Injecting security as aspectable NFR into Software Architecture

Hamid Bagheri
Seyed Hassan Mirian Hosseinabadi

Software Quality Research Lab
Department of Computer Engineering
Sharif University of Technology
h_bagheri@ce.sharif.edu, hmirian@sharif.ir

Abstract

Existence of crosscutting concerns in software requirements often intensifies complexity of software development process; moreover, Software security as a particular non-functional requirement of software systems is often addressed late in the software development process. Modeling and analyzing of these concerns and especially security in the software architecture facilitate detecting architectural vulnerabilities, decrease costs of the software maintenance, and also reduce finding tangled and complex components in the ultimate design. Aspect oriented ADLs have emerged to overcome this problem; however, imposing radical changes to existing architectural modeling methods is not easily acceptable by architects. In this paper, we present a method to enhance conventional software architecture description languages through utilization of aspect features with special focuses on security. To achieve the goal, aspectable NFRs have been clarified; then, for their description in the software architecture, an extension to xADL 2.0 [5] has been proposed; finally, we illustrate this material along with a case study.

1. Introduction

Software requirements are conventionally categorized in two groups: functional requirements and non-functional requirements (NFRs). Although the applicability of a software system mainly depends on its functional requirements, its value will be assessed based on the quality of its NFRs such as security and performance [1]. Moreover, the importance of security in the development of software systems has increasingly become more critical as software becomes increasingly durable, and more extensive in our everyday lives.

While decisions made at every phase of the development process are important, architectural decisions have the greatest impact on the NFRs such as security; therefore, there is a need for modeling NFRs at architectural level of software development.

Because NFRs are usually considered as crosscutting concerns [8] which may appear in different components of the software architecture, and it is generally accepted that security requirements are NFRs that are crosscutting in nature, Aspect concept seems to be a well-behaved candidate for their modeling. Most of the current ADLs (Architecture Description Languages) have shortcoming in modeling NFRs. On the other hand, only a few of ADLs provide facilities for modeling aspects in the software architecture description [20, 21]; however, they are not generally used. Thus, it might be worthwhile to extend a generally accepted and widely used ADL for modeling NFRs as aspectual components, instead of proposing a new ADL.

For the purpose of this work, we have extended xADL 2.0 and by affixing an aspectual layer over the conventional architecture, achieved a higher degree of encapsulation which results in modifiability, reusability and testability enhancement of the software architecture.

The rest of this paper is organized as follows. In Section 2 a brief introduction to related work is presented. Following this, in Sections 3 and 4 aspectable NFRs will be discussed and aspect enhanced software architecture will be delineated. Section 5 presents the extended modeling language we have designed and Section 6 provides a case study to illustrate modeling of NFRs as aspectual components in
the software architecture. In Sections 7 the outcomes and future work of this paper will be presented.

2. Related Works

Whereas focus of our work is on injecting security as aspectable NFR into software architecture, in this section we first survey existing researches on modeling NFRs at the software architecture; afterward, revolve around Aspect Oriented ADLs.

Despite the importance of modeling non-functional requirements at architectural level of software design, many ADLs such as C2SADEL [7], Darwin [10], Rapide [13], and Wright [14], have not provided explicit method for their description. On the other hand, ADLs that provided NFR modeling components can be further categorized in two groups. ADLs of the first group are those having a general perspective of NFR description, while ADLs of the second group are designed for modeling specific NFRs such as Security. ACME [17], AESOP [15], and Weaves [11] are among ADLs of the first group, UMLSec [6], MetaH [18], and UniCon [19] can be categorized as ADLs of the second group.

Shortcoming of ADLs in presenting crosscutting concerns resulted in evolution of Aspect Oriented (AO) ADLs. Avoiding tangled components through their encapsulation in aspect components, resulted in complexity reduction of AO software architectures. ADLs like DAOP-ADL [20], PRISMA [21] are instances of AO ADLs.

Pinto et al. [20] proposed DAOP-ADL with components and aspects as first-order elements. In the Prisma approach [21], aspects are new ADL abstractions used to define the structure or behavior of architectural elements (component and connectors), according to specific system viewpoints. Such ADLs introduced a new description language for the software architecture.

Another approach for aspect modeling in the software architecture is to extend current ADLs with Aspect components. Pessemier et al [9] extended the Fractal ADL with Aspect Components (ACs) and they supposed cross-cutting concerns can be found only in components. In this paper we have the same approach, however the extended ADL is xADL 2.0 and the proposed method is not limited to components and can be applied on any architectural elements (component, connector and configuration).

3. Aspectable NFRs

This paper is not going to propose any aspect elicitation method; in fact, it is just to highlight those NFRs which can show crosscutting behavior and name them Aspectable NFRs.

An Aspectable NFR is a one that some of its dimensions have the potential of being replicated among architectural elements.

Bass et al. [1] accentuate Availability, Performance, Security, Modifiability, Testability and Usability as six most common and important quality attributes (the term which can be used interchangeably by NFR).

Modifiability and Testability are commonly known as internal quality attributes [12], which means that their importance is due to software engineers' point of view, not software users. Therefore, these quality attributes cannot be considered as Aspectable NFRs because their dimensions (portability and adaptability for modifiability, modularability and readability for testability) do not have this potentiality to be explicitly replicated by architectural elements and not able to make a complex design. Afterwards, it will be explained that the primary result of this work is achieving a higher level of quality in these two NFRs.

According to Smith and Williams [3] there are two dimensions for performance: Responsiveness and Scalability. Responsiveness is the ability of a system to meet its objective for response time or throughput, and Scalability is its ability to continue to meet the responsiveness objectives as the demand for the software functions increases.

Performance is not a quality attribute achieved in isolation. Performance crosses the whole system globally and due to this global nature it must be considered over incorporation of all architectural elements. Furthermore, performance is a quality attribute which will be achieved through a trade off between other quality attributes such as: availability, security and modifiability.

Regarding performance dimensions and their application mechanisms, it is not common that its mechanisms be replicated among architectural elements independently, as they have overlapping affects. Thus due to intrinsic association of performance mechanisms, we cannot set a general rule on extracting them from architectural elements and encapsulating them in a separate component, as it may directly affect the functionality of system.

The following sub-sections will clarify Aspectable NFRs and provide evidences for each one.
3.1. Usability

Usability is concerned with how easy it is for the user to accomplish a desired task and the kind of user support the system provides [1]. As it can be inspired from usability definition, its primary dimensions are mostly related to the quality of learning features and the degree of user friendliness, of the ultimate product. Although usability decisions may be made during later phases of software design, some of these decisions can have direct impact on architectural components.

For instance, consider a usability strategy which imposes active component at run-time, to provide proper help on user demand. Consequently, this strategy must be replicated in many of architectural components which may increase the complexity of ultimate design, excessively.

Moreover, as another supporting example, consider the strategy of a software system in handling errors. According to Bass et al. [1] minimizing the impact of errors is one of usability dimensions. Thus, exception handlings must be localized within any of the architectural elements. Having a proper categorization of system exceptions can be used to aspectize this strategy which has crosscutting behavior and move toward more cohesive architectural elements.

3.2. Security

According to Bashir et al. [4] there are seven critical dimensions for a secure software system: (1) Authentication, (2) Access Control, (3) Audit trail, (4) Confidentiality, (5) Integrity, (6) Availability, and (7) Non-Repudiation. A security attack can be any attempt to affect any of these dimensions adversely.

Considering any of these dimensions, they have this potential to be replicated among architectural components. For instance, audit trail which will be used for the purpose of evaluating the effectiveness of security policies and mechanisms, is dealt with logging record of security related events. It is obvious that logging mechanism has crosscutting behavior.

Section 6 presents a case study which describes how to consider integrity, access control, and audit trail constraints as a crosscutting concerns and illustrates how to extend that idea for other dimensions of security NFR.

3.3. Availability

Somo Banerjee et al. [16] provide a classification of availability techniques in order to explain in which architectural element, any of them can be applied. They have identified five availability techniques as: (1) Caching, (2) Hoarding, (3) Queuing Remote Procedure Calls, (4) Deployment and Mobility and (5) Replica Reconciliation.

Caching and Hoarding is related to ability of an architectural element for caching and pre-fetching data, respectively; such strategies can be deployed in architectural elements that are concerned with data storage and retrieval. Queuing Remote Procedure Calls is a mechanism that ensures reception of remote calls to the architectural elements' procedures, while the element is disconnected. These three techniques of availability can have crosscutting behavior as they are defined at the local level of architectural elements.

On the other hand, Deployment and Mobility which relate to relocating component code or state, also Replica Reconciliation which implies architectural elements' replication, will be defined at the global level of whole architecture and their consideration as a crosscutting concern is not trivial.

To sum up briefly, an aspectual NFR seems to be one that can be satisfied by adding crosscutting functionality to the design.

4. Aspect Enhanced Software Architecture Modeling

Functional and non-functional requirements are orthogonal [1] and it is possible to consider any level of NFRs for functionality of any software component. Thus, leaving NFRs scattered among architectural components and having no distinct separation of concerns in their modeling approach, can result in sophisticated and tangled components along the rest of software development process.

In this paper, aspectual components will be utilized for modeling crosscutting behavior NFRs in the software architecture. Meanwhile, affixing aspectual components into the software architecture needs some special considerations which are not supported by conventional architectural styles. For instance, suppose the security policy of a software system which imposes an authentication service to any component that may change the system status. Therefore, a series of actions must be added to any of those components, while they can be set aside in an aspectual component. However, with respect to many of architectural styles, adding an aspectual component is not allowed because a single component cannot provide crosscutting service for components of all layers.

In order to enhance the software architecture for supporting aspectual components, we have considered
an Aspectual hyper-layer which will be affixed on the top of base architecture model. This hyper-layer holds aspectual components and their weavers will be placed in a middleware between aspectual hyper-layer and base architecture. Figure 1 depicts the stance of hyper-layer regarding base architecture.

It would be worthy of mention that the proposed method of software architecture modeling does not depend on any kind of conventional styles. Thus, it is just an enhancement on architectural styles for supporting aspectual components. Any aspectual component is responsible for encapsulation of a crosscutting NFR and its associated weavers are responsible for mapping the appropriate method of that NFR to the relevant elements of underlying architecture model.

Encapsulation of NFRs within aspectual components will result in more precise separation of concerns in the software architecture modeling. Achieving more cohesive architectural components is direct outcome of this method. Besides, Reusability and Portability which are dimensions of Modifiability will enhance substantially. Furthermore, Testability of the software architecture will be improved, as it is defined with respect to the ease of finding errors in architectural elements. Thus, software architectures with high level of separation of concerns and explicitly defined responsibilities of components can facilitate software testing.

Encapsulation of NFRs within aspectual components will result in more precise separation of concerns in the software architecture modeling. Achieving more cohesive architectural components is direct outcome of this method. Besides, Reusability and Portability which are dimensions of Modifiability will enhance substantially. Furthermore, Testability of the software architecture will be improved, as it is defined with respect to the ease of finding errors in architectural elements. Thus, software architectures with high level of separation of concerns and explicitly defined responsibilities of components can facilitate software testing.

5. Aspect Enhanced ADL

In order to describe an aspect enhanced architecture we need an ADL which can be extended for description of aspectual components and definition of weavers. Any ADL which supports these two properties can be extended to become aspect enhanced ADL. In this paper we have focused on xADL 2.0 which is an extensible, XML based and modular architecture description language [5].

xADL 2.0 has a simple core for describing fundamental elements of the software architecture (component, connector, and configuration). This core is based on xArch and due to its reliance on XML can be easily extended. In this work, we have extended the xADL 2.0 core by adding “AspectWeaver.xsd” schema to its schema collection for definition of weavers. This ADL supports definition of aspectual components by itself.

5.1. Aspectual Component Description

As eluded before, the purpose of defining aspectual components is to achieve a higher degree of separation of concerns in the software architecture, with respect to non-functional requirements. Thus, any aspectual component will be responsible for encapsulating definitions of all methods which are related to one primary NFR.

Aspectual components can be easily modified to reflect any of the NFRs’ crosscutting methods which are scattered among architectural components. In fact, any aspectual component can be considered as library of one primary NFRs’ methods. A component specification is a packaging of methods corresponding to a single NFR; likewise, we define subcomponent whose specification is a packaging of methods corresponding to a one dimension of a primary NFR. Figure 2 depicts an aspectual component for non-functional requirement of Security.
5.2. Definition of Weaver

Any aspectual component must have a weaver for mapping its methods onto base architectural elements. A weaver will be described in an XML file which is separated from base architecture description. Thus, adding aspectual components over base architecture will impose no change on conventional description of architectural elements.

As described in figure 3, "AspectWeaver.xsd" defines weaverCollection, weaver, pointcutSet, pointcut, adviceSet and advice. Each weaverCollection consists of a set of weavers that any of them holds pointcutSet and adviceSet. Realization of this schema will generate a middleware between aspectual components and architectural components.

At the highest level, a weaverCollection holds description of all weavers. Any weaver represents a dimension of one NFR. It is responsible for weaving appropriate aspectual components' methods into relevant architectural components and will perform its responsibility by set of pointcuts and advices. Each pointcutSet consists of a set of pointcuts, also adviceSet defines a set of advices and contains component attribute, identifying aspectual component, and responsible for definition of all methods which are related to one primary NFR.

Each Pointcut defines the point of description in the architectural elements at which crosscutting concerns are needed to be applied. In other words, NFRs'...
methods will be inserted to the software architecture at the points which are defined by pointcuts. Each pointcut has three attributes:

- **id**: A unique identifier of that pointcut.
- **type**: Pointcuts may refer to any of architectural elements (component, connector and configuration). The type of referenced element must be apparent at pointcut description.
- **pattern**: A repeated behavior in architectural elements describes the weaving condition.

On the other hand, each **Advice** specifies the crosscutting NFR’s method which must be inserted at the relevant pointcut. Each advice has also three attributes:

- **type**: determination of how the method of crosscutting concern should be applied to the affected architectural element. There is three different moment for application: **before** means that method of the advice executed before the method call of base architectural element, **around** is like before but it allows to cancel the method execution, and **after** is executed when the method call finished.
- **pointcutRefID**: the relevant pointcut id for which method of this advice will be applied.
- **method**: one appropriate method from aspectual component which is specified by component attribute of advices tag.

6. Case Study

In this section we will describe Content Management System (CMS) architecture and illustrate how to achieve better separation of concerns in aspectually enhanced architecture in comparison with base architecture. The focus of this case study is on deriving out non-functional requirements of Access Control, Integrity, and Auditing which are known as dimensions of security.

6.1. CMS Base Architecture

CMS is a software system used for content management. This includes files, images, audios, and electronic documents. Figure 4 depicts part of the CMS architecture. Document Manager Component is responsible for maintaining the structure (folders) of the content and fetching requested content on demand; besides, Version Manager Component is responsible for managing multiple versions of contents in the system; therefore, contents can have many versions in the system. In addition, Communication Component is responsible for communication of the system with other systems; in fact, sending and receiving of the request is performed by this component.

![Figure 4, part of the CMS architecture](image)

**6.2. Security Aspectual Component**

One of non-functional requirements of CMS is to ensure users that documents will be transmitted without any changes over the network. This NFR imposes utilization of integrity methods as preventive or detective mechanism of unauthorized information modification [2]. On the other hand, some CMS functionalities should be only accessed by certain groups of users according to an organizational policy; for instance, documents can be delivered only by authorized users. Therefore, we need to utilize Access control mechanism. The other NFR, which we focus on, is audit trail. Auditing is used for the purpose of evaluating the effectiveness of security policies and mechanisms, and is dealt with logging record of security related events.

As all of these NFRs are subsets of Security. We can, therefore, use the aspect component described in section 5.1, but only those methods are important for us which are related to integrity, access control, and auditing. The digestCreator and the integrityChecker methods are from the integrity subcomponent. The digestCreator method uses a hashing algorithm for creating digest of the content and attaches to the content. In contrast, the integrityChecker method which will be executed on the receiver is responsible for generating the digest of received content with the same hashing algorithm and checking its equivalence with the attached digest. In addition, the policyControl method is from the access control subcomponent; the policyControl method validates whether access to functionality must be allowed or denied; it is obvious that this validation can be implemented in various
ways. Finally, the `audit` method is from the `AuditTrail` subcomponent and is responsible for registering security related events.

### 6.3. CMS Weavers Middleware

Eliciting crosscutting methods of integrity, access control, and auditing NFRs from base architecture elements, and encapsulating into security aspectual component, formed the aspectual hyper-layer. In order to weave these aspectized methods into base architecture elements, weavers middleware must be described.

Figure 5 depicts this middleware as an instance of "AspectWeaver.xsd". This instance specifies security as a weaver with a pointcutSet and adviceSet. As described before, each pointcut has three attributes: id, type and pattern. In this case study, four pointcuts have been defined for specifying components from base architecture. Pointcut with `read` id defines the components which invokes the readDocument method; furthermore, two more pointcuts describe components which may invoke send/receive methods, and assigned appropriate id to each of them. Similarly, `versioning` pointcut presents the component which invokes the addVersion method.

Any of referenced pointcuts must be supported by at least one advice that specifies an appropriate method for that pointcut. Each advice has also three attributes: type, pointcutRefID and method. In this case study, ten advices have been defined. Two of them are defined for specifying digestCreator and integrityChecker methods from integrity subcomponent which must be inserted before `send` and `receive` pointcuts in order that the integrity needs of the system can be fulfilled. Moreover, four other advices are described for specifying policyControl method which must be inserted before `send`, `receive`, `read`, and `versioning` pointcuts; then, it validates if any of the user roles contains the minimum permission required to access the method, and lets the system continues its normal flow providing that it is allowed; otherwise, it cancels execution of the method. Furthermore, four advices are defined for specifying `audit` method which must be inserted after the pointcuts for logging these security related methods.

This case study explains how to model non-functional requirements which has crosscutting behavior, using aspectual components and how to weave theirs behaviors into base software architecture. It is obvious that integrity, access control, and auditing are just some dimensions of the security NFR which have aspectable behaviors. Confidentiality, authentication and non-repudiation are the other dimensions of security which have this potential to be derived out of base architectural elements, declared within security aspect component, and weaved back to those elements. There is similar situation for Availability NFR with respect to its caching dimension.

It is important to notice that, this method is very useful when having a system lacking of control in functionality access. It brings in minimal changes in the
base software architecture, as it just affixes independent aspectual components to the base software architecture.

7. Conclusions and Future Work

In this paper we described a method for enhancing software architectural modeling of NFRs by taking advantages of aspect concepts. To achieve the goal we will focus on based architecture, elicit aspectable NFRs, encapsulate them into components of aspectual hyper-layer and then weave them into base architecture elements. Besides, we have extended xADL 2.0 for describing aspect enhanced software architecture. All of these steps have been explained through a case study on the Content Management System and have been described in extended xADL 2.0.

The primary advantages of this method are:

- Injecting NFRs like security into the Software Architecture without performing any changes in the base Software Architecture.
- Achieving higher degree of separation of concerns.
- Designing more cohesive components.
- Improving maintainability, testability and modifiability of architectural elements.
- Extending a commonly used ADL instead of introducing a new Aspect Oriented ADL.

For future work we are going to extend our idea not to limit ourselves into non-functional requirements, and move toward eliciting all of aspectable behaviors from base architecture. Furthermore, having such model of the software architecture that has a salient focus on separation of concerns, can lead us toward introducing an aspect enhanced software architecture analysis method.

8. References