Science of Computer Programming 98 (2015) 439–463

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Managing the evolution of a software architecture at minimal cost under performance and reliability constraints

Vittorio Cortellessa [a], Raffaela Mirandola [b,*], Pasqualina Potena [c]

[a] DISIM, Univ. dell'Aquila, Coppito (AQ), Italy
[b] Dip. di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Milano, Italy
[c] Computer Science Department, University of Alcalá, Alcalá de Henares, Madrid, Spain

HIGHLIGHTS

- We introduce an optimization-based approach for choosing evolution actions.
- It provides support for the decisions that architects make after deployment.
- It suggests the "best" actions to be taken according to a set of new requirements.
- It minimizes the evolution cost under reliability and performance constraints.

ARTICLE INFO

Article history:
Received 9 February 2012
Received in revised form 29 May 2014
Accepted 3 June 2014
Available online 18 June 2014

Keywords:
Software evolution
Software cost
Software reliability
Software performance
Optimization model

ABSTRACT

Managing a software architecture after the deployment phase is a very complex task due to frequent changes in the software requirements and environment. The software architecture must evolve in order to tackle such changes. The goal of this paper is to provide support for the decisions that software architects make after deployment. This approach is based on an optimization model whose solution suggests the "best" actions to be taken according to a given change scenario (i.e., a set of new requirements that induce changes in the structural and behavioral aspects of the software architecture). The model aims to minimize the costs while keeping the reliability and the performance of the software architecture within certain thresholds. The approach has been implemented as a framework named SHEPhERd, which is composed of a UML case tool, a model builder and a model solver. We show how SHEPhERd works on a smartphone mobile application example, and we highlight its potential to drive architectural decisions through sensitivity analysis. The achieved results are compared with those obtained by two groups of (human) maintainers composed of experts and non-experts with respect to the system and the execution environment, and we show that SHEPhERd outperforms the human judgment-based approaches.

1. Introduction

In the last few years, development processes have primarily focused on the maintenance phase, due to the frequent changes required by software after the deployment phase. Thus, software maintenance is a phase dedicated to more than just the removal of possible remaining bugs and/or the release of new software versions; it also needs to consider new requirements, and it has to help the system evolve in a way that satisfies such requirements. For example, in business

* Corresponding author.
E-mail addresses: vittorio.cortellessa@univaq.it (V. Cortellessa), raffaela.mirandola@polimi.it (R. Mirandola), p.potena@uah.es (P. Potena).

http://dx.doi.org/10.1016/j.scico.2014.06.001
0167-6423/

management systems for cellphone companies, new billing procedures associated with new marketing offers have to be implemented every week. Hence, the evolution represents most of the after-deployment activities.

Software evolution clearly involves the functionalities that it provides, but it also has a considerable impact on its non-functional characteristics. Evolution actions should not compromise the overall software quality, and this is hard to control without modeling support. Therefore, the software architecture represents a powerful instrument to drive software evolution, as changes should be driven by quality assessment based on the software architecture as a whole. Consequently, the architecture should evolve to represent the currently deployed product.

The goal of this paper is to assist software architects in the decisions they take after software deployment. Such assistance aims to take into account several quality attributes of the architecture, i.e., cost (such as that of software components' adaptation/integration, or the cost to introduce new components), reliability and performance. This approach is automated through the SHEPhERd (Software arcHitecture Evolution based on cost, PErformance and Reliability) framework. SHEPhERd is based on an optimization model that suggests the "best" actions to be taken upon a certain *change scenario* arising. A change scenario is a set of new requirements that induce changes in the structural and behavioral aspects of the software architecture. In particular, in our model, for each new requirement in a change scenario we consider different sets of evolution actions (called *evolution plans*) that are able to guarantee these new requirements. We aim to obtain a set of decisions that lead to the definition of a new architecture that minimizes cost, while keeping the reliability and the response time within certain thresholds. To keep the modeling aspects of our approach as simple as possible, we assume that the evolution plans are able to address each requirement independently (i.e., changes brought by a plan do not compromise the application of other plans).

A new software architecture is obtained by modifying its structure and behavior. To modify the structure, our approach suggests replacing existing components with different available instances and/or to introduce new components into the system.[1] With regard to the system behavior, our approach works on the system scenarios (expressed, for example, as UML Sequence Diagrams) by removing or introducing interaction(s) between existing or new components. The platform architecture (modeled, for example, with an UML deployment diagram) can also be modified by re-deploying existing components and/or deploying new components.

Non-functional attributes are expressed both as functions of the software component attributes (e.g., resource demands) and platform architecture features (e.g., the probability of failure of the links connecting components or the processing capacity of hardware devices). As will be shown in this paper, the compositional aspect of our model allows considering different types of evolution actions, and it takes into account their consequences on the overall architecture quality.

This paper extends our previous work [1], where an optimization model suggests how to change the software structure and behavior at a minimum cost while guaranteeing a certain level of software reliability. The new contributions that we present in this paper can be summarized as follows: (i) the proposed reliability model is enhanced by taking into account the reliability of links connecting components in addition to the reliability of software components (i.e., some platform features are also modeled); (ii) besides the composition of software components, the model solution suggests the best allocation of components to platform nodes; (iii) performance constraints on response times are added to the model, allowing study of the trade-offs between performance and reliability attributes.

This paper confirms the relevance of automated support in non-functional validation of software by showing that SHEPhERd produces better results than the ones obtained by human expert/non-expert judgments alone. The SHEPhERd framework is composed of an UML case tool, a model builder and a model solver. It allows for specifying ranges for model parameters, and, consequently, sets of alternative optimization models can be automatically generated (by sampling parameters in the given ranges) and solved. The output of SHEPhERd, for each model, is a set of evolution actions.[2] It suggests how to adapt both the static and dynamic aspects of the software architecture. Moreover, the platform architecture can be modified by re-deploying existing components and/or deploying new components on the existing nodes.

In our previous work [2], we have provided basic premises of a framework, based on an optimization model, that dynamically adapts a service based system (i.e., both the structural and behavioral software and platform architecture) while minimizing the adaptation costs and guaranteeing required levels of the system qualities. Adaptation actions can be triggered by a user request, by the runtime violation of system quality constraints, or by the appearance/disappearance of services into the environment.[3] In this paper, we instantiate the optimization model of [2], where a general formulation is provided, by considering the system evolution cost, reliability and response time. At the same time, we also take into account the deployment of components on the platform nodes.

The paper is organized as follows: in Section 2 we present an overview of the SHEPhERd framework; in Section 3 we provide the formulation of the optimization model that represents the core of our approach; in Section 4 we apply the model to an example; in Section 5 the achieved results are compared with those obtained by two groups of (human) maintainers composed of experts/non-experts with respect to the system and execution environment; Section 6 introduces

[1] Notice that we assume that all components are still used after the application of the evolution plans. However, in Section 3.2 we discuss how to relax our assumption.

[2] The tool underlying the SHEPhERd framework is under maintenance to create a user-friendly interface that enables easy input even from users that are not the framework authors.

[3] As far as the service-oriented domain is concerned, an instantiation of the model in [2] can be found in our previous work [3], where we have only considered software aspects.