Towards Modeling Non-Functional Requirements in Software Architecture

Lihua Xu, Hadar Ziv, Debra Richardson
University of California, Irvine
{lihuax, ziv, djr} @ ics.uci.edu

Zhixiong Liu
California State University, Fullerton
zl135161@student.fullerton.edu

Abstract

Functional requirements (FRs) capture the intended behavior of the system, in terms of the services or tasks the system is required to perform, while non-functional requirements (NFRs) are requirements that impose restrictions on the product being developed [3]. Despite the obvious importance and relevance of non-functional requirements, NFRs specified during requirement engineering are almost always left to be verified after the implementation is finished, which means NFRs are not mapped directly and explicitly from requirements engineering to architectural design. This leaves software development with potential exacerbation of the age-old problem of requirements errors that are not detected until very late in process. Modeling and analyzing NFRs in software architectures at least partially overcomes this deficiency.

This position paper introduces our early research, which proposes a grouping mechanism, similar to the separation of concerns achieved in Aspect-Oriented Programming (AOP), to model NFRs in software architectures directly and explicitly. This eventually directs our ultimate research goal, which is to verify software architecture with respect to NFRs defined in requirements engineering. An example of how we model NFRs in software architecture is provided using the canonical KLAX example for the C2 architectural style.

1. Introduction

We are conducting research in the area of analysis and testing of software architectures against one or more desirable software qualities. The importance of system-wide software qualities has long been recognized in software engineering [1], specifically that the earlier the analysis of these qualities is conducted, the less effort is needed later in the software lifecycle.

We are specifically interested in the well-documented distinction between functional requirements (FRs) and non-functional requirements (NFRs), since it leads to new and intriguing research questions, including: Can FRs and NFRs be specified in the same language or formalism, and should they be? Are the processes for mapping FRs and NFRs to software architectures the same or different? Are techniques for analysis and testing of architectures against FRs and NFRs the same or different? and so on. In particular, analyzing the architecture against a number of critical NFRs, such as security, usability, integrity, and performance, early in the architecture design phase, becomes more crucial [8], and NFRs have been observed to be frequently neglected or forgotten in software design [7]. It would be very expensive if the resulting system is unable to support certain NFRs due to an inappropriate software architecture design. Our research is therefore targeted at analyzing non-functional requirements in software architectures to verify that the architecture is designed properly to support certain desirable attributes.

To analyze a given software architecture with respect to NFRs, NFRs must be represented in software architectures. Since NFRs are observed often as cross-cutting concerns in the literature, we follow the trend to modularize these cross-cutting concerns as aspectual components, which, in our approach, semantically describe the operations needed to satisfy [1] the corresponding NFRs. Introducing NFRs as aspectual components into an existing architecture is not, however, straightforward, since most architectural styles have rules and constraints that would typically prevent the addition of such components (in other words, an aspect would be an "illegal" component in most styles). To that end, we propose the use of a grouping mechanism following the principle of Separation of Concerns similar to its use in Aspect Oriented Programming (AOP). The approach considers the software architectural design includes two phases: one is the traditional software architecture design, with the goal of structuring the system to provide functional services; another one is to push the constraints, the NFRs defined in requirements engineering phase, into the architecture to verify that the architecture designed in the first phase is appropriate in terms of meeting NFRs.

The motivations for our work are two-folded:

1. Providing improved support for early specification of NFRs during software architecture design hence offering a better means of separation of cross-cutting functional and nonfunctional concerns in architecture level.
2. Supporting architectural analysis against NFRs early in the software lifecycle hence establishing confidence of properly chosen architecture style and designed architecture before the architecture is implemented.

This paper is organized as following: Section 2 outlines our overall proposed approach with descriptions

---

[1] NFR satisfying suggests that the solution used is expected to satisfy within acceptable limits, this is a similar notation used by Mylopoulos [4].

of the conceptual design model that supports layered representation and product line architectures. Section 3 shows a small example with C2 architecture with carrying out our approach. Section 4 reviews our current work and introduces challenges and directions for future work.

2. The Proposed Approach

The approach rests on two well-known software engineering principles – namely, separation of concerns and modularity -- and applies them to software architectural design and analysis. We argue that modeling and analyzing FRs and NFRs should be considered separately; while still modeling and maintaining their many-to-many interrelationships (i.e., one FR is related to and affected by many NFRs, one NFR relates to many FRs, etc.). This follows the same argument as used for Aspect-Oriented Programming (AOP), namely that without separation of concerns, we face the risk that FRs and NFRs will be modeled and analyzed in a scattered and tangled fashion and that resulting software architectures, designs, and implementations will be "scattered and tangled" as well. Our approach to architectural modeling uses *aspectual components* to describe the operations for satisfying corresponding NFRs. An aspectual component differs from a regular component in a traditional software architecture language such as UML or C2xADL, in that it has different rules and constraints. For instance, if one set of operations of satisfying security is to add one more authentication step for every component that changes the system's state, then the corresponding aspectual component will be distributed to (i.e, "woven with") all of these components; hence, it is inappropriate to simply add a normal component into the designed architecture to describe the operations. In fact, the composition rules of most architectural styles – such as C2 – do not support and will not allow a component with the characteristics of an aspect.

Our approach therefore proposes, a conceptual architectural design model, which views the traditional architecture model for software systems as one layer, and where the operations of satisfying NFRs are represented as aspectual components in another layer in the model, and the means of weaving these aspectual component into the traditional architecture is done by a special connector: XML Binder, which uses eXtensible Markup Language (XML) to represent the semantics of where and how the aspectual component should be woven with the corresponding components in the traditional architecture.

Previous work on systematically identifying and representing NFRs [2, 4] has shown that NFRs are often cross-cutting concerns of the system-to-be, and that most of them are operationalizable. Our approach builds upon these earlier observations, as follows:

- We propose that aspectual components be used to represent the semantics of those operationalized NFRs; these components correspond to advice tasks in the aspect-oriented world,
- The connectors between the software architecture layer and the NFR layer describe binding rules, thus corresponding to the pointcut from the aspectual component to the normal components.

Figure 1 illustrates the conceptual design model proposed in this paper. The first layer in the model is the "typical" structural description of the system – the traditional software architecture, while the second layer captures the semantics of operationalized NFRs, which represent the constraints designed to be distributed over (i.e., "woven with") one or more components in the traditional architecture of the first layer. The connectors of the design model are actually the semantics of weaving rules for each aspectual component, represented by XML Binders.

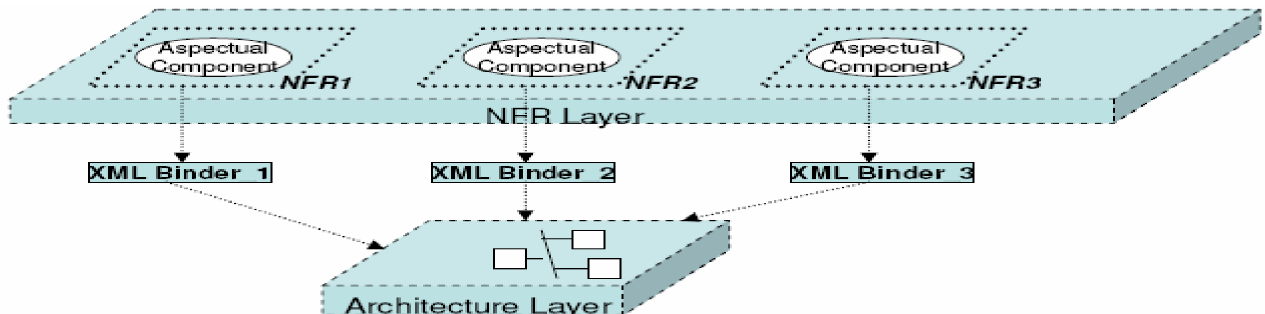

Figure 1. Conceptual Design Model with one architecture

We propose to use the same XML Binders detailed in the Aspect Markup Language (AML), proposed by Lopes et. al. [10], specifically, the format of *Aspect Binding* proposed there. Our XML Binders are therefore XML-based binding specifications that provide weaving instructions to determine how aspectual components and the traditional software architecture are to be composed together.

Moreover, with the shift in software engineering paradigm from monolithic, standalone application to componentized reusable product line systems, proper

reuse of available components and other architectural elements is crucial to successful enterprise architecture design. We contend that the architectural model presented in this paper also supports product line architectures. As depicted in Figure 2, aspectual components are maintained in the NFR layer, while multiple architectures described in the architecture layer can be woven with aspectual components in the NFR layer by using different XML Binders. This effectively allows a many-to-many-to-many architectural style, in that one or more aspectual components can be linked by one or more XML Binders to one or more traditional components.

into aspectual components. For the sake of space, we are only going to discuss one resulting operationalized NFRs for illustration, but all NFRs can be discovered and operationalized accordingly.

We consider the crosscutting requirement *confidentiality*, whereby the game requires only one person play at a time to ensure the score earned is attributed correctly. Thus, the operationalized NFR is to verify the events passing around the system are the ones requested by the only player, hence, all the events received by any component should be checked against their attached playerID before any update action taken.

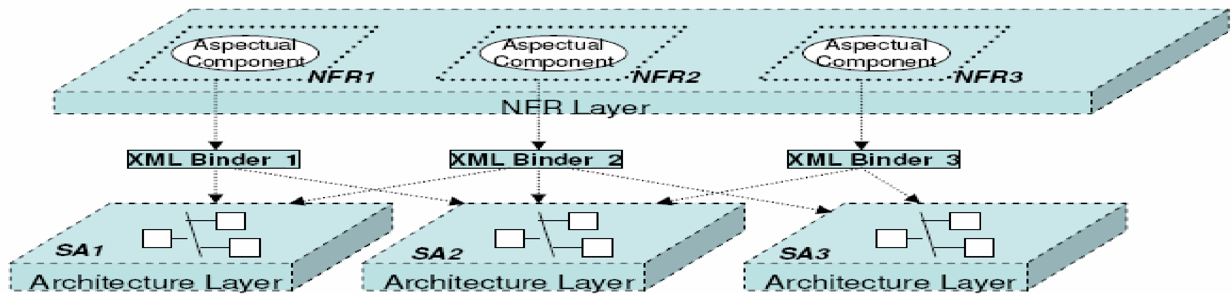

Figure 2. Conceptual Design Model with product line architectures

3. Example

To formulate a clear understanding of our approach, a small example is presented in this section that applies our approach to a known example for the C2 architectural style.

3.1. KLAX Example

The canonical example used for demonstrating the C2 architectural style is a video game called KLAX presented, for example, in [9]. The game includes three parts: KLAX Chute drops tiles of random colors at random times and locations; KLAX Palette catches tiles coming down the Chute and drops them into the Well, and Well removes horizontal, vertical, and diagonal sets of three or more consecutive tiles of the same color the collapse down the tiles above them to fill in the newly-created empty spaces.

The game calculates the scores accordingly. The C2 architecture designed for the game of KLAX is shown in Figure 3, where ADT components encapsulate the game's state, Logic components request changes of ADT state in accordance with game rules and interpret ADT state change notifications to determine the state of the game in progress, and artist components maintain the state of a set of abstract graphical objects.

In the first step of applying our approach to KLAX, the first layer of our architectural design model would maintain the original C2 architecture as shown in Figure3.

3.2. Aspectual Components

By following the discovering procedure proposed in [2], we are able to derive and operationalize the NFRs

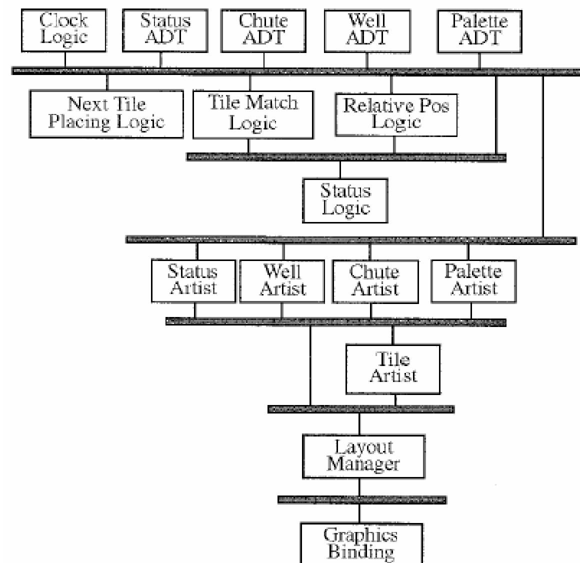

Figure 3. C2 architecture for KLAX [9]

Therefore, the derived aspectual components would include the advising task *PlayerIDProtection()*. The syntax of representing the aspectual component is shown below:

```
Component aspect ConfidentialityInterceptor {
        PlayerIDProtection () {
            // the code for checking PlayerID goes here
        }
        …
}
```

3.3. XML Binder

The XML Binder is actually the connector in our conceptual design model, each of which consists of *pointcut* and *interceptor* definitions. The pointcut definition specifies the matching conditions, represented as signature patterns, in the traditional software architecture components. The interceptor definition provides a means to weave the crosscutting behaviors, it specifies the name of the an interceptor component among aspectual components, and the methods of the interceptor are invoked when certain matching conditions are matched;

```
<xml>

  <Binder id = "confidentiality">

    <pointcut  id="Update">
      <or>
        <pointcut type="component"
    pattern="Received(event)&&Preparing(action)"  />
      </or>
    </pointcut>

    <interceptor
        Component = "ConfidentialityInterceptor">
      <advice
        type = "before"
        pointcut-refid = "update"
        interceptor-method  =  "PlayerIDProtection()"
      />
    </interceptor>

  </Binder>

</xml>
```

Figure 4. XML Binder for Confidential NFR

Figure 4 shows how the operationalized *confidentiality* NFR can be woven with the normal component and what are the joint points that define the matching conditions. The first section of the snippet defines the binder for *confidentiality* that consists of a pointcut and an interceptor. The pointcut picks out all the components that received events and are preparing to take some update actions. The interceptor definition uses a nested advice definition to specify that *PlayerIDProtection()* method of *ConfidentialityInterceptor* aspectual component (shown in Section 3.2) is an advice for the *update* pointcut. This method is invoked just before any *update* point is executed.

In this example, we showed how one NFR can be operationalized and represented as an aspectual component, and the binding rules for this particular aspectual component with the traditional architecture. For systems that have scattered and tangled crosscutting NFRs, or product line architectures, architects can define their own *pointcut* and *interceptor* definitions accordingly.

4. Conclusions and Future Work

The architectural design model proposed here offers a multi-dimensional view of software architecture design: one dimension includes the traditional architecture design, following an architectural style such as UML or C2 to structure the system to support required services; the second dimension reflects cross-cutting non-functional requirements, and imposes constraints for making the architecture designed correspond and "implement" those NFRs. Finally, an implied third dimension represents the resulting enterprise software architecture design, which satisfies both functional requirements and nonfunctional requirements. Here, we plan to investigate the possibility of "weaving" the architectural views corresponding to FRs and NFRs into the resulting enterprise architecture – what does it mean and can it be done at the architectural level?

We anticipate several potential benefits to modeling NFRs in architectural designs; the primary benefit of interest to us is the ability to analyze and verify the software architecture before it implemented. Currently, we are experimenting with modeling and analyzing several NFRs for C2 architectures by extending Argus-I [5], a comprehensive set of specification-based analysis tools focusing on both the component and architecture levels. More specifically, the existing Argus-I toolset supports behavior descriptions for C2 architectures by StateChart specifications, which may be considered as the first layer at our model. We are trying to augment it with an NFR layer consisting of multiple NFRs, such as security, performance, and modifiability, etc.; this augmented model will then be subject to various analyses.

Acknowledgement

The authors would like to appreciate great comments and help given by Prof. Andre van der Hoek. The authors wish to also thank the meaningful discussions with ROSATEA group at UC Irvine.

References

1. L. Chung, D. Gross and E. Yu, "Architectural Design to Meet Stakeholder Requirements". Working IEEE/IFIP Conference on Software Architecture (WICSA99).
2. Y. Yu, J.C. Leite, J. Mylopoulos. "From Goals to Aspects: Discovering Aspects from Goal Models", RE'04. 2004.

3. G. Sousa, and J.F.B. Castro. "Supporting Separation of Concerns in Requirements Artifacts". First Brazilian Workshop on Aspect-Oriented Software Development (WASP'04), 2004, Brazil.
4. J. Mylopoulos, L. Chung and B. Nixon. "Representing and Using Non-Functional Requirements: A Process-Oriented Approach". IEEE Transactions on Software Engineering, Vol18. June, 1992.
5. M. Dias, M. Vieira, D. Richardson, "Software Architecture Analysis based on Statechart Semantics', Proceedings of the 10th International Workshop on Software Specification and Design (IWSSD-10), 8th International Symposium on the Foundations of Software Engineering (FSE 2000), San Diego, USA, Nov. 2000.
6. A. Rashid, A. Moreira, and J. Araujo, "Modularization and Composition of Aspectual Requirements." 2nd International Conference on Aspect-Oriented Software Development. ACM.
7. L.M.Cysneiros, E.Yu, J.C.S.P. Leite, "Cataloguing Non-Functional Requirements as Softgoal Networks" in Proc. Of. Requirements Engineering for Adaptable Architectures @ 11th International Requirements Engineering Conference, 2003 p.13-20
8. M.R. Barbacci, "SEI Architecture Analysis Techniques and When to Use Them", Technical Note, CMU/SEI-2002-TN-005.
9. R.N.Taylor, N.Medvidovic, K.M.Anderson, E.J.Whitehead Jr., J.E.Robbins, K.A. Nies, P.Oreizy, and D.L.Dubrow, "A Component- and Message-Based Architectural Style for GUI Software", IEEE Transactions on Software Engineering, June 1996.
10. C.V. Lopes and T.C. Ngo, "The Aspect Markup Language and its support of Aspect Plugins", ISR Technical Report UCI-ISR-04-8, University of California, Irvine.