A Practical Architecture-Centric Analysis Process

Antonio Bucchiarone[1,3], Henry Muccini[2], and Patrizio Pelliccione[2]

[1] Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo" (ISTI-CNR)
Area della Ricerca CNR di Pisa, 56100 Pisa, Italy
antonio.bucchiarone@isti.cnr.it
[2] University of L'Aquila, Computer Science Department
Via Vetoio 1, 67010 L'Aquila, Italy
{muccini, pellicci}@di.univaq.it
[3] IMT Graduate School
Via San Micheletto, 3 - 55100 Lucca , Italy

Abstract. When engineering complex and distributed software and hardware systems (increasingly used in many sectors, such as manufacturing, aerospace, transportation, communication, energy and health-care), dependability has became a must, since failures can have economics consequences and can also endanger human life.

Software Architectures (SA) can help improving the overall system dependability, providing a system blueprint that can be validated and that can guide all phases of the system development. Even if much work has been done on this direction, three important topics require major investigation: how different analysis techniques can be integrated together, how results obtained with SA-based analysis can be related to requirements and coding, and how to integrate new methodologies in the industrial software development life-cycle.

In this paper we propose an architecture-centric analysis process which allows formal analysis driven by model-based architectural specifications. This analysis process satisfies the industrial requirements, since it is tool supported and based on semi-formal (UML-based) specifications.

1 Introduction

One of the most used definitions for software architecture is the following: *"The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them"* [1]. Researchers in industry and academia have integrated the Software Architecture (SA) [2] description in their software development processes (e.g. [3,4]). However, putting SA in practice, software architects have learned that the SA production and management is, in general, an expensive task. Therefore the effort is justified if the SA artifacts are extensively used for multiple purposes helping on assuring the desired levels of dependability. Typical use of SA is as a high level design blueprint of the system to be used during the system development and later on for maintenance and reuse. At the same time, SA can be used in itself in order to *analyze and validate architectural choices*, both behavioral and quantitative. Thus, the problem of assuring as early as possible the correctness of a

C. Hofmeister et al. (Eds.): QoSA 2006, LNCS 4214, pp. 127–144, 2006.

software system, occupies an ever increasing portion of the development cycle cost and time budgets.

Analysis techniques have been introduced to understand if the SA satisfies certain expected properties, and tools and architectural languages have been proposed in order to make specification and analysis rigorous and to help software architects in their work (e.g., [2]). Even if much work has been done on this direction, the application of such techniques into industrial systems can be still very difficult due to some extra requirements and constraints imposed by industrial needs [5]: first of all, we cannot assume that a *formal* modeling of the software system exists. What we may reasonably assume, instead, is a semi-formal, easy to learn, specification language. Moreover, the approach should be *time* reducing and *tool supported* (automated tool support is fundamental for strongly reducing analysis costs).

The main *goals* that industries try to catch by modeling and analyzing architectural specifications are:

g1: to produce highly-dependable systems, while limiting analysis costs and
g2: to create a suitable process, where architecture-level decisions may be aligned to requirements and propagated down to the deployed system.

Regarding *g1*, the SA community has observed, in a few years, a proliferation of architecture description languages (ADLs) [6] for rigorous and formal SA modeling and analysis and the introduction of supporting tools. Although several studies have shown the suitability of ADLs, industries still tend to prefer model-based (semi-formal) notations. In particular, since UML is the de-facto standard for modeling software systems and it is widespread adopted in industrial contexts, many extensions and profiles have been proposed to "adapt" UML to model software architectures (e.g., [7]).

Regarding *g2*, in the last few years there has been a lot of interest in understanding how SA may be integrated with the other phases of the software life cycle [3,4], and how results gained at a certain level may be propagated to lower levels [8]. However, an automated, tool supported process which permits to consistently move towards such development phases is missing. As a result, it is still very difficult to propagate results from one phase to another.

Software *model-checking* and *testing* are some of the most used techniques to analyze software systems and identify hidden faults. Unfortunately, both approaches suffer some limitations: while current testing techniques are typically of difficult automation, model-checking requires an adequate developers expertise and skills on formal methods. In summary, industries are not encouraged to use the above mentioned techniques: industrial requirements are not met by either selective or exhaustive analysis.

In this paper we propose an *architecture-centric design and validation process* which enables the validation of a software system during the entire life-cycle. The analysis process is architecture-centric, since it validates the architecture specifications with respect to requirements, and then uses the validated SA as the starting point for any other analysis. It combines model-checking and testing techniques: we apply a model-checking technique to higher-level (architectural) specifications, thus governing the state-explosion problem, while testing techniques are used to validate the implementation conformance to the SA model. We check and test the architecture and the system implementation with respect to architectural (functional) properties. This