An Aspect Enhanced Method of NFR Modeling in Software Architecture

Hamid Bagheri
Hesam Chiniforooshan Esfahani
Seyed Hassan Mirian Hosseinabadi

Software Quality Research Lab
Department of Computer Engineering
Sharif University of Technology

{h_bagheri, chineeforoushan}@ce.sharif.edu, hmirian@sharif.ir

Abstract
Existence of crosscutting concerns in software requirements often intensifies complexity of software development. Modeling and analysis of these concerns at software architecture can decrease possibility of finding tangled and complex components in ultimate design. Aspect oriented ADLs have emerged to overcome this problem but imposing radical change to existing architectural modeling methods is not easily acceptable by architects. Thus, this paper presents a method on how to enhance conventional software architecture description languages through utilization of aspect features with special focus on non-functional requirements (NFRs). To achieve the goal, aspectable NFRs have been clarified and for their description in software architecture, an extension to xADL 2.0 [5] has been proposed. Encapsulation of crosscutting NFRs in aspectual components, which is the result of this method, will provide a higher degree of separation of concerns in software architecture and prepares a straightforward baseline for software architecture analysis. The method is applied for description of a Chat System architecture and by encapsulating Integrity constraints in an aspectual component, illustrates how to enhance quality attributes like Reusability, Testability and Maintainability.

1- Introduction
Software requirements are conventionally categorized in Functional and non-Functional. Although the applicability of a software system mainly depends on its functional requirements, its value will be assessed based on the quality of its non-functional requirements (NFRs) such as security and performance [1].

In spite of the NFRs’ important role, they are not usually considered during early phases of software development lifecycle, and will be left for system verification at the end of the implementation phase [2]. This problem is the result of having no direct mapping between NFRs defined at requirement engineering to the design of software system. While decisions made at every phase of the development process are important, architectural decisions have the greatest impact of the NFRs such as modifiability, reusability and testability. So, there is a need for modeling NFRs at architectural level of software development.

As NFRs are usually considered as crosscutting concerns [8] which may appear in different components of software architecture, 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 software architecture description [20, 21] but 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 software architecture. In Sections 7 the outcomes and future work of this paper will be presented.

2 Related Works
Despite the importance of modeling non-functional requirements at architectural level of software design, many ADLs such as C2SADEL, Darwin, Rapide, and Wright [7, 10, 13, 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, AESOP and Weaves [17, 22, 23] are among ADLs of the first group, MetaH and UniCon [25, 24] 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, PRISMA [21, 20] are instances of AO viewpoints. 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 software architecture.

Another approach for aspect modeling in software architecture is to extend current ADLs with Aspect components. Pessemier et al [26] extended the Fractal ADL with Aspect Components (ACs) and they supposed crosscutting 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 can be applied on any architectural element (component, connector and configuration).

3 Aspectable NFRs

This paper is not going to propose any aspect elicitation method while, 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. So these quality attributes can not be considered as Aspectable NFRs because their dimensions (portability and adaptability for modifiability, modularity and readability for testability) do not have this potential to be explicitly replicated by architectural elements and 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 can not 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.

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. [15] 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 constraint as a crosscutting concern and illustrates how to extend that idea for other dimensions of security NFR.

3.3 Availability

Some 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, respec-
tively. 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 elate 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.

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 software architecture. Meanwhile, affixing aspectual components into 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 can not provide crosscutting service for components of all layers.

In order to enhance 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 software architecture. 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 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.

![Figure 1, Aspect Enhanced model of Software Architecture](image)

The following section will describe the extended description language which will be used for modeling aspectual components and weavers.

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 software architecture (component, connector, and configuration). This core is based on xArch [6] 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 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. Figure 2 depicts an aspectual component for non-Functional Requirement of Security which is described in xADL 2.0.
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 Appendix 1, "AspectWeaver.xsd" defines weaverCollection, weaver, pointcut and advice. Each weaverCollection consists of a set of weavers that any of them holds a set of pointcuts and advices. 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 Pointcut defines the point of description in the architectural elements at which crosscutting concerns are needed to be applied. In another word, NFRs' methods will be inserted to the software architecture at the points which are defined by pointcuts. On the other hand, each Advice specifies the crosscutting NFR's method which must be inserted at the relevant pointcut.

6 Case Study

In this section we will describe a Chat system 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 to derive out non-functional requirement of Integrity, which is known as a subset of security.

6.1 Chat System Base Architecture

Figure 3 depicts Chat System architecture prior to its enhancement with aspectual components. This architecture takes advantage of Client / Servers model in which Clients are responsible for receiving users' messages, sending them to the server and receiving other users' messages from server.

6.2 Security Aspectual Component

The first step of aspectually enhancing typical software architecture is to elicit aspects from its constituent elements. [4, 9, 18, 19] proposed some methods for aspect elicitation. However, in this paper we are not going to explain any specific aspect elicitation method and just the resultant aspect is of our interest.

One of Chat System's non-functional requirements is to ensure users that their messages will be transmitted without any change over the network. This NFR imposes utilization of integrity methods as preventive or detective mechanism of unauthorized information modification [11]. Thus, digest creation and integrity checking which are well known methods of integrity insurance, will be replicated in all of the clients and server components.

As Integrity is a subset of Security we can use the aspect component described in section 5.1, but only those methods are important for us which are related to Integrity. digestCreator method uses a hashing algorithm for creating message digest and attaches to the message. On the other hand, the integrityChecker method which will be executed on the receiver is responsible for generating the digest of received message with the same hashing algorithm and checking its equivalence with the attached digest.

As there is no direct link between clients, server plays an intermediate receiver / sender role at any communication. Thus, digestCreator and integrityChecker methods are replicated in all client and server components. And in this way integrity NFR can be considered as a crosscutting concern.

6.3 Chat System Weavers Middleware
Eliciting crosscutting methods of integrity NFR 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 4 depicts this middleware as an instance of "AspectWeaver.xsd". This instance specifies Integrity as a weaver with a set of pointcuts and advices. 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.

In this case study two pointcuts have been defined for specifying those architectural elements which may invoke send/receive methods.

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**: determination of how the method of crosscutting concern should be applied to the affected architectural element.
- **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.

This case study explains how to model a non-Functional Requirement which has crosscutting behavior, using aspctual components and how to weave its behaviors into base software architecture. It is obvious that integrity is just one of NFRs which have aspectable behavior. Confidentiality, authentication and audit trail are the other dimensions of security which have this potential to be derived out of base architectural elements, declared within security aspect component, and weave back to those elements. There is similar situation for Availability NFR with respect to its caching dimension.

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 a Chat System and been described in extended xADL 2.0.

The primary advantages of this method are:

- 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 software architecture that has a salient focus on separation of concerns, can lead us toward introducing an aspect enhanced software architecture analysis method.

Acknowledgment

Special thanks to Professor Mary Shaw for her guidelines on aspectual NFRs, and kindest appreciations to Mrs. Neda Noroozi for her contributions as xADL 2.0 expert.

References:


Appendix 1 - AspectWeaver.xsd

```xml
<xsd:schema xmlns="http://ce.sharif.edu/~seResearchLab/aspect.xsd" >

  <xsd:complexType name="Pointcut">
      <xsd:attribute name="type" type="MyString"/>
      <xsd:attribute name="id" type="ID"/>
      <xsd:attribute name="pattern" type="string"/>
  </xsd:complexType>

  <xsd:complexType name="Pointcuts">
      <xsd:sequence minOccurs="1" maxOccurs="unbounded">
         <xsd:element name="pointcut" type="Pointcut"/>
      </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="Advice">
      <xsd:attribute name="type" type="string"/>
      <xsd:attribute name="pointcut-refid" type="ID"/>
      <xsd:attribute name="method" type="string"/>
  </xsd:complexType>

  <xsd:complexType name="Advices">
      <xsd:attribute name="Component" type="string"/>
      <xsd:sequence minOccurs="1" maxOccurs="unbounded">
         <xsd:element name="advice" type="Advice"/>
      </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="Weaver">
      <xsd:attribute name="id" type="ID"/>
      <xsd:sequence maxOccurs="1" minOccurs="1">
         <xsd:element name="pointcuts" type="Pointcuts" minOccurs="1" maxOccurs="1"/>
         <xsd:element name="advices" type="Advices" minOccurs="1" maxOccurs="1"/>
      </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="WeaverCollection">
      <xsd:sequence minOccurs="1">
         <xsd:element name="weaver" type="Weaver"/>
      </xsd:sequence>
  </xsd:complexType>

  <xsd:element name="weavercollection" type="WeaverCollection"/>

</xsd:schema>
```