Chapter 1

UML as Front–End Language for Embedded Systems Design

Lisane Brisolara de Brisolara
Universidade Federal de Pelotas (UFPEL), Brazil

Márcio Eduardo Kreutz
Universidade de Santa Cruz do Sul (UNISC), Brazil

Luigi Carro
Universidade Federal do Rio Grande do Sul (UFRGS), Brazil

ABSTRACT

This chapter covers the use of UML as a modeling language for embedded systems design. It introduces the UML language, presenting the history of its definition, its main diagrams and characteristics. Using a case study, we show that using the standard UML with its limitations one is not able to model many important characteristics of embedded systems. For that reason, UML provides extension mechanisms that enable one to extend the language for a given domain, through the definition of profiles covering domain-specific applications. Several profiles have been proposed for the embedded systems domain, and some of those that have been standardized by OMG are presented here. A case study is also used to present MARTE, a new profile specifically proposed for the embedded system domain, enabling designers to model aspects like performance and schedulability. This chapter also presents a discussion about the effort to generate code from UML diagrams and analyses the open issues to the successful use of UML in the whole embedded system design flow.

INTRODUCTION

Embedded systems design has become an important research area due to the high complexity of the new generations of such systems. This increased complexity derives from the amount of functionality that is associated with those systems, which evolved from simple, dedicated applications (like motor control) to portable multimedia devices (like modern portable phones). Efforts in all areas of system level design, like specification, modeling, synthesis, simulation, and verification are required to cope with this increasing complexity.

At the same time, the life cycle of embedded products has become increasingly tighter. Current research on embedded systems design emphasizes
that the use of techniques starting from higher abstraction levels is crucial to the design success. Some authors like Selic (2003) and Gomaa (2000) argue that this approach is the only viable way to cope with the complexity that is found in the new generations of embedded systems. Using abstraction, models of embedded systems should evolve from high level views into actual implementations, ensuring a relatively smooth and potentially much more reliable process as compared to traditional forms of engineering.

The Unified Modeling Language (UML) (OMG, 1997) is a standard notation for modeling and documentation of object-oriented software. UML2 (OMG, 2007c), the current version of the language, supports thirteen different diagrams, which allows one to model system structure (e.g. class, deployment) and behavior (e.g. collaboration, sequence and timing diagrams). UML has gained popularity as a tool for specification and design of embedded systems, mainly due to the abstraction provided by this language (Lavagno, Martin & Selic, 2003; Martin & Mueller, 2005). The first investigation in this direction has been conducted by Grant Martin, Luciano Lavagno, and Jean Louis-Guerin in 2001 (Martin, Lavagno & Louis-Guerin, 2001). After that, several efforts that describe the use of UML during the different phases of an embedded system design process can be found in (Lavagno, Martin & Selic, 2003), including the investigation of the use of UML for modeling hardware by Edwards and Green (2003). Such popularization comes from the fact that UML is by far the most-used modeling notation for conventional computational systems. An advantage of UML is that it provides extension mechanisms that allow customization of the language, enabling the definition of profiles for specific domains. UML-SPT (OMG, 2002) and MARTE (OMG, 2007a) are examples of profiles proposed by OMG to model schedulability, performance, time and other implementation aspects that are paramount to the development of embedded products. Besides the language extension, UML also improves the clarity, readability and communicability amongst designers in the same domain.

This chapter introduces the UML language as a strong candidate for being the specification formalism for embedded system design, and discusses its usage in the embedded field, including limitations and the evolution of the application of this language in this domain.

The remaining parts of the chapter are divided in the following way. First, we present the history and evolution of the UML language, and the main supported diagrams in the Background section. An embedded system example is used to explain different diagrams and discuss limitations of the language, which have motivated extension proposals to support embedded systems design. The Modeling Embedded Systems with UML section addresses the use of UML for embedded systems modeling and presents MARTE through of a case study. The section Code Generation from UML diagrams discusses some weaknesses of UML that must be further tackled to allow complete code generation from an UML front-end. Future trends are discussed in the sequence followed by the main conclusions of this chapter.

BACKGROUND

Models provide a way to cope with complex systems by letting designers work at a higher abstraction level. A model can hide details and provide a visual representation for the system. Several modeling approaches have been proposed in the software engineering domain (Atlee et al., 2007). With the popularity of object-oriented approach, the community decided to define a default modeling language. In this context, UML was proposed, and since October 1994 has been under development. The intention behind the definition of the language was to consolidate the various OO languages, methods, and notations in a single modeling language, in a vendor independent fashion. UML was based on the OMT