An Evaluation Method for Aspectual Modeling of Distributed Software Architectures
Hamid Bagheri, Vajih Montaghami, Gholamreza Safi
Seyed-Hassan Mirian-Hosseinabadi

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

Abstract

Dealing with crosscutting requirements in software development usually makes the process more complex. Modeling and analyzing of these requirements in the software architecture facilitate detecting architectural risks early. Distributed systems have more complexity and so these facilities are much useful in development of such systems. Aspect oriented Architectural Description Languages (ADL) have emerged to represent solutions for discussed problems; nevertheless, imposing radical changes to existing architectural modeling methods is not easily acceptable by architects. Software architecture analysis methods, furthermore, intend to verify that the quality requirements have been addressed properly. In this paper, we enhance ArchC# through utilization of aspect features with an especial focus on Non-Functional Requirements (NFR). ArchC# is mainly focused on describing architecture of distributed systems; in addition, it unifies software architecture with an object-oriented implementation to make executable architectures. Moreover, in this paper, a comparative analysis method is presented for evaluation of the result. All of these materials are illustrated along with a case study.

1. Introduction

In previous work [7, 10] we have proposed a method to inject Aspectable NFRs into the software architecture, which is specified by xADL 2.0 [5]. For the purpose of this work, we have extended ArchC# [8], which is designed for modeling of distributed systems, and by affixing an aspectual layer over the conventional architecture, lead to a trustable design of a distributed system, and achieve a higher degree of encapsulation which results in modifiability, reusability and testability enhancement of the software architecture.

By injecting new NFRs into Software Architecture, other NFRs will be affected. This side effect, however, should be controlled by business goals of the system. For this reason, new Architecture in which new NFRs injected could be compared by the old architecture based on given business goals; for this comparison, we introduce a new lightweight method that can compare our proposed architecture with other relevant architectures. Consequently, the resulted improvements and other side effects will be shown.

The rest of this paper is organized as follows. First, in Section 2 a brief introduction to related work is presented. Following this, Section 3 describes aspect enhanced ADL. Afterward, Section 4 defines our evaluation and comparison method. Then, Sections 5 and 6 provide a case study to illustrate our method for aspectual modeling and analysis of software architecture. Finally, in Sections 7 the outcomes of this paper are presented.

2. Related Work

In the software architecture literature, a number of methods have been developed to evaluate the quality related issues; using scenario-based approaches is common. One of the related methods to our evaluation approach is Scenario-based Architecture Analysis Method (SAAM) which assesses software architectures via scenarios, quality attributes and quality objectives [1]. On the other hand, Architecture Tradeoff Analysis Method (ATAM) concentrates on finding crucial tradeoff decisions and tries to detect the risks involved [4]. Active Reviews for Intermediate Design (ARID), moreover, focuses on the architecture of subsystems; this lightweight evaluation approach provides a procedure for the early phases of architectural design [2]. Software Architecture Comparison Analysis Method (SACAM) proposes a method for comparison of a set of software architectures possibly of different application domains [9]. Apart from SACAM, none of these approaches facilitates the comparison of several architectures of the same application domain. Nevertheless, SACAM is too sophisticated for our methodology.

3. 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 ArchC# [8] which is a new ADL for distributed systems and supports communication integrity.

As elucidated before [7, 10], aspectual components can be easily modified to reflect any of the Aspectable NFRs’ methods which are scattered among architectural components. In fact, any aspectual component can be considered as the library of one primary NFRs' methods. A component specification is a package of methods corresponding to a dimension of a primary NFR; likewise, we define subcomponent whose specification is a package of methods corresponding to a dimension of a primary NFR.

3.1 Definition of Weaver

Any aspectual component must have a weaver for mapping its methods on to based architectural elements. A weaver will be described through weaver components in ArchC#. Weaver components are defined using weaver primitive with the component keyword as you can see in figure 1. Each pointcut in a weaver component 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. As it is shown in figure 1, a pointcut could be defined in a weaver component using pointcut keyword. In addition to

```java
public weaver component componentname{
    public pointcut pointcutName{
        pattern "m()";
        type component;
        // connector| configuration
    }
    public advice adviceName{
        type before; // after| around;
        pointcutRefID pointcutName;
        method "MethodName";
        aspectComponent "ComponentName";
    }
    ...
}
```

**Figure 1, A weaver component in ArchC#.**

The name of pointcut, a pattern and also a type should be indicated in a pointcut definition. Type determines that to which of architectural elements (component, connector and configuration) the pointcut refers. The type of referenced element must be apparent at pointcut description. Pattern is a repeated behavior in architectural elements that describes the weaving condition.

Advice are also some information that should be provided for a weaver component. As you can see in figure 1, each advice will be defined using advice keyword. The type keyword is used for determination of when 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. Using method keyword, each advice indicates a method name which is an appropriate method from aspectual component. The pointcutRefID keyword is used for indicating the pointcut for which method of this advice will be applied. The aspectComponent determines the name of appropriate aspectual component.

4. Comparison Method Definition

After reorganization, we suspected that the new system is an improvement, but how do we evaluate the software architecture? To address this issue, we designed a new comparative method based on the concepts of scenario-based software evaluation methods such as ATAM, SAAM and ISO-9126-1 quality model [6].

A generalized description of the process we follow for the study is composed of the following steps:

- **Select Business Goals**: Select Business Goals for Evaluation and Refine them to quality characteristics
- **Extract architectural services**: Extract architectural services from architecture models and architectural styles
- **Quality Model Identification**: Define, Select and Customize quality model
- **Quality Measurement**: Measure quality metrics are refined from quality characteristics
- **Compare and Reporting**: Prioritize quality characteristics and architectural services upon business goals

4.1 Select Business Goals

A system can be evaluated with different goals in mind and from many different perspectives. An evaluation can be established upon whether the system implements the specified functional requirements or whether it fulfills the non-functional requirements.

However, we want the architecture evaluation and comparison to focus on those aspects of architecture that are important for achieving the business objectives of the system. The business goals that should be supported by envisioned system, or systems, are elicited from stakeholders. Business goals can be found in business environment, stakeholder needs, business constraints, a description of technical constraints, quality attribute requirements and from what business needs they are derived [3]. After business drivers are specified, an architect or an evaluation team could easily relate them to quality characteristics.

Quality characteristics, like those found in the ISO9126-1 standard, define quality specification refinement and criteria for a certain capability area of the architecture. The quality characteristics are used as the targets for validation and verification at the various stages of development. They, moreover, facilitate the interactions among software architect and evaluation team.
4.2 Extract Architectural Services
In our method, “Architectural Service” is a definition of the basic functionality and purpose of the software intensive system in an abstract way. Compared to application scenarios [4], which are often very specific and may change over the lifetime of the architecture, architectural services provide a more high-level and stable view. Conceptually, services are roughly similar to architectural view types [9], as they determine which architectural aspects are assessed. However, in contrast to architectural view types, services are motivated by the functional requirements and not by technical issues found in the architectural documentation.

4.3 Quality Model Identification
In general, different domains handle different terminology, and a kind of dictionary has to be established. For example, if we mention performance, it must be specified whether we deal with time behavior or resource utilization and all of these attributes must be defined. We select ISO9126-1 as the base quality model; it proposes a set of six independent high-level quality characteristics. The model, however, should be adapted or customized to the specific application or product domain. Quality characteristics demonstrated in the first step, are the subset of ISO9126-1 quality model. They are refined into sub-characteristics, until the attributes or measurable properties are obtained.

4.4 Quality Measurement
The concrete service implementation of the evaluated architecture is examined and assessed with respect to the identified quality characteristics and sub characteristics of the system. This is done by employing the quality attribute metrics. It is important to select right metric and measurement method, not just any metric to any quality attribute. Each of these metrics yields a single evaluation result on a normalized scale from 0 to 100. Two kinds of measurement methods exist at architectural level specification. First, some quality characteristics of the software, like performance, can be precisely pointed out. On the other hand, some quality characteristics, such as security, cannot be quantified, but may be defined as present in the software in a certain level. This certain level is appointed by extracting architectural mechanism. An architectural mechanism is a structure where objects collaborate to provide some behavior that helps achieving specific quality goals [3].

4.5 Compare and Reporting
When all architecture services have been assessed with respect to all quality attributes, the evaluation results may be entered into the weighted quality comparison matrix. This leads to a single, normalized evaluation result, characterizing the fitness of the architecture compared to other architectures and providing a hint for its absolute value with respect to the criteria captured within the quality model. These weights are assigned by two approaches. First, they are extracted from business goals which demonstrated in the first step. As a case in point, the main goal of the system is to make it easier to add and modify functionality to the system in the future and support distributed development. According to this goal, we assign high weights to some quality characteristics, such as maintainability and portability. Second, they are extracted from architectural styles. After extracting the patterns in architecture, we can relate major quality attributes to the architectural services. By this relation, the priority of each quality characteristic will be distinguished in the context of whole architecture and architectural services.

5. Case Study
In our previous work [10], we described Content Management System (CMS) architecture. The focus of this case study is on deriving out Aspectable NFRs of Access Control, Integrity, and Auditing which are known as dimensions of security, and evaluating the result.

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

![Figure 2, part of the CMS architecture [10]](image)

Figure 3 depicts this middleware which specifies security as a weaver with some pointcuts and corresponding advices. As described before, each pointcut has three attributes: id, type and pattern. In figure 3, one pointcut has been shown for specifying components from based architecture. This pointcut describes components which may invoke send methods for sending a document. 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 let the system continue 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.

```java
public weaver component security{
    public pointcut send{
        pattern *Send(content)*;
        type component;
    }...
    //other pointcuts are not shown here

public advice al{
    type before;
    pointcutREPID send;
    method "Integrity.digestCreator";
    aspectComponent "SecurityAspectualComponent";
}...
//other advices are not shown here
```

Figure 3, The Weaver Middleware of CMS System

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.

6. Evaluating new Architecture

Now we are ready to compare our new CMS architecture with the other CMS architectures like based architecture, but we do not restrict our comparison model to specific architecture. To do this, we define a matrix which is used to compare our solution to any other solutions. This matrix is constructed by our comparison method described in the Section 4. If we want to compare other CMS architecture to this architecture, just iterate step 5 of this method and fill comparative matrix.

6.1 Step 1

First of all, we should specify CMS definition and extract business goals which architecture should provide. In spite of specific usage of CMS, we choose a general definition of it as a source for extract business goals. The following list describes its business goals.

- The ability to publish the content to the repository to support on demand access to the content
- The ability to search and retrieve as soon as requests arrived to the repository
- The ability to identify all users and their content management roles
- The ability to track and manage multiple versions of a single instance of content
- Ensure users that documents will be transmitted without any changes over the network
- The ability to manage new content formats and support new clients

Based on ISO9126-1 quality characteristics, the first and second goals relate to efficiency characteristic; moreover, the third to fifth ones involve in functionality of the system; finally, the last goal is related to maintainability. Consequently, Functionality, Maintainability, and Efficiency quality characteristics are vital for the system architecture of a CMS.

6.2 Step 2

In order to evaluate architectures using our evaluation model, it is necessary to extract the architectural service based on the based architecture which is defined in Section 5.1. CMS server software provides various services for its clients. The following architecture services have been identified as crucial for the evaluation of different architectural solutions:

- **Document Management**: it is responsible for maintaining the structure (folders) of the content and fetch requested content on demand.
- **Version Management**: it is responsible for managing multiple versions of content in the system.
- **Communication Management**: sending and receiving of the request with other systems and clients is performed by this component.

6.3 Step 3

To regulate comparison process, based on the result of the first step, functionality, maintainability and efficiency are the most significant quality characteristics. These characteristics will be refined to several sub-characteristics for better evaluation and measurement. As we mention to business goals which are related to functionality characteristic; these goals lead us to security sub-characteristic of functionality. Besides, Time behavior or performance is one of the most important aspects of this system derived from Efficiency. Based on the other business goal, maintainability is reduced to
changeability and Stability. At the architectural level these two sub-characteristics expressed by Coupling and Cohesion [11].

6.4 Step 4
Now, we define a set of measurement methods to measure security, performance, Coupling and Cohesion. Security is the ability to prevent any attempt to affect any of security dimensions adversely. At the architectural level, it means to have a mechanism or device (software or hardware) to perform explicitly this task. It may be a component or functionality integrated into a component. We measure security attribute by assigning a level of securitizing architecture. Presenting of each security mechanism increases the level of this sub-characteristic. Access Control, Authentication, Integrity, Audit Trail and Non-Repudiation are some of these mechanisms.

Performance is the capability of the software product to provide appropriate response time, processing time and throughput rates when performing its function under stated conditions. It is an attribute that can be measured for each of the system’s functionalities. The attribute is computed by summing components’ time behavior involved in each architectural service.

Coupling is a global property of the architecture relative to the exchanges between components [11]; its attributes can be measured for each couple of connected component using fan-in, fan-out metrics. Cohesion expresses the topology of the architecture, as the number of components depending on one component [11]. It is an attribute computed for each component by metrics involving size.

To imply these attributes to services, we cut architecture by services. On the other words, we extract components related to a service and compute these metrics on them.

6.5 Step 5
The quality computation technique is based on a two-dimensional matrix with the architecture services in its rows and the quality attributes in its columns.

For every combination of quality sub-characteristic and architecture service, the method provides a quality attribute metrics which determines a value normalized to the range from 0 to 100. Usually, the same quality attribute metrics will be used for all architecture service alike although this is not mandatory. In tables 1 and 2, these values can be found in the Value columns.

In our method, we have three types of weight. Characteristic Weights were obtained from business goals and the domain of application, and prioritized them. In our case study, we assign 50 to functionality, because there are many goals corresponding to it, and allot 20 points to maintainability on account of the fact that there are no high level modifications in this system. Furthermore, sub-characteristics are prioritized against to other sub-characteristics of the same quality characteristic; therefore, as there is only one sub-characteristic in each of functionality and efficiency characteristics, these weights set to 100 for security and performance, and we assign equal points to coupling and cohesion in maintainability characteristic.

Finally, each architectural service gets a weight in the context of sub-characteristic; these weights are determined from two source of information: business goals and architectural styles. In security sub-characteristic, based on the fundamental roles of Document management and Communication management services in security concerns of CMS system, all data is stored and transferred using these services, we allot 40 points to each of them. Moreover, in performance column, based on their high level frequency of invoking these services, we allot 40 points to each of them. Owing to the fact that CMS should support new format of contents, Document management and version management have high rates of changeability.

<table>
<thead>
<tr>
<th>Architectural Services</th>
<th>Quality Characteristics</th>
<th>Functionality</th>
<th>Efficiency</th>
<th>Maintainability</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Security</td>
<td>Performance</td>
<td>Coupling</td>
</tr>
<tr>
<td>Document Management</td>
<td>W 40</td>
<td>V N/A</td>
<td>50(1/2)</td>
<td>40 100 (1)</td>
</tr>
<tr>
<td>Version Management</td>
<td>20  N/A</td>
<td>20 50(1/2)</td>
<td>40 100 (1)</td>
<td>40 10 (1)</td>
</tr>
<tr>
<td>Communication Management</td>
<td>40 N/A</td>
<td>20 100 (1)</td>
<td>20 50(1/2)</td>
<td>20 10 (1)</td>
</tr>
<tr>
<td>Sub-Characteristic Weight</td>
<td>100 100</td>
<td>50</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Characteristic Weight</td>
<td>50</td>
<td>30</td>
<td>20</td>
<td></td>
</tr>
</tbody>
</table>

Table 1. Comparison matrix for based Architecture

After prioritizing services, the values of each service should be computed in the context of related sub-characteristic. As a case in point, for the Communication Management service in enhanced architecture, Table 2, communication manager component interact to three aspectual components. Consequently, three security mechanisms are explicitly demonstrated and this service takes 3 points in the context of security. Based on the measurement method which is described in the step 4, Communication Management service should be done by four components. By reversing this value, an appropriate
value has been initiated and the value of service in the context of efficiency is 1/4.

Coupling and Cohesion are computed by the components that involved in the scope of Communication Management services. As is mentioned before, coupling can be measured for each couple of connected components using fan-in, fan-out metrics. Presumably, each component has a suitable cohesion; thus, we can compute cohesion of each service by accumulating all of constituent components’ cohesion.

The results of other services are computed and normalized with this routine and depicted in tables. These two tables show our enhanced architecture is better than based architecture in security concept and cohesion, but performance and coupling attributes are decreased. On the based architecture, no explicit definition of security is available; thereby, there are assigned to not available.

<table>
<thead>
<tr>
<th>Architectural Services</th>
<th>Quality Characteristics</th>
<th>Functionality</th>
<th>Efficiency</th>
<th>Maintainability</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Security</td>
<td>Performance</td>
<td>Coupling</td>
</tr>
<tr>
<td>Document Management</td>
<td></td>
<td>W 20(2)</td>
<td>40 14(1/7)</td>
<td>40 33(1/3)</td>
</tr>
<tr>
<td>Version Management</td>
<td></td>
<td>20 20(2)</td>
<td>40 14(1/7)</td>
<td>40 33(1/3)</td>
</tr>
<tr>
<td>Communication Management</td>
<td></td>
<td>40 30(3)</td>
<td>40 25(1/4)</td>
<td>20 25(1/4)</td>
</tr>
<tr>
<td>Sub-Characteristic Weight</td>
<td></td>
<td>100</td>
<td>100</td>
<td>50</td>
</tr>
<tr>
<td>Characteristic Weight</td>
<td></td>
<td>50</td>
<td>30</td>
<td>20</td>
</tr>
</tbody>
</table>

Table 2. Comparison matrix for Enhanced Architecture

7. Conclusion

In this paper we described a method for enhancing software architectural modeling 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 based architecture elements. Furthermore, we have proposed an evaluation method for assessing 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 ArchC#.

The primary advantages of this method are:
- Injecting crosscutting concerns like security into the Software Architecture without performing any changes in the base Software Architecture.
- Reducing native complexity in the modeling of the distributed systems by eliciting crosscutting concerns
- Achieving higher degree of separation of concerns.
- Designing more cohesive components.
- Improving maintainability, testability and modifiability of architectural elements.
- Assuring the communication integrity and solving inconsistency problems between implementation and software architecture
- Providing a detailed analysis of system in early-phases of software development process
- Extending our measuring metrics upon distinctive domains
- Enhancing business goal refinement method to determine quality attributes and their priorities

8. References