Towards a Model-based Toolchain for the High-Confidence Design of Embedded Systems

János Sztipanovits, Gábor Karsai, Sandeep Neema, Harmon Nine,
Joseph Porter, Ryan Thibodeaux, and Péter Völgyesi
Institute for Software Integrated Systems
Vanderbilt University
Nashville, TN 37235, USA
janos.sztipanovits@vanderbilt.edu

Abstract

While design automation for hardware systems is quite advanced, this is not the case for practical embedded systems. The current state-of-the-art is to use a software modeling environment and integrated development environment for code development and debugging, but these rarely include the sort of automatic synthesis and verification capabilities available in the VLSI domain. This paper introduces concepts, elements, and some early prototypes for an envisioned suite of tools for the development of embedded software that integrates verification steps into the overall process.

1. Introduction

Embedded software often operates in environments critical to human life and subject to our direct expectations. We assume that a handheld MP3 player will perform reliably, or that the unseen aircraft control system aboard our flight will function safely and correctly. Embedded environments require far more care than provided by the current best practices in software development. Often formal verification and system certification are required to insure correct behavior and conformance to legal standards. Embedded systems design challenges are well-documented [4], but industrial practice still falls short of these expectations.

Consider one style of modern development practice: graphical modeling and simulation tools (e.g. Mathworks' Simulink/Stateflow or National Instruments' Matrix-X) represent physical systems and engineering designs using block diagram notations for dataflows or state models. Design work revolves around simulation and test cases, with code generation following once the design is considered complete. Such methods frequently ignore software engineering constraints on the design and neglect issues that arise from embedded platform choices. At early stages of the design, often the platform is vaguely specified to the engineers as a set of possible tradeoffs, with incomplete details regarding actual platform function and performance.

Similarly, another development style uses UML (or similar) tools to capture software engineering concepts such as components, interactions, timing, fault handling, and deployment. These workflows focus on source code creation and management followed by testing and debugging on target hardware. In this case the physical and environmental constraints are not represented by the tools. At best such constraints may be provided informally as notes or documentation to developers and may remain poorly understood.

The interplay between these two prevalent development styles creates problems. Designers lack tools to model the interactions between the hardware, software, and the environment. For example, software generated from a carefully simulated functional dataflow model may fail to perform correctly when its functions are distributed over a shared network of processing nodes. Neither style of development supports comprehensive verification of certification requirements. To move towards a solution to these problems, we propose a suite of tools that address many of these challenges. Currently under development at Vanderbilt's Institute for Software Integrated Systems (ISIS), these tools use domain-specific modeling languages (DSMLs) to integrate the disparate aspects of an embedded systems design.

The tool suite described here is built on the concept of platform-based design [8], and is shown conceptually in Figure 1. Componentization and higher-level services enable the designer to build correct systems from validated components. Additionally, if the DSMLs used in tool integration have formally defined behavioral semantics and well-defined models of computation (MoCs) for component interactions [7], system properties and models can be

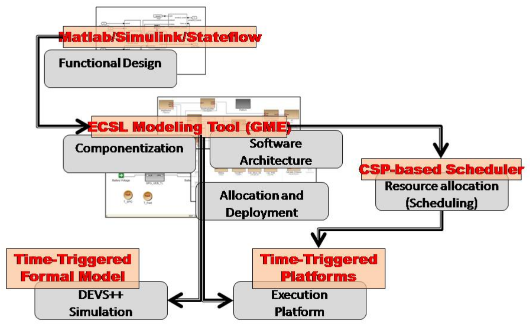

Figure 1. Existing elements of the tool suite.

expressed formally and verified with appropriate external tools. In the sequel we briefly describe the current state of the tool suite and conclude with a discussion of the direction of our future goals.

2. Elements of the Tool Suite

The domain of choice for this research is that of distributed and embedded control systems. Accordingly, the formal MoC chosen is that of the Time-Triggered Architecture (TTA) [6]. Time-triggered systems provide a number of essential guarantees for safety-critical control systems designs. In particular, the TTA provides precise timing for periodic tasks, distributed fault-tolerance, and replica determinism in redundant configurations. These basic guarantees and their implementations constitute some of the important high-level component services needed for our platform-based designs.

2.1. Software architecture specification

Simulink/Stateflow (SL/SF) models can be imported into a well-defined modeling format that allows for analysis, extension, and code generation. Graphical modeling tools can read these models and perform software engineering design tasks. The SL/SF models are embedded in software components with well-defined interfaces, and then mapped to well-defined distributed hardware models.

2.2. Code generation

Model transfomations [3] can convert imported SL/SF models into a model representing an abstract syntax tree (AST) for C code fragments. Interpreters for the new AST model can create code or directly perform simple static analyses such as checking variable initializations. Generated C code is generic – the tools currently support execution on a hardware implementation of the TTA (hardware available from TTTech[2]) or on a time-triggered virtual machine (VM) running on Linux (described below).

2.3. Scheduling

Resource allocation in the TTA is controlled by a pre-generated cyclic schedule created from task specifications and their communication dependencies. We have created a simple schedule generation tool that uses the Gecode finite-domain constraint programming library to search for cyclic schedules that meet the specifications. Constraint models are an extension of earlier work in this area [9].

2.4. Modeling the execution platform

The chosen time-triggered model of computation has been formalized using the DEVS formalism (Figure 2) and simulated using the DEVS++ simulator [5]. Simulation results for a time-triggered triple modular redundancy experiment were consistent with observed performance of a time-triggered implementation [10].

2.5. Implementation of the execution platform

In addition to tests on available time-triggered hardware, we have developed a portable time-triggered VM running on a networked cluster of processors running standard Linux. The portability of the VM allows the direct exploration of the capabilities and limitations of the services provided by the underlying operating system, and the effects of those limitations on the guarantees provided by the chosen MoC [10].

3. Future work

As this research effort is a work in progress, we conclude with a brief summary of the next steps and future objectives for each of the tools presented. We must keep in mind the final goal of verifiable and certifiable software for embedded systems. This section contains forward-looking statements.

3.1. Software architecture specification

The chief limitation of our software architecture tools is the one-way design flow from the SL/SF design, through componentization, down to the final code. We aim to improve the ability to send design information back to the earlier stages of the design as neeeded. For example, platform-specific simulations may indicate that jitter or quantization

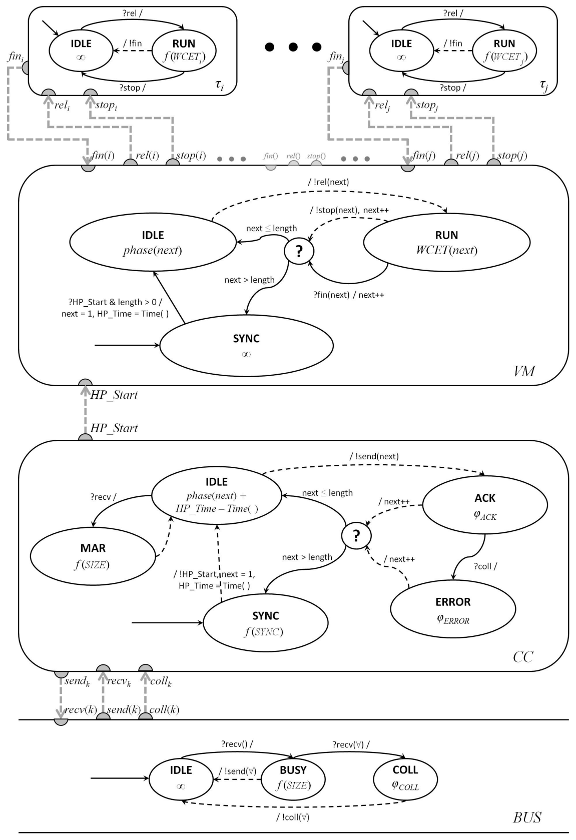
Figure 2. DEVS models for time-triggered virtual machine

effects will impact the initial assumptions of a control design. Representing that data to control designers in a meaningful way will allow design changes without excessive workflow iterations. Schedulability is another area where downstream software design tools can provide meaningful feedback to the original design engineers.

3.2. Code generation

The abstract model in the code generator opens the door for a number of potential static analysis and verification opportunities. The current toolchain includes two code generators that produce C (and Java) source code from (single-rate) subsystems in Simulink and Stateflow models. The code generators have been implemented using graph transformation techniques, and they produce an AST from which the actual code is printed. To assist in system-level or functional code verification the AST could be extended to carry over information from the original model, thus providing guidance for the source code-level verification tool regarding the original model from which the code was generated and its properties. We believe this can significantly improve the performance of the verification step because the verifier does not have to reverse engineer the high-level abstractions from the source code, as the abstractions are readily available in the models.

3.3. Scheduling

We aim to expand the scheduling tools to include specific time-triggered models. One simple example is that of adding constraints to support the requirements of the TT-Tech TTP/C hardware. Another avenue for research is the exploration of interactions between the resource allocation model (via schedules) with other system objectives which can be modeled by constraint or optimization problems in other domains (such as continuous stability in the control design).

3.4. Extending the modeling of the execution platform

The formal DEVS model is a big step towards providing guaranteed safety and performance in time-triggered control system designs. DEVS also supports pure event-triggered behaviors in addtion to timed models. Experimentation with this capability will hopefully lead to a better understanding of the limitations of heterogeneous component interactions in our system designs.

Platform simulation also opens up opportunities for exploration. The TrueTime tool suite from Lund University [1] extends Simulink models with concepts for modeling distributed platforms, scheduling policies, and communication protocols. TrueTime promises to help characterize behavioral changes due to the distribution of functionality over networked processors.

3.5. Extending the capabilities of the execution platform

As the capabilities of the formal models expand, we aim to extend our portable VM implementation to manage heterogeneous behaviors. The VM will also be ported to other operating platforms, including diverse hardware and RTOSes such as QNX and uC-OS. Different platforms provide different levels of assurance regarding timing, determinism, and resource management. These differences will need to be reflected in the models. New features may also be added to the VM as required to support interaction idioms such as remote procedure calls or rendezvous. We may also require additional component services such as health monitoring, fault management, robust clock synchronization, or failover.

4. Acknowledgements

This work was sponsored (in part) by the Air Force Office of Scientific Research, USAF, under grant/contract number FA9550-06-0312. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Office of Scientific Research or the U.S. Government.

References

[1] Truetime: Simulation of networked and embedded control systems. http://www.control.lth.se/truetime/.

[2] TTTech TTP/C Cluster. http://www.tttech.com/.

[3] Aditya Agrawal, Gabor Karsai, Sandeep Neema, Feng Shi, Attila Vizhanyo. The design of a language for model transformations. *Journal on Software and System Modeling*, 5(3):261–288, Sep 2006.

[4] T. Henzinger and J. Sifakis. The embedded systems design challenge. In *FM: Formal Methods*, Lecture Notes in Computer Science 4085, pages 1–15. Springer, 2006.

[5] M. H. Hwang. *DEVS++: C++ Open Source Library of DEVS Formalism*. http://odevspp.sourceforge.net/, first edition, May 2007.

[6] H. Kopetz and G. Bauer. The time-triggered architecture. *Proceedings of the IEEE, Special Issue on Modeling and Design of Embedded Software*, Oct 2001.

[7] E. A. Lee and A. L. Sangiovanni-Vincentelli. A denotational framework for comparing models of computation. Technical Report UCB/ERL M97/11, EECS Department, University of California, Berkeley, 1997.

[8] Sangiovanni-Vincentelli, A. Defining Platform-based Design. *EEDesign of EETimes*, February 2002.

[9] K. Schild and J. Würtz. Scheduling of time-triggered real-time systems. *Constraints*, 5(4):335–357, Oct. 2000.

[10] R. Thibodeaux and G. Karsai. Model-based specification and implementation of a model of computation. In preparation for ECMDA 2008, February 2008.