A comparative study of formal verification techniques for software architecture specifications *

Jeffrey J.P. Tsai and Kuang Xu

Department of Electrical Engineering and Computer Science, University of Illinois at Chicago, 851 South Morgan Street, Chicago, IL 60607, USA
E-mail: tsai@eecs.uic.edu

With the rapid growth of network computing, the demand for large-scale and complex software systems has increased dramatically. However, the development of large-scale and complex software systems is much more difficult and error prone. This is due to the fact that techniques and tools for assuring the correctness and reliability of software systems lag far behind the increasing growth in size and complexity of software systems. The concept of software architecture has recently emerged as a new way to improve our ability to effectively construct and maintain large-scale complex software systems. The architecture based development of software systems focuses on the architectural elements and their overall interconnection structure. Several Architectural Definition Languages (ADLs) have been proposed for specifying domain specific or general purpose architectures. On the other hand, formal verification is rapidly becoming a promising and automated method to ensure the accuracy and correctness of software systems. In this paper, we survey several architecture description languages and formal verification methods. We present an environment to conduct experiments to study the performance of five different verification tools on software architecture specifications. Based on these experiments, we are able to compare the efficiency of these verification tools in verifying certain software property.

1. Introduction

With the rapid growth of network computing, the demand for large-scale and complex software systems has increased dramatically. However, the development of large-scale and complex software systems is much more difficult and error prone. This is due to the fact that techniques and tools for assuring the correctness and reliability of software systems lag far behind the increasing growth in size and complexity of software systems [Tsai and Weigert 1994; Tsai and Yang 1995; Tsai *et al.* 1996]. The results are unreliable and poorly performing applications, delayed projects (often for years), and considerable cost overruns. The developed software should posses important properties of reliability: e.g., correctness, robustness, performance, and security. To achieve this, new techniques and development tools need to be created.

* This research is supported in part by NSF and DARPA under Grant CCR-9633536.

The concept of software architecture has recently emerged as a new way to improve our ability to effectively construct and maintain large-scale complex software systems. The architecture based development of software systems focuses on the architectural elements and their overall interconnection structure. Several Architectural Definition Languages (ADLs) have been proposed for specifying domain specific or general purpose architectures. Some of these languages use a formal model to define their semantics. For instance, *Wright* [Allen and Garlan 1994] is an ADL which represents interface points as ports, whose interaction semantics is specified in CSP. *UniCon* [Shaw *et al.* 1995] has a predefined, enumerated set of types, the component semantics is represented as event traces. *Darwin* [Magee and Kramer 1996] uses π*-calculus* to model the component interaction and composition properties. *Rapide* [Luckham and Vera 1995] is an event-based ADL used for defining and simulating behaviors of system architectures.

ADLs are typically used for specifications at higher levels of software life cycle. At lower levels other notations/representations may be used. As software evolves from early architectural designs to final code a variety of notations may be used. There may be more than one representation for the same stage in the development life cycle. Sometimes, some notations are better suited for specifying requirements, others are easy to model and code in programming languages. This is particularly true in a heterogeneous distributed environment. All these notations/representations are usually based on certain standard formalisms/models such as process algebras, temporal logic, Petri Nets, automata, label transition systems, real-time logic and so on.

On the other hand, formal verification, based on static analysis, allows us to precisely specify the correctness of the system and analyze it systematically and exhaustively. Formal verification is rapidly becoming a promising and automated method to ensure the accuracy and correctness of software systems at an early stage during software development process. Studies have repeatedly shown that most of the cost of software development and maintenance stems from design (or requirements) defects. Formal verification enables us to identify those defects at early stages, such as the requirement and design phase of the software life-cycle. Such detection and correction of bugs in early stages, significantly reduces the cost of debugging, maintenance, and re-development.

Several verification tools have also been developed for system analysis based on different formal models. Such tools include *Symbolic Model Checker (SMV)* [Clarke *et al.* 1996], *SPIN* [Holzmann], *Symmetric-based Model Checker (SMC)* [Sistla *et al.* 1997], *Input–Output Traces Analyzer (IOTA)* [Juan *et al.* 1998], etc. Thus comes the problem of how to apply these existing tools to verify the properties of software architectures described by various ADL specifications.

In this paper, we survey several architecture description languages and formal verification methods. We present an environment to conduct experiments to study the performance of five different verification tools that are based on different techniques, such as *partial-order reduction*, *symbolic model checking*, *symmetric-based model checking*, *bit-hash*, and *compositional verification*. Based on these experiments,