Defining Architectural Viewpoints for Quality Concerns

Bedir Tekinerdogan[1] and Hasan Sözer[2]

[1] Bilkent University, Department of Computer Engineering
Bilkent 06800 Ankara, Turkey
bedir@cs.bilkent.edu.tr
[2] Ozyegin University, Department of Computer Engineering
Istanbul, Turkey
hasan.sozer@ozyegin.edu.tr

Abstract. A common practice in software architecture design is to apply architectural views to model the design decisions for the various stakeholder concerns. When dealing with quality concerns, however, it is more difficult to address these explicitly in the architectural views. This is because quality concerns do not easily match the architectural elements that seem to be primarily functional in nature. As a result, the communication and analysis of these quality concerns becomes more problematic in practice. We introduce a general and practical approach for supporting architects to model quality concerns by extending the architectural viewpoints of the so-called V&B approach. We illustrate the approach for defining recoverability and adaptability viewpoints for an open source software architecture.

Keywords: Software Architecture Modeling, Architectural Views, Quality Concerns.

1 Introduction

An architectural view is a representation of a set of system elements and relations associated with them to support a particular concern [2]. Having multiple views helps to separate the concerns and as such support the modeling, understanding, communication and analysis of the software architecture for different stakeholders. Architectural views conform to viewpoints that represent the conventions for constructing and using a view. Because of the different concerns that need to be addressed for different systems, the current trend recognizes that the set of views should not be fixed but multiple viewpoints might be introduced instead. Certainly, existing multi-view approaches are important for representing the structure and functionality of the system and are necessary to document the architecture systematically. Yet, an analysis of the existing multi-view approaches reveals that they still appear to be incomplete when considering quality concerns. The ISO/IEC 42010 [4] standard intentionally does not define particular viewpoints to address the different concerns. In the V&B approach, quality concerns appear to be implicit in the different views but no specific viewpoints have been proposed to represent quality concerns. One could argue that for addressing quality concerns software architecture analysis approaches have been introduced. The difficulty here is that these approaches usually apply a separate quality model, such as queuing networks or process algebra, to analyze the quality properties. Although these models represent precise calculations they do not depict the decomposition of the architecture and an additional translation from the evaluation of the quality model needs to be performed. To represent quality concerns more explicitly, preferably an architectural view is required to model the decomposition of the architecture based on the required quality concern. In this context, we introduce an approach for defining architectural viewpoints for modeling quality concerns. We illustrate the approach for two different quality concerns; recoverability and adaptability. The approach is applied to the open source media player application, MPlayer.

The remainder of this paper is organized as follows. Section 2 introduces the case study and the problem statement in which we describe the need for architectural decomposition for quality concerns. Section 3 describes the concepts for modeling architectural viewpoints for quality concerns. Section 4 provides the related work. Finally, section 5 provides the conclusions.

2 Problem Statement

2.1 Case Study: MPlayer

MPlayer [6] is a media player, which supports many input formats, codecs and output drivers. It is available under the GNU General Public License. Figure 1 presents a simplified module view of the MPlayer software architecture with basic implementation units and direct dependencies among them.

Here *Stream* represents the module that reads the input media and provides buffering, seek and skip functions. *Demuxer* demultiplexes (separates) the input to audio and video channels, and reads them from buffered packages. *Mplayer* connects all the other modules, and maintains the synchronization of audio and video. *Libmpcodecs* embodies the set of available codecs. *Libvo* displays video frames. *Libao* controls the playing of audio. *Gui* provides the graphical user interface (GUI) of MPlayer.

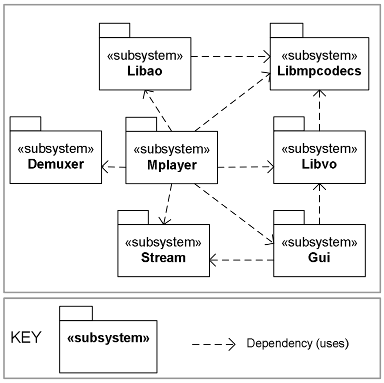


Fig. 1. Module View of the MPlayer Software Architecture

2.2 Architectural Decomposition for Quality Concerns

When designing the architecture for the MPlayer besides of the functional concerns also non- functional concerns have to be taken in to account. In the following, we will consider the recoverability and adaptability concerns for MPlayer.

Recoverability refers to the ability to recover from errors [1]. Recoverability has a separate impact on the system and is usually not always aligned with the individual components in the system. Figure 2a represents an example of the required decomposition of the architecture for recoverability in the MPlayer case. Here, a decomposition unit is called *recoverable unit (RU)*. Each RU should be independently recoverable. As we can see in Figure 2a, three recoverable units have been defined: *RU AUDIO, RU MPCORE*, and *RU GUI*. In fact, Figure 2a provides two views on top of each other, the module view and the view related to recoverability, which overlays the module view. Obviously many different decomposition alternatives are possible. Each design alternative will require a different impact on the system. Unfortunately, the architectural decomposition in Figure 1 is not sufficient to communicate design decisions about the recoverability. On the other hand, although Figure 2a provides the impact of recoverability it models two concerns at the same time and likewise it violates the separation of concerns principle. A more complicated decomposition for recovery would be harder to model, and in case more than one concern needs to be modeled the model becomes less useful for communication about the concerns. Both figures are also less suitable to support the analysis of recoverability and/or to guide the implementation of the system based on the architecture.

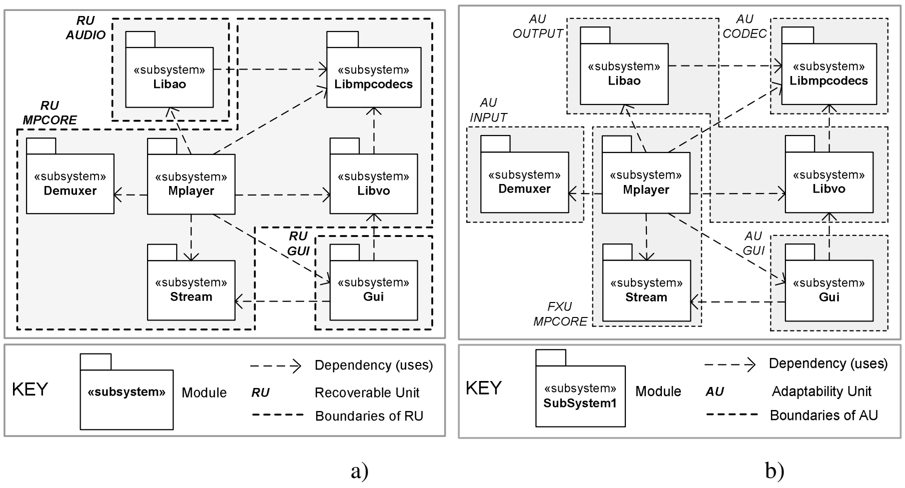


Fig. 2. Required decomposition for MPlayer architecture for Recoverability (a) and Adaptability (b)

Adaptability is defined as the ease with which a system can change [5]. There are several types of adaptation techniques applied in practice. These techniques are applied at different phases (i.e., compile-time, run-time) and at different abstraction levels (e.g., source code, architecture description). Knowing the adaptability properties of architectural components early on is important to communicate and guide the system development. In addition, similar to recoverability, one may define different architecture design alternatives that behave differently with respect to adaptability properties. Figure 2b shows an example decomposition of the architecture that might be required for adaptability. Here, the decomposition unit is called adaptability unit (AU). Each AU shows whether the unit is adaptable or fixed, and should define the adaptability properties. As we can see in Figure 2b five adaptability units have been defined: AU INPUT, AU OUTPUT, AU GUI, AU CODEC, and FXU MPCORE (fixed unit). Again this provides two views on top of each other, the module view and the view related to adaptability which actually overlays the module view. Unfortunately, the architectural decomposition of Figure 1 alone is not suitable to support the communication, analysis and guidance of the implementation for adaptability.

When we consider other quality concerns the situation does not seem to be different than in the case for recoverability and adaptability. Reusability will require a different view on the architecture in which, for example, the reusable components need to be depicted. Performance will require, for example to view the elements of the system based on their influence on the performance, etc. We could try to visualize these quality concerns on the base view that we are working on (dominant decomposition), however this will clutter the module view and eventually will decrease the understandability of the architectural description. In fact, this would also not be in alignment with the overall strategy in architectural view modeling, i.e. define an architectural view for the relevant concerns. As such, we believe that the relevant quality concern should also be represented using the corresponding views.

3 Quality Viewpoints

In this section, we provide an approach for defining architectural viewpoints for quality concerns. The overall process is shown in Figure 3. The process starts with defining the stakeholders of the concerns. For each stakeholder the concerns are defined which are categorized as *functional concerns* and *quality concerns*.

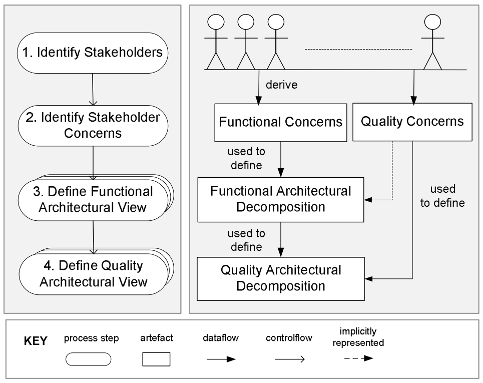

1. Identify Stakeholders
2. Identify Stakeholder Concerns
3. Define Functional Architectural View
4. Define Quality Architectural View
derive
Functional Concerns
Quality Concerns
used to define
Functional Architectural Decomposition
used to define
Quality Architectural Decomposition
used to define
KEY
process step
artefact
dataflow
controlflow
implicitly represented

Fig. 3. General Approach for Architectural View Modeling

The stakeholder concerns form an input for the architecture view modeling process. Hereby the architectural view that represents the *functional view* is described. On the other hand, quality concerns are modeled using *quality views*. For different quality concerns the architect might need to define a different architectural view. Similar to functional views, quality views will be based on architectural viewpoints. Defining a new architectural viewpoint implies writing a *viewpoint guide*. This is similar to the notion of *style guide* as defined in [2]. The *viewpoint guide* defines the vocabulary of the architectural element and relation types, and defines the rules for how that vocabulary can be used. For defining a viewpoint guide for a particular quality concern we apply the template as defined in Table 1. The viewpoint guide for quality concerns is largely the same as for the viewpoints that address functional concerns. The important difference here is that the architectural elements now are used to explicitly represent quality concerns in the architectural decomposition. Further, the quality view is applied to a functional view.

In the template of Table 1, this is defined by the field *Base View,* representing the view on which the quality view is applied. The base view could be for example the module view, component and connector view or deployment view. To make a distinction among these, the name of the viewpoint should be described accordingly, e.g. *Recoverability:Decomposition*, *Recoverability:Deployment*, *Adaptability: Process* etc. Here the symbol : refers to the mapping of the quality view on the functional view. In the following, we will give two distinct examples of the application of the viewpoint guide for quality concerns.

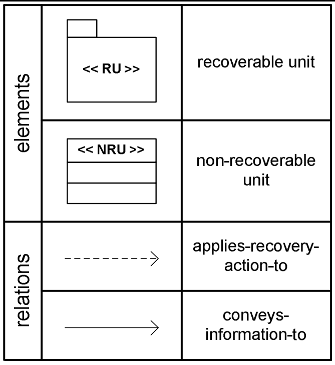
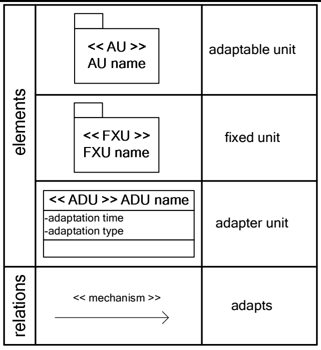
Table 1. Viewpoint Guide Template for Quality Concerns

Viewpoint Element	**Description**
Name	Unique name for the viewpoint concatenated with the view it overlays
Element Types	The architectural element types native to the viewpoint
Relation Types	The relation types among architectural elements
Properties of Elements	Additional information on the element types
Properties of Relations	Additional information on the relation types
Topology Constraints	The rules of composition of the elements and relations.
Notation	The adopted notation for the element types and relation types. The notation can be textual or visual.
Base View	The view that can be overlaid
Relation to other views/viewpoints	The relation to other viewpoints other than the base viewpoint

3.1 Example – Recoverability

Similar to the case where we separate the views for different concerns (e.g. deployment view separate from module view), we also provide a separate view for recoverability. To define the template for the recoverability view we introduce the recoverability viewpoint (Table 2) as an explicit viewpoint for depicting the architecture from the recoverability viewpoint. Unlike conventional analysis techniques that require different models, recoverability views directly represent the decomposition of the architecture and as such help to understand the structure of the system related to the recoverability concern. In essence, the recoverability viewpoint considers RUs as first class elements and represents the units of isolation, error containment and recovery control. The relation types define the relations for coordination and application of recovery actions.

Table 2. Recoverability Viewpoint (left) and Adaptability Viewpoint Guide (right)

Viewpoint Element	**Description**	
Name	Recoverability Viewpoint:Module View	Adaptability Viewpoint:Module View
Element Types	• Recovery Unit (RU) – represents a set of modules that can be recovered together, independently from other elements it is connected to. • Non-Recovery Unit (NRU) – an element that cannot be recovered independent of other RUs and NRUs.	• Adaptable Unit (AU): a set of modules that can be adapted independently from other modules of the system. • Fixed Unit (FXU): a set of modules that cannot be adapted. • Adapter Unit (ADU): an entity, which implements an adaptation mechanism.
Relation Types	• applies-recovery-action-to • conveys-information-to	• adapts
Properties of Elements	• RU: set of system modules, criticality, reliability, types of errors that can be detected, supported recovery actions, type of isolation. • NRU: types of errors that can be detected.	• AU: the set of modules, adapted properties • FXU: the set of modules • ADU: adaptation time, type of adaptation
Properties of Relations	• applies-recovery-action-to: type of communication, timing constraints. • conveys-information-to: type of communication, timing constraints.	• adapts: the type of mechanism used for adaptation.
Topology Constraints	• The target of an applies-action-to relation can only be a RU.	• the adapts relation can only be defined from an ADU to AU.
Notation	 elements << RU >> recoverable unit << NRU >> non-recoverable unit relations applies-recovery-action-to conveys-information-to	 elements << AU >> AU name adaptable unit << FXU >> FXU name fixed unit << ADU >> ADU name -adaptation time -adaptation type adapter unit relations << mechanism >> adapts
Base view	Module View	Module View
Relation to other views/viewpoints	• Dynamic views for depicting recovery scenarios.	• Deployment view for relating platform-specific adaptations.

We can document the viewpoint by using the viewpoint guide as defined in Table 1. An example application of the viewpoint guide to the MPlayer case is shown in Figure 4 (left). The figure represents the case as defined in Figure 2 but now we view the system solely from a recoverability concern perspective. The view includes three RUs and two non-recoverable units (NRUs) as first class abstractions. The relations represent the specific recovery mechanisms among the recovery units. Typically, this view can be used by reliability engineers to communicate about the reliability and fault tolerance of the system, to use this for guiding the implementation of recovery mechanisms in the corresponding units, and to analyze the different decompositions for recoverability.

3.2 Example – Adaptability

The adaptability viewpoint guide that we have defined is shown in the right column of Table 2. Here, we have identified three units and one relation. We have defined an *adaptable unit* and a *fixed unit* to differentiate software modules that are considered for adaptation from the ones that are not. We have defined an additional unit, *adapter unit*, which represents the implementer of the adaptation mechanism. This can be a part of the system or an external entity. Its attributes identify the time (compile-time or run-time) and type (manual or automatic) of adaptation implemented. The only relation defined is the *adapts* relation, which is defined from an adapter unit to an adaptable unit, emphasizing the mechanism used for adaptation. Focusing on this property of the system led us to define a decomposition (*Figure 4*) that comprises fixed and adaptable units and additional modules to support adaptability.

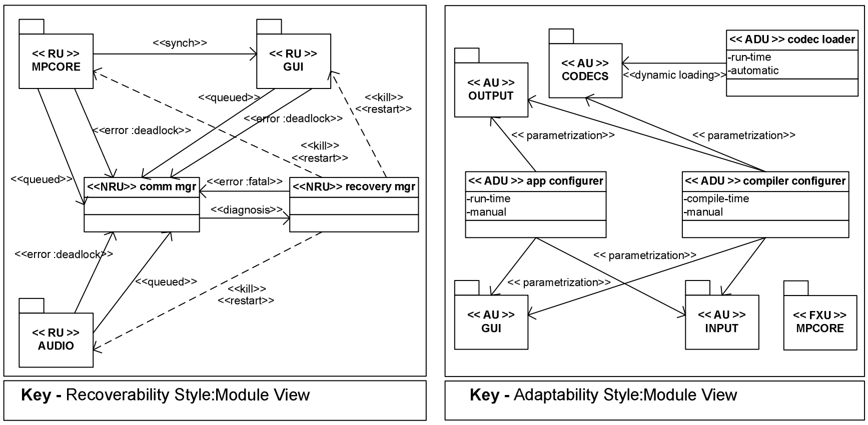


Fig. 4. Recoverability View (left) and Adaptability View (right) for MPlayer case

4 Related Work

Architectural Perspectives [8] are a collection of activities, tactics and guidelines to modify a set of existing views to document and analyze quality properties. Architectural perspectives as such are basically guidelines that work on multiple views together. An analysis of the Architectural Perspectives and our approach shows that the crosscutting nature of quality concerns can be both observed *within* an architectural view and *across* architectural views. Both approaches focus on providing a solution to the crosscutting problem. We have chosen for providing separate architectural viewpoints for quality concerns. It might be interesting to look at integrating the guidelines provided by the *Architectural Perspectives* and the definition/usage of the viewpoints developed by our approach. In that sense the approaches can also be considered as complimentary to each other.

Architectural tactics [1] aim at identifying architectural decisions related to a quality attribute requirement and composing these into an architecture design. Defining explicit viewpoints for quality concerns can help to model and reason about the application of architectural tactics.

Several software architecture analysis approaches have been introduced for addressing quality properties. The goal of these approaches is to assess whether or not a given architecture design satisfies desired concerns including quality requirements. The main aim of the viewpoint definitions in our approach, on the other hand, is to communicate and support the architectural design with respect to quality concerns. As such our work can directly support the architectural analysis to select feasible design alternatives.

5 Conclusion

The evolution of architectural view modeling can be characterized as a gradual shift from defining fixed set of multiple views to an understanding in which the set of views for architecture description is not bounded but open, dependent on the stakeholder concerns. Yet another step in the evolution of architectural view modeling is the focus on quality concerns in architectural views. From both our research activities and practical experiences in an industrial context [7] we can observe that quality concerns cannot be easily represented in current architectural views and tend to crosscut elements within an architectural view. We have proposed a solution to this problem by providing an approach for defining architectural viewpoints for quality concerns. From our experience, the explicit modeling of architectural viewpoint for quality concerns seemed to be a practical instrument [7]. Explicit viewpoints for quality concerns do not only improve the understanding and communication of the architecture but also support the analysis of these concerns. Our future work will include the analysis of quality concerns based on the architectural viewpoints that we have developed.

References

[1] Avizienis, A., et al.: Basic concepts and taxonomy of dependable and secure computing. IEEE Trans. Dependable Secur. Comput. 1(1), 11–33 (2004)

[2] Clements, P., et al.: Documenting Software Architectures: Views and Beyond. Addison-Wesley, Reading (2002)

[3] Garlan, D., Barnes, J.M., Schmerl, B.R., Celiku, O.: Evolution styles: Foundations and tool support for software architecture evolution. In: Proc. of the 7th Working IEEE/IFIP Conference on Software Architecture (WICSA 2009), pp. 131–140 (2009)

[4] [ISO/IEC 42010:2007] Recommended practice for architectural description of software-intensive systems (ISO/IEC 42010) (identical to ANSI/IEEE Std1471–2000) (July 2007)

[5] Kell, S.: A Survey of Practical Software Adaptation Techniques. Journal of Universal Computer Science 14(13), 2110–2157 (2008)

[6] MPlayer official website, `http://www.mplayerhq.hu/` (accessed March 2010)

[7] Sözer, H., Tekinerdogan, B., Akşit, M.: FLORA: A Framework for Decomposing Software Architecture to Introduce Local Recovery. Wiley Software Practice and Experience Journal 39(10), 869–889 (2009)

[8] Rozanski, N., Woods, E.: Software Systems Architecture – Working with Stakeholders using Viewpoints and Perspectives. Addison-Wesley, Reading (2005)