

Using UML as Front-end for Heterogeneous Software Code Generation Strategies

Lisane B. Brisolaro, Marcio F.S. Oliveira, Ricardo Redin, Luis C. Lamb, Luigi Carro, Flavio Wagner
{lisane, mfsoliveira, rmredin, lamb, carro, flavio}@inf.ufrgs.br
Institute of Informatics– Federal University of Rio Grande do Sul

Abstract

In this paper we propose an embedded software design flow, which starts from an UML model and provides automatic mapping to other models like Simulink or finite-state machines (FSM). An automatic synthesis of an executable and synthesizable Simulink model is also proposed, enabling the use of UML as front-end for a multi-model design strategy that includes a Simulink-based MPSoC target design flow. In addition, the proposed synthesis tool automatically handles processor allocation, mapping of threads to processors, and insertion of required Simulink temporal barriers, ports, and dataflow connections. Following this approach, the UML model is mapped to the more appropriated model and specialized code generators are used. Therefore, this approach allows designers to employ UML to model the whole system and reuse this model to generate code using different strategies and targeting different platforms.

1. Introduction

The rising costs of hardware development motivate the reuse of pre-defined platforms and the use of software-based implementations, where product differentiation is achieved by the software. Nowadays, software development responds for most of the design time spent and is the largest cost factor in embedded systems design [1]. This scenario motivates the investigation of strategies to accelerate embedded software development by process automation.

UML [2] supports the whole software development process, beginning with requirements analysis and supporting object-oriented system specification, design, and modeling, thus being an attractive modeling language to the software community. Further, UML provides high abstraction and increases maintainability. The growing interest for using UML in software projects has led to an increase in its popularity, in particular in the embedded system community [3].

Embedded systems are usually compositions of subsystems, each of which may be based on a different Model of Computation (MoC) [3]. Each MoC, with

associated modeling notation, is chosen with respect to its adequacy to the subsystem domain. UML is widely used in the development of event-based systems in the software engineering domain, since the UML modeling style maps nicely to the underlying object-oriented paradigm. However, UML is not well suited to model dataflow or other MoCs required by heterogeneous systems, like those present in most current or emerging embedded systems (e.g. mobile phones).

Simulink [4] is another language currently used for embedded systems design, with a focus on control and signal processing. This language supports dataflow and continuous time models, but it lacks the desired high-level abstraction and the support of other software-oriented models.

Recent efforts have shown that both UML and Simulink are considered attractive for electronic system-level design [5, 6, 7], a fact that motivates researchers to aim at finding a way to simultaneously exploit the benefits of both languages. One strategy to achieve this aim could be the use of either UML or Simulink whenever the underlying software best fits a certain model. However, we advocate that the use of a single language facilitates the communication among the designers from both software and embedded systems domains.

In this paper, we propose a design flow to use UML as the modeling language of the whole system, and provide automatic generation of an executable and synthesizable Simulink model, whenever the application requires it. In this way, designers have to know a single modeling language, a high-level abstraction can be used for the system specification, and specific details of different MoCs can be abstracted and handled by our synthesis tool.

However, there is no clear and straightforward way to effortlessly translate from an UML model to a functional-block model such as Simulink [8]. Specific details must be added during the synthesis process. We focus on the translation of UML to Simulink, targeting an MPSoC design flow [9]. Our proposed model and tool automatically handle the allocation of processors, the mapping of threads to processors, and the insertion of required Simulink temporal barriers, ports, and dataflow connections, in order to generate a synthesizable Simulink model from the UML model. We also illustrate the

effectiveness of our methodology by means of two case studies.

The remaining of the paper is organized as follows. Section 2 describes previous work on the integration of UML and Simulink. Section 3 introduces the proposed methodology. Section 4 details the automatic mapping from UML to Simulink models. Section 5 presents the experimental results and analyzes them. Section 6 concludes and point out directions for future work.

2. Related work

Several authors have already investigated the need to use a single language for embedded system design [8, 10]. Several efforts have also been made proposing the integration between UML and Simulink, varying the provided abstraction and effective integration [10,11,12,13].

As stated in [6,14], two different approaches for integration of UML and Simulink have been proposed so far: co-simulation and integration based on the target implementation language. As an example of the co-simulation approach, the Exite tool [11] allows the coupling of a Simulink model with ARTiSAN Software Real-Time Studio or I-Logix Rhapsody. In another effort, the Rhapsody UML2.0 tool has been integrated with Matlab/Simulink, enabling the building of UML mixed models which allow for modules to be described in Simulink [12].

The alternative approach is the use of a common execution language. This solution is adopted in the Constellation framework [13] and in the GeneralStore integration platform [10]. Although the authors claim that a bidirectional mapping between UML and Simulink is provided in the GeneralStore framework, only the capturing of a Simulink model in an UML model is supported, and the integration is effectively made at the code-level.

Both approaches focus on the use of different modeling languages to specify each system module. Differently from these approaches, we propose the use of UML as a single modeling language for initial specification, which provides the advantage of using a standard language that is widely accepted in the software engineering community. Moreover, neither of the previous approaches provide the synthesis of a Simulink model like the one we propose here. Our approach directly derives the Simulink model from the UML modeling strategy chosen by the designer.

3. Proposed design flow

Figure 1 illustrates the proposed design flow for embedded software development. In this flow, the UML-based code generation can be used to generate code for event-based (control-flow) subsystems, using available tools that generate code from state diagrams or FSM-like models [15]. On the other side, Simulink-based strategies can be used to generate code for the dataflow subsystems.

Moreover, in case a Simulink compiler is not available, the same UML model can be used to generate multithreaded code for other languages, e.g. Java.

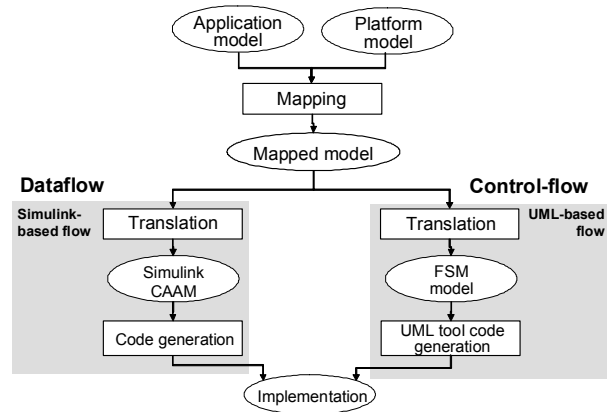


Figure 1: Proposed design flow

Figure 2 presents the mapping flow defined to capture an UML model and transform it into another modeling language, which has two main steps. The first step is the UML model construction, and this is made by the designer. In the second step, the UML model is traversed to find constructions that can be directly mapped into the target modeling language, e.g. Simulink, which is defined in a meta-model. This is a model-to-model transformation, following a model-driven engineering approach. This step produces another XML file that follows the target language meta-model, which can be Simulink or FSM, as illustrated in Fig. 2. In order to be flexible, technologies for model transformation, such as smartQVT [16] and ATL [17], should be used to assist in this translation.

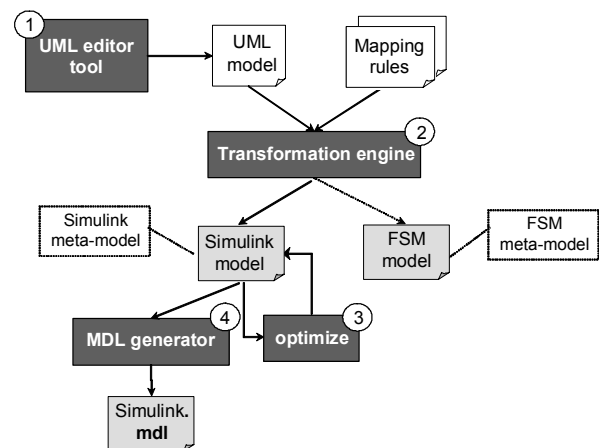


Figure 2: Proposed mapping flow

Our focus here is on the support of the dataflow paradigm, by the mapping from UML to Simulink, and later this model can be used as input to the MPSoC design flow [9]. This design flow uses as input a Simulink Combined Architecture Algorithm Model (CAAM), which is manually built by the Simulink GUI Interface. Differently from Simulink, CAAM provides information

about the processors and thread allocation. The use of the tool presented in this paper eliminates this manual step. Therefore, it allows designers to employ UML to model the whole system, which is the preferred language to model software, besides endowing them with a higher abstraction model.

The third and fourth steps in Fig. 2 are specifically tailored to the generation of a Simulink model from an UML one. The third step receives as input the model resulting from the model-to-model transformation, which follows the Simulink meta-model semantics, and performs some optimizations (described in Section 4) before generating the final Simulink model. After that, from the optimized model, a Simulink *mdl* file is generated using model-to-text transformation in the fourth step. Although we have focused on generating the Simulink model, the proposed transformation approach can be extended to support mappings to other languages, such as UML state diagrams, other FSM-like languages, or KPN (Kahn Process Network).

4. Mapping from UML to Simulink CAAM

In order to apply the proposed model transformation, the target language needs to be defined as a meta-model. We defined a meta-model for the Simulink CAAM. The proposed mapping can be used to generate both conventional Simulink models as well as CAAM Simulink models. Section 4.1 explains the proposed mapping. A prototype that implements it has been developed and is detailed in Section 4.2.

4.1 Mapping rules

Our mapping uses information from the UML deployment and sequence diagrams to obtain the Simulink CAAM. To explain the mapping, a didactic example is presented in Fig. 3. Figures 3(a) and (b) depict the deployment diagram and a partial sequence diagram of the system, respectively. Figure 3(c) shows the Simulink CAAM obtained by the mapping.

The allocation of threads to processors is captured from the deployment diagram. The `<<SAEngine>>` and `<<SASchedRes>>` UML-SPT stereotypes indicate processors and threads, respectively, as shown in Fig. 3(a). For each processor and thread, a Simulink hierarchical subsystem is instantiated in the CAAM model to represent a CPU subsystem (CPU-SS) and a Thread subsystem (Thread-SS), respectively, as illustrated in Fig. 3(c).

The thread behavior is captured from sequence diagrams, once it represents messages exchanged between components. This capturing is used to build the CAAM Thread-layer, which is composed of Simulink blocks. Method calls are translated to Simulink blocks or to communication blocks in the Simulink CAAM. When a method of a passive object is called from a thread, a Simulink block is instantiated. To use pre-defined blocks, the designer needs to indicate its usage by the invocation

of a method from the special object *Platform*, which represents the Simulink library. When the method name does not match the pre-defined component names, a user-defined Simulink block called S-function is instantiated. An S-Function can have its behavior described in a C code that is compiled and linked to the model. In the example shown in Fig. 3(b), the *dec* and *mult* methods are invoked from the *Dec* and *Platform* objects, respectively, by the thread T1. Notice that in the resulting Simulink model, shown in Fig. 3(c), a *Product* block and an S-function were instantiated in the T1 subsystem.

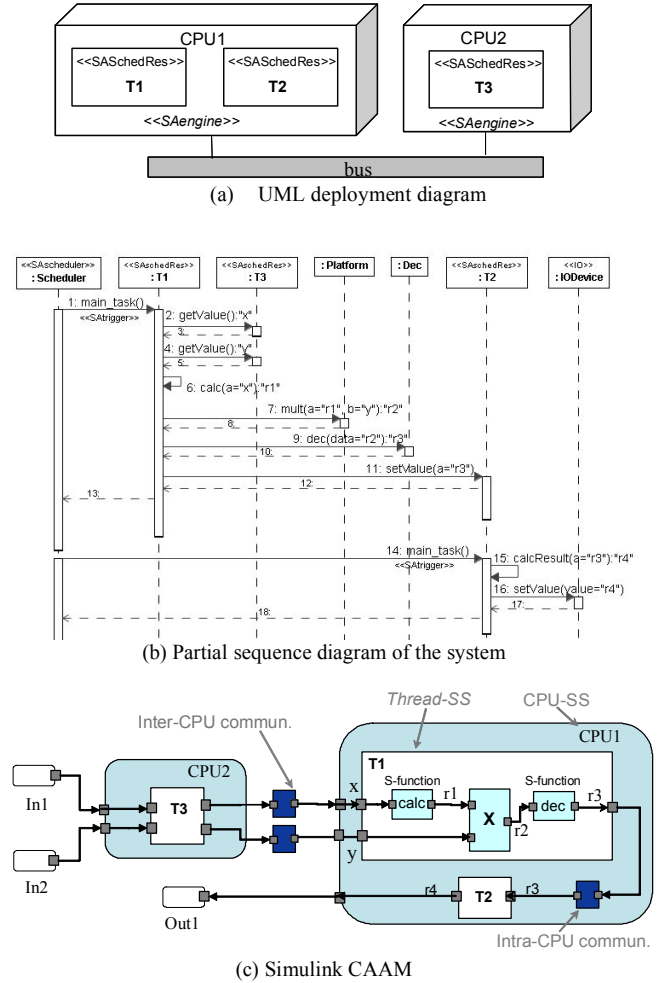


Figure 3: Didactic mapping example

The direction of method parameters (*in/out*) and the return are translated to input and output ports of subsystems/blocks, and message arguments to connection (data links) between different subsystems/blocks. The *a* parameter from *calc* method and its return are mapped to an input port and an output port in the *calc* S-function, as shown in Fig. 3(c). The *r1* argument is passed from *calc* to *mult*, thus a connection is instantiated between these ports when generating the Simulink model.

When a thread invokes a method from another thread, ports and communication channels are instantiated in the Simulink CAAM. In this case, the designer is asked to use a default prefix in the method name, *Set* or *Get*, to

indicate send or receive operations, respectively. After that, connections are created between the ports of these thread subsystems. The instantiation of communication channels is discussed in detail in Section 4.2.1.

The communication with external systems is indicated through method invocations from a special object decorated with the <<IO>> stereotype. This is a new stereotype we have defined. Methods with the prefix *get* and *set* are used to indicate the reading and writing operations and, during the mapping, they are mapped to system's input and output ports. In the example (see Fig. 3(b)), T2 invokes the *getValue()* method from the *IODevice* object marked as <<IO>>. This invocation is translated to a system output port in the Simulink CAAM, as shown in Fig. 3(c).

The deployment diagram defines the number of processors and allocates the threads to processors. We propose the automation of these decisions by the use of an optimization algorithm that groups threads into clusters according to their data dependency. This is detailed in Section 4.2.3. The use of this algorithm makes the deployment diagram unnecessary when generating the Simulink CAAM from an UML model.

4.2 Mapping tool prototype

We have developed a prototype that implements the mapping proposed in Section 4.1 and that uses the flow depicted in Fig. 2. For building the UML model, MagicDraw or other EMF/UML2 compliant tool is used. During the second step, the UML model is traversed and translated into a Simulink model. This step produces an XML file, which conforms to the Simulink CAAM meta-model. This transformation was implemented in Java using the API provided by the Eclipse EMF, according to the required mapping rules described in Section 4.1. The third step receives as input the resulting Simulink CAAM model represented using the E-core format (XML-like) and performs some optimizations before generating the final Simulink CAAM model. After that, from the resulting model, we generate a file in *mdl* format used as input in the Simulink environment.

During the optimization step, our tool can perform three types of optimizations: inference of communication channels, loop detection, and thread allocation, which are detailed in Sections 4.2.1, 4.2.2 and 4.2.3, respectively.

4.2.1 Inference of communication channels

In the Simulink CAAM, the communication is explicitly represented by communication channels that can be either inter-subsystem (inter-SS) or intra-subsystem (intra-SS). When the communicating threads are in different CPUs, an inter-SS channel is required. Otherwise, an intra-SS channel is instantiated. To instantiate the channels, we use information from the sequence and deployment diagrams or from the result of the thread allocation algorithm.

In the sequence diagram depicted in Fig. 3(b), T1 invokes the method *getValue()* from T3, which indicates

that T1 receives data from T3. As both threads are allocated in different processors, an inter-SS channel is instantiated in the Simulink model, as shown in Fig. 3(c). The method call *setValue(r3)* in Fig. 3(b) indicates that T1 sends data to T2. The same argument *r3* is also used by the *dec* method, indicating that the value produced by this method must be sent to T2. This communication is translated to an output port in T1, and an intra-SS channel is instantiated, since both threads are mapped to the same CPU.

The communication protocol is indicated in the Simulink CAAM using a parameter. At present, we use two different protocols, the SWFIFO for intra-SS channels and the GFIFO for inter-SS ones. Our tool instantiates communication channels and sets their parameters.

4.2.2 Insertion of temporal barriers

When describing a dataflow model, cyclic paths need to be found and temporal barriers are required to avoid deadlocks. In order to do that, the Simulink model obtained from the translation (step2) is searched for cyclic paths. Simulink Delay blocks are then inserted in the resulting model. Our tool automatically detects the cyclic paths and inserts a Simulink *UnitDelay* block in the data link where the loop is detected.

4.2.3 Thread allocation

This optimization allocates threads with more data dependencies in the same processor, in order to reduce the inter-processor communication. When this optimization is applied, the deployment diagram is not needed to generate the Simulink CAAM. The data dependency between threads is captured from the sequence diagrams, and a task graph is built, where the nodes are threads and the edges have a cost. This cost is determined by the amount of transferred data.

An algorithm based on Linear Clustering [18] is used to evaluate the graph. This algorithm separates parallel tasks into different clusters and groups threads with more data dependencies into the same cluster, which means that they will be allocated to the same processor. The result of this optimization step is used to generate the top-level description of the Simulink CAAM. This step is optional, and when the designer wants to decide the mapping by himself, one can use the deployment diagram.

It is interesting to notice that this algorithm allocates all threads that are in the system critical path to the same processor. This is a good practice to reduce the communication cost, once the cost for intra-CPU communication is lower than the cost for communication between different CPUs.

5. Experiments

We are now in position to illustrate the effectiveness of our approach through two case studies: a crane control system and a synthetic example.

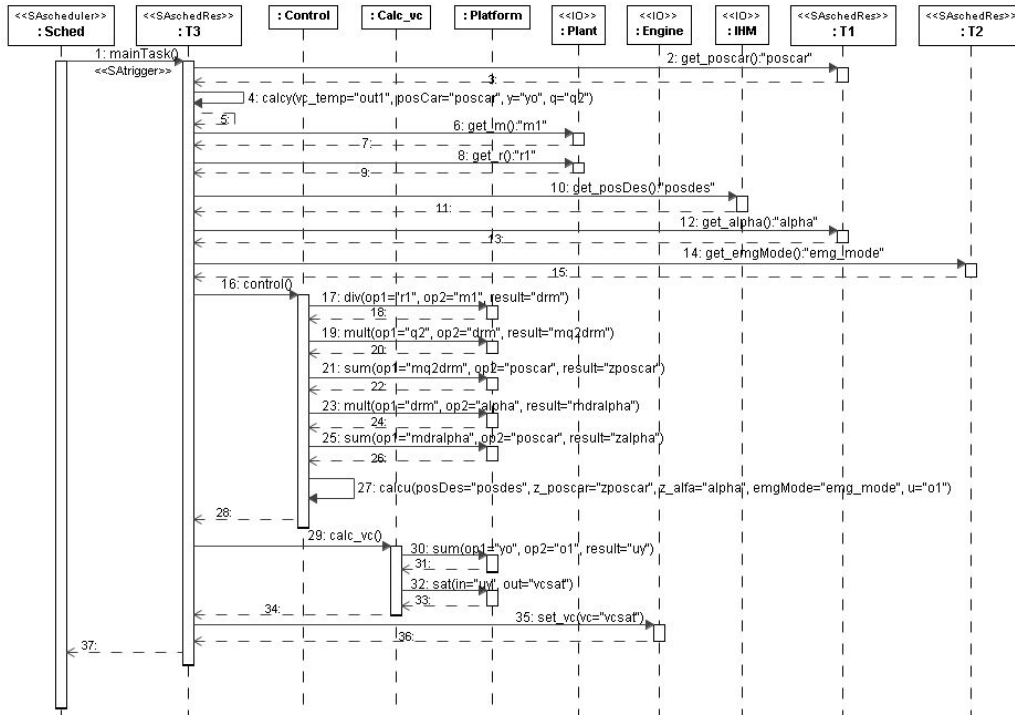


Figure 4: UML sequence diagram for the Crane - T3

5.1 Crane control system

The crane control system [18] is used as case study to show the capabilities of our approach to generate a Simulink model from an UML model. We also show that our tool automatically inserts the required temporal barriers in the generated Simulink model. We divide the system into three threads, where each one is specified using UML sequence diagrams. The three threads were mapped to the same processor, which was defined through a deployment diagram. Figure 4 illustrates the UML sequence diagram for thread T3, and Figure 5 shows the Simulink model generated for T3, which is composed of one S-function and two subsystems and a Delay that is automatically inserted. The subsystem *control* has its behavior detailed in Fig. 5.

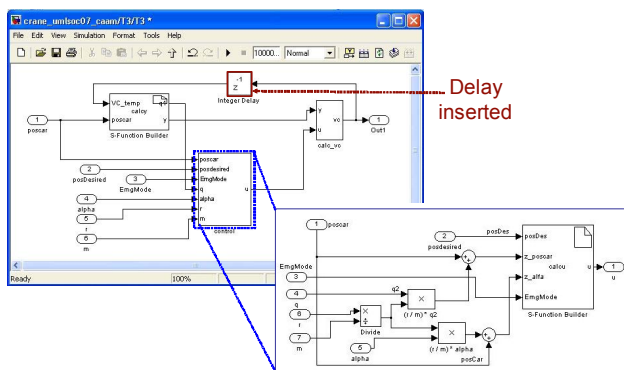


Figure 5: Generated Simulink for the Crane

5.2 Synthetic example

To validate the proposed thread allocation in the optimization step, we have developed a synthetic example, which has twelve communicating threads. The application was specified using a sequence diagram that expresses the communication between the application threads. Figure 6 illustrates a block of interactions of this sequence diagram, since the whole diagram is too large to be shown here.

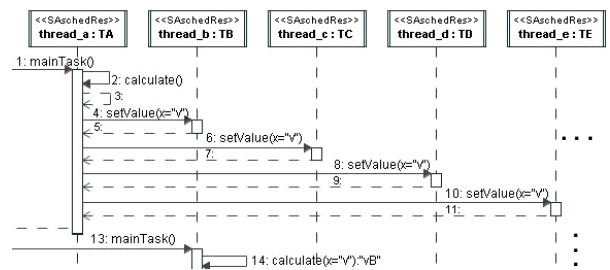


Figure 6: Partial UML model for the synthetic example

The communications captured from the sequence diagrams are used to build a task graph, as shown in Fig. 7(a). Using our proposed optimization, the thread allocation is automatically determined and the result is shown in Fig. 7(b).

After applying our approach, a Simulink CAAM model was generated, which is depicted in Fig. 8. This figure shows the top-level model, where four CPU

subsystems communicate through inter-SS channels. The inference of communication channels is also automatically performed to build this Simulink CAAM.

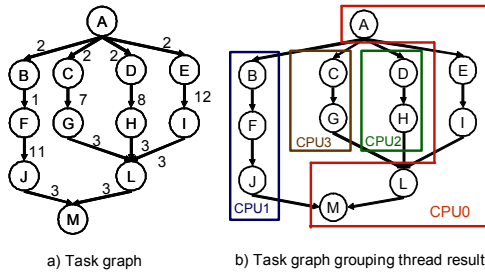


Figure 7: UML sequence diagram for the synthetic example

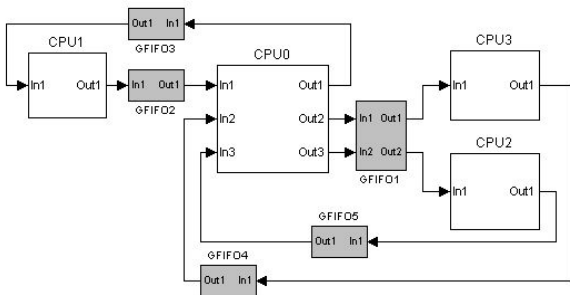


Figure 8: Simulink CAAM top-level for the synthetic example

6. Conclusions and future work

We have proposed a design flow which starts from an UML model and provides automatic mapping to other models such as Simulink or FSMs. Using this approach, the UML model is mapped to a more appropriate model and specialized code generators are used to obtain an implementation. Moreover, this approach allows designers to employ UML to model the whole system and reuse this model to generate code using either traditional UML tools or a Simulink-based approach.

In addition, an automatic mapping from UML to Simulink CAAM has been proposed. This allows us to eliminate the need of manually building the Simulink model used as input for a Simulink-based MPSoC design flow, which generates HW and SW for an MPSoC platform. The proposed mapping is based on sequence and deployment diagrams.

We have also shown that some UML constructions can have a direct mapping to Simulink. However, the one-to-one mapping is not able to capture the whole model. There is still the need to make inferences performed in the optimization phase of our mapping tool. Two case studies have been presented to show how the proposed optimizations are computed during the mapping from UML to Simulink.

Currently, the designer needs to partition the system into threads and to describe thread behavior using sequence diagrams to apply the mapping. As future

work, we plan to integrate an estimation step in the proposed development flow to automatically determine the best partitioning and mapping solution. This would avoid the need for the designer to specify the deployment model and partition the system into threads, while supporting design space exploration.

Moreover, we have used sequence diagrams to capture thread behavior in our mapping. However, other behavior diagrams could also be used by a designer, since UML provides them. Thus, we plan to extend this mapping to support other UML diagrams, such as activity diagrams.

References

- [1] B. Graaf; M. Lormans; H. Toetenel. "Embedded Software Engineering: The State of the Practice". IEEE Software, v. 20, n. 6, 2003, p. 61-69.
- [2] OMG. Unified Modeling Language (UML). <http://www.omg.com/>
- [3] Martin, G. "UML for Embedded Systems Specification and design: Motivation and overview". DATE 2002.
- [4] Mathworks. Simulink. <http://www.mathworks.com>.
- [5] D. Densmore; R. Passerone; A. Sangiovanni-Vincentelli. "A Platform-Based Taxonomy for ESL Design". IEEE Design and Test of Computers, v.23, n.5, p.359-374. 2006.
- [6] Y. Vanderperren; W. Dehaene. "From UML/SysML to Matlab/Simulink: Current State and Future Perspectives". DATE 2006. p.93-93.
- [7] L. Brisolara et al. "A Comparison between UML and Function Blocks for Heterogeneous SoC Design and ASIP Generation". In: G. Martin; W. Mueller (Eds.): UML for SoC Design. Springer-Verlag, 2005. p. 199-222.
- [8] Tranoris, C.; Thramboulidis, K. Integrating UML and the function block concept for the development of distributed control applications. ETFA 2003. p.87-94.
- [9] Huang, K. et al. "Simulink-Based MPSoC Design Flow: Case Study of Motion-JPEG and H.264". DAC 2007.
- [10] C. Reichmann et al., "Model Level Coupling of Heterogeneous Embedded Systems", Proc. of 2nd RTAS Workshop on Model-Driven Embedded Systems, 2004.
- [11] Extessy. Exite tool. <http://www.extessy.com/>
- [12] R. F. Boldt. "Combining the Power of MathWorks Simulink and Telelogic UML/SysML- based Rhapsody to Redefine MDD". Telelogic White Paper, Oct. 2007.
- [13] Real-Time Innovation. Constellation framework. <http://www.rti.com/>
- [14] W. Mueller et al. UML for ESL Design – basic Principles, Tools and Applications. ICCAD 2006.
- [15] Mentor Graphics. BridgePoint UML Suite. <http://www.projtech.com>.
- [16] SmartQVT. <http://smartqvt.elibel.tm.fr/>.
- [17] Eclipse Development Team. ATLAS Transformation Language (ATL). <http://www.eclipse.org/m2m/atl/>.
- [18] Gerasoulis, A. and Yang, T. "On the Granularity and Clustering of Directed Acyclic Task Graphs". IEEE Transactions on Parallel and Distributed Systems. v. 4, n. 6, June, 1993.
- [19] E. Moser, W. Nebel. "Case Study: System Model of Crane and Embedded Control". DATE 1999.

Acknowledgments: This work has been partly supported by the Brazilian Research Council CNPq.