The Journal of Systems and Software 82 (2009) 319–332

Contents lists available at ScienceDirect

The Journal of Systems and Software

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

Quality attribute tradeoff through adaptive architectures at runtime

Jie Yang [a,c], Gang Huang [a,b,*], Wenhui Zhu [a,b], Xiaofeng Cui [a,b], Hong Mei [a,b]

[a] School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China
[b] Key Laboratory of High Confidence Software Technologies, Ministry of Education, Beijing 100871, China
[c] IBM China Research Laboratory, Beijing 100193, P.R.China

ARTICLE INFO

Article history:
Received 29 April 2007
Received in revised form 22 June 2008
Accepted 22 June 2008
Available online 8 July 2008

Keywords:
Quality attribute tradeoff
Software architecture
Quality attribute scenario
Adaptive architecture
Reflective middleware

ABSTRACT

Quality attributes, e.g. performance and reliability, become more and more important for the development of software systems. One of the critical issues on quality assurance is how to make good enough tradeoffs between quality attributes that interfere with each other. Some architecture-based quality design and analysis methods are proposed to make tradeoffs at design time. However, many quality attributes depend on runtime contexts; it may be difficult and even impossible to make tradeoffs between them at design time. In this paper, we use an adaptive architecture model to capture candidate strategies for different quality attributes; the tradeoff, i.e. which strategies are more appropriate and thus applied, is postponed to runtime. The contribution of our approach is threefold. First, it makes use of existing architecture-based quality design and analysis methods to identify why and where quality attribute tradeoffs are necessary. Second, a traditional architecture description language is extended to support the modeling of an adaptive architecture, which records strategies for different quality attributes under different conditions. Third, a reflective middleware is used to monitor the runtime system, collect required information to determine appropriate strategies, and adapt the application's architecture to achieve expected quality attributes. This approach is demonstrated on J2EE.

1. Introduction

Software systems should satisfy not only functional requirements, but also quality requirements. By quality, we mean "the degree to which software possesses a desired combination of attributes (e.g., reliability, interoperability)" (IEEE-1061, 1992). Generally, quality requirements can be categorized into development quality requirements and operational quality requirements (Bosch and Mollin, 1999). Typical development quality requirements include configurability, flexibility, maintainability, and demonstrability. Typical operational quality requirements include performance, reliability, safety, and security (Bosch, 2000). In recent decades, software architecture (SA) has emerged as an appropriate level for dealing with software qualities (Dobrica and Niemela, 2002). Some architecture-based design methods emphasize the satisfaction of quality requirements at the design stage of software development. For example, the research of attribute-based architecture style (ABAS) associates an architecture style with an attribute-reasoning framework (Klein and Kazman, 1999). Typical styles are collected and thus form the foundation of an architect's handbook to deal with quality requirements. The attribute driven design (ADD) is an approach that defines an SA model based on a decomposition process driven by desired quality attributes (Bass et al., 2003). A quality attribute is a non-functional characteristic of a component or a system (Dobrica and Niemela, 2002). In ADD, quality attributes are considered as drivers in the design process that yields the system's conceptual software architecture. The output architecture satisfies not only the functional requirements, but also the important qualities the system must possess. Secondary qualities are satisfied within the constraints of achieving the most important ones. Bosch et al. propose a design method that employs an iterative evaluation and transformation of SA models in order to satisfy desired quality requirements (Bosch and Mollin, 1999).

Being one of the most important artifacts in software development, an SA model not only sets the boundary for software qualities of the resulting system (Dobrica and Niemela, 2002), but also embodies tradeoffs made by designers (Kazman et al., 1999). Efforts to explore such tradeoffs can be seen on scenario-based SA analysis methods such as scenario-based architecture analysis method (SAAM), SAAM founded on complex scenarios (SAAMCS), extending SAAM by integration in the domain (ESSAMI), architecture tradeoff analysis method (ATAM), scenario-based architecture reengineering (SBAR), and so on. These methods analyze the architecture with respect to multiple quality attributes to explore inherent tradeoffs concerning software qualities in the design. The outputs of such analysis include potential risks of the architecture and the verification result of the satisfaction of quality requirements.

When the above methods are applied to analyze such operational quality attributes as performance and availability that

* Corresponding author.
E-mail address: huanggang@sei.pku.edu.cn (G. Huang).

0164-1212/$ - see front matter © 2008 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2008.06.039

involve runtime information, architects often estimate quality attributes through simulating, mathematical modeling or experience-based reasoning and thus make tradeoffs between them (Bosch and Mollin, 1999). This would be helpful in calling attention to potential quality-related problems with the evaluated architecture; however, these methods do not guarantee the quality attributes of the target system at runtime, because in practice, the ever-changing runtime environment is too complex to predict or estimate; the estimated quality attributes may not accord with real situations.

A possible solution to the above problem is to obtain runtime information as accurate as possible by profiling before the implementation of the target software. Then trade-offs involving runtime information could be made. This approach may be feasible when the runtime environment is stable enough to simulate. In practice, many software systems would be distributed into the Internet or some enterprise-wide Intranet. It is rather difficult to simulate the target environment. Therefore, runtime information obtained through profiling are usually approximate only. Tradeoffs based on such information may not achieve desired qualities during the real execution.

Another possible solution is to postpone runtime-related decisions or tradeoffs until runtime, because the most accurate runtime information could only be obtained during the execution. There are many middleware researches dealing with quality goals of runtime software, such as real-time systems (Schmidt, 2002), multimedia systems (Aurrecoechea et al., 1998), and adaptive middleware (Agha, 2002). The solutions in these fields generally remain at the runtime stage; they are either application specific or quality attribute specific. However, they prove the possibility and effectiveness of changing software systems at runtime and thus allow architects to design with more flexibility and practicability. That means when designing, architects can predefine a set of configurations, each of which can satisfy desired quality attributes in different predictable situations. When a predicted situation occurs, the underlying middleware can automatically select and apply the right predefined configuration.

Based on current researches on SA and middleware, we propose a novel approach to the tradeoff between quality attributes regarding runtime information. In our approach, candidate strategies for the tradeoff are designed and recorded in the SA model (and thus make it an adaptive SA model); the decision of which strategy should be applied is postponed until runtime, when there is enough runtime information to determine the most appropriate strategy. The SA for the target system is therefore adapted during runtime to satisfy desired quality attributes. Our approach consists of three phases. Firstly, the SA for the target system is analyzed to locate the potential tradeoff points for various quality attributes; secondly, corresponding solutions (i.e. adaptive strategies) are designed and recorded in the SA model; finally, the target system is deployed and executed on some adaptive middleware, which is responsible for collecting runtime information, selecting a right configuration and adapting the runtime system.

The rest of this paper is organized as follows. Section 2 presents the overview of our approach. Section 3 briefly introduces Java Pet Store, a real case that is used throughout this paper to demonstrate the feasibility of this approach. Sections 4–6 detail the three phases of our approach; they are scenario-based quality attribute tradeoff process, adaptive architecture modeling and runtime validation, respectively. Section 7 presents some discussions and related work. Section 8 concludes this paper with a brief summary and future work.

2. Approach overview

2.1. Quality attribute tradeoff process

Traditional quality attribute (QA) tradeoff process in architecture-based development consists of four successive steps (Fig. 1a). Firstly, the expected QAs of the target system are estimated by mathematical modeling, simulating or experience-based reasoning to determine the satisfaction of individual QAs. Secondly, the architecture is analyzed to identify conflicts between different QAs. The fact that every decision to improve some QAs may impact others shows that the conflict is usually caused by satisfaction of certain QAs (In and Flores-Mendoza, 1999). Thirdly, architects make a tradeoff solution for every conflict identified. Conflicts may be resolved by introducing some architectural tactics or design patterns that usually satisfy QAs with higher priorities and sacrifice less important ones (Rajeshwari and Sarkar, 2005; Bachmann et al., 2003). Lastly, the solution is applied with actual actions as reconstructing the architecture with proper architecture approaches.

Different architecture-based development methods may manifest the process in different ways; however, these methods share a common characteristic, that is, after the tradeoff solutions are

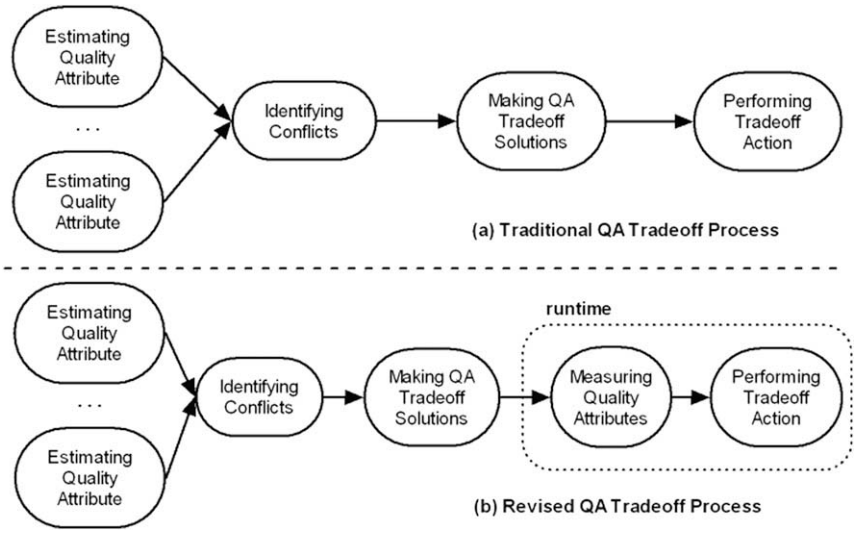

Fig. 1. Quality attribute tradeoff process.