Model-driven Development with Predictable Quality

James Ivers Gabriel A. Moreno

Software Engineering Institute, Carnegie Mellon University
Pittsburgh, PA, USA
{jivers,gmoreno}@sei.cmu.edu

Abstract

The PACC Starter Kit is an eclipse-based development environment that combines a model-driven development approach with reasoning frameworks that apply performance, safety, and security analyses. These analyses predict runtime behavior based on specifications of component behavior and are accompanied by some measure of confidence.

Categories and Subject Descriptors D.2.4 [*Software Engineering*]: Software/Program Verification—Model checking, Correctness proofs; C.4 [*Performance of Systems*]: Modeling techniques

General Terms Design, Performance, Reliability, Security, Verification

Keywords Predictable assembly, components, performance, model checking, model-driven development

1. Introduction

Many software systems have stringent quality attribute requirements. For example, industrial robots must perform tasks with strict deadlines, medical devices must comply with safety requirements, and most software must minimize security vulnerabilities. Although analysis theories and techniques addressing these requirements have existed for many years, they are not widely used because of the resources and expertise required to create, maintain, and evaluate analysis models. Consequently, developers usually rely on testing to verify the satisfaction of these requirements, incurring expensive overruns when they are not met.

The PACC[1] Starter Kit (PSK), shown in Figure 1, is an eclipse-based development environment that combines a model-driven development approach with reasoning frameworks (RFs) that package the expertise needed to apply quality attribute analyses. A reasoning framework is used to achieve some level of confidence (statistical or proof-based) in predictions of runtime behavior based on specifications of component behavior. RFs ensure that designs satisfy analytic assumptions and then automatically generate analysis models appropriate for reasoning about specific runtime behaviors.

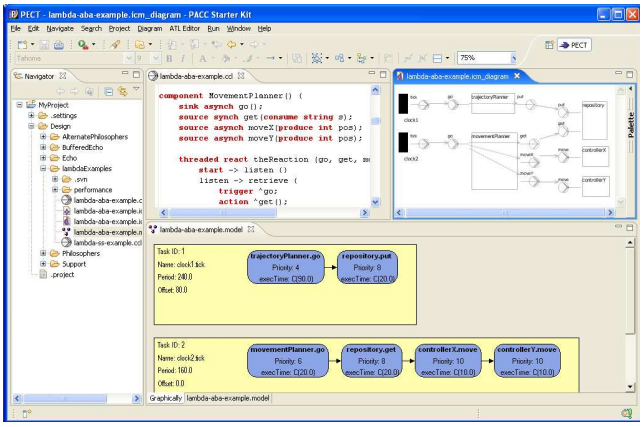

Figure 1. PACC Starter Kit

The PSK demonstration focuses on two RFs. The performance RF predicts latency in soft and hard real-time systems. The model checking RF analyzes runtime behavior for safety properties, such as detecting buffer overflows, and can generate proofs that binary code satisfies such properties. The demonstration also touches on the included code generator and runtime environment.

2. Basic PSK Features

The PSK is designed for use in model-driven development environments. As such, it includes tools supporting many fundamental model-driven development tasks, the most important of which are

- A design language: CCL is a design language for specifying the behavior of components and the manner in which they are assembled to form systems [8]. CCL includes

[1] The PSK was developed by the Predictable Assembly from Certifiable Components (PACC) group at the Carnegie Mellon Software Engineering Institute (http://www.sei.cmu.edu/pacc).

OOPSLA'07, October 21–25, 2007, Montréal, Québec, Canada.
ACM 978-1-59593-865-7/07/0010.

syntax for component wiring and a behavioral notation based on UML statecharts and a C-like action language.

- An execution environment: CCL is specialized for use with components built for the Pin component technology [4]. Pin offers typical features, such as standard component lifecycles and interaction mechanisms. Pin sits on top of RTOS, a simple real-time operating system extension for Windows.
- A code generator: Complete component and assembly implementations can be generated from CCL specifications for deployment in the Pin runtime environment. Code generation is an important element of ensuring that analysis results based on CCL specifications are applicable to code executing in Pin.

3. Performance Reasoning Framework

The performance reasoning framework in the PSK can predict average and worst-case response times of a component-based application. The performance analysis, which is completely automated, involves two steps. The first step, called interpretation, transforms the specification of the assembly, creating a performance model. The second step evaluates the performance model using a suitable evaluation procedure.

In component-based designs, the response to some event is usually realized by an assembly of components that interact with each other either synchronously or asynchronously. This results in non-linear sequences of interactions that include complex interactions between components and threads of execution. The interpretation exploits the knowledge of priority assignments, thread allocation, and interaction semantics, and computes an equivalent model with linear sequences of interactions that can then be analyzed by using real-time analysis theories [7] or simulation.

The RF supports several evaluation procedures to evaluate the performance model. Three different discrete-event simulators can be used to predict average response time. Worst-case prediction can be done using RMA for varying priorities [3]. Also, a closed-formula evaluation procedure predicts average response time of a sporadic server task [5].

4. Model Checking Reasoning Framework

The ComFoRT reasoning framework [6] uses software model checking to determine whether a system satisfies desired safety or security policies.[2] ComFoRT uses component behavior specifications written in CCL to generate equivalent C programs for use with the Copper software model checker. These programs are functionally equivalent to the C programs that are generated for execution in the Pin runtime environment, but are optimized for verification. The generated C program (model) is then searched to determine if all possible executions through the program (regardless of event arrival order or concurrency interleavings) satisfy the desired policies, which are expressed in a state/event linear temporal logic (SE-LTL) [1].

When a policy is found to be violated, a counterexample is generated as evidence. Counterexamples are sequences of actions through the model that leads to a state in which a policy does not hold and are shown in the PSK's Counterexample view.

When a policy is found to hold, a certified component can be generated that includes a proof certificate as evidence [2]. The certified component is generated for execution in the Pin runtime environment, and the embedded proof certificate is automatically generated using certifying model checking and proof-carrying code techniques.

[2] A version of the PSK that includes ComFoRT can be downloaded from http://www.sei.cmu.edu/pacc/comfort.html

Acknowledgments

The PACC Starter Kit is a result of the contributions of the whole PACC team: Sagar Chaki, Arie Gurfinkel, Jeff Hansen, Scott Hissam, Mark Klein, Paulo Merson, Linda Northrop, Dan Plakosh, and Kurt Wallnau. We are also indebted to all those developing the tools that we've built on.

References

[1] S. Chaki, E. M. Clarke, J. Ouaknine, N. Sharygina, and N. Sinha. State/Event-Based Software Model Checking. volume 2999, pages 128–147.

[2] S. Chaki, J. Ivers, P. Lee, K. Wallnau, and N. Zeilberger. Certified binaries for software components. Technical Report CMU/SEI-2007-TR-001, Software Engineering Institute, Pittsburgh, PA, 2007.

[3] M. Gonzalez Harbour, M. Klein, and J. Lehoczky. Timing analysis for fixed-priority scheduling of hard real-time systems. *IEEE Trans. Softw. Eng.*, 20(1):13–28, 1994.

[4] S. Hissam, J. Ivers, D. Plakosh, and K. C. Wallnau. Pin component technology (V1.0) and its C interface. Technical Note CMU/SEI-2005-TN-001, Software Engineering Institute, Pittsburgh, PA, 2005.

[5] S. Hissam, M. Klein, J. Lehoczky, P. Merson, G. Moreno, and K. Wallnau. Performance property theories for predictable assembly from certifiable components (PACC). Technical Report CMU/SEI-2004-TR-017, Software Engineering Institute, Pittsburgh, PA, September 2004.

[6] J. Ivers and N. Sharygina. Overview of ComFoRT: A model checking reasoning framework. Technical Note CMU/SEI-2004-TN-018, Software Engineering Institute, Pittsburgh, PA, 2004.

[7] M. H. Klein, T. Ralya, B. Pollak, R. Obenza, and M. Gonzalez Harbour. *A practitioner's handbook for real-time analysis*. Kluwer Academic Publishers, Norwell, MA, USA, 1993.

[8] K. Wallnau and J. Ivers. Snapshot of CCL: A language for predictable assembly. Technical Note CMU/SEI-2003-TN-025, Software Engineering Institute, Pittsburgh, PA, 2003.