Model Based Approach for Heterogeneous Application Modelling for Real Time Embedded Systems

Sanna Määttä[1] Leandro Soares Indrusiak[2] Luciano Ost[3] Leandro Möller[4] Manfred Glesner[4] Fernando Gehm Moraes[3] Jari Nurmi[1]

[1] Department of Computer Systems, Tampere University of Technology, P.O.Box 553, FIN-33101 Tampere, Finland
[2] Department of Computer Science, University of York, York, YO10 5DD, United Kingdom
[3] Catholic University of Rio Grande do Sul (PUCRS), Av. Ipiranga, 6681 - P.32 - 90619-900, Porto Alegre, Brazil
[4] Institute of Microelectronic Systems, Technische Universität Darmstadt, Karlstrasse 15, 64283 Darmstadt, Germany

Abstract. Modelling and simulation are essential methods in state of the art embedded system design. Especially modelling at high levels of abstraction helps designers to handle the design complexity and heterogeneity. This paper presents a model based approach for modelling and validation of heterogeneous application together with a multicore Network-on-Chip (NoC) platform. We use the MARTE profile to capture the structure and real time features of our system model and we model the application using UML sequence diagrams and directly simulate the diagrams in the Ptolemy II framework. As the case study shows, our approach sufficiently covers both communication and computation aspects of real time and heterogeneous systems.

1 Introduction

In model based design, a system model is the key element of the design process from the specification to implementation. Modelling helps designers to manage complex systems, better understand the system under development, visualise a system, specify the structure or behaviour of the system, and document the decisions [1]. Moreover, modelling reduces development time and costs. A system can be modelled many different ways, at many levels of abstraction, and presenting different level of formalism. Very often the system model is built ad hoc. However, without modelling, it is likely to either build a wrong system or build the system wrong.

Modelling systems having real time features sets requirements for the modelling language. The Unified Modelling Language (UML) responds to the real time requirements through profiles, such as the Profile for Modeling and Analysis of Real-Time and Embedded Systems (MARTE) [2].

The MARTE profile customises UML for model driven development of Real Time and Embedded (RTE) systems. For instance, it supports the specification, design, and validation stages of embedded system design. MARTE extends UML with constructs for modelling RTE software applications, high level RTE hardware, and their nonfunctional properties. Moreover, MARTE provides designers with necessary extension units in order to address performance and schedulability analysis of RTE systems. The MARTE profile is organised into a set of packages, such as foundations, design model, and analysis model. The MARTE foundation package contains basic elements for modelling nonfunctional properties, timing, and general resources. The allocation concept associates application functions with the execution platform resources. The design model package includes elements for modelling generic components, software and hardware components, and the application. Furthermore, the analysis package contains modelling capabilities for scheduling and performance analysis and enables designers to perform for instance timing analysis directly from the UML description instead of building a separate model for analysis [2].

This paper presents a model based approach for real time embedded system design using UML and MARTE. Our approach is actor oriented and besides addressing real time requirements, it also covers another important aspects of embedded system design, namely heterogeneity. The contribution of this paper is to first combine our previous work on joint validation of application and platform models [3] [4], a MARTE based UML profile for describing the application constructs [5], and heterogeneous application modelling [6] into a model based approach. Second, the contribution is to extend the previous work with real time features. Moreover, we also demonstrate the potential of our approach and its suitability for real time embedded system design in a case study.

The rest of the paper is organised as follows: Section 2 describes related work on using UML profiles or model based approach for embedded system design. Section 3 describes our approach more in detail. In Section 4 we present an application modelling example with a simulation case study. Finally, Section 5 draws conclusions.

2 Related Work

Modelling using UML diagrams is getting more acceptance among the hardware and embedded system community [7]. Especially, when profiles, such as MARTE enables also the RTE system design. However, while MARTE provides designers with the concepts and syntax for modelling real time features, such as deadlines and periods of the system, we also need a tool or framework for system simulation, such as Simulink® [8], Accord|UML [9], or Ptolemy II [10].

The MathWorks Simulink® is a commercial environment for model based design for dynamic embedded systems [8]. Accord|UML framework provides a facility for code generation while transforming UML and MARTE based models into executable models [9]. Ptolemy II is a framework for modelling heterogeneous embedded systems [10]. Although Ptolemy II does not support the simulation of UML diagrams as such, it can be extended to be able to simulate for instance UML sequence diagrams, as done in [11].

Our approach uses the execution semantics of the Ptolemy II framework without the need for code generation when simulating the system model. Our model based approach for application modelling is described more in detail in the next Section.

3 Modelling and Simulation of Heterogeneous Systems

We use an autonomous vehicle application as an example to demonstrate the potential of our design approach. The functionality of the vehicle is described as UML sequence diagrams, illustrated in Figure 1. The vehicle has sensors for sensing for instance speed, tyre pressure, and vibration as well as two cameras for snapshots. The vehicle can adjust for instance its direction and speed according to the obstacles shown in the snapshots and its tyre pressure according to the information received from the sensors. The platform we simulate the application on runs under the Discrete Event (DE) Model of Computation (MoC) as well as most of the application. However, the application parts processing pictures run under Synchronous Data Flow (SDF) MoC (the Obstacle Recognition sequence diagram in Figure 1), whereas the speed controlling of the vehicle is modelled using the Continuous Time (CT) MoC, as described more in detail in [6].

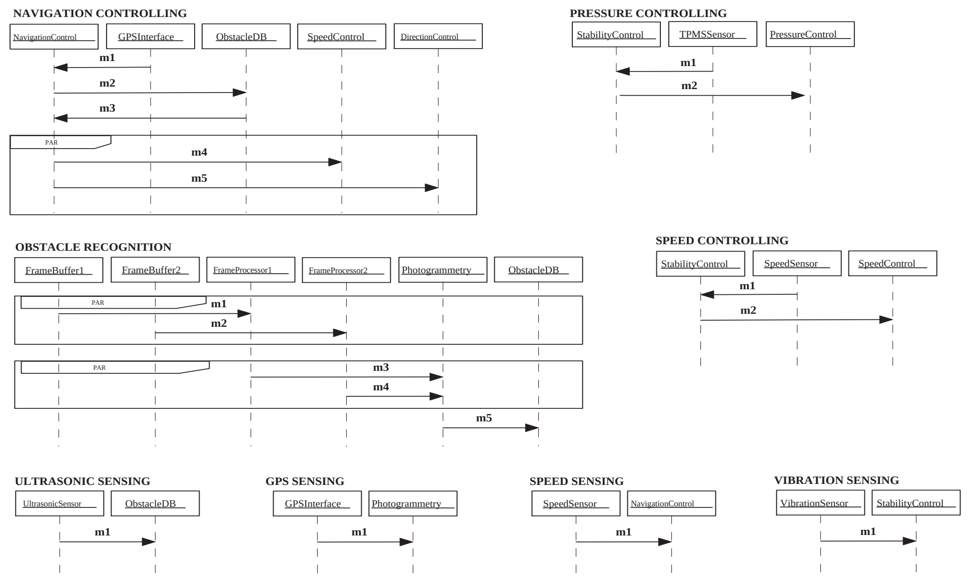

Fig. 1. Sequence diagrams describing an autonomous vehicle application

3.1 System Modelling

Figure 2 depicts a composite structure diagram of both the application, platform, and the mapping of each application actor onto a Processing Element (PE) on the platform. The lower part of the figure illustrates the application model and the upper part the platform. In our previous work, we defined only the structure of the application without much support for real time features [5]. In this paper we have provided our application model with soft real time features, such as deadlines for messages in order to be able to monitor for instance timing constraint violations.

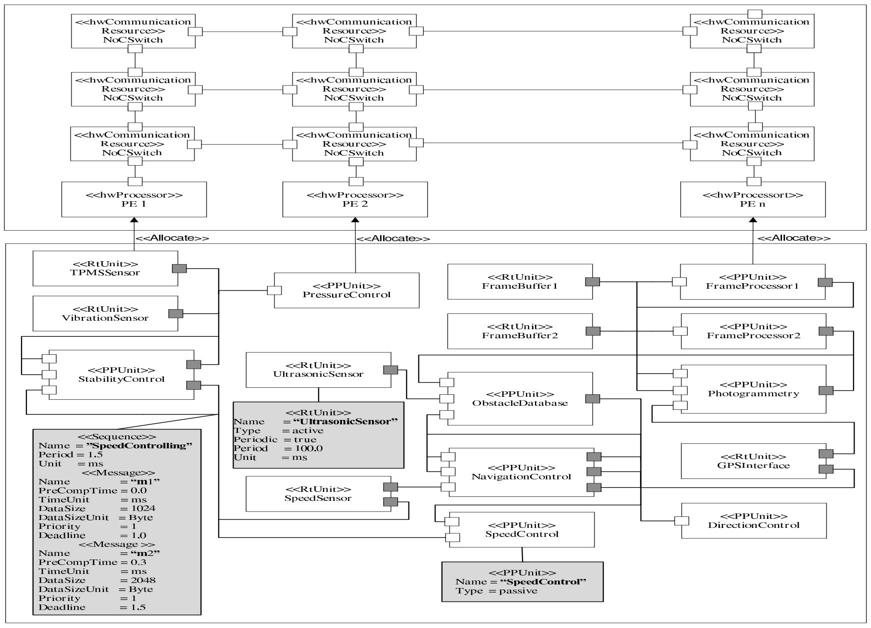
<<hwCommunication Resource>> NoCSwitch
<<hwProcessor>> PE 1
<<hwProcessor>> PE 2
<<hwProcessort>> PE n
<<Allocate>>
<<RtUnit>> TPMSSensor
<<RtUnit>> VibrationSensor
<<PPUnit>> PressureControl
<<PPUnit>> StabilityControl
<<RtUnit>> UltrasonicSensor
<<RtUnit>> Name = "UltrasonicSensor" Type = active Periodic = true Period = 100.0 Unit = ms
<<Sequence>> Name = "SpeedControlling" Period = 1.5 Unit = ms
<<Message>> Name = "m1" PreCompTime = 0.0 TimeUnit = ms DataSize = 1024 DataSizeUnit = Byte Priority = 1 Deadline = 1.0
<<Message >> Name = "m2" PreCompTime = 0.3 TimeUnit = ms DataSize = 2048 DataSizeUnit = Byte Priority = 1 Deadline = 1.5
<<RtUnit>> SpeedSensor
<<PPUnit>> ObstacleDatabase
<<PPUnit>> NavigationControl
<<PPUnit>> SpeedControl
<<PPUnit>> Name = "SpeedControl" Type = passive
<<RtUnit>> FrameBuffer1
<<RtUnit>> FrameBuffer2
<<PPUnit>> FrameProcessor1
<<PPUnit>> FrameProcessor2
<<PPUnit>> Photogrammetry
<<RtUnit>> GPSInterface
<<PPUnit>> DirectionControl

Fig. 2. Composite structure diagram of the application and platform models

Our platform is a JOSELITO Network-on-Chip (NoC) [12] interconnect arranged as a mesh topology. Each network switch is connected to a PE. For the sake of clarity, we show only three PEs in Figure 2. MARTE stereotype <<*hw-CommunicationResource*>> is used for each network switch, whereas <<*hw-Processor*>> describes the PEs. We use MARTE's <<*allocate*>> concept to describe the mapping of a function into the architecture, that is, mapping an application actor onto a PE. Figure 2 shows the allocation of three application actors to the PEs.

The stereotypes <<*RtUnit*>> and <<*PPUnit*>> describe the active and passive application actors. Active actors can initiate communication and passive actors communicate only as a response to a received message. Figure 2 depicts the application actors as well as their input and output ports (white and grey squares). Each contiguous set of line segments corresponds one sequence diagram describing a part of the application model. Each application actor and sequence diagram has parameters and constraints. Whereas passive actors have only their name and type as parameters, active actors have also a period parameter defining the time interval at which they initiate communication. Also the sequence diagrams have the period parameter and in addition they contain the messages. The messages can be constrained with computation time (the delay caused to the packet at the processing element as if a processor would execute the task), data size (the packet size that carries the message in the NoC), and priority (for scheduling purposes) [5]. In this paper, we have also set a deadline for each message indicating the maximum time a computation and communication of the message can take.

3.2 System Interaction

Figure 3 illustrates the interaction between different elements of our system. The lifelines represent either a Director (SDDirector and Director), an attribute (Mapper), a class (PrecedenceGraph), or actors (CompositeActor, Producer, NoCSwitch, Consumer). Directors implement the MoC in the Ptolemy II framework, in our case a Director is capable of executing UML sequence diagrams.

The UML sequence diagrams describing the application model are first transformed into acyclic, directed message graphs (an example of a sequence diagram and its corresponding task graph can be seen in Figure 4). The super class of directors (SDDirector) ask the executing Director to create a graph for each sequence diagram describing our application model. The executing Director passes this request to the PrecedenceGraph class. When the graphs are created, the Director needs to know whether the application is simulated as a platform independent stand-alone model or whether it is running on a platform. This is done by a method call that returns the Mapper if it exists.

Inside the alternative fragment of the sequence diagram in Figure 3 the upper part describes the application simulation as a stand-alone model and the lower part with the platform. In the upper part a Composite Actor, inside which each sequence diagram is encapsulated, can fire the Director as soon as an incoming token in its input port indicates that two application actors need to communicate. The Director then checks whether the precedence for that message is satisfied. The sequence diagram defines the messages' precedence so that messages between actors happen in the order the appear in the diagram. For instance considering the sequence diagram and message graph in Figure 4, communication defined by messages m1 and m2 must happen before m3 can happen, and m1, m2, and m3 need to happen before m4 and m5. Messages m4 and m5 are parallel, that is, the communication between actors can happen simultaneously. A new execution round of the diagram can start either as soon as the current round has finished, or when in pipelined mode, any time after the current round has started.

The precedence of a message is satisfied if the message has not been sent on that execution round and all its preceding messages has been sent on that round.

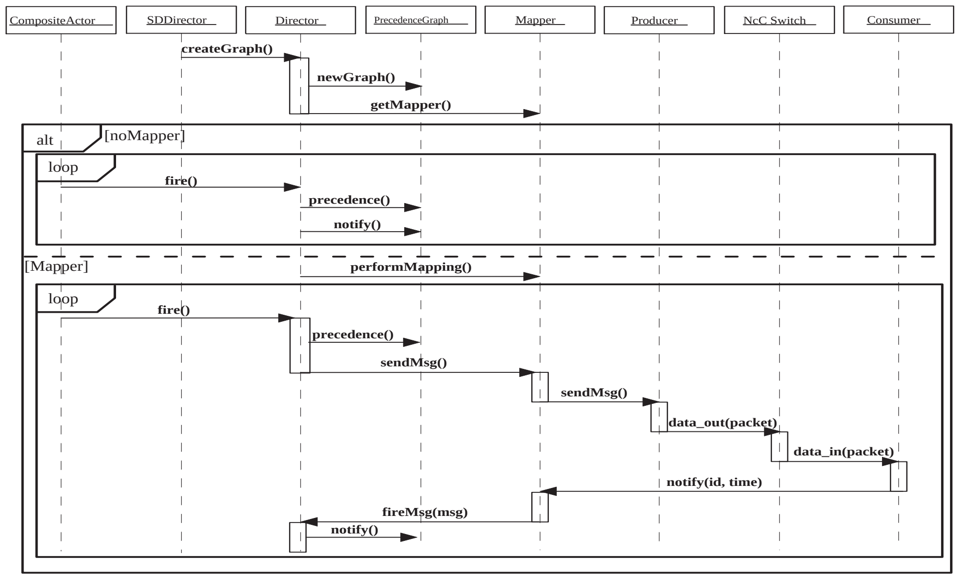

Fig. 3. System functionality illustrated as a sequence diagram

Alternative, optional, and loop combined fragments (CF) set some restrictions for that rule. If a message is inside an alternative CF, the precedence is satisfied, if the alternative condition is true. Likewise, if a message is inside an optional CF, the precedence is satisfied if that optional branch condition is true. Moreover, if a message is inside a loop CF, the precedence can be satisfied even if it has been sent on that execution round as soon as the loop option is still true. Otherwise the precedence of the message is not satisfied and the token at the input port of the CompositeActor is stored until the message's precedence is satisfied.

Getting back to Figure 3. If the message's precedence is satisfied, the Director notifies the PrecedenceGraph to update the status of the message graph, that is, change the status of a node containing the message to fired on that execution round. The execution of the sequence diagram is inside a loop and will continue until the simulation is stopped.

The lower part of the alternative fragment in Figure 3 depicts the application simulation with the platform. First, the Director asks the Mapper to perform the mapping of application actors onto a PE. Then if the Composite Actor has an incoming token, it fires the executive Director, which checks the message's precedence. If the precedence is satisfied, the Director asks the Mapper to communicate with the platform. The Director is unaware of the mapping of application actors on the platform, therefore, the Director does not communicate with the platform directly. Each PE corresponding a processor is divided into a Producer that sends packets and Consumer that receives them. Each lifeline in a sequence diagram represent an application actor that is mapped to a PE. A message between two lifelines in a sequence diagram indicates a packet sent by the PE's Producer, where the sending actor is mapped and received by the PE's Consumer, where the receiving actor is mapped.

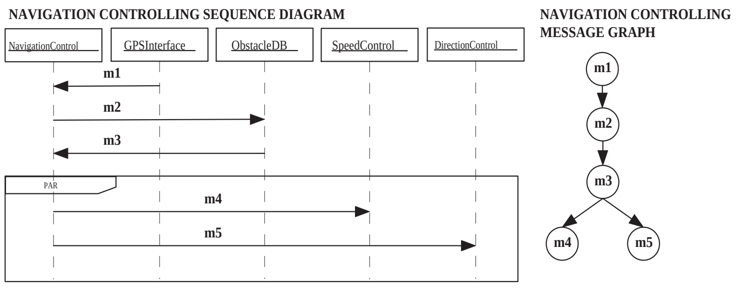

Fig. 4. A sequence diagram with its corresponding message graph

When the Mapper asks a Producer to send a message, following procedure happens: First, the Producer creates a packet containing the message as payload and the receiving PE's address among other header information. Because each message also describes the computational workload (the preCompTime parameter in Figure 2) imposed by the application task to a processor, the Producer delays the sending of the packet as long as the computation time indicates. After that the packet is passed to the NoC that delivers the packet to the correct Consumer.

When the Consumer receives a packet, it notifies the Mapper that the packet is delivered correctly. The Mapper can let the Director to ask the PrecedenceGraph to set status of a node containing the message into fired on that execution round.

4 Simulation Case Study

We simulated the autonomous vehicle application with High Definition Television (HDTV), Video Object Plane Decoder (VOPD), and MPEG4 decoder on the JOSELITO platform using 3x3, 4x4, 5x5, and 6x6 mesh topologies. For reference, we first simulated the application models without priority based scheduling, that is, using the first in first serve scheduling policy. Then we set higher priority to the critical application messages, that is, the messages used for controlling the vehicle's speed or direction. When a higher priority message arrives, the Producer sends it before lower priority messages.

Using the interactions described in Section 3.2, we simulated the application models in the Ptolemy II framework. We explored, how many timing constraint violations happens for critical messages using different scheduling policies and different platform configurations. A timing constraint violation means that a message exceeds its deadline.

Table 1 depicts how many per cent of messages violate their timing constraints and Table 2 illustrates the average violation time in milliseconds for all topologies and both communication and computation. A dash means no violation. From the tables we can see that usually a bigger network results in less and shorter violations (with a few exceptions). We used random mapping, which explains the anomalies in the results (especially first in first serve communication in 4x4 and priority based communication in 6x6 in Table 1. Moreover, priority based scheduling results in less violations and the violations are shorter in bigger topologies.

Table 1. Percentage of messages violating timing constraints

	Scheduling	**3x3**	**4x4**	**5x5**	**6x6**
Communication	First in first serve	7.7	1.2	3.6	3.0
Communication	Priority based	9.5	7.7	4.2	6.0
Computation	First in first serve	9.5	–	3.0	–
Computation	Priority based	2.3	–	–	0.6

Table 2. Average timing constraint violation in milliseconds

	Scheduling	**3x3**	**4x4**	**5x5**	**6x6**
Communication	First in first serve	1.91	0.00	0.02	0.02
Communication	Priority based	0.03	0.01	0.01	0.01
Computation	First in first serve	1.6	–	0.1	–
Computation	Priority based	1.4	–	–	0.86

5 Conclusions

In this paper we presented a model based approach for modelling heterogeneous applications and simulating them with a NoC based multicore platform. We described the system structure and functionality using UML composite structure and sequence diagrams and used MARTE profile to provide our application model with some real time features. We also presented a case study of how network configuration and scheduling policy affect the number of timing constraint violations in real time systems. We can conclude that our approach covers fundamental requirements of embedded system design, such as handling design complexity and heterogeneity. Moreover it sufficiently covers the design and validation of both communication and computation aspects of real time application models.

Acknowledgment

This work was supported by the SYSMODEL project (http://www.sysmodel.eu)

References

1. Booch, G., Rumbaugh, J., and Jacobson, I.: The Unified Modeling Language User Guide (1999)
2. Object Management Group: UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems http://www.omg.org/spec/MARTE/1.0/PDF/
3. Määttä, S., Indrusiak, L.S., Ost, L., Möller, L., Nurmi, J., Glesner, M., and Moraes, F.G.: Validation of Executable Application Models Mapped onto Network-on-Chip Platforms. *3rd International Symposium on Industrial Embedded Systems* (2008) 118–125
4. Määttä, S., Möller, L., Indrusiak, L.S., Ost, L., Glesner, M., Nurmi, J., and Moraes, F.G.: Joint Validation of Application Models and Multi-Abstraction Network-on-Chip Platforms. *International Journal of Embedded and Real-Time Communication Systems* vol. 1, no. 1 (2010) 85–100
5. Määttä, S., Indrusiak, L.S., Ost, L., Möller, L., Glesner, M., Moraes, F.G., and Nurmi, J.: Characterising Embedded Applications using a UML Profile. *International Symposium on System-on-Chip* (2009) 172–175
6. Määttä, S., Indrusiak, L.S., Ost, L., Möller, L., Glesner, M., Moraes, F.G., and Nurmi, J.: A Case Study of Hierarchically Heterogeneous Application Modelling Using UML and Ptolemy II. *International Symposium on System-on-Chip* (2010) To be published
7. Martin G.: UML for embedded systems specification and design: motivation and overview. *Design, Automation and Test in Europe Conference and Exhibition* (2002) 773–775.
8. The MathWorks Simulink® : http://www.mathworks.cn/access/helpdesk/help/toolbox/simulink/gs/brc3u5l.html
9. Mraidha, C., Tanguy, Y., Jouvray, C., Terrier, F., and Gerard, S.: An Execution Framework for MARTE-Based Models. *IEEE International Conference on Engineering of Complex Computer Systems* (2008) 222–227
10. Eker, J., Janneck, J.W., Lee, E.A., Liu, J., Liu, X., Ludvig, J., Neundorffer, S., Sachs S., and Xiong, Y.: Taming Heterogeneity – the Ptolemy Approach. *IEEE* vol. 91, no. 1 (2003) 127–144
11. Indrusiak, L.S. and Glesner, M.: SoC Specification using UML and Actor-Oriented Modeling. *Baltic Electronics Conference* (2006) 1–6
12. Ost, L., Moraes, F.G., Möller, L., Indrusiak, L.S., Glesner, M., Määttä, S., and Nurmi, J.: A simplified executable model to evaluate latency and throughput of networks-on-chip. *Symposium on Integrated circuits and system design* (2008) 170–175