Using UML as a front-end for an efficient Simulink-based multithread code generation targeting MPSoCs

Lisane B. Brisolara, Marcio F. S Oliveira, Francisco A. Nascimento, Luigi Carro, Flávio R. Wagner

Instituto de Informática, Universidade Federal do Rio Grande do Sul (UFRGS), Porto Alegre, Brazil
{lisane, mfsoliveira, fanascimento, carro, flavio}@inf.ufrgs.br

Abstract. UML provides all benefits from the object-oriented paradigm, such as encapsulation and reusability, while Simulink is targeted to support multiple models of computation such as stream processing and control. Recently, a Simulink-based design flow was proposed to address MPSoC systems, generating multithread code. However, MPSoC modeling with Simulink is error-prone and does not adequately cope with MPSoC modeling. In addition, software engineers usually prefer to employ UML, due to its higher abstraction level. This work proposes the mapping of system models from UML to Simulink, thus enabling a Simulink-based MPSoC design flow. In this way, the same UML model can be used as a front-end for both a traditional software design flow, relying on the usage of several tools that are already available to handle the complexity of this kind of design starting from UML, and a Simulink-based one.

1 Introduction

Present days’ embedded applications impose real-time constraints, in such a way that the system should be able to process complex algorithms within a given time slack, with minimum power dissipation. Conventional single processor architectures can no longer cope with these constraints. Therefore, the investigation of multiprocessor architectures (MPSoC) is required, on which several threads are executed concurrently. It would be interesting to use an appropriate high-level specification that can express parallelism in the target application and can be transformed into a multithread code in an automatic way.

Traditionally, the functional block (FB) modeling approach has been used by the signal processing and control engineering communities for the development of embedded systems. This approach has been widely accepted in industrial designs, driven by an extensive set of design tools, as for instance Matlab/Simulink [1]. On the other hand, the UML modeling language is considered the de facto modeling notation for any object-oriented (OO) system and has gained in popularity also for real-time embedded systems specification and design. Efforts that describe the use of UML in different phases of an embedded system design are shown in [2].
A previous comparison between UML and Simulink [3] concluded that both languages are somehow equivalent, but UML presents some advantages for requirements specification and represents a higher abstraction level when compared to Simulink. Moreover, UML provides all benefits from the OO paradigm, like modularity, encapsulation, reusability, etc. Although some UML tools support automatic code generation, designers are asked to write code for some methods in order to obtain the complete application code.

On the other side, Simulink supports multiple models of computation, and the whole code can be automatically generated from a Simulink model, since it relies on the use of pre-defined libraries and the designer is asked to write code for the user-defined functions. Real-time Workshop (RTW [4]) can be used to automatically generate code targeted to single-processor architectures from a Simulink model. Recently, a Simulink-based MPSoC design flow [5] was proposed to address multiprocessor architectures. This flow starts from a unified Simulink model, called Simulink CAAM, covering both the algorithm and the abstract target MPSoC architecture, and generates multithread code. However, building the CAAM using the Simulink graphical user-interface can be error-prone, and usually software engineers prefer to employ UML.

In this context, we propose the mapping from a UML model to a Simulink model, thus allowing the use of UML as a front-end for the Simulink-based multiprocessor design flow. The proposal also allows the integration of both modeling languages in a unique design flow, allowing one to exploit the benefits of UML for requirements specification and software design, while providing a way to obtain complete executable code for MPSoC architectures from the high-level specification. Moreover, the same UML model can be reused for different approaches.

The remaining of this paper is organized as follows. Section 2 discusses related work. Section 3 presents the proposed mapping. Section 4 describes a case study that demonstrates the utilization of this approach and shows the obtained results. Section 5 concludes the paper.

2 Related Work

Recent efforts show that both UML and Simulink are considered attractive for Electronic System Level (ESL) design [2][6][7]. This motivates researchers to find a way to simultaneously exploit the benefits of UML and Simulink modeling languages. Recently, SysML [6] was proposed as an extension to UML for systems engineering applications, with a higher degree of integration with the FB paradigm. However, the first SysML language specification was so close to UML that it is difficult to clearly define its improvements. Additionally, the available SysML modeling tools have not been evaluated yet for compliance or modeling capabilities.

In another effort to integrate both languages, the Rhapsody UML2.0 tool has been integrated with Matlab/Simulink, allowing the inclusion of elements in a Rhapsody model that are linked to a Simulink model [7]. This allows the use of Simulink resources to describe signal processing algorithms and simulation of heterogeneous
models that can include physical models like a plant, while at the same time UML is used for requirements specification. However, the proposed tool uses the code generation approach provided by RTW, which does not support multithread programming for MPSoC.

In this work, we propose the use of UML as front-end for the Simulink-based design flow, allowing one to exploit the benefits of UML, while generating executable code for MPSoC from high-level models. This allows designers to work at a higher abstraction level, avoiding the necessity of building or modifying Simulink models directly, which means abstracting about low-level details like signals and ports.

3 Proposal of Mapping from UML to Simulink

The Simulink-based MPSoC design flow was proposed in [5]. In this flow, a Simulink functional model and a hardware architecture template are combined in a mixed model, called Combined Architecture Algorithm Model (CAAM). From the Simulink CAAM, multiprocessor hardware architecture models and a multithreaded code targeted to the MPSoC architecture are generated.

The Simulink CAAM is built manually by a Simulink GUI Interface. From the Simulink functional model, the designer partitions functions into tasks and groups them into different subsystems, thus defining threads and mapping them to processors. Three hierarchical layers are used in the Simulink model to specify the CAAM. The first layer describes a system architecture that contains CPU subsystems and inter-CPU communication channels. The second layer describes a CPU-SS (CPU subsystem) architecture, which contains software threads and intra-CPU communication channels. The third layer describes a software thread that contains Simulink blocks and links. Intra or inter-CPU communication channels specify the desired communication protocols.

To maintain UML abstraction capabilities and eliminate the necessity of manually building the Simulink CAAM, this work proposes the mapping from UML to Simulink, thus allowing one to specify a system at a higher abstraction level through UML diagrams and automatically generating the Simulink CAAM from it. Then, multithread code can be generated from the system model. It must be noticed that the same UML model can be also used to generate code by UML commercial tools, thus enabling the reuse of models in different platforms or a comparison of different design alternatives.

Figure 1 illustrates the flow of the proposed mapping, which can be divided in three steps. The input of our flow is a UML model built using TopCased [8] or other EMF/UML2 compliant tool. In the first step, the UML model is traversed to find constructions that can be directly mapped to Simulink primitives using specified ATL transformation rules. This step produces another XML file that follows the target language meta-model, which is Simulink CAAM in this work. The transformation mechanism is supported by the TopCased/ATL engine, for which we have defined the required mapping rules. The second step has as input the resulting Simulink CAAM model and performs some optimizations before generating the final Simulink CAAM
model in ECore format. After that, from the resulting model, we generate an mdl file format. In this article, we have focused on the generation of Simulink, but this approach can be extended to support the mapping to other languages, such as UML, state diagrams, other FSM-like languages, or KPN.

![Diagram](image)

**Fig. 1.** Flow for the proposed UML-Simulink mapping

Our mapping rules are based on the UML deployment and sequence diagrams. Both diagrams are compulsory in order to capture the necessary information to generate the Simulink CAAM. Activity diagrams can also be used to detail the behavior of complex algorithms. Other diagrams like class and collaboration diagrams can be used during the modeling, but our tool does not capture information from them at the moment. Figure 2 illustrates an example of mapping, where 2(a) and (b) depicts UML diagrams and 2(c) shows the generated Simulink CAAM.

From the deployment diagram, the number of processors and threads are captured. Processors and threads are indicated by the `<Processor>` and `<SchedRes>` UML-SPT (UML profile for Schedulability, Performance and Time) stereotypes, respectively. For each processor, a Simulink hierarchical subsystem is created in the CAAM representing a CPU-SS. For each thread mapped to a processor, a Thread subsystem is created inside the corresponding CPU-SS to represent the Thread-SS that will use Simulink blocks to specify its behavior.

The data flow can be captured from the sequence diagram. The `<STrigger>` stereotype indicates a timer event and the invoked method for which the scheduler selects a thread to run. Other method calls are translated to Simulink blocks (user-defined or pre-defined blocks) or to communication blocks in Simulink CAAM. When a method of a passive object is called from a thread, a subsystem is created inside a Simulink block. In Figure 2(c), the subsystem `Sensor` contains one block generated from the method invocation `read`. The information about the direction of method parameters is used to define input and output ports for subsystems. To use pre-defined Simulink blocks, the designer needs to indicate its usage by a stereotype in the method call. If no stereotype is found, a Simulink S-function is instantiated for a method invocation.

When a thread invokes a method from another thread, this indicates that there is a communication between them. In order to represent it in Simulink, ports are created in the Thread subsystem. To indicate the direction of these ports, the designer must use a default prefix (`Set` and `Get`) to indicate send and receive operations, respectively.
Beside the ports, an intra-SS COMM or an inter-SS COMM subsystem is instantiated, according to the mapping indicated in the deployment diagram. After that, connections are created between the ports of these subsystems. In Figure 2(c), the method `getAngle()` indicates that the thread `Nav` receives data from `MovCtrl`. As both threads are mapped to the same CPU, an intra-SS COMM is instantiated to perform this communication, as shown in Figure 2(d). In figure 2(c), the thread `Nav` invokes a method from an actor, indicating that the value `r` generated by this thread is sent to a system output.

During the optimization step, the tool performs three kinds of optimizations: inference of communication channels, loop detection, and thread grouping. From the sequence diagrams, where there are method invocations between different threads, the tool can capture the kind of communication (inter-CPU or intra-CPU), thus setting the appropriate protocol. When a variable is used as input and output of a function, we have a cyclic path (or loop). In a Simulink model, to avoid deadlock, one needs to insert a temporal barrier (`Delay`) to guarantee that a valid value is available for the input function. The tool looks for cyclic paths in the model and inserts temporal barriers in the generated Simulink model. Moreover, our tool analyzes the model and groups threads whenever possible, in order to reduce the communication overhead.

As future work, this tool will be integrated with an estimation tool to improve design space exploration, such as to automatically generate the deployment model.

4 Case Study

In this work, two case studies are used to validate the proposed mapping. The crane control system, proposed in [9], was used to show the capabilities regarding the
capture of a dataflow from the UML model and the generation of the corresponding Simulink model. Using this case study, we also show that our tool can automatically insert the required temporal barriers in the generated Simulink model.

In addition, a wheelchair movement controller was also used as case study to show the effective optimizations supported for our mapping tool. In this experiment, the designer specifies the system in UML, which is composed of four threads. During the mapping, our tool can group threads in order to reduce the communication cost.

Due to the lack of space, we are not able to give more details of these case studies in this article.

5 Conclusions

This work proposes an automatic mapping from UML to Simulink CAAM, thus eliminating the necessity of manually building the Simulink model used as input for the Simulink-based design flow for MPSoC architectures. The mapping is based on sequence and deployment diagrams, and we show that some UML constructions can have a direct mapping to Simulink. Whenever this is not possible, inferences are necessary and performed in the optimization phase of our tool. The proposed mapping allows one to exploit the benefits of UML for requirements specification and software design, while providing a way to obtain complete executable code for MPSoC architectures from the high-level specification. Moreover, the same UML model can be used to generate code using either traditional UML tools or a Simulink-based approach.

References