Chapter 2.2
Architecture-Centered Integrated Verification

Yujian Fu
Alabama A & M University, USA

Zhijang Dong
Middle Tennessee State University, USA

Xudong He
Florida International University, USA

ABSTRACT

This chapter presents an architecture-centered verification approach to large scale complex software systems by integrating model checking with runtime verification. A software architecture design provides a high-level abstraction of system topology, functionality, and/or behavior, which provides a basis for system understanding and analysis as well as a foundation for subsequent detailed design and implementation. Therefore, software architecture plays a critical role in the software development process. Reasoning and analysis of software architecture model can detect errors in an early stage, further reduce the errors in the final product and highly improve the software quality. First identified are the two main streams of software architecture research groups–the groups that work on the architectural abstraction and semantic foundation, and the group works on the framework using object oriented concepts. Problematically, both architecture designs cannot generate correct products due to two reasons. On one hand, not all properties can be verified at design level because of the state space explosion problem, verification costs, and characteristics of open-system. On the other hand, a correct and valid software architecture design does not ensure a correct implementation due to the error-prone characteristics of the software development process.

DOI: 10.4018/978-1-61350-456-7.ch2.2

The approach aims at solving the above problems by including the analysis and verification of two different levels of software development process–design level and implementation level-and bridging the gap between software architecture analysis and verification and the software product. In the architecture design level, to make sure the design correctness and attack the large scale of complex systems, the compositional verification is used by dividing and verifying each component individually and synthesizing them based on the driving theory. Then for those properties that cannot be verified on the design level, the design model is translated to implementation and runtime verification technique is adapted to the program. This approach can highly reduce the work on the design verification and avoid the state-explosion problem using model checking. Moreover, this approach can ensure both design and implementation correctness, and can further provide a high confident final software product. This approach is based on Software Architecture Model (SAM) that was proposed by Florida International University in 1999. SAM is a formal specification and built on the pair of component-connector with two formalisms – Petri nets and temporal logic. The ACV approach places strong demands on an organization to articulate those quality attributes of primary importance. It also requires a selection of benchmark combination points with which to verify integrated properties. The purpose of the ACV is not to commend particular architectures, but to provide a method for verification and analysis of large scale software systems in architecture level. The future research works fall in two directions. In the compositional verification of SAM model, it is possible that there is circular waiting of certain data among different component and connectors. This problem was not discussed in the current work. The translation of SAM to implementation is based on the restricted Petri nets due to the undecidable issue of high level Petri nets. In the runtime analysis of implementation, extraction of the execution trace of the program is still needed to get a white box view, and further analysis of execution can provide more information of the product correctness.

INTRODUCTION

A software architecture (SA) design provides a high-level abstraction of system topology, functionality, and/or behavior ((Shaw, M. and Garlan, D., 1996), (Perry, D. E. and Wolf, A. L.,1992), (Taylor, R. N. et al., 2009)), which provides a basis for system understanding and analysis as well as a foundation for subsequent detailed design and implementation. Therefore, software architecture plays a critical role in the software development process. In the past decade, tremendous research ((Luckham, D., et al., 1995), (Taylor, R. N., et al., 1996), (Roshandel, R., et al.,2004),(Medvidovic, N., et al., 1996,2002,200,2006),(He, X., et al.,2002,2004),(Fu, Y., et al.,2007)) has been done on software description languages and their analysis.

There are two main research groups in the field of software architectures: one group has focused on the architectural abstraction, and semantic analysis of architectures, while the other present a framework adopting object oriented reuse concepts for software architectures. The first group has focused on architectural design abstractions called styles and the semantics underpinning (Shaw, M. and Garlan, D., 1996). Various formal architecture description languages (ADLs) ((Luckham, D., et al.,1995),(Allen, R. J., 1997), (Taylor, R. N., et al., 1996),(Lu L., et al., 2002), (Vestal, S., 1998)) and their supporting tools ((Medvidovic, N., et al., 1996), (Vestal, S., 1998) have emerged from this body of research over the decades (N. Medvidovic & R.N. Taylor, 2000). To date, most architectural tools have focused on the simulation and analysis of architectural models to exploit the semantic power of ADLs. However, the analysis