Information Sciences 259 (2014) 396–411

Contents lists available at ScienceDirect

Information Sciences

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

Reusing design experiences to materialize software architectures into object-oriented designs

Germán Vazquez [a,b,*], J. Andres Diaz Pace [c], Marcelo Campo [a,b]

[a] ISISTAN Research Institute, Facultad de Cs. Exactas, UNCPBA Campus Universitario, Paraje Arroyo Seco, Tandil 7000, Argentina
[b] CONICET, Comisión Nacional de Investigaciones Científicas y Técnicas, Argentina
[c] Software Engineering Institute, Carnegie Mellon University, 4500 Fifth Ave., Pittsburgh, PA 15232, USA

ARTICLE INFO

Article history:
Received 7 March 2008
Received in revised form 11 March 2010
Accepted 12 March 2010
Available online 20 March 2010

Keywords:
Architecture design
Object-oriented design
Architecture materialization
Software reuse
Case based reasoning

ABSTRACT

Software architectures capture early design decisions about a system in order to fulfill relevant quality attributes. When moving to detailed design levels, the same architecture can accept many different object-oriented implementations. A common problem here is the mismatches between the quality-attribute levels prescribed by the architecture and those realized by its object-oriented materialization. A significant step towards reducing those mismatches is the provision of tool support for assisting developers in the materialization of software architectures. Prerequisites to develop materialization tools are the organization of a body of design knowledge and the definition of quality-driven reasoning procedures. Since materialization activities are mainly driven by past developers' experiences, we propose a Case-based Reasoning (CBR) approach that, through the codification of design experiences, permits to establish links between software architecture structures and object-oriented counterparts. This approach is supported by an Eclipse-based tool, called SAME (Software Architecture Materialization Environment), which is a reuse-oriented assistant to the developer. SAME is able to recall and adapt successful architecture materializations for particular quality attributes, in order to help the developer to derive an appropriate object-oriented design for his/her architecture.

© 2010 Elsevier Inc. All rights reserved.

1. Introduction

The design of software systems is a very complex activity as it is strongly influenced by the quality required in the final products. Consequently, software designers are compelled to make the right design decisions in early development stages, in order to fulfill quality-attribute requirements such as: modifiability, performance, or security, among others [9]. Over the last years, the software community has paid great attention to software architectures as a key abstraction in the software design process. A software architecture enables the description of the high-level organization of a system independently from implementation issues, so as to capture key design decisions and allow stakeholders to reason about quality attributes [12].

Being abstract design models, software architectures happen to be usually realized by means of object-oriented designs. Given an architecture and a set of quality-attribute drivers as input, there is a variety of object-oriented designs that can be derived from that input. We refer to this mapping between architecture and object-oriented designs to as *object-oriented materialization of software architectures* [13]. Ideally, the quality attributes prescribed at the architectural level must be

* Corresponding author at: ISISTAN Research Institute, Facultad de Cs. Exactas, UNCPBA Campus Universitario, Paraje Arroyo Seco, Tandil 7000, Argentina.
E-mail addresses: gvazquez@exa.unicen.edu.ar (G. Vazquez), adiaz@sei.cmu.edu (J. Andres Diaz Pace), mcampo@exa.unicen.edu.ar (M. Campo).

0020-0255/$ - see front matter © 2010 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.ins.2010.03.013

preserved at the object-oriented level in the final design. That is, if the architecture prescribes a layered design as a mechanism for achieving modifiability, the object-oriented design should be implemented in such a way the modifiability concerns still hold. However, due to the differences in the abstractions used by architecture design and object-oriented design, mismatches between the architecture "as intended" and the architecture "as implemented" are common. In spite of the current body of design knowledge and technologies, we believe that research is still needed to clarify the relationships between the architecture and the object-oriented worlds. Furthermore, little progress has been made regarding tool support for materialization activities.

In practice, the quality of a software design depends on the developer's knowledge and experience. Knowledge in the form of design patterns is important here as they codify guidelines for deriving object-oriented designs [21]. However, a pattern-based approach still presents some problems. A comprehensive catalog of design patterns has not yet been developed. The common template for expressing patterns is not conceived to put the pattern's knowledge directly into a design tool. Besides, patterns do not explicitly link architectural knowledge to quality attributes, so the choice among candidate patterns for achieving a quality-attribute goal in an architecture materialization is not straightforward. Most organizations rely on "gurus" that know how to implement predefined architectures in object-oriented terms [31,8], and these people are able to adapt their object-oriented solutions to new situations. In this context, a tool approach able to integrate the experiences from both the pattern community and expert developers can be very valuable for managing both architectural and object-oriented knowledge. We envision a design assistant that, fed with appropriate design experiences, helps developers in the exploration of object-oriented design alternatives for a given architecture. In order to implement these ideas, we face two challenges. First, we need a way to codify developers' design experiences along with their quality drivers in a knowledge repository. Second, we need a process to operationalize that knowledge in order to make it applicable to architecture materializations with similar characteristics. Among the different AI reasoning mechanisms, the case-based reasoning (CBR) paradigm fits well with the memory-based process employed by expert developers, in which certain types of materialization problems tend to recur and in which similar problems have often similar solutions [19].

In this article, we describe a novel tool approach based on the CBR paradigm, in which materialization experiences carried out by developers to implement a previous architecture provide insights for the implementation of a new architecture with similar quality-attribute goals. The ideas behind this approach were initially outlined in a previous work [40]. In [40], we proposed the use of architectural connectors as the focal entities for structuring the materialization experiences. Thus, a materialization experience is a case that describes the design context around a connector (the problem), and also captures the object-oriented design devised to implement that context (the solution). The design context includes the quality-attribute goals that drive the materialization (e.g., performance, modifiability, etc.). In the present work, we elaborate on the algorithms used for retrieving, comparing and adapting connector-based experiences, and present a prototypical integration of these algorithms into a tool called SAME (**S**oftware **A**rchitecture **M**aterialization **E**xplorer). SAME is an Eclipse-based tool that implements the typical CBR process: retrieve, reuse, revise and retain [30], and it is able to propose alternative object-oriented designs to the user. The main contribution of this article is a knowledge-based automation perspective to bridge the gap between architecture and object-oriented design. We have also performed a case-study to test the quality of the object-oriented solutions explored by the tool, and we have compared the time spent by developers to derive materialization alternatives with and without the tool.

The rest of the paper is organized around 8 sections as follows. Section 2 introduces the concepts underlying the SAME approach using a blackboard design example. In Section 3, we focus on the representation of designers' experiences in terms of architectural connectors. Sections 4 and 5 discuss the algorithms used to compare and adapt connector-based experiences respectively. Section 6 describes the SAME tool that we developed to support the approach. Section 7 discusses preliminary experimental results and lessons learned. Section 8 analyses related work. Finally, Section 9 comments on lines of future research and rounds up the conclusions of the paper.

2. Architectural connectors as reusable elements

Existing approaches to the problem of architecture materialization have been either focused on solutions given by architectural styles [32,15] or based on object-oriented frameworks applicable to specific domains [17,25] (see Related Work). Unfortunately, architectures are often heterogeneous and it is difficult to consider a particular style as the predominant structure for them [1]. This suggests that the materialization process must consider the elements that compose the architecture rather than the architecture as a whole. In several software development projects [13], we have observed that an architecture materialization is a combination of fine-grained design elements that separately reify the components and connectors [14] of the base architecture. An architectural component is a computational entity with runtime presence (e.g., process, client, server, data repository), while an architectural connector is a communication path among components (e.g., information flow, access to shared data, procedure call). Moreover, we have found that the same set of object-oriented elements used to reify specific connectors occur recurrently across different domains. The concrete implementations of a connector may differ (e.g., the specific communication protocols in a socket, or the particular method being invoked in a remote procedure call) but the generic object-oriented structure that makes the connector work remains constant across system implementations. These observations led us to revisit our materialization approach and move its focus from architectural styles to architectural connectors [40].

Download English Version:

https://daneshyari.com/en/article/6858509

Download Persian Version:

https://daneshyari.com/article/6858509

Daneshyari.com