EMBEDDED SW DESIGN SPACE EXPLORATION AND AUTOMATION USING UML-BASED TOOLS

Flávio R. Wagner and Luigi Carro
Computer Science Institute, Federal University of Rio Grande do Sul (UFRGS), Brazil
{flavio, carro}@inf.ufrgs.br

Abstract: This tutorial discusses design space exploration and software automation based on an UML front-end. First, we review software automation tools targeted at the embedded systems domain. Following, we present an approach for the estimation of memory, performance, and energy of a given application modeled from an initial UML specification. We proceed with an analysis of the possibilities of linking different modeling environments for software generation (Simulink and UML, for example). Finally, we show the possibilities of using other specification languages to obtain more abstraction and allow design space exploration together with software automation.

Key words: Embedded Software, Design Space Exploration, High-level modeling, UML, MDA, code generation

1. INTRODUCTION

The increasing complexity of embedded systems design, which is derived from the amount of functionality that is required from these systems, together with the shortening of the life cycle of embedded products, results in a design scenario where productivity and quality are simultaneously required in order to deliver competitive products.

Selic[1] emphasizes that the use of techniques starting from higher abstraction levels is crucial to the design success. The UML language has gained in popularity as a tool for specification and design of embedded systems. There are many efforts that describe the use of UML during the different phases of an embedded system design [2].

It is widely known that design decisions taken at higher abstraction levels can lead to substantially superior improvements. This context suggests the support to a fast design space exploration in the early design steps. However, software engineers, when developing an application using UML, do not have a concrete measure of the impact of their modeling decisions on issues such as performance and energy for a specific embedded platform.

On the other hand, the specification of typical embedded systems requires several models of computation, and in a single product (like a cell phone) many of these models can be simultaneously found, like event-driven, sequential code, and synchronous dataflow. This pushes the need for supporting software automation under different models of computation, a task not completely supported by any current software automation tool.

This tutorial discusses two critical aspects of embedded systems software automation, namely the design space exploration (DSE) from high-level UML models, and automation techniques to serve several models of computation. The tutorial is organized in the topics that follow.

1) Review of current software automation tools

Most of the academic and commercial solutions for software automation focus on the management of huge domain-specific systems, because conventional software is usually suited for a single domain. Some tools automate code generation for any application specified in a proper way, such as Unimod[3], which extracts code from UML diagrams and follows the Model Driven Architecture[4] approach. However, tools that automate general and conventional software development are not aware of code optimizations for memory or power, a crucial step for embedded systems because of their tight restrictions.

More recent approaches targeted to embedded software development[5] keep working on extensions of state machines, in an Esterel[6] fashion, and try to improve final code optimizations for control-dominated reactive systems. The Model-Integrated Computing (MIC) approach fully adopts the model-based development paradigm[7]. As the approach relies on domain-specific modeling languages[8], it is also not suited for applications that mix different models of computation.

The Ptolemy framework[9] provides simulation and prototyping for heterogeneous embedded systems. Although Ptolemy provides abstraction for many domains, it requires knowledge of several different languages, and this may increase development time. As a matter of fact, none of the presented approaches targets the ultimate goal of providing high abstraction to increase software production, with the necessary design space exploration to meet embedded systems' tight constraints.

2) UML modeling and high-level design space exploration

UML modeling solutions for the same application may result in very different physical costs, as demonstrated by Oliveira[10]. We propose the estimation of physical costs such as performance, energy, and memory footprint directly from UML models. We show how these estimations may be integrated into an automatic DSE process, which is guided by optimization heuristics (such as simulated annealing or ant colony optimization), following desired design goal and constraints.

3) Software synthesis using an MDA-based approach

In the context of platform-based design, software synthesis is mainly based on the reuse of a library of previously designed components. For a rapid DSE process, it is essential that a designer can quickly generate software for different mappings from a specification of the application into the platform. This can be accomplished by an MDA-based approach, where the application and the platform are specified according to appropriate meta-models and the mapping between them is implemented as a sequence of model transformations. We present a DSE tool that can automatically optimize the mapping of PIM (Platform Independent Model) into PSM (Platform Specific Model). From this mapping, and according to an implementation meta-model, the final software may be generated.

Our approach uses a meta-modeling infrastructure proposed by Nascimento[11], which is based on OMG standards, such as MOF and XMI. Different languages for modeling the application, the platform, and the final implementation may be used, given the existence of appropriate translators from these languages into instances of the generic meta-models. This approach, in fact, may be used also for the synthesis of dedicated hardware modules, in a HW-SW co-design methodology. We support UML and Simulink because recent efforts[2,12,13] show that both languages are considered attractive for System Level design.

4) Software modeling and synthesis for different models of computation

A comparison between UML and Simulink modeling approaches concluded that both have pros and cons[14]. UML provides all benefits from the object-oriented paradigm, like modularity, encapsulation, reusability, etc. In the other side, Simulink supports multiple models of computation and complete code generation. To address multiprocessor systems, a Simulink-based software synthesis approach was recently developed[15], which starts from a Combined Architecture Application Model (CAAM) described in Simulink and generates multithread code. It includes optimization techniques to generate memory-efficient[15] and communication-efficient code[16].

However, building the CAAM model using the Simulink GUI can be error-prone, and usually software engineers prefer to employ UML. Then, Brisolara[17] proposes mapping rules to translate a UML model into a Simulink CAAM, thus allowing the use of UML as a front-end for the Simulink multithread code generation flow. It allows the integration of both modeling languages in a unique design flow, which combines the benefits of UML and Simulink.

REFERENCES

[1] B.Selic. Models, Software Models and UML. UML for Real: Design of Embedded Real-Time Systems. Kluwer Academic Publishers, 2003. Chapter 1, p. 1-16.
[2] L.Lavagno, G.Martin, B.Selic. UML for Real: Design of Embedded Real-Time Systems. Kluwer Academic Publishers, 2003.
[3] Executable UML Unimod. v.1.3, 2003. http://unimod.sourceforge.net.
[4] OMG. Model Driven Architecture. White Paper v. 3.2, November 2000.
[5] F.Balarin et al. Synthesis of Software Programs for Embedded Control Applications. IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems, v. 18, n.6, June 1999.
[6] G.Berry et al. The Esterel Synchronous Programming Language: Design, Semantics, Implementation. Science of Computer Programming, v. 19, n. 2, 1992.
[7] J. Sztipanovits et al. Model-Integrated computing. IEEE Computer. April 1997.
[8] G.Karsai et al. Model-Integrated Development of Embedded Software. Proceedings of the IEEE, v. 91. n. 1, January 2003.
[9] J.Buck et al. Ptolemy: a framework for simulating and prototyping heterogeneous systems. International Journal in Computer Simulation, v. 4, 1992.
[10] M.Oliveira et al. Early Embedded Software Design Space Exploration Using UML-based Estimations. RSP'06, Chania, Greece. June 2006.
[11] F.A.Nascimento et al. ModES: Embedded Systems Design Methodology and Tools based on MDE. MOMPES 2007, Braga, Portugal, March 2007.
[12] SysML: Systems Modeling Language. http://www.omgsysml.org/. July 2006.
[13] R.Boldt. Combining the Power of MathWorks Simulink and Telelogic UML/SysML-based Rhapsody to Redefine the Model-Driven Development Experience. Telelogic White Paper, June 2006. http://www.ilogix.com/whitepaper-overview.aspx.
[14] L.Brisolara et al.. A Comparison between UML and Function Blocks for Heterogeneous SoC Design and ASIP Generation. In: G.Martin and W.Mueller (Ed.). UML for SoC Design. Chapter 9. Springer, 2005.
[15] K. Huang et al. Simulink-Based MPSoC Design Flow: Case Study of Motion-JPEG and H.264. DAC'07, San Diego, USA, June 2007.
[16] L.Brisolara et al. Reducing Fine-grain Communication Overhead in Multithread Code Generation for Heterogeneous MPSoC. SCOPES'07. Nice, April 2007.
[17] L.Brisolara et al. Using UML as a front-end for an efficient Simulink-based multithread code generation targeting MPSoCs. UML-SoC'07, San Diego, USA, June 2007.