A Hybrid Framework for Resource Verification in Executable Model-based Embedded System Development

Honguk Woo, Aloysius K. Mok, James C. Browne
Department of Computer Sciences,
The University of Texas at Austin
{honguk, mok, browne}@cs.utexas.edu

Abstract

In this work, we consider the integration of resource safety verification into a design methodology for development of verified and robust real-time embedded systems. Resource-related concerns are not closely linked with current xUML model-based software development although they are critical for embedded systems. Our work employs the hybrid verification framework combining static resource analysis and run-time monitoring schemes.

1 Introduction

Model-based development has focused on verification and testing for functional or timing aspects, yet embedded software also involves para-functional resource-related aspects, termed *resource (bound) properties*. These properties enforce resource limits on CPU time, memory, battery power, network bandwidth, etc. Unlike functional properties, the verification and testing for resource bound properties has not been completely addressed in executable model-based development. Specifically resource-related language constructs are not incorporated in the action semantics of executable models since early design is intended to be platform independent. Accordingly resource bound properties have not been entirely linked with functional verification from the beginning of the development cycle. This limitation often renders the process of resource safety verification (the verification of resource bound properties) non-systemic or ad hoc at best leading to excessive cost for monitoring and analysis. Furthermore, resource safety verification is usually deferred to testing during/after the implementation phase. Resource safety violations detected during implementation testing commonly require redesign and reimplementation of the system. Recently we proposed the software engineering discipline incorporating resource safety verification into a design and development methodology for embedded systems in [8]. This work focuses on how to integrate the verification of para-functional resource properties into the software development cycle of executable model-based approaches. As briefly shown in Figure 1, the proposed methodology integrates:

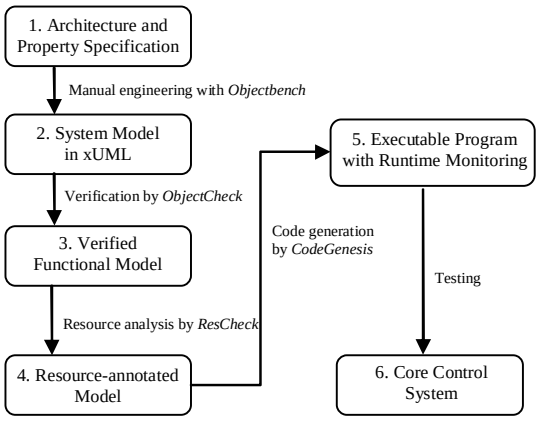

Figure 1. Development Cycle

- xUML (eXecutable Unified Modeling Language [2]) modeling and simulation-based model testing processes supported by the commercial software modeling and simulation environment Objectbench [5]
- Automatic code generation from xUML models by CodeGenesis code generator [4]
- Formal functional verification by model checking for xUML models [7]
- Resource bound checking based on efficient dynamic monitoring [3, 1]

ObjectCheck [7] is used to validate the xUML model with respect to selected functional properties while the resource verifier for embedded systems, named ResCheck [8], deals with resource properties to provide the comprehensive autonomous support for resource safety verification.

Verification for functional or resource properties is done by combining model checking, resource analysis and runtime monitoring. Using xUML model checking for functional properties has been well studied by in [7], which is briefly introduced in the following. In ObjectCheck, designers of the system use the Property Specification Interface and xUML Visual Modeler to specify the properties of

the system and xUML model. An xUML-to-S/R translator converts them to S/R query and S/R model respectively. The COSPAN model checker accepts these inputs and checks whether the query is valid in the model. In case that the verification fails, COSPAN model checker generates an error track and then Error Report Generator produces an error report in xUML from the error track. To help the debugging process, Error Visualizer creates a test case from the error report and reproduces the error by running the xUML model with the test case.

The complementary problem, *efficient run-time monitoring* has been recently introduced in [8] where resource properties are treated as the primary concern on verification. Notice that due to their inherently dynamic nature, resource properties are not usually addressed in the static-verification-only context. It is also important to note that resource properties can be translated in the form of *continuous constraint queries* in [6].

Currently our work primarily aims at incorporating the verification process for resource properties into the early development cycle, thereby lowering development cost and enhancing the quality of resource critical embedded software. Resource safety verification requires additional resource-related specification in the model as input to further analysis at the design phase. It is worthwhile to note, however, the developer's labor can be reduced by exploiting the executable semantics of xUML and the autonomous tool support for resource analysis and monitoring code insertion. The executable semantics of xUML enables the model specification to be tested in the simulation environment and to be automatically translated into an executable program [2, 7], and furthermore it allows resource-related operations to be specified as part of state actions in the state model.

Our approach employs a hybrid framework where static analysis and monitoring techniques cooperatively work to provide resource safety verification. In the framework, the run-time monitoring makes explicit use of static analysis results to cope with the possible performance overhead of traditional run-time monitoring mechanism. To do so, the static analysis in the framework first translates a given executable model containing resource-related code into a tree-based resource evaluation structure. The resource evaluation structure may involve statically-unbounded variables e.g., loop bounds that can only be dynamically determined. This often renders the verification inherently incomplete at analysis time and necessitates run-time monitoring support. For monitoring efficiency, the static resource analysis simplifies run-time operations by having in-lined monitoring code. The run-time monitoring relies on static analysis and thus monitors discrete updates of a small set of specific variables in the program execution, instead of directly tracking and managing dynamic resource usage information. Since in practice testing alone cannot completely guarantee system correctness, it is natural that a product system employs run-time monitoring support at the execution environment as part of exception handling. This would in turn incur inordinate performance overhead to the execution environment unless the monitoring algorithm is carefully designed. The hybrid framework – specifically lightweight monitoring based on static analysis results – addresses this problem of run-time overhead. More importantly, it can cover a wide variety of property types in software safety requirements.

The critical functionalities identified in a system design can be verified by model checking and/or completely tested in the development cycle above, and then the corresponding software component can be treated as core segment. The hybrid, lightweight monitoring technique is also naturally consistent with desirable extension to the implementation and integration of non-core segment where the static verification may be inherently limited. The monitoring technique for providing the safe interplay between core and non-core segments of a control-oriented embedded system is one of our future directions. Out-stream of unreliable components and dynamic environmental information can be relatively unreliable to some extent, thus necessitating probabilistic approach for handling those information streams.

This research is supported in part by the National Science Foundation under Grant Number 0613665 and the authors have been working with Jianliang Yi, Fei Xie, and Ella Atkins.

References

[1] C. Ajay, E. David, and I. Nayeem. Enforcing Resource Bounds via Static Verification of Dynamic Checks. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 29(5), January 2007.

[2] S. J. Mellor and M. J. Balcer. *Executable UML: A Foundation for Model-Driven Architecture*. Addison-Wesley, 2002.

[3] A. K. Mok and W. Yu. TINMAN: A Resource Bound Security Checking System for Mobile Code. In *Proc. of the European Symposium on Research in Computer Security(ESORICS)*, pages 178–193, Octobor 2002.

[4] SES. *Code Genesis Manual*. 1996.

[5] SES. *Objectbench User Reference Manual*. 1996.

[6] H. Woo and A. K. Mok. Real-time Monitoring of Uncertain Data Streams using Probabilistic Similarity. In *Proc. of IEEE Real-Time Systems Symposium(RTSS)*, December 2007.

[7] F. Xie, V. Levin, and J. C. Browne. ObjectCheck: A Model Checking Tool for Executable Object-Oriented Software System Designs. In *Proc. of Fundamental Approaches to Software Engineering(FASE)*, April 2002.

[8] J. Yi, H. Woo, J. C. Browne, A. K. Mok, F. Xie, E. Arkins, and C.-G. Lee. Incorporating Resource Safety Verification to Executable Model-based Development for Embedded Systems. Technical report, University of Texas at Austin, 2007.