Future Generation Computer Systems 47 (2015) 80–96

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Harmonizing architectural decisions with component view models using reusable architectural knowledge transformations and constraints

Ioanna Lytra *, Huy Tran, Uwe Zdun

Research Group Software Architecture, University of Vienna, Austria

HIGHLIGHTS

- Reusable AK transformation language for automated transformation of ADDs into designs.
- Reusable constraints for consistency checking between decisions and designs.
- Tool support based on integration of two existing tools—ADvISE and VbMF.

ARTICLE INFO

Article history:
Received 4 November 2013
Received in revised form
24 October 2014
Accepted 10 November 2014
Available online 18 November 2014

Keywords:
Architectural decisions
Architectural design
Architectural knowledge
AK transformation language
Consistency checking

ABSTRACT

Architectural design decisions (ADDs) have been used in recent years for capturing design rationale and documenting architectural knowledge (AK). However, various architectural design views still provide the most common means for describing and communicating architectural design. The evolution of software systems requires that both ADDs and architectural design views are documented and maintained, which is a tedious and time-consuming task in the long run. Also, in lack of a systematic and automated support for bridging between ADDs and architectural design views, decisions and designs tend to become inconsistent over time. In our proposal, we introduce a reusable AK transformation language for supporting the automated transformation of reusable AK knowledge to component-and-connector models, the architectural design view used most commonly today. In addition, reusable consistency checking rules verify the consistency between decisions and designs. We evaluate our approach in an industrial case study and show that it offers high reusability, provides automation, and can, in principle, deal with large numbers of recurring decisions.

1. Introduction

From the various architectural views [1–3] used to document software architectures the component-and-connector (C&C) view is often considered the one that contains the most significant architectural information [1]. In many enterprises today, software architecture is mainly documented using component-and-connector diagrams, usually in an informal or semi-formal fashion (e.g., as box-and-line diagrams). However, architectural documentations based only on components and connectors have many disadvantages, such as limited reusability of and reasoning about architectural knowledge (AK), and lack of sharing support of this knowledge among stakeholders [4]. Therefore, the software architecture community has proposed a new perspective on software architecture through the explicit documentation of architectural design decisions (ADDs) [5]. The actual solution structure, or architectural design, is merely a reflection of those design decisions.

Several approaches have been proposed for capturing architectural design decisions. Akerman and Tyree defined a rich decision capturing template [3]. Kruchten et al. presented an ontology for architectural decisions that defines types of architectural decisions, dependencies between them, and a decision lifecycle [6]. Zimmermann et al. suggested a meta-model for decision capturing and modeling [7]. To minimize the effort of documenting architectural decisions, approaches for reusable architectural decision modeling [8] and using design patterns as a basis for documenting reusable ADDs (see, e.g., [9]) have been proposed.

ADDs play a crucial role not only during architectural design but also during development, evolution, reuse, and integration of

* Corresponding author.
E-mail addresses: ioanna.lytra@univie.ac.at (I. Lytra), Huy.Tran@univie.ac.at (H. Tran), Uwe.Zdun@univie.ac.at (U. Zdun).

http://dx.doi.org/10.1016/j.future.2014.11.010
0167-739X/

software architectures [5]. In practice, the ADDs frequently are neither maintained nor synchronized over time with the corresponding C&C diagrams (or other design views), that is decisions and designs drift apart over time [5]. This leads potentially to the loss of architectural knowledge, a phenomenon which is known as *architectural knowledge vaporization* [5,9]. Lacking of an adequate harmonization between software architectures and design decisions often leads to more severe consequences [10].

Until now, the establishment and preservation of consistency between decisions and designs have not been addressed or supported systematically. That is, so far there is no formal mapping or automated translation between ADDs and design views. Thus, the task of harmonizing decisions and designs remains ad hoc and tedious. Making matters worse, the actual documentation of ADDs is also time consuming [8], especially for kinds of ADDs that need to be made repeatedly throughout a software design process, such as many of the ADDs documented in [11,12].

In our previous work [13], we partially addressed the problem of bridging ADDs and designs. This approach introduced a formal mapping model between different ADD types, on the one hand, and elements and properties of C&C models, on the other hand. Based on this formal mapping model, preliminary component models and OCL-like constraints for consistency checking can be derived. Yet, so far this mapping model had to be manually created and modified for each ADD separately, making this approach inefficient for large numbers of ADDs and/or complex design models. Moreover, in reality, several ADDs can be reused in different software design contexts and domains [7]. Thus, taking advantage of the reusability of such recurring decisions would significantly enhance the productivity in creating and maintaining the formal mappings between the decisions and the designs. Unlike our previous work [13], we now set the focus on reusable architectural knowledge.

The approach presented in this paper aims at addressing the aforementioned challenges. In particular, our proposal introduces an architectural knowledge transformation language. This domain-specific language supports the specification of primitive and complex actions whose enactment leads to automatic updates of design models (i.e., C&C diagrams) based on the corresponding documented ADDs. The outcomes of a certain decision can be expressed either by executing individual actions, such as the creation of new elements or the deletion, modification, or grouping of existing elements in the C&C diagrams, or by executing composite actions (e.g., for capturing reusable pattern-based ADDs) that can be formally modeled through certain architectural primitives [14] or other composite actions. To ensure consistency between ADDs and C&C views, constraints are automatically generated from the execution of transformation actions. That is, the constraints ensure, for instance, that manual changes in C&C views do not violate the ADDs.

In our approach, we exploit template-based generation rules and model-driven techniques for automatically instantiating and enacting the actions, as well as generating corresponding constraints. The linking of reusable ADDs to reusable actions and constraints (in template form) offers higher reusability and automation and results in less complexity and modeling effort for software architects. The reusability is achieved here (1) through the automatic derivation of parts of the C&C diagrams and consistency checking rules using model-driven templates and (2) by reusing common abstractions shared among common design patterns (see [14]).

In order to demonstrate our approach, we integrated two tools from our previous work: ADvISE[1]—a tool for assisting architectural decision making for reusable ADDs, and VbMF[2]—a tool for describing architectural view models and performing model-driven code generation. Using this prototypical implementation we evaluated our proposal in the context of an industrial case study from the warehouse automation area, in terms of reusability and modeling effort.

Our approach solely addresses C&C views at the moment, but it is possible to integrate other architectural design aspects with architectural decisions. The incorporation of other views is supported in VbMF via its view integration techniques [15]. Illustrative examples of different aspects that are integrated using VbMF include data [16], human [17], event and runtime monitoring [18], and compliance [19]. Investigating architecture-specific information leveraging the integration of different VbMF views is beyond the scope of this paper and part of our future work.

This article is an extension of our conference publication [20] with following significant additions and improvements: (a) a more complete list of compound transformation actions and reusable constraints has been included along with further elaborations, (b) the case study and evaluations have been extended, and (c) the related work has been discussed further and compared to our contributions.

The remainder of the paper is structured as follows. First, in Section 2 we introduce some basic concepts and present briefly the ADvISE and VbMF tools and their meta-models. In Section 3 we describe the details of our approach, namely we present the reusable AK transformations and consistency checking rules along with some illustrative examples. We apply our approach in an industrial case study in Section 4 and discuss the evaluation results in Section 5. Finally, we compare to the related works in Section 6 and summarize the key contributions of our work in Section 7.

2. Background

The main focus of our study is the harmonization of architectural design decisions and architectural models. This involves the tasks of making and documenting design decisions and creating and manipulating architectural models. Thus, in this section we briefly present ADvISE and VbMF, the two tools that support these tasks and are leveraged to illustrate our approach. Before that, we introduce some key concepts that we will use throughout the paper.

2.1. Basic concepts

According to the ISO/IEC/IEEE 42 010 standard [2] an ADD affects various architectural elements, pertains to or raises concerns, and is justified by architecture rationale. ADDs capture knowledge that may concern a software system as a whole, or one or more components of a software architecture. Rather than documenting the structure of software systems (e.g., components and connectors) ADDs entail the design rationale that led to that structure (e.g., justification about the ADDs that were made and the architectural alternatives not chosen).

For capturing reusable ADDs, architectural decision modeling has been introduced in the existing literature (refer to [21] for a comparison of existing architectural decision models and tools). The advantage of these architectural decision models is that they are reusable and can thus provide guidance for architectural decision making activities, whenever recurring design issues emerge. Reusable architectural models share common concepts with patterns (see [9]) which give proven solutions to recurring problems

[1] http://swa.univie.ac.at/Architectural_Design_Decision_Support_Framework_(ADvISE).

[2] http://swa.univie.ac.at/View-based_Modeling_Framework.