NFRs can be decomposed into sub-requirements, which is expressed by embedding them in the parent NFR (c.f. Section 4). Figure 6c shows a subsequent step in the system's evolution which illustrates the refinement of NFRs. For example, the *Performance* requirement consists of a sub-requirement for the response time, which in turn consists of a sub-requirement for the *ConnectionBandwidth*. The *Performance* requirement and its sub-requirement are partial. The *Security* requirement is refined to a partial sub-requirement *Authentication*. The partial indicator of *Security* is removed, which means that its evolution is finished at this point.

Join relationships can be refined in two ways: they are either relocated or multiplied. Both evolution steps are caused by the evolution/refinement of its originating aspect or target element. For example in Fig. 6b, the component *LibrarySystem* was evolved resulting in having a new sub-component *BorrowManager* causing the relocation of the particular join relationship to the new sub-component. Figure 6c shows a subsequent evolution step of the library system where the out-going join relationships of the *Authentication* and the *Performance* aspect are multiplied. In contrast to these examples, the *Maintainability* NFR is neither decomposed to a sub-NFR nor is its join relationship refined.

In subsequent steps, the evolution of the requirements is continued in a similar way. This process ends when the NFRs are operationalized. Type (i) NFRs are operationalized if they are described functionally, which is illustrated for the security concern in Fig. 2. The elaboration of NFRs of type (ii) and type (iii) ends at the point where they turn into decisions in the architecture or design of the software system.

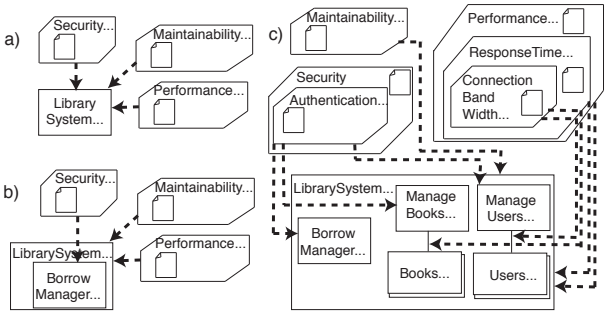

Figure 6. Three evolution steps of the Library System.

6 Related Work

Chitchyan et al. [2] provide an overview of existing work on aspect-oriented requirements engineering.

Goal-oriented approaches, such as [3, 19], support the evolution of functional and non-functional requirements. However, they do not support the separation of crosscutting concerns.

Araújo et al. [1] model both aspectual and non-aspectual behavior as scenarios, transform these into state machines, and integrate the state machines. However, the resulting model can be rather difficult to read and modelers cannot switch forth and back between aspectual and woven views.

Sousa et al. [16] combine a use case driven approach with the NFR framework. The approach is able to evolve aspect-oriented software systems from early requirements

to design, but does not provide a coherent graphical notation.

Jacobson [6] as well as Clarke and Baniassad [13] mainly support the separation of functional concerns.

Moreira et al. [8] introduce a n-dimensional separation of concerns in requirements models, treating functional and non-functional concerns uniformly. However, their approach works on textually represented requirements only.

7 Conclusions

Achievements. We have presented an aspect-oriented modeling approach which supports the evolution of functional and non-functional crosscutting concerns in requirements engineering and early architectural design. We have demonstrated our approach with a focus on evolving non-functional crosscutting concerns. Our concepts have been implemented in the prototype of the ADORA tool [11]. Our weaving semantics, together with the visualization mechanisms in the ADORA approach [4, 7] allow a modeler to switch between aspect-oriented and conventional views of a model. Furthermore, the view concept allows to abstract from details that are not in the focus of interest and therefore supports the scalability of our approach.

Limitations and future work. Our approach is currently restricted to requirements and architecture. It supports neither evolution nor traceability to artifacts in later stages of the software process. Furthermore, conflicts between aspects are currently handled by prioritizing the competing aspects, i.e. a tradeoff analysis and conflict resolution for competing partial aspects have to be done manually. Finally, aspect weaving is not yet fully implemented in the ADORA tool. In our future work, we plan to provide better support for tradeoff analysis of competing aspects and to improve the implementation of aspect weaving.

References

[1] J. Araújo, J. Whittle, and D.-K. Kim. Modeling and Composing Scenario-Based Requirements with Aspects. In *Proc. 12th IEEE International Requirements Engineering Conference (RE'04)*, pages 58–59, 2004.

[2] R. Chitchyan, A. Rashid, P. Sawyer, J. Bakker, M. P. Alarcon, A. Garcia, B. Tekinerdogan, S. Clarke, and A. Jackson. Survey of Aspect-Oriented Analysis and Design. In *R. Chitchyan, A. Rashid (eds.): AOSD-Europe Project Deliverable No. AOSD-Europe-ULANC-9.*, 2005.

[3] A. Dardenne, A. van Lamsweerde, and S. Fickas. Goal-directed Requirements Acquisition. *Science of Computer Programming*, 20(1):3–50, 1993.

[4] M. Glinz, S. Berner, and S. Joos. Object-Oriented Modeling with ADORA. *Information Systems*, 27(6):425–444, 2002.

[5] M. Jackson. *Principles of Program Design*. Academic Press, New York, 1975.

[6] I. Jacobson. Use Cases and Aspects Working Seamlessly Together. *Journal of Object Technology*, 2(4):7–28, 2003.

[7] S. Meier, T. Reinhard, C. Seybold, and M. Glinz. Aspect-Oriented Modeling with Integrated Object Models. In *Modellierung 2006*, pages 129–144, 2006.

[8] A. Moreira, J. Araújo, and A. Rashid. Multi-Dimensional Separation of Concerns in Requirements Engineering. In *Proc. 13th IEEE International Requirements Engineering Conference (RE'05)*, pages 285–296, 2005.

[9] H. Ossher and P. Tarr. Using Multidimensional Separation of Concerns to (Re)Shape Evolving Software. *Communications of the ACM*, 44(10):43–50, 2001.

[10] A. Rashid, A. Moreira, and J. Araújo. Modularisation and Composition of Aspectual Requirements. In *Proc. 3rd Aspect-Oriented Software Development Conference (AOSD 2004)*. ACM Press, 2003.

[11] RERG. The ADORA Tool Web Site of the Requirements Engineering Research Group (RERG) of the University of Zurich http://www.ifi.unizh.ch/rerg/research/projects/adora/tool/.

[12] L. Rosenhainer. Identifying Crosscutting Concerns in Requirements Specifications. In *Early Aspects 2004 Workshop: Aspect-Oriented Requirements Engineering and Architecture Design (AOSD 2004)*, 2004.

[13] S. Clarke and E. Baniassad. *Aspect-Oriented Analysis and Design - The Theme Approach*. Addison-Wesley, Upper Saddle River, New Jersey, USA, 2005.

[14] S. Robertson and J. Robertson. *Mastering the Requirements process*. Addison-Wesley, ACM Press, 1999.

[15] C. Seybold, S. Meier, and M. Glinz. Evolution of Requirements Models By Simulation. In *Proc. 7th International Workshop on Principles of Software Evolution (IWPSE'04)*, pages 43–48, 2004.

[16] G. Sousa, S. Soares, P. Borba, and J. Castro. Separation of Crosscutting Concerns from Requirements to Design: Adapting the Use Case Driven Approach. In *Early Aspects Workshop: Aspect-Oriented Requirements Engineering and Architecture Design (AOSD 2004)*, 2004.

[17] J. Stanley M. Sutton and I. Rouvellou. Modeling of Software Concerns in Cosmos. In *AOSD'02: Proc. 1st International Conference on Aspect-Oriented Software Development*, pages 127–133, New York, NY, USA, 2002. ACM Press.

[18] Y. Xia and M. Glinz. Extending a Graphic Modeling Language to Support Partial and Evolutionary Specification. In *Proc. 11th Asia-Pacific Software Engineering Conference (APSEC 2003)*, pages 186–196.

[19] E. Yu. Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering. In *Proc. 3rd IEEE International Symposium on Requirements Engineering (RE'97)*, pages 226–235, 1997.