UML for Embedded Systems Specification and Design: Motivation and Overview

Grant Martin
Cadence Design Systems
2001 Addison Street, Third Floor, Berkeley CA 94704, U.S.A.
gmartin@cadence.com

Abstract

The specification, design and implementation of embedded systems demands new approaches which go beyond traditional hardware-based notations such as HDLs. The growing dominance of software in embedded systems design requires a careful look at the latest methods for software specification and analysis. The development of the Unified Modeling Language (UML), and a number of extension proposals in the realtime domain holds promise for the development of new design flows which move beyond static and traditional partitions of hardware and software. However, UML as currently defined lacks several key capabilities. In this paper, we will survey the requirements for system-level design of embedded systems, and give an overview of the extensions required to UML that will be dealt with in more detail in the related papers. In particular, we will discuss how the notions of platform-based design intersect with a UML based development approach.

1. Introduction: The Nature of Embedded Systems

Modern embedded systems have certain characteristics that demand new approaches to their specification, design and implementation. These approaches must be a combination of traditional hardware and software methods, but must seek new ways of linking them together in order to allow designers to develop such products rapidly and with low risk.

These systems are composed of multiple subsystems or functional units that carry out computation and communication using a heterogeneous set of models of computation. These functional subsystems can be implemented using a variety of components, both hardware and software, composed together into flexible platform architectures. The mapping of function to architecture is not fixed: design space exploration allows developers to find optimal ways of implementing functions by analysis of a variety of alternatives in both hardware and software domains. This emphasises both early modelling of systems, and methods that delay commitment to particular components or implementations to the end of the design process.

1.1. Heterogeneous

As technology has developed, embedded systems have moved from single function products to ones incorporating multiple roles. The convergence of functionality from the multiple domains of computing, signal processing of audio and video, and wired and wireless communications have led to increasingly complex embedded systems which integrate subsystems reflecting a heterogeneous set of Models of Computation (MOCs). Modelling such systems requires a set of heterogeneous notations to reflect this: continuous time, finite-state-machine, dataflow, discrete event, reactive, etc. In addition to such heterogeneous logical domains, there are also multiple physical implementation domains in embedded systems – function is realised through combinations of dataflow and control-oriented software, hardware components such as microprocessors, DSPs, analogue and mixed-signal components, digital HW blocks, and RF, optical and MEMS components. These heterogeneous implementation, or ‘architectural’ choices, also need to be modelled.

1.2. Compositional

Today’s embedded systems are not usually modelled and implemented monolithically from scratch. Rather they are usually compositions of subsystems, each of which may be based on a different MOC. Each MOC, and modelling notation, is chosen to be ‘natural’ for the subystem domain. Both functional and architectural descriptions of the system may be based on composition of subsystems. With such a design strategy, the emphasis must be equally on the validity of the composition as on the correctness of the constituent parts.

1.3. Complexity, Driving Reuse and Synthesis

In line with the 'convergence' and 'compositional' attributes discussed earlier, modern embedded system products are too complex to be designed from scratch. The complexity of the embedded software for a 3G mobile handset exceeds that of a 2G phone by at least one order of magnitude. Complexity from composing heterogeneous functional requirements demands more design knowledge than many single product groups possess. All these factors point to a strong need to maximise software and indeed system development productivity through use of embedded system platforms and reuse and synthesis methods driven from system-level models [1].

1.4. System Context

As well as modelling the system from functional requirements through executable specifications, it is important to be able to model the context for an embedded system – both environmental (e.g. channel characteristics and noise scenarios for a wireless system) and user-driven (use-cases for multi-function embedded systems).

2. Design Methodology Requirements

In a related paper in this session [2], embedded and real-time systems design methodologies based on UML and SDL will be discussed. Suffice it here to summarise the basic design methodology requirements based on the characteristics described earlier. We need a methodology to support:

- Heterogeneous modelling of system function and architecture, including target implementation SW-HW platforms
- The system modelled within its use-context and environment.
- Mapping from function to architecture in order to support design space exploration, alternative implementations, and reusable components
- Strong analytical and verification techniques
- Strong linkages from models through to implementation in both SW and HW domains, including synthesis, refinement, decomposition and reuse approaches.

3. UML: Capabilities and Lacks

3.1. What UML has or will have

What makes UML a reasonable meta-language to model embedded/real-time systems, and solve these kinds of design problems? First, from the current UML (1.4) [3,4,5], we can identify several key attributes of UML important to embedded systems:

Heterogeneous set of notations

- A concept that the UML is not a single language, but a set of notations, syntax and semantics to allow the creation of families of languages for particular applications. In other words, UML is a 'meta-language'.
- Extension mechanisms via profiles, stereotypes, tags, and constraints for particular applications
- Use-case modelling to describe system environments, user scenarios, and test cases
- A large standards organisation, the Object Modelling Group [4], which is promoting UML among software designers and sponsoring its evolution via task forces, working groups and the like. OMG also promotes the development of application-domain-specific profiles.
- Support for object-oriented system specification, design and modelling, thus appealing to the software community.
- Growing interest in UML from the embedded systems and realtime community.
- Support for state-machine semantics which can be used for modelling and synthesis
- Support for object-based structural decomposition and refinement

However, UML is not static. It is evolving in many directions and in particular, there are a number of activities relevant to embedded and real-time systems:

- UML 2.0 evolution [2, 6, 7] which is looking at functional encapsulation methods, annotations to UML to allow modelling of real-time aspects such as schedulability, performance and time, and other important extensions such as dataflow.
- Plans to support SDL, used for many years in realtime and embedded software development, as a profile of UML 2.0 [8].
- Action semantics, driven by OMG and the Action Semantics Consortium, which OMG wishes to incorporate into UML 2.0, and that provide formal underpinnings for more complete executable specifications and synthesis.
- Model-driven architecture [4] that aims to separate the specification of software applications from the particular middleware 'platform' which implements them. This is similar to function-architecture codesign.

3.2. What UML Lacks and needs

Given existing UML and the evolutions promised in UML 2.0, what are the key lacks remaining to support embedded systems design [9]? These are:

- the platform model
- the mapping and refinement methodology to move from one platform level to another
- the constraint definition and budgeting methodology to complement the movement of design from requirements to implementation, and which provide a necessary control for optimisation processes in design transformation and synthesis.

4. "UML-Platform"

Some work has been done on fleshing out the requirements to model SoC platforms in UML, and to develop methodologies for embedded systems design using these concepts [10]. The proposal involves several extensions to UML and assumes that most of the current proposals – for real-time UML and 2.0 extensions – will be accepted and standardised. In this sense, the UML-Platform profile describes the following extensions to UML, as a set of stereotypes and tags:

- "uses" and "needs" stereotypes to specify realisation relationships between applications or platform components, and services offered by other platform components
- a "stack" stereotype to describe hierarchical, layered implementations of platform services
- a "peer" stereotype for components and service offerings at the same abstraction level
- "coupling" relationships to show necessary linkages between components
- tags to define Quality of Service (QoS) parameters for platform services, and application requirements. These can be derived from specifications, mapped into constraints, and used to select appropriate implementations.
- defined platform layers – Application Specific Programmable (ASP), Application Programming Interface (API) and Architectural (ARC) within which services can be classified for deployment
- a method for depicting platform "extension points" for future application requirements and new or variant service offerings.

The notion of "UML-Platform" thus offers a base for the mapping and refinement concepts which support the move from a target-independent system model to an optimised platform-based implementation. A variety of analysis capabilities can be built using the profile, which allow the notion of 'software-software' codesign, in which function is primarily mapped onto a variety of software-based implementations, to be supported. QoS tags allow constraint definition, decomposition into subsystem budgets and the effective choice of possible components that meet overall system requirements.

Further research is planned to apply the "UML Platform" profile to commercial SoC platform offerings, and to extend its concepts to support complete UML-based platform design methodologies for embedded systems.

5. Conclusions

The Unified Modelling Language is an interesting and reasonably compelling basis for embedded system design methodology. The current state of the language is not complete enough to build comprehensive tools, flows and methodologies, and the anticipated changes in 2.0, although vital, are still not complete enough to meet all needs. The UML-platform concept, complemented by further work on methodology, should provide much of the additional concepts required for a UML-based embedded and realtime software design flow.

6. References

1. Alberto Sangiovanni-Vincentelli and Grant Martin, "Platform-Based Design and Software Design Methodology for Embedded Systems", IEEE Design and Test of Computers, Volume 18, Number 6, November-December 2001, pp. 23-33.
2. Gjalt de Jong, "A UML-Based Design Methodology for Real-Time and Embedded Systems", DATE 2002, March 2002.
3. J. Rumbaugh, I. Jacobson, and G. Booch, The Unified Modeling Language Reference Manual, Addison-Wesley, 1998.
4. The Object Modeling Group, URL: http://www.omg.org/.
5. Thomas Weigert, "What Really is UML?", presentation, October 18, 1999.
6. Bran Selic, "The Real-Time UML Standard: Definition and Application", DATE 2002, March 2002.
7. Bran Selic, "A Generic Framework for Modeling Resources with UML", IEEE Computer, June 2000, p.64-69.
8. Morgan Björkander, "Graphical Programming Using UML and SDL", IEEE Computer, December 2000, pp. 30-35.
9. Grant Martin, Luciano Lavagno, and Jean Louis-Guerin, "Embedded UML: a merger of real-time UML and co-design", CODES 2001, Copenhagen, April 2001, pp.23-28.
10. Rong Chen, Marco Sgroi, et. al. "Embedded System Design Using UML and Platforms", Unpublished paper, U.C. Berkeley, September, 2001.