Applying Static Analysis to Software Architectures

Gleb Naumovich, George S. Avrunin, Lori A. Clarke and Leon J. Osterweil

email: {naumovic|avrunin|clarke|jo}@cs.umass.edu
Laboratory for Advanced Software Engineering Research
Computer Science Department
University of Massachusetts
Amherst, Massachusetts 01003

Abstract. In this paper we demonstrate how static concurrency analysis techniques can be used to verify application-specific properties of an architecture description. Specifically, we use two concurrency analysis tools, INCA, a flow equation based tool, and FLAVERS, a data flow analysis based tool, to detect errors or prove properties of a WRIGHT architecture description of the gas station problem. Although both these tools are research prototypes, they illustrate the potential of static analysis for verifying that architecture descriptions adhere to important properties, for detecting problems early in the lifecycle, and for helping developers understand the changes that need to be made to satisfy the properties being analyzed.

1 Introduction

With the advent of improved network technology, distributed systems are becoming increasingly common. Such systems are more difficult to reason about than sequential systems because of their inherent nondeterminism. In recognition of this, software architecture research is attempting to define architecture description languages to help developers describe distributed system designs. These high-level descriptions allow developers to focus on structural, high-level design issues before lower level details are addressed, thereby helping to discover areas of high risk and to address these risks as early in the lifecycle as possible. To be truly beneficial, developers should be given tools to help them reason about their architecture descriptions, to help them discover problems as early as possible, and to help them verify that desired properties would indeed be maintained by these designs as well as by any systems correctly derived from these designs. It has been demonstrated that detecting errors early in the lifecycle [3] greatly reduces the cost of fixing those errors. Architecture description languages combined with appropriate analysis tools could therefore be an important means for reducing costs and improving reliability.

A number of architecture description languages have been developed, such as WRIGHT [2], Rapide [13], Darwin [14, 15], and UniCon [20]. There has also been some work on validating aspects of architecture designs. Using architectures specified in UniCon, for instance, developers can estimate local timing information and use those estimates to check time-dependent properties with the RMA real-time analysis tool [12]. Another approach is to use model-theoretic proof techniques to verify conformance of
elaborated architecture descriptions to higher-level architecture designs [14, 18]. Developers using the Rapide architecture description language can simulate executions of the system and verify that the traces of those executions conform to high-level specifications of the desired behavior [13]. Although one would expect the number of traces through an architecture description to be much less than the number of possible executions in the corresponding software system, for most interesting systems there are still far too many such traces to explore them all. Thus, this is basically a sampling technique, and while it increases confidence in the architecture, it does not verify that all executions conform to the specifications. Another validation approach that has been explored is the use of static analysis techniques to verify general properties of architecture descriptions. When successful, this type of analysis does verify that all possible executions conform to the specification. Allen and Garlan [1] use the static analysis tool FDR [7] to prove freedom from deadlock as well as compatibility between the components and connectors in an architecture description. These are general properties that are desirable for all architecture descriptions.

The primary goal of this work is to investigate the applicability of existing static analysis techniques for verifying application-specific properties of architectures. We investigate one example architecture, a WRIGHT description of the gas station problem, and illustrate the kinds of properties that can be verified and the kinds of errors that can be found early in the lifecycle. Two versions of a WRIGHT architecture specification of the gas station example were graciously provided to us by David Garlan. We applied two static analysis tools: INCA, which is based on flow equations, and FLAVERS, which is based on data flow analysis. Both of these tools are research prototypes that illustrate the potential for static analysis to verify that architecture descriptions adhere to important properties, to detect problems early in the lifecycle, and to help developers understand the changes that need to be made to satisfy the properties being analyzed.

The next section gives a high-level overview of the two static analysis tools used in this case study. Section 3 gives a brief description of the gas station problem and the WRIGHT specification of the problem. Section 4 introduces the properties we selected to prove about this architecture and describes the analysis process and the results of that process. Section 5 summarizes the overall results, describes the benefits of this approach, and points out some interesting directions for future research.

2 Tools Used

A number of automated static concurrency analysis techniques have been proposed. They span such approaches as reachability analysis (e.g. [11, 21, 8]), symbolic model checking [4, 17], flow equations [5], and data flow analysis [6, 16]. The goal of this work is to demonstrate the applicability of static analysis techniques to architecture descriptions but not, at least at this point in time, to determine which approach might be best. Thus, we selected two different static analysis tools, based on fundamentally different approaches, with which we have considerable expertise. One tool, INCA [5], is based on flow equations, and the other, FLAVERS [6], is based on data flow analysis. Both these tools can be used to check whether all executions of a concurrent system satisfy a property, such as the mutually exclusive use of some resource. Although these