Propel: Tools and Methods for Practical Source Code Model Checking[1]

Abstract for DSN-2003 Workshop on Model-Checking for Dependable Software-Intensive Systems

Massoud Mansouri-Samani
Peter Mehlitz
Computer Sciences Corporation

Lawrence Markosian
Owen O'Malley
QSS Group, Inc.

Dale Martin
Lantz Moore
Clifton Labs, Inc.

John Penix
NASA

Willem Visser
RIACS

Computational Sciences Division
NASA Ames Research Center

The work reported here is an overview and snapshot of a project to develop practical model checking tools for in-the-loop verification of NASA's mission-critical, multithreaded programs in Java and C++. Our strategy is to develop and evaluate both a *design concept* that enables the application of model checking technology to C++ and Java, and a model checking toolset for C++ and Java. The design concept and the associated model checking toolset is called *Propel*. It builds upon the Java PathFinder[1] (JPF) tool, an explict state model checker for Java applications developed by the Automated Software Engineering group at NASA Ames Research Center.

The *design concept* that we are developing is Design for Verification (D4V). This is an adaption of existing "best design practices" that has the desired side-effect of enhancing verifiability by improving modularity and decreasing "accidental" complexity. D4V, we believe, enhances the applicability of a variety of V&V approaches; we are developing the concept in the context of model checking.

The *model checking toolset,* Propel, is based on extending JPF to handle C++. Our principal tasks in developing the toolset are to build a translator from C++ to Java, productize JPF, and evaluate the toolset in the context of D4V. Through all these tasks we are testing Propel capabilities on customer applications.

Development of the "Design for Verification" (D4V) concept. Real applications typically exceed the capabilities of today's V&V tools in terms of size and complexity. As a result, these applications often need to be manually translated into tool-specific models—an expensive process that comes with a potential for introducing fidelity problems. This effectively renders V&V a one-time effort inconsistent with the evolutionary nature of large-scale system development. As part of Propel, we are creating a methodology that explicitly adds V&V goals to the design phase. Our purpose is to turn V&V into a development co-process by mapping key system properties to dedicated, separately checkable design components. This is achieved mainly by using domain-specific design pattern systems, where each pattern instance has its own set of tool-supported usage checks and guarantees. These, in turn, are used to identify suitable pattern candidates based on the system specification. The whole application design is centered around three concepts: *Extension Points* (base classes with overridable methods, delegation objects/types), *Conceptual Branch Points* (potentially blocking operations), and *Check Points* (consistent states that require verification in terms of reachability and evaluation). The methodology can be seen as an adaption of existing "best design practices", with the desired side-effect of improving modularity and decreasing "accidental" complexity.

The initial target defect classes for Propel are deadlocks and race conditions, with additional properties including temporal logic properties to be identified from Design for Verification concept development.

Translation of C++ to Java. To leverage the existing JPF model checker, we are building an automated translator from C++ to Java. The much simpler and better-defined semantics of Java lend themselves to be model-checked more easily and correctly. We considered defining a new virtual machine to support C++, rather than using Java VM in JPF, but by using the standard JVMs we can use the large collection of tools

1 The research and development reported in this paper is funded by the Engineering for Complex Systems program at NASA Ames Research Center.

that support Java manipulation and verification. Naturally, the tradeoff is that some parts of C++, such as multiple inheritance and the loose type model, are difficult to represent in Java. Therefore, the translated system is more complex than the original system. Part of the ongoing research is focused on understanding and minimizing the impact of this translation overhead on verification.

The translator is implemented with an Edison Design Group-based C++ parser, which we extended to create an XML-based abstract syntax tree (AST). Various tree-based transformations to replace the non-Java-like parts of C++ with Java equivalents are then applied to the AST and, finally, Java code is published from the AST.

Productization of JPF. To evaluate the capability of model-checking technology on real applications, the tools themselves must be robust and scalable. Therefore, we are putting significant effort into refactoring JPF for faithfulness to real JVMs; maintainability, extensibility and testability; and implementing performance enhancements to reduce the size of the state space, reduce the size of the state representation, and speed up model checking. As an example, we have implemented a mechanism to abstract components of a target system so that they can be partly executed in the underlying JVM instead of entirely in the JPF JVM. This is especially important for implementing system library abstractions that try to minimize the relevant state space of the verified system. It also supports extensibility of JPF by greatly simplifying the process of adding support for system libraries that are required for real applications.

We are also enhancing usability and are planning to integrate JPF with development environments. We have identified and are addressing engineering risks and their mitigations, and we have identified research issues as well. Examples of engineering or research issues include: rendering the JPF output in a way that supports traceability and debugging; model checking applications that have large or complex environments that cannot be modeled adequately in JPF; and handling concurrency libraries that are too low-level to model in JPF.

In addition to the tasks described above, we need to address technical marketing questions such as how model checking is best integrated with other V&V approaches, and how significantly the limitations on our approach (or more generally on model checking) affect the set of suitable applications of interest to NASA. Thus an important activity for us is characterizing the relevant attributes of suitable applications (those that increase risk or enhance applicability of our tools), obtaining NASA mission applications (both C++ and Java), analyzing these applications, and testing our tools with them. We are in the process of developing collaborations with NASA missions to provide early feedback using the productized JPF on their Java applications and to use their Java and C++ applications to focus our product development.

We are also interested in exploring the question of return on investment of applying model checking, but we have encountered difficulty finding data to support a baseline ROI estimate.

[1] W. Visser, K. Havelund, G. Brat, S. Park. Model Checking Programs. *Proceedings of the 15th International Conference on Automated Software Engineering (ASE)*, Grenoble, France, September 2000.