TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

Sanna Määttä

Modelling Embedded Applications for On-Chip Multiprocessing Platforms

Julkaisu 968 • Publication 968

Tampere 2011

Tampereen teknillinen yliopisto. Julkaisu 968
Tampere University of Technology. Publication 968

Sanna Määttä

Modelling Embedded Applications for On-Chip Multiprocessing Platforms

Thesis for the degree of Doctor of Science in Technology to be presented with due permission for public examination and criticism in Tietotalo Building, Auditorium TB109, at Tampere University of Technology, on the 3rd of June 2011, at 12 noon.

Tampereen teknillinen yliopisto - Tampere University of Technology
Tampere 2011

ISBN 978-952-15-2579-7 (printed)
ISBN 978-952-15-2617-6 (PDF)
ISSN 1459-2045

ABSTRACT

The complexity of state-of-the-art embedded systems requires designers to focus on abstraction levels much higher than Register Transfer Level (RTL). As the designers are familiar with using RTL, system design often starts at levels of abstraction that are too close to implementation. Higher levels of abstraction substantially reduce the amount of details designers need to consider enabling complex system design in shorter time.

Modelling and simulation are essential methods in state-of-the-art embedded system design. In model-based design, a system model is the key element of the design process from the specification to the implementation. Modelling helps designers to manage complex systems, better understand the system under development, visualise a system, specify the structure and behaviour of the system, validate the system behaviour, and document the design decisions. Moreover, modelling reduces development time and costs.

This thesis describes a model-based approach for embedded application modelling and validation together with an on-chip multiprocessing platform. The aim of the work was to facilitate the programming of multiprocessing systems as well as to enable early system validation, design space exploration, and performance evaluation.

PREFACE

The work presented in this thesis has been carried out in the Department of Computer Systems at Tampere University of Technology, Finland during the years 2005-2010 and in the Institute of Microelectronic Systems at Technical University Darmstadt, Germany during the academic year 2007-2008.

I would like to thank my supervisor Prof. Jari Nurmi for his guidance and encouragement through all these years as well as for providing such a positive atmosphere in the Team Nurmi. Moreover, I would like to express my gratitude to my ”surrogate” advisor Dr. Leandro Soares Indrusiak, who took me under his wings during my stay in Germany and from that time on. Also, many thanks to Prof. Manfred Glesner who enabled my stay at MES. I am also grateful to the reviewers of this thesis Prof. Johan Lilius and Prof. Lisane Brisolara for providing constructive comments on this thesis.

I would like to thank my parents and brother as well as all my friends for their tolerance and support. I would also like to thank all the members of Team Nurmi for all kinds of co-operation and all entertaining (especially coffee table) conversations. Also, many thanks to Dr. Luciano Ost and M.Sc. Leandro Möller for their wonderful company in Germany and for never letting me swim alone...

This thesis was financially supported by the Tampere Graduate School in Information Science and Engineering (TISE), the SYSMODEL project, and Tampereen kaupungin tiederahasto, which are gratefully acknowledged.

Tampere, May 2011

Sanna Määttä

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

AMS	Analogue and Mixed Signal
ANSI	American National Standards Institute
CF	Combined Fragment
CSP	Concurrent Sequential Processes
CT	Continuous Time
DE	Discrete Event
DSP	Digital Signal Processor
EDA	Electronic Design Automation
ESL	Electronic System Level
FIFO	First In First Out
flit	Flow Control Digit
ForSyDe	Formal System Design
FSMD	Finite State Machine with Datapath
GPS	Global Positioning System
HDL	Hardware Description Language
HDTV	High-Definition Television

HetSC	Heterogeneous Specifications using SystemC
HW	Hardware
IP	Intellectual Property
ITRS	International Technology Roadmap for Semiconductors
JVM	Java Virtual Machine
LUT	Look-Up Table
MARTE	Modelling and Analysis of Real-Time and Embedded Systems
MESCAL	Modern Embedded Systems, Compilers, Architectures, and Languages
MoC	Model of Computation
MPEG	Moving Picture Experts Group
MPSoC	Multiprocessor System-on-Chip
MSC	Message Sequence Chart
NoC	Network-on-Chip
PAT	Payload Abstraction Technique
PBD	Platform Based Design
PE	Processing Element
PTRT	Packet Trailer Release Time
RT	Register Transfer
RTE	Real-Time and Embedded
RTL	Register Transfer Level

RSM	Repetitive Structure Modelling
SDF	Synchronous Data Flow
SoC	System-on-Chip
SPT	Schedulability, Performance and Time
SW	Software
SysML	Systems Modelling Language
TLM	Transaction Level Modelling
UML	Unified Modelling Language
VHDL	Very High Speed Integrated Circuit Hardware Description Language
VOPD	Video Object Plain Decoder
XML	Extensible Markup Language

1. INTRODUCTION

Embedded systems, such as consumer electronics, kitchen appliances, mobile phones, medical systems, and traffic control systems, can be found everywhere. Embedded systems are dedicated to perform a certain function. They are often portable devices and targeted to mass production. Most of them also interact continuously with their embedding environment. Thus, they need to be energy efficient, small, low cost, and operate in real-time [25] [69].

The diversity and complexity of applications, increased number of nonfunctional requirements, increased need for integration and networking, increased heterogeneity of products, increased flexibility, and shortened time to market characterise the state-of-the-art embedded systems and set requirements for their design [25].

State-of-the-art embedded systems are complex and heterogeneous, containing for example analogue and digital parts, as well as hardware (HW) and software (SW) [26]. The hardware-driven design flow poorly addresses the dominance of SW in embedded systems [32]. The integration of the HW and SW late in the design flow might lead to a system that does not work at all, does not work as it should, or does not meet its performance requirements. Therefore, it is important to model the whole system already early at the design process in order to validate its correct functionality and explore the design space in order to make necessary trade-offs for instance in terms of area, speed, and power consumption.

System complexity can be addressed by raising the level of abstraction above the Register Transfer Level (RTL) [50]. Higher levels of abstraction are usually closer to human way of reasoning. For instance, it is very difficult to catch

the system functionality when it is described as a schematic or by a Hardware Description Language (HDL). Furthermore, system level methodologies narrow the gap between application and hardware designers, since the application designer can validate the whole embedded system without detailed knowledge about system manufacturing [29].

The increasing complexity of embedded systems cannot lead to increasing design costs. The design cannot be based on for instance repetitive prototyping, since it is expensive and time consuming. More systematic and well-defined approaches are necessary in order to make systems meet their requirements [25].

1.1 Objective and Scope of Research

This thesis presents an approach for modelling and validating embedded applications together with on-chip multiprocessing platforms. This approach relies on executable specifications; however, without excluding (semi) formal description of the application. Even if the main focus is on embedded systems, the application modelling approach does not exclude general purpose applications.

The novelty of this approach is it being fully model-based using actor orientation and executable Unified Modelling Language (UML) sequence diagrams. Moreover, neither code generation nor HW emulation is required when simulating applications on a multiprocessing Network-on-Chip (NoC) platform.

The scope of the thesis is modelling at high levels of abstraction (that is, system level). Thus, this thesis considers neither the HW/SW division nor the process of system refinement into lower levels of abstraction. SW means exclusively application SW, operating systems or middleware are not addressed.

To summarize, the main contributions of this thesis are the following:

- An approach for joint modelling and validation of application and platform models using UML sequence diagrams, actor orientation, Java,

and the Ptolemy II framework

- Implementation of a mapping between the application and platform models
- Implementation of Processing Elements (PEs) to be connected to the NoC switches
- Design space exploration of application models mapped on different platform models and configurations
- Describing the application model using the UML profile for Modelling and Analysis of Real-Time and Embedded Systems (MARTE) and defining a few necessary extension to the MARTE profile
- Describing the platform model using the MARTE profile and its Repetitive Structure Modelling (RSM) notation

1.2 Thesis Outline

This thesis is a monograph, which contains some unpublished material, but is mainly based on the author's publications [64], [65], [66], and [67]. Publications [47], [83], and [84] are directly complementary, whereas publication [85] is used as a reference only.

This thesis can be divided in three parts: background theory (Chapters 2 – 3), closely related previous work not carried out by the author (Chapters 4 and 5), and the actual contribution of this thesis (Chapters 6 – 7).

Chapter 2 gives an overview of the concepts and terminology the author considered relevant for the rest of this thesis. Chapter 2 first presents different abstraction levels. Then, it introduces the terminology and concepts of system design and modelling, design space exploration, heterogeneity, and Models-of-Computation (MoCs). Moreover, Chapter 2 discusses the effect of the accuracy of a simulatable model on the simulation speed. Finally, Chapter

2 presents a few system level design and modelling languages, from which UML is discussed in more detail, since it is relevant for the rest of this thesis.

Chapter 3 gives first an overview of a few system level modelling frameworks and approaches. The emphasis of Chapter 3 is on academic approaches that are freely available. Chapter 3 then presents the Ptolemy II framework, within which the application modelling approach presented in this thesis is implemented.

One of the contributions of this thesis is the joint validation of application and platform models. Therefore, Chapter 4 presents the models of NoC interconnects used as a part of the platform models (the PEs are presented in Chapter 6). Chapter 4 presents first the reference interconnect, HERMES, since the more abstract models are based on it. Then, Chapter 4 presents three models of NoC interconnects, RENATO, JOSELITO, and BOÇA that are all implemented on system level yet still presenting different level of accuracy. The work presented in this Chapter is not carried out by the author.

Chapter 5 describes the previous work on creating executable application models using UML sequence diagrams within the Ptolemy II framework. The work presented in this Chapter is not carried out by the author.

Chapter 6 describes the contribution of this thesis and explains the application modelling approach for on-chip multiprocessing platforms.

Chapter 7 presents a few case studies in which the application modelling approach is demonstrated. Finally, Chapter 8 draws conclusions and presents some future work.

2. RISING ABOVE RTL: SYSTEM DESCRIPTION AT HIGHER LEVELS OF ABSTRACTION

According to the International Technology Roadmap for Semiconductors (ITRS) the cost effective and reliable design of such a complex system on a single chip that is possible with the current technology would need a fifty-fold increase in design productivity. Hence, the specification, design, verification, and design space exploration of complex systems require a level of abstraction that is above the RTL [50].

Despite the clear need for system level design, neither industry nor academia has been able to sufficiently formalise a system level design technology or methodology [29]. In fact, the term system level does not even have a clear or unified definition [91] [96]. Usually system level is described to be a level above RTL including both HW and SW [50] [96].

2.1 Levels of Abstraction

As the definition of the term system level is not unified, neither is the terminology describing the rest of the abstraction levels: Rabaey et al. divide the abstraction levels in digital circuit design from the lowest level to the highest in device, circuit, gate, functional module, and system levels [93], whereas Gajski et al. call them circuit, logic, processor, and system levels [29]. At the device level, the basic component of the models is a transistor, whereas circuit level components are standard cells consisting of them. Further, at the gate level (or logic level according to Gajski et al. [29]), components are logic gates and flip-flops building register transfer components. At the functional

module level (or processor level) systems are described using components such as adders that compose bigger systems (such as processors) that are used at the system level [29] [93].

RTL is a low level of abstraction and is usually associated with HDLs, such as Very High Speed Integrated Circuit Hardware Description Language (VHDL) or Verilog [34]. At the Register Transfer (RT) level, a design has clocked behaviour described in terms of data transfers between storage elements (such as registers) [40].

Transaction Level and Electronic System Level (ESL) are not exactly levels of abstraction, even though both of them are placed in a level above RTL [34] [59] [99]. Transaction Level Modelling (TLM) is a high level modelling style that enables the separation between the implementation of the communication functionality and the functional units that use the communication functions. Especially SystemC [41] advocates the use of TLM, in which the communication functionality is modelled as channels and the SystemC modules can request a transaction by calling the interface functions of other modules [34].

ESL is an intermediate design phase above RTL (or rather a new ”buzzword” in the Electronic Design Automation (EDA) industry). The raise of the abstraction level has been a trend during the past few years; therefore, the EDA industry has had room for a new definition [59] [99]. ITRS defines ESL to be a level above RTL that consists of behavioural and architectural levels. At the behavioural level the system functions have not yet been partitioned into SW and HW, whereas at the architectural level this partitioning has been already done [49].

2.2 System Design and Modelling

Benini and de Micheli define a system to be a collection of components providing a useful service when operating together [9]. System design is the process of refining a functional specification of a system into the final system

implementation [91]. However, the gap between the system specification and implementation is too large to be closed in a single step. Thus, closing the gap requires a successive stepwise refinement of system models [29].

A model is a simplification of reality. Building models helps us to better visualise and understand the system we are designing. Especially complex systems cannot easily be comprehended without an abstract model of them [10]. An abstract model is a source of nondeterministic behaviour and the implementation process makes it a deterministic system [56]. Hence, according to Selic, the main purpose of models is to understand the aspects of a complex system before constructing it [98]. Systems models can be created many ways, most of the ways being ad hoc. However, without modelling it is likely to either build a malfunctioning system or fail building it at all [10].

Modelling is an accepted engineering technique [10]. It is the process of creating or generating models. According to ITRS, modelling aims at supporting technology development and optimisation as well as reducing development time and costs [51]. Modelling also enables the analysis of existing systems, for instance when it is too impractical to experiment with the actual system [98].

Models have a central role in model-based design, model-integrated engineering [100], and model-driven architecture [75]. All of them depict the same idea using models; how it is called depends on the user. For instance, the Mathworks has adopted the concept of model-based design, in which a system model is used as an executable specification throughout the development process [70]. Furthermore, according to Sztipanovits and Karsai, model-integrated engineering uses models as a backbone for the development of computer-based systems. Though, they claim that model-integrated computing uses models in a more general sense than model-based design. In model-integrated computing models do not only capture for instance the SW architecture but also its environment [100]. The Object Management Group's (OMG) model-driven architecture is targeted mostly to SW domain, since it considers a platform-independent model to be a base of the application development [75].

Regardless of how the use of models is called, usually a single model is not sufficient for the whole system design process. Hence, we need different models at different levels of abstraction [29] [91]. A model serves as a specification of the desired functionality to be implemented in the next lower level of abstraction and as a description for the validation of the design decisions through simulation or analysis [29].

2.3 Design Space Exploration

The early stages of any system design process can be characterised by incompleteness and exploration. Therefore, system level design deals with incomplete and inconsistent information and the evaluation of different design decisions based on it [91].

Plantin and Stoy emphasise the importance of models to capture the existing knowledge of the system, even if the knowledge is incomplete. The incomplete system level models are the base of making trade-offs between available system solutions [91]. The process of finding out the trade-offs by testing different solutions and optimising the design under a set of constraints is called design space exploration [29]. Hence, the purpose of system design is to realise a desired functionality while satisfying design constraints that delimit the design space and making trade-offs between system performance versus costs [9].

The Y-chart method, depicted in Figure 1 emphasises the separation of architecture and application models, which facilitates the design space exploration. Different methods can be used for designing and refining the application models and platform architecture separately until the mapping of the application models onto different architectures for performance evaluation [58] [63].

Design space exploration is easier and faster at the system level than at the lower levels of abstraction. Too detailed simulation, such as cycle accurate simulation, is not very suitable for design space exploration of for instance state-of-the-art embedded systems. As their design space is large, simulators

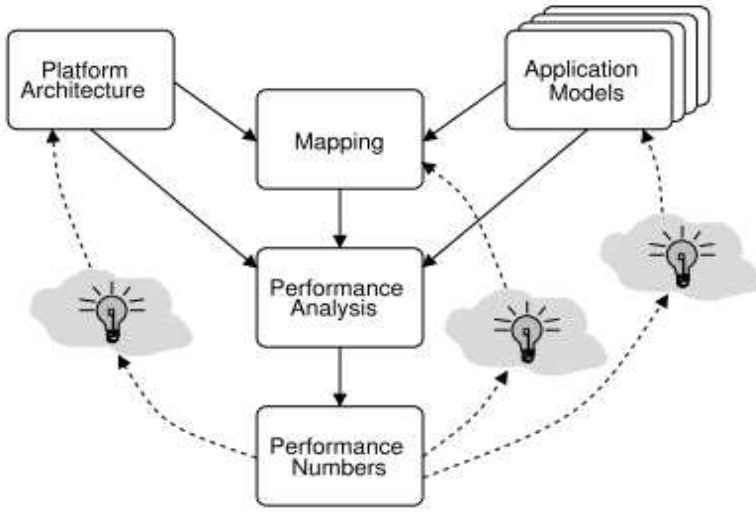

Fig. 1. *Y-chart [63] ©IEEE, 1999*

get too complicated and are too slow [90]. Even though at the system level designers need to trade-off between the simulation speed and accuracy, it is not always necessary to sacrifice accuracy for speed, as shown for instance in [43].

2.4 Model Accuracy versus Simulation Speed

Modelling systems at the higher levels of abstraction reduces the number of objects designers need to consider by an order of magnitude [29]. Moreover, at the system level, designers can exploit all the freedom without making any design decisions, such as the HW/SW partitioning [56]. However, when using higher level models, designers need to trade-off between the simulation speed and accuracy.

Ideally, the designers should start the system design process by using fast but not very accurate system models, and stepwise refine the models towards more accurate ones at the expense of the simulation speed. However, very often the RTL model exists before the more abstract models. Nevertheless, this enables the back-annotation of for instance timing information to the more

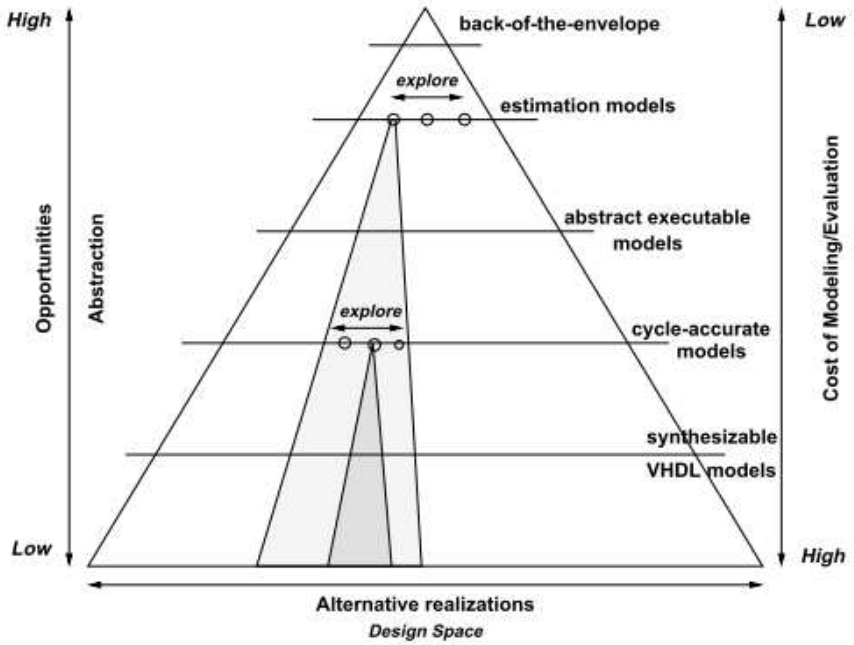

Fig. 2. *The abstraction pyramid depicts the trade-off between the modelling effort and level of detail [57] ©A.C.J. Kienhuis, 1999*

abstract models making them also more accurate.

Figure 2 depicts the abstraction pyramid. The bottom of the pyramid presents the whole design space, whereas the top of the pyramid is the designer's original idea about the system. The abstraction pyramid addresses various modelling issues: cost of modelling and evaluation, opportunity to change, level of detail, and accuracy. On top of the pyramid, system models are very abstract, meaning that their level of detail is low. However, the possibility to explore various design choices is high, because the more detailed the system description is, the more difficult and costly it is to change it [57].

When refining the system model towards the bottom of the abstraction pyramid, the modelling and evaluation costs increase: the number of different design choices increases making the system simulation and design space exploration more time consuming [57].

2.5 Heterogeneity and Models-of-Computation

State-of-the-art embedded systems are often heterogeneous containing for instance analogue, digital, and mixed-signal parts as well as hardware and software [20] [33]. The various domains of heterogeneous systems require modelling and design using different MoCs.

According to Keutzer et al. a MoC refers to a mathematical model that describes the system behaviour in terms of specifying the semantics of computation and concurrency [56]. The definition of a MoC has been often refined to apply for instance to the context it is used in: For example, Eker et al. define a MoC to be a framework that specifies the interaction of components in a subsystem covering both the flow of data and the flow of control between them [26], whereas Jantsch and Sander describe a MoC to represent time and the semantics of communication and synchronisation between processes in a process network [54].

Even though MoCs have been described in detail [53], compared [62], and categorized [24], it is difficult to model or implement a system using various MoCs. However, it is beneficial to choose the right MoC and to understand the impact of different MoCs on the design and implementation choices [53]. MoCs provide the designers with useful properties, such as determinism and deadlock protection [54]. Therefore, choosing a MoC has a significant impact on the quality of the system design [13]. Moreover, an appropriate MoC speeds up the simulation of the system since only the issues relevant to that particular MoC need to be simulated [54].

MoCs can be either timed or untimed. Timed MoCs (such as Discrete Event (DE) or Continuous Time (CT)) have temporally ordered events, whereas untimed MoCs (such as Synchronous Data Flow (SDF)) have only partially ordered events [15] [24] [59].

The DE MoC is often used in HDL simulators, synchronous languages, and in general in all time oriented models of systems (such as communication networks or digital hardware). The CT MoC is suitable for systems requiring

continuous time, such as analogue circuits, mechanical systems, or embedded systems interacting with continuous environments. The CT MoC supports continuous time mixed-signal modelling because a model in the CT domain can include both continuous signals and discrete events. The SDF MoC suits for modelling simple dataflow systems without complicated flow of control, such as signal processing systems. The execution order of actors in the SDF domain (and the number of produced and consumed data samples of each actor) is statically determined prior to execution. This results in minimal execution overhead, bounded memory use, and the occurrence of no deadlocks [15].

The modelling languages and frameworks need to support system modelling and design using multiple MoCs. Whereas SpecC supports various MoCs (such as Concurrent Sequential Processes (CSP), Finite State Machine with Datapath (FSMD), and DE) [30], the SystemC standard simulation kernel supports only the simulation of DE models [41]. This poorly addresses the heterogeneity of the state-of-the art embedded systems.

Heterogeneous specifications in SystemC (HetSC) [36], Analogue and Mixed-Signal extensions to SystemC (SystemC-AMS) [102], SysteMoC [27], and SystemC kernel extensions [86] [87] enable the simulation of hybrid and mixed-signal SystemC models. The latter approach modifies the SystemC kernel directly, whereas HetSC, SystemC-AMS, and SysteMoC are built on top of SystemC. Also Verilog-AMS [2] and VHDL-AMS [39] enable the analogue and mixed-signal description and simulation.

Besides languages, also some frameworks, such as the Formal System Design (ForSyDe) [94] or Ptolemy II [26] (described in more detail in Chapter 3) enable heterogeneous system modelling using multiple MoCs.

2.6 System (Level) Design Languages

System design languages need to capture not only the behaviour of the HW but also to enable the description of the system SW [34] [96]. Sangiovanni-Vincentelli points out that the current industrial approaches of system level de-

sign address either HW or SW but not both. HW approaches, such as VHDL or Verilog, have poor or no support for SW, whereas SW languages, such as C or C++ are not able to describe concurrency and time [96].

C and C++ are popular among SW engineers for creating executable specifications. However, although they execute fast, they lack the notion of concurrency and time, which are supported by for instance the American National Standards Institute (ANSI) C based SpecC [30] or C++ based SystemC [41]. One of the main goals of the development of SystemC was to enable system level modelling of systems that include both HW and SW [34]. SpecC is a system level design language that facilitates the specification and design of digital embedded systems containing both HW and SW [22] [30].

HDL based system level languages, such as System Verilog [42], preserve all the features of the underlying language [42], but might be hard to use by SW engineers [96]. For instance, System Verilog is built on top of the Verilog 2001 standard [42]. While preserving all the Verilog features it also supports system level and object oriented modelling as well as the specification, design, and verification of HW [1] [42]. However, it does not enable the modelling of system SW.

The proportion and complexity of SW in embedded systems have increased in recent years [16]. This increasing dominance of SW in embedded systems needs to be addressed with other methods than using traditional HDLs. One potential approach is UML [68].

2.7 UML

UML is a standardised modelling language for visualising, specifying, constructing, and documenting especially software intensive systems. Hence, one of the ideas behind the development of UML was to bring some stability to the object oriented marketplace in forms of a unified and mature modelling language and tools [10].

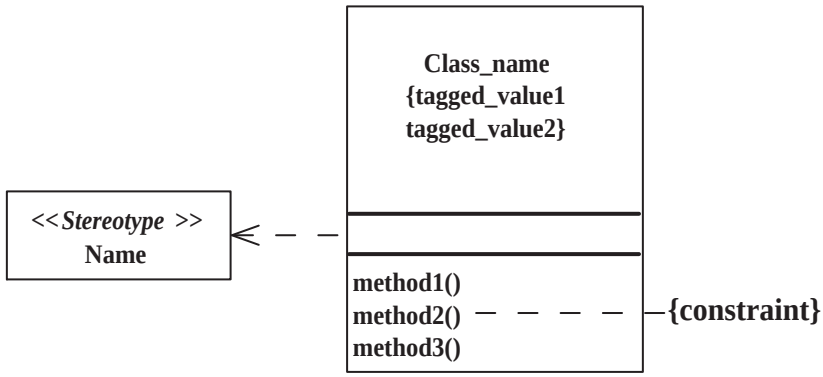

Fig. 3. *UML extension mechanisms: stereotypes, constraints, and tagged values*

UML can be applied to various types of systems, domains, methods, or processes [4]. The growing interest of the embedded systems and real-time communities in UML, UML's extension mechanisms, and support for object orientation among others are considered the strengths of the language. The shortcomings, such as the lack of platform models or mapping methodology [68], are later addressed for instance by the MARTE profile [78].

2.7.1 UML Profiles

A metamodel defines UML syntax and semantics. UML can be extended by defining new metaclasses, extending the existing metaclasses, or defining new constraints or more precise semantics. UML extension mechanisms (that do not alter the UML metamodel) are stereotypes, constraints, and tagged values [10] [80], whose syntax is shown in Figure 3.

Stereotypes extend the vocabulary of UML and allow the creation of new building blocks. Furthermore, constraints extend the semantics of UML building blocks and enable the definition of new rules, whereas tagged values extend the properties of UML building blocks. The extensions can be arranged as profiles. A profile is a set of stereotypes, constraints, and tagged values that customise UML to a specific domain [10] [80].

Sangiovanni-Vincentelli points out that UML profiling can be considered ei-

ther a strength or a weakness of the language. The over 300 existing UML profiles will for sure customise UML for various needs while they most likely also overlap [96]. Well known UML profiles include the Systems Modelling Language (SysML) [76], a profile for Schedulability, Performance and Time (SPT) [79], and the MARTE profile [78].

The UML profile for MARTE customises UML for model-driven development of real-time and embedded systems [78]. Already MARTE's predecessor, the SPT profile extended UML with periodic tasks, schedulable objects, timing, and concurrency aspects [79].

The MARTE profile supports the specification, design, and validation stages of embedded system design. MARTE extends UML to provide constructs for modelling Real-Time and Embedded (RTE) SW applications, high-level RTE HW, and their non-functional properties. Moreover, MARTE provides designers with necessary extension units in order to address performance and schedulability analysis of RTE systems [78].

The MARTE profile is organised into a set of packages, such as foundations, design model, and analysis model. The foundation package contains basic elements for modelling non-functional properties, timing, and general resources. Moreover, an 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. Finally, 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 [78].

Several approaches use the UML extensibility through profiles in order to customise UML for embedded system design. For example, Kukkala et al. introduce a UML 2.0 profile for embedded system design [60], whereas Arpinen et al. explore the embedded SW platform modelling using that profile [5]. Moreover, Brisolara et al. show the benefit of using the MARTE profile when defining the same system with and without the profile [12]. Boulet et al. and

Cuccuru et al. demonstrate the expressiveness of the MARTE profile's notation for describing repetitive structures [11] [17].

2.7.2 Repetitive Structure Modelling using MARTE

The MARTE profile has also high level modelling constructs to describe (parallel) data computations using several computation units. The RSM notation provides a compact way to depict the regularity of such repetitions of structural elements connected via a regular connection pattern [17] [78].

The RSM notation extends the UML multiplicity concept in order to enable to specify a shape of the repetitive structure elements. The RSM modelling aspects can be used to model HW platforms, applications, and the allocation of the application onto the HW platform [17] [78].

Figure 4 illustrates a 3x3 torus network using the RSM notation. The $<<$*HwCommunicationResource*$>>$ stereotype (which inherits from the $<<$*HwResource*$>>$ stereotype) describes NoC switches. The repetitionSpaceDependence attribute defines the relative position of an adjacent instance for each element (a vector $\{1,0\}$ means that the adjacent instance is one hop along the x-axis and zero hops along the y-axis, while $\{0,1\}$ is zero hops along the x-axis and one hop along the y-axis). The modulo attribute describes whether the topology is cyclic (modulo is true) or not (modulo is false). In the case depicted in Figure 4, a true modulo value means a torus topology, whereas a false modulo value would mean a mesh topology.

RSM is necessary for describing massive regular parallelism: it makes the model more compact, readable, and maintainable. However, it only handles regular topologies [17].

2.7.3 UML Diagrams

The earlier version of UML, UML 1.1, included nine different diagrams: class, object, component, and deployment diagrams depict the system struc-

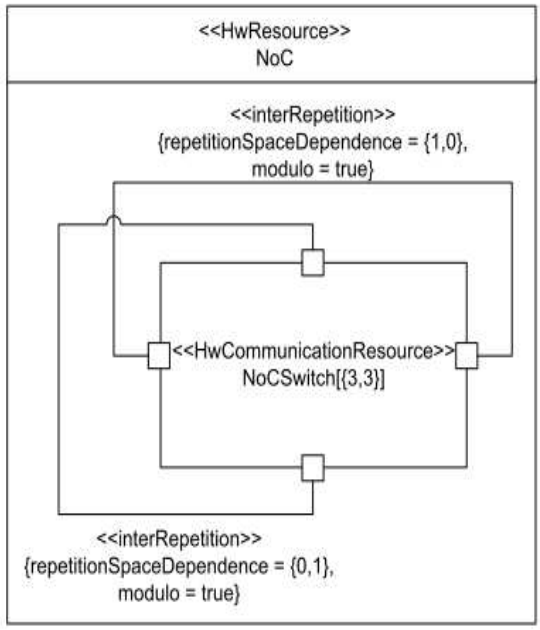

Fig. 4. *A 3x3 torus topology described using the RSM notation*

ture while use case, activity, statechart, sequence, and collaboration diagrams depict the system behaviour [10]. Later, in UML 2.0 the number of diagrams has increased to thirteen: composite structure and package diagrams are the additional structural diagrams, whereas timing and interaction overview diagrams are behavioural. Also the collaboration diagram is nowadays called communication diagram and the statechart diagram state machine diagram. Moreover, some of the behavioural diagrams (sequence, communication, timing, and interaction overview diagrams) are subcategorised into interaction diagrams [77]. The sequence diagram is described next in more detail, since it is used to create the executable application models as explained in Chapters 5 and 6.

UML sequence diagram is an interaction diagram and depicts the time ordering of messages. Booch defines an interaction to be a behaviour that consists of a set of messages exchanged among a set of objects. Moreover, a message specifies a communication between the objects [10]. Figure 5 illustrates a

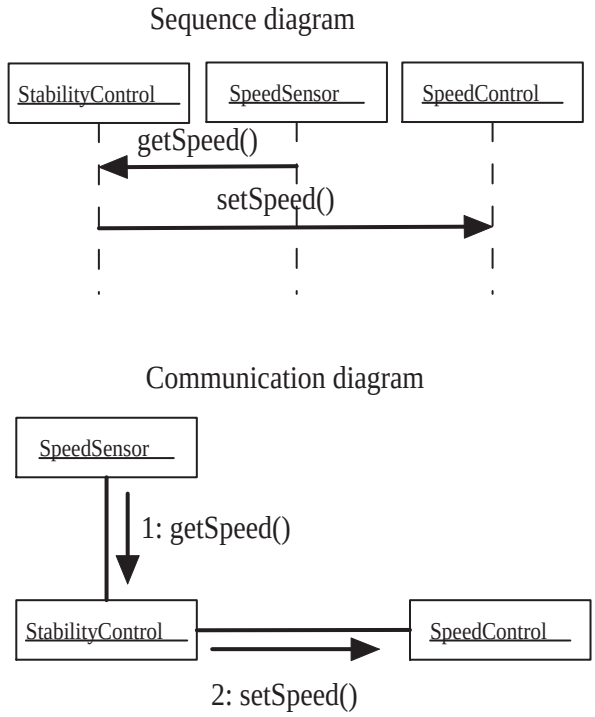

Fig. 5. *UML sequence diagram and the corresponding communication diagram*

sequence diagram that has three lifelines and two messages. In the sequence diagram, each object has a lifeline that describes its existence over a period of time. The messages represent the communication between the objects.

Some of the UML diagrams are almost overlapping, as can be seen from Figure 5. The lower part of Figure 5 depicts the corresponding communication diagram. While the sequence diagram emphasises the temporal ordering of messages, the communication diagram emphasises the structural organisation of objects. These two diagrams are semantically equivalent [10]: as can be seen from Figure 5 both diagrams contain the same objects and messages, and no information is lost when converting one to the other.

A sequence diagram is one variant of Message Sequence Charts (MSCs). MSCs are a visual formalism used to capture system requirements during the early design stages. MSCs specify scenarios that describe interaction patterns between processes or objects [35].

However, as much as the visual notation of UML is expressive and the diagrams have underlying formalism, the UML diagrams can be misinterpreted. For instance Graaf et al. have noticed that in many companies UML is used for drawing instead of modelling and the interpretation of the drawings (not always being syntactically correct or consistent) might get obscured [32]. Therefore, executable models are necessary.

3. PTOLEMY II: A SOFTWARE FRAMEWORK FOR ACTOR ORIENTED EXPERIMENTING

System design at various levels of abstraction requires support from design automation tools, but they are hardly mature enough for system level design [50]. Moreover, system level design tools lack universal applicability [53] and instead of creating a unified flow from the system level specification to the implementation, they are merely a bunch of unlinked tools requiring informal techniques and human intervention during the design flow [8].

Several methods and frameworks exist for embedded system design. Platform Based Design (PBD) aims at reducing system design costs by reusing applications and architectures [28] [56]. The ForSyDe framework aims at heterogeneous system modelling using multiple MoCs and the development of transformational design refinement methodology for embedded systems and Systems-on-Chip (SoCs) [94] [95].

Polis is an approach for the design and verification of control dominated reactive systems. Polis is a full design methodology and a design framework for HW/SW co-design of embedded system [7]. Metropolis continues the work done within Polis. Metropolis is a design environment for heterogeneous systems designed to support PBD [8] [96] [101]. The goal of the development of Metropolis has been to obtain a unified environment for unambiguous system presentation at various levels of abstraction [101].

MILAN is a model-based simulation framework facilitating the design and optimisation of embedded systems [6]. The Modern Embedded Systems, Compilers, Architectures, and Languages (MESCAL) project aims at creating a disciplined approach to produce reusable architectural platforms [72].

Artemis and SESAME are frameworks aiming at efficient design space exploration of heterogeneous embedded systems architectures at multiple abstraction levels [89] [90]. The Ptolemy II framework [21] [26] is presented next in further detail.

3.1 The Ptolemy Project

The Ptolemy project of the University of California at Berkeley aims at modelling, simulating, and designing of concurrent, real-time embedded systems [21]. The first outcome of the project was the Gabriel software for signal processing at 1986. Gabriel included a code generator to produce efficient assembly code for Digital Signal Processors (DSPs) as well as a HW/SW co-simulator. Early 1990s, the project announced Ptolemy Classic, which is a modelling environment supporting multiple MoCs. At 1996 the project started working on the Ptolemy II software framework [13].

Although the Ptolemy II framework used some of Ptolemy Classic's capabilities, it also introduced several new features: data and domain polymorphism, new MoCs, a new visual editor (called Vergil), actor oriented classes and subclasses, and the use of the Extensible Markup Language (XML) for representing models [13].

Whereas Gabriel was implemented in Lisp and Ptolemy Classic in C++, the Ptolemy II's implementation language was changed to Java due to its capability to for instance built-in threading and building user interfaces. Even though Java has several advantages, which is why it was chosen to implement the Ptolemy II framework in the first place [13], the fact that it runs over the Java Virtual Machine (JVM) is also one of its biggest shortcomings. The simulation of Ptolemy II models may be slow in some cases and also dependent on the size of the Java heap space.

One of the aim of the Ptolemy project is to create an environment for actor oriented experimenting [21].

3.2 Actor Orientation and Hierarchical Heterogeneity

An actor is a component that communicates with its environment by sending and receiving data tokens (that encapsulate messages) via ports using channels [13] [61]. The actor model derives from the mathematical model of concurrent computation [37] and later from the formal models of concurrency [3].

The development of actors is influenced by the concept of objects in the Simula language [3]. Agha defines that each actor has an independent thread of control and the actors communicate via asynchronous message passing [3].

Actor orientation is the theoretical foundation behind the Ptolemy II framework. Thus, Lee at al. have further refined the concept of actor orientation: Even though the actors are still parallel, they can share the thread of control and the message passing does not necessarily need to be asynchronous. Moreover, each actor can run in its own thread or the whole system can run sequentially in a single thread [61].

Ptolemy II actors belong to a domain, which defines both a receiver and a director. The receiver implements the communication of data tokens and can be either a QueueReceiver (containing a First In First Out (FIFO) queue) or a Mailbox (a FIFO with a capacity of one). The director is an object that defines the interaction semantics of components. Thus, a director controls the communication and execution of actors in the domain defining when the actors communicate and update their internal state (thus following the rules defined by a MoC) [13] [14] [26].

Ptolemy II actors can be either atomic or a composition of other actors (called composite actors). Atomic actors are primitive and cannot contain other actors. Composite actors contain other composite actors, atomic actors, or both. Opaque composite actors have a local director, whereas transparent composite actors do not and their execution follows the rules of the executive director [13] [26]. Figure 6 depicts executive and local directors as well as a composite actor and atomic actors (Actor1, Actor2, and Actor3).

Lee et al. consider the most important advantage of actor orientation to be

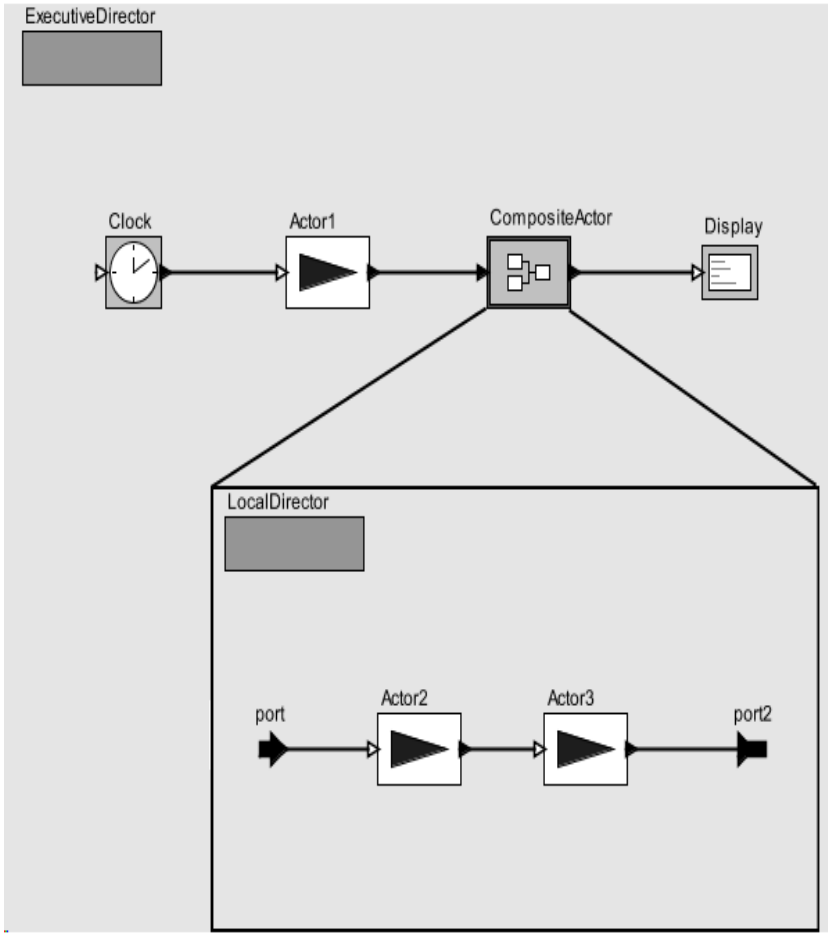

Fig. 6. *Vergil workspace showing composite and atomic actors and executive and local directors*

the use of MoCs in the actor interaction [61]. A MoC defines the details of scheduling and communication and how the actors are related if they are related. The actors themselves do not do that, unlike objects in object oriented design. This increases reusability and the model can be easier analysed and understood [26]. The use of MoCs in actor interaction is necessary in order to comprehend the heterogeneity of the state-of-the-art embedded systems.

The Ptolemy II framework, as targeted at the design of heterogeneous embedded systems, enables the integration of various MoCs by using hierarchi-

cal heterogeneity. That is, the composite actors can encapsulate subsystems, whose components' interaction a MoC controls, as seen in Figure 6. Furthermore, the MoC also turns the subsystem into a component that can be used in other systems. This hierarchical heterogeneity enables the analysis and modification of subsystems without affecting the overall systems [26].

3.3 Advantages of the Ptolemy II Framework

Various Ptolemy II frameworks characteristics have been useful for the application modelling approach presented in this thesis:

- The Ptolemy II framework is open source and can be modified and extended. For instance, as described in further detail in Chapter 5, the visual editor has been extended with a UML editor, in which UML sequence diagrams can be drawn.
- The Ptolemy II framework defines a large, domain polymorphic component library from which components can be selected to the workspace of the visual editor.
- The Ptolemy II framework enables the definition of more components. For instance, the Ptolemy II composite actor has been modified in order to enable it to encapsulate a UML sequence diagram.
- One of the objectives of the Ptolemy II is to enable modelling using different MoCs. Various directors supporting different MoCs can be selected from the component library and the implementation of new ones is ongoing work.
- The Ptolemy II framework enables the creation of new directors. For instance, the implementation of two additional directors was necessary in order to define total and partial order execution semantics for the UML sequence diagrams.

- One of the Ptolemy II framework's advantage over other system level design methods and tools is the visual editor, called Vergil. Vergil and the component library make the building of models fast and quite effortless.
- Using the Ptolemy II framework, both application and platform models can be described and simulated in the same model using the visual editor.
- Ptolemy II can also be set up with Eclipse software development environment [23] making the programming of user defined components convenient.

4. MORE AND LESS ABSTRACT MODELS OF NETWORK-ON-CHIP INTERCONNECTS

A platform has various definitions depending on the domain of application [96]. In the context of PBD, a platform is an abstraction that covers several possible lower level refinements [97] or a library of both computational and communication components that can be composed to a design at a certain level of abstraction. The selected parameterised components form a platform instance [96]. Jantsch and Tenhunen define a NoC platform to include not only the communication infrastructure, but also middleware and operating system communication services, and a design methodology and tools to map applications onto a NoC platform [55].

Traditional von Neumann uniprocessor architectures will not meet the power, performance, and cost requirements of the state-of-the art and future systems [56] [81]. Thus, multiprocessor systems are needed in order to increase the performance without increasing the power consumption of a processor [81]. When programming multiprocessing system, we try to exploit parallelism in order to achieve increased performance. The increase in parallelism causes also increasing communication overhead [88]. Therefore, the interconnection network has a central role in state-of-the-art multiprocessing systems.

The design and verification of the inter-task communication of applications is hard if the interconnect is a bus based system due to the unpredictability of the communication performance [55]. Even though on-chip buses are cost efficient and have high performance, they do not scale when the number of communicating components increases. This makes their behaviour unpredictable. The concept of a NoC replaces ad hoc wiring with a more structured

approach and increases predictability [18] [55].

In this thesis, a platform denotes a NoC based multiprocessing platform that consists of one of the models of a NoC interconnect described in the next subsections and Processing Elements (PEs) connected to local ports of the NoC switches.

The rest of this Chapter describes first a RT level reference model of a NoC infrastructure called HERMES and then three different system level models based on it. The modelling and implementation of the NoC interconnects are not carried out by the author and are therefore not a contribution of the thesis. The models of the NoC interconnects are presented here, because the system level models RENATO, JOSELITO, and BOÇA are used as the platform interconnects in the case studies described in more detail in sections 7.1 – 7.5.

4.1 HERMES: The RTL Reference Model

HERMES is an infrastructure for low area overhead packet-switching NoC. A switch is the basic element of the network and can be connected to four other switches and to a local Intellectual Property (IP) core implementing especially 2D mesh topologies [74].

Each HERMES switch contains routing control logic and five bi-directional input ports that establish the connection to four adjacent switches and to the local core. Each port uses input buffering for incoming packets. The arbitration logic gives a priority to the port, which has been granted routing longest time ago. The routing control logic implements the routing strategy, arbitration logic, and a packet switching algorithm with wormhole switching mode [74].

The routing strategy chosen for HERMES is XY routing [74]. In the XY routing flow control digits (flits) are first routed in the X direction until the right X-coordinate is found and then in the Y direction until the right Y-coordinate is found.

HERMES switches use wormhole switching because it allows efficient use of buffers, resulting in less buffer space needed than for instance in cut-through switching. Moreover, wormhole switching is able to multiplex a physical channel into more than one logical channel [19]. However, in HERMES switches, only one logical channel is used of one physical channel in order to reduce the switch complexity and cost [74]. In wormhole switching, packets are divided into entities called flits [19].

The flit size in HERMES can be parameterised, but the number of flits in a packet is fixed to be $2^{flit\ size\ in\ bits}$. The first and second flits of the packet form the header and contain the address of the target switch of the packet and the size of the payload in flits respectively. The payload is carried by the rest of the flits, which do not contain any routing information. Therefore, the payload flits must follow the same path as the header flits. Each switch has a unique address expressed as X and Y coordinates corresponding the horizontal and vertical position of the switch in the network. This representation facilitates the XY routing of the packet [74].

HERMES aims at small design size and a NoC switch with low latency. The HERMES switch is described in VHDL and prototyped on FPGA Virtex II. A NoC configuration having a fixed flit size of 10 bits (2 bits for control and 8 bits data), buffer size 8, and 5x5 mesh topology has 500 Mbits per second peak performance at 25 MHz operation frequency. With this configuration, the switch area is 555 Look-Up Tables (LUTs) [74].

4.2 RENATO: Modelling the HERMES Switch Using UML Interactions

RENATO is an actor oriented system-level NoC model written in Java within the Ptolemy II framework. RENATO is based on the UML interactions that describe the functionality of the HERMES switch [46]. The interactions are illustrated as UML sequence diagrams in Figure 7. These interactions only visualise the RENATO model's behaviour and are not associated with the ex-

ecutable UML sequence diagrams for the application modelling explained in Chapters 5 and 6.

The leftmost sequence diagram in Figure 7 describes an arbitration request of an input buffer in a switch. An input buffer requests arbitration if it has received a packet. The buffer sends the packet's header flit to the router controller and waits for a response (denoted by a synchronous message in the sequence diagram). The controller requests arbitration from the arbiter for the packet the header flit belongs to [46].

The arbiter handles all incoming requests and grants arbitration to one of the buffers requesting it according to an arbitration scheme, such as round robin. If the arbitration is granted, the arbiter sends the flit to the router that determines which of the 5 output ports the flit should be sent to. After the routing is done, the router controller verifies that the chosen output port is free. If it is free, the input buffer can establish the connection to the output port and send the flit, otherwise the connection is refused and the input buffer has to request for arbitration again [46].

The rightmost sequence diagram in Figure 7 describes the transmission of a flit between two adjacent routers. The controller receives the flit from the local input buffer and determines to which output port it belongs to. After sending the flit to the correct output port, the controller waits for an acknowledgement. After a positive acknowledgement, the controller removes the flit from the source router's input buffer, whereas a negative acknowledgement causes resending of the flit [46]. Since there is only one condition in the last two interactions of the rightmost sequence diagram, the alt combined fragments could also be replaced with opt combined fragments.

Originally, RENATO was untimed. In order to increase its accuracy, timing information extracted from the cycle accurate HERMES model was annotated to RENATO's interactions. The simulation results of RENATO have been compared to the results of HERMES using various traffic scenarios and NoC configurations. For a long lasting traffic, the error is up to 10 per cent. This can be considered as a good result since the actor oriented model is based on

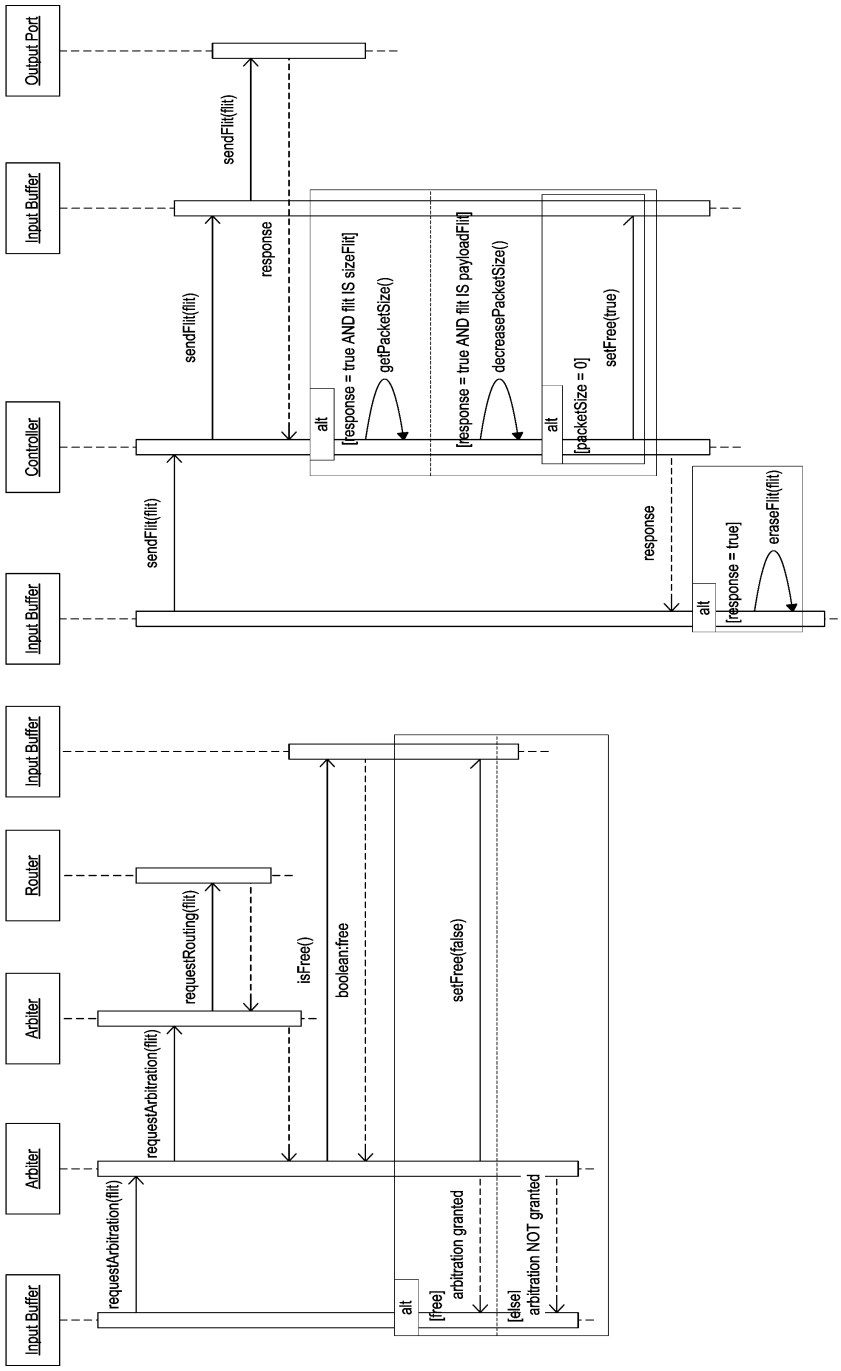

Fig. 7. *RENATO's UML interactions [46] ©IEEE, 2008*

only the interactions and works without a synchronising clock signal [46].

4.3 JOSELITO: Number of Model Details Versus Simulation Speed

JOSELITO is a simplified NoC model that uses Payload Abstraction Technique (PAT) and allows the performance evaluation of a NoC using both simulation and analytical methods. Even if JOSELITO uses the same UML interactions as RENATO, it is more abstract. The PAT decreases the simulation time of JOSELITO in comparison with RENATO, because it reduces the number of communication events by abstracting the payload and therefore omitting the flit by flit payload forwarding. Whereas RENATO forwards all packets flit by flit, JOSELITO defines the packet as a header and a trailer and uses an analytical method for denoting the transmission time of the payload, which is actually the packet trailer release time (PTRT) [85].

Using JOSELITO, three different transmission scenarios are possible: blocking free, header blocking, or header and trailer blocking. In the blocking free scenario no resource conflicts occur. Therefore, the latency and throughput can be measured with no loss of accuracy. The packet header can be blocked if the input buffer of the next router on the header's way is full or the output port of the current router is used by another packet. If the header cannot proceed further within the time the trailer is released and reaches the header, also the trailer is blocked. Both blocking situations increase the possibility of the loss of accuracy in the latency evaluation, because either the header or trailer may block other packets [85].

JOSELITO performs in average 2.3 times faster than RENATO in 88 per cent of the test cases. Moreover, when compared to the results of HERMES, JOSELITO has 5.26 per cent error in latency and 0.1 per cent error in throughput [85]. Moreover, the difference in the average energy consumption between HERMES and JOSELITO is close to zero per cent (0.001247 mJ) [82].

4.4 BOÇA: Analytical Calculation of Communication Latency

BOÇA is a fully analytical model and the most abstract of the NoC models used in the simulation cases used in Chapter 7. A BOÇA model is parameterised with the network topology and it uses that information to analytically calculate the latency of each communication that happens. BOÇA disregards any interference between different traffic. Although being a very simplified model, BOÇA enables fast simulation and early analysis of the effect of the network topology or different application mappings [67].

When simulating an application model on the abstract platform models, hundreds of actors are simulated. As the simulation case in section 7.2 indicates, there is a clear need for more abstract models, such as BOÇA, at least as long as the system is modelled and simulated within the Ptolemy II framework.

5. UML EXTENSION TO THE PTOLEMY II FRAMEWORK

This Chapter covers first how UML sequence diagrams can be encapsulated inside the Ptolemy II's composite actors and then how the execution semantics are assigned to the sequence diagrams in order to create executable UML models. The work and ideas presented in this Chapter are not carried out by the author; the starting point of the contribution of this thesis was the already existing idea and work of simulating UML sequence diagrams within the Ptolemy II framework.

Most attempts of using UML as a system level language use either static analysis, code generation, or manual transformation into an executable form. Static analysis provides designers with information they can use when building models for instance with SystemC or implementing systems, whereas code generation requires model transformation. A manual transformation is however a slow and error-prone task. Even though UML lacks execution semantics, actor-orientation has the coexistence of multiple execution semantics as its major feature. Therefore, it is reasonable to have a joint approach of UML and actor orientation where the shortcomings of one of them are compensated by the strengths of the other [45].

5.1 Encapsulating UML Sequence Diagrams Inside Composite Actors

Figure 8 depicts a UML sequence diagram with one asynchronous and one synchronous message (messages x and y respectively). When the sequence

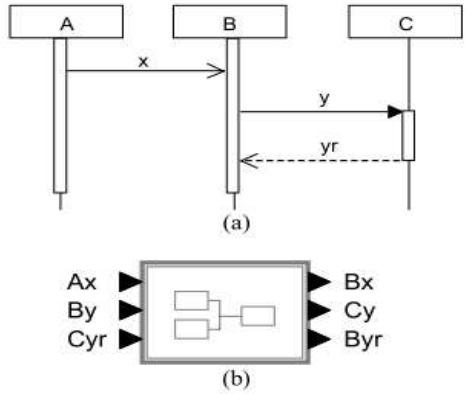

Fig. 8. *Encapsulating a UML sequence diagram inside a composite actor [44]* ©*IEEE, 2006*

diagram is encapsulated inside a composite actor, input and output ports are created for each message so that one input and output port is created for the asynchronous message (ports Ax and Bx in Figure 8), whereas two input and output ports are created for each synchronous message (ports By, Cy, Cyr, and Byr in Figure 8).

The input ports are connected to the source of the message while output ports are connected to the receiver of the message, as depicted in Figure 9 [44]. This can be used for the creation of generic communication patterns that are reusable in various application domains in a similar way that Gamma defines in [31].

Each lifeline of the UML sequence diagram represents an application actor: As seen in Figures 8 and 9, lifeline A represents the actor ”source A”, lifeline B the actor ”filter B”, and lifeline C the actor ”adaptation C”. Moreover, the messages between the lifelines in the sequence diagram contain the data tokens that the actors send to each other. The sequence diagram defines the sequence of the messages between actors and the director associated with the sequence diagram enforces their ordering [44].

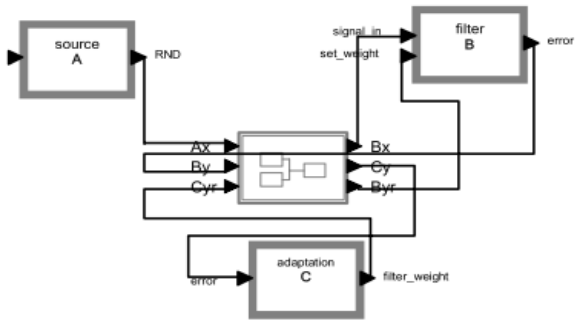

Fig. 9. *Actors connected to the input and output ports of a composite actor [44]* ©*IEEE, 2006*

The Ptolemy II framework's visual editor, Vergil, was extended to support the description of UML sequence diagrams as seen in Figure 10 [44]. In the Figure, all directors on the left, the lifelines, messages (both synchronous and asynchronous), and the Combined Fragments (CFs) belong to the extension: the directors PO SD Director and Lin SD Director, lifelines, messages, and the parallel CF (par) were added within the work described in [44]. The directors Total Order, Partial Order, optional, loop, and alternative CFs (as well as the functionality of all the CFs) were added later, within the work described from Chapter 6 onwards.

5.2 Simulating UML Sequence Diagrams within Executable System Models

Indrusiak et al. describe how to directly validate UML models by combining them with executable system models in [44], [45], and [48]. The Ptolemy II framework's hierarchically heterogeneous modelling style that allows the

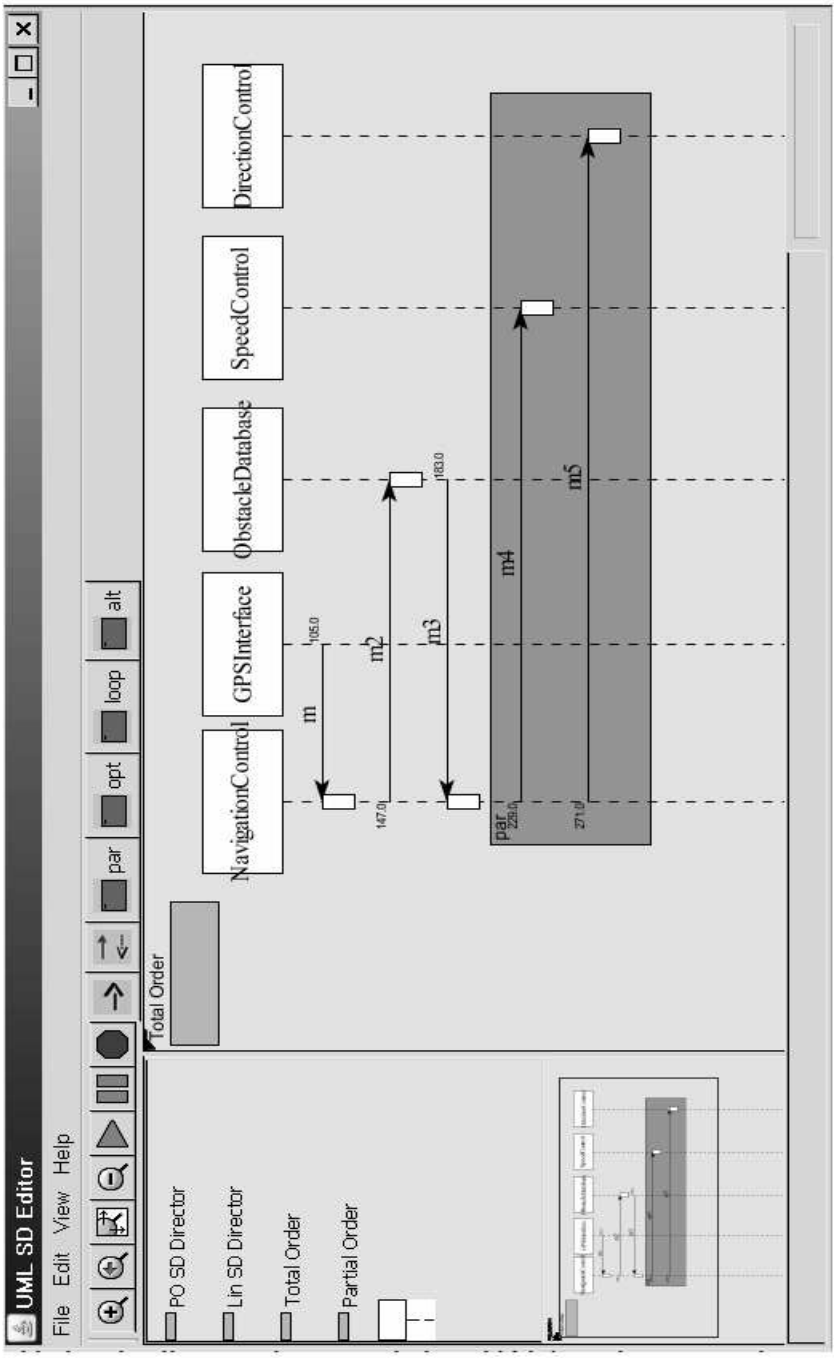

Fig. 10. *UML editor extension to the Ptolemy II framework*

inclusion of different execution semantics for each hierarchical level enables the execution of UML sequence diagrams.

When UML sequence diagrams are encapsulated inside composite actors, in addition to the creation of the input and output ports, the sequence diagrams need to implement a firing scheme so that it can be simulated with other actors and commanded by a director. The firing scheme depends on the precedence of messages (the precedence of the message occurrence specification within a lifeline). The formalism behind sequence diagrams does not formally determine the precedence between messages that are not sent and received by the same lifeline. However, in that case, a total ordering of messages can be achieved by taking into account the messages' position on the y-axis of the lifeline, since a lifeline has a temporal dimension. That is, time passes when going downwards along the lifeline [45].

The firing scheme determines in which order the tokens arriving at the input ports of the composite actor containing the sequence diagram are forwarded to the respective output ports of the actor. On each firing, the sequence diagram triggers all messages that have tokens in their respective input port and whose all preceding messages have been triggered. The firing scheme of each opaque composite actor is modelled by a director. It is possible to create directors implementing different firing schemes, such as total order (director Lin SD Director in Figure 10) or partial order (director PO SD Director) [45].

In Ptolemy II, a director defines the execution semantics of actors. The Lin SD Director defines the messages' firing order to be dependent on the messages' position on the y-axis: a message that is higher on the axis is triggered before a message located lower on the axis. The PO SD Director maintains a total order within each lifeline, but a partial order among messages on different lifelines [45]. This means that in case two messages are sent and received by completely different lifelines, their execution order does not depend on their position on the y-axis (that is, on the lifeline). But if they have at least one common lifeline as a sender of receiver, their position of the y-axis defines their firing order.

The simulation of the same sequence diagram with both total and partial order directors show that the input buffers can be slightly smaller in partial order. However, the implementation of total order is simpler and the precedence can be determined in advance (can be simply a round robin arbiter), whereas partial order requires dynamic analysis of the precedence [45].

6. SIMULATING EMBEDDED APPLICATIONS TOGETHER WITH ON-CHIP MULTIPROCESSING PLATFORMS

According to ITRS, to increase the performance of the SW execution requires heterogeneous parallel processing using various application-specific processors for system functions. Moreover, ITRS emphasises the importance of executable specifications written in a formal language, since they reduce the verification effort by allowing automated verification early at the design process at high levels of abstraction. However, more research and solutions for both heterogeneous parallel processing and executable specifications are needed in the near future [50].

The programming environment for multicore systems should be application-centric [38]. Even though the application programmers should be aware of the characteristics of the hardware to better exploit the capabilities of it [88], they should be protected from as many HW features as possible [38]. A compiler can parallelise sequential code, but this is not the most optimal solution. In order to achieve highest possible performance of a multiprocessor system, the programmers need to change to a parallel programming model [81].

This Chapter presents the approach for modelling and validating embedded application models together with on-chip multiprocessing platforms. This approach addresses the need for application-centric parallel programming methods by using actor orientation and UML. The approach enables joint validation of the application and platform models by simulation; therefore, it addresses the need of executable specifications and awareness of the underlying HW without excluding any (semi) formal definition of the application mod-

els. Moreover, the simulation within the Ptolemy II framework provides the means for creating heterogeneous system models.

6.1 An Approach for Embedded Application Modelling for on-Chip Multiprocessing Platforms

The previous work explained in Chapter 5 enabled the simulation of application models within the Ptolemy II framework. However, it did not allow the simulation of the application models mapped on platform models.

Following terminology is used from now on:

- A message represents a communication between two actors. Moreover, within this work, it also represents application tasks. Messages are parameterisable and the delay parameter set by the application designer indicates how long each task would be executed by a real processor.
- Communication latency of a message is the time that the packet(s) corresponding to that message spends on the network. The latency consists of the model time during which the packets are either routed between switches or waiting for routing in a buffer. The communication latency of messages depends on network congestion, data size of the corresponding packet, and the mapping (that is, if the sending and receiving actors of the packet are mapped to nodes that are close to each other or not).
- Computation latency is the time that a task would execute on a processor. Since the level of abstraction of the work this thesis describes is system level, the processors are modelled as PEs. The PEs take into account the latency by delaying the delivery of the messages until the computation time is over.
- Firing or executing a message happens when the composite actor encapsulating a sequence diagram has an incoming data token that indicates

that two application actors want to communicate. The message firing or execution means that a PE creates a packet containing the message, delays the packet according to the computation delay, and sends it to the receiver.

- Active actor can initialise communication.
- Passive actor never initialises communication but can respond to a received communication.
- Firing period indicates how often active actors are fired (that is, how often they initialise communication).

The next three sections describe the application model, platform model, and the mapper in more detail.

6.2 Application Modelling

Using the application modelling approach presented in this thesis, the application models are described using UML sequence diagrams. Even though for instance UML state machine or activity diagrams would also enable the description of application behaviour, sequence diagrams have a clear advantage over them: the time ordering of messages.

Even though the sequence diagram is the only diagram that can be simulated, other diagrams are not excluded: as a sequence diagram present only a part of the application behaviour, a composite structure diagram is used to illustrate the structure of the whole application model and all connections between application actors. Moreover, the composite structure diagram is also extended with the ordering of the communication between actors, as explained later in this Chapter in section 6.7.

The application modelling process starts with defining the application behaviour using sequence diagrams. Figure 11 depicts an autonomous vehicle

application model that is used as an example to demonstrate the potential of this modelling approach. The application model consists of nine sequence diagrams: navigation controlling, pressure controlling, speed controlling, obstacle recognition 1, obstacle recognition 2, ultrasonic sensing, Global Positioning System (GPS) sensing, speed sensing, and vibration sensing. The vehicle can determine its speed, location, vibration, and whether there are obstacles ahead so that it can avoid collisions. The vehicle can adjust its direction and speed according to the obstacles shown in the obstacle databases and its tyre pressure according to the information received from the vibration sensor.

After the sequence diagrams are defined, they are drawn to the UML editor that extends the Ptolemy II's visual editor. In the sequence diagrams depicted in Figure 11 lifelines represent application actors and messages between the lifelines communication between actors. The sequence diagrams are encapsulated inside composite actors and input and output ports are created for each message of the sequence diagram. Each application actor is connected to these input and output ports, as shown in Figure 12, which depicts the speed controlling sequence diagram and its encapsulating composite actor connected to application actors.

A director defines the execution semantics of the sequence diagrams maintaining the order of the messages. Two different directors are implemented: total order and partial order. They implement either total or partial ordering of messages and are based on the Lin SD Director and PO SD Director described in Chapter 5.

Even though this application model is a synthetic example (the sequence diagrams do not depict any real-life vehicle), it can be parameterised to have realistic delays and workloads. For instance, the messages carrying and processing images are bigger and have longer delays in comparison with messages carrying sensor readings or control information.

The application can run either independently from a platform as a stand-alone model or if the platform and mapper exist, mapped on the platform.

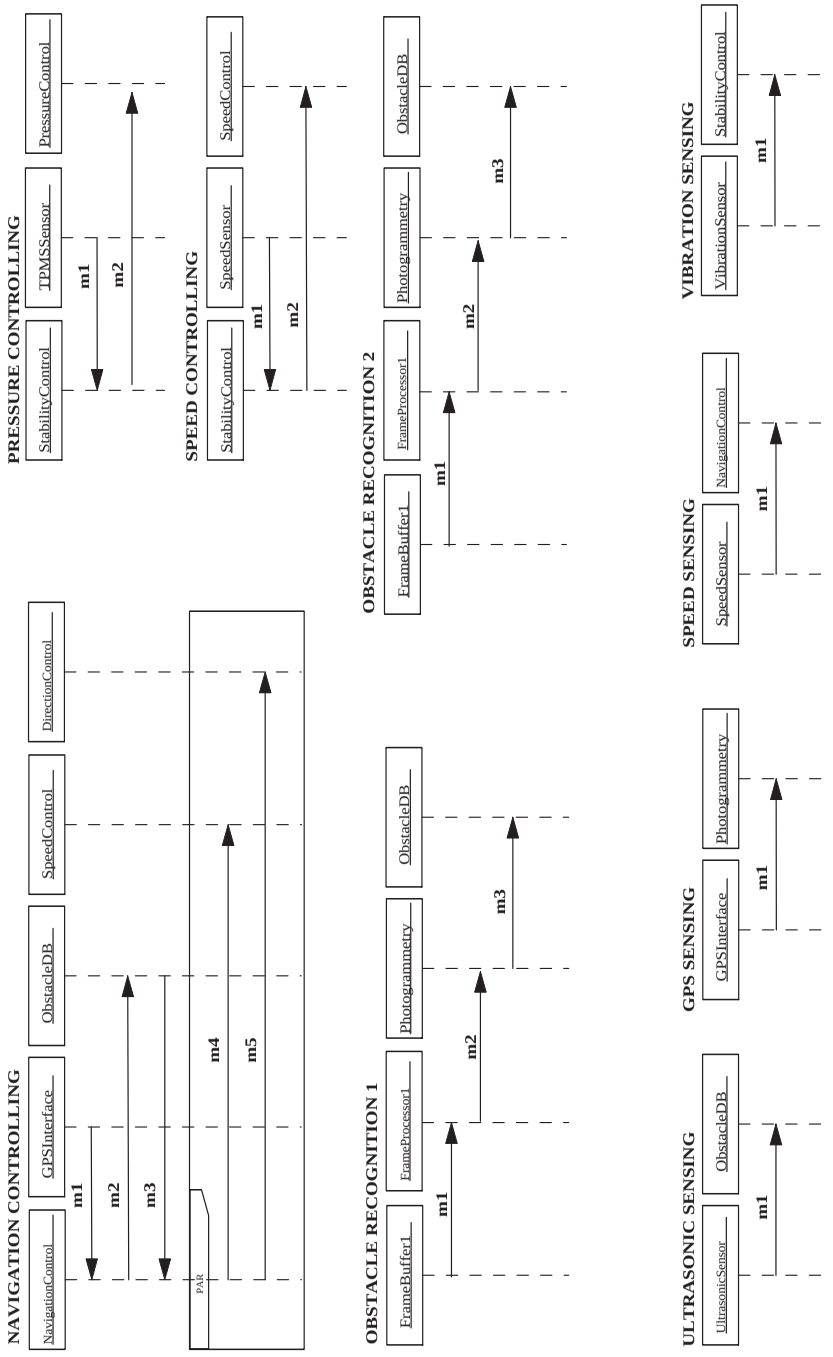

Fig. 11. *Sequence diagrams describing an autonomous vehicle application*

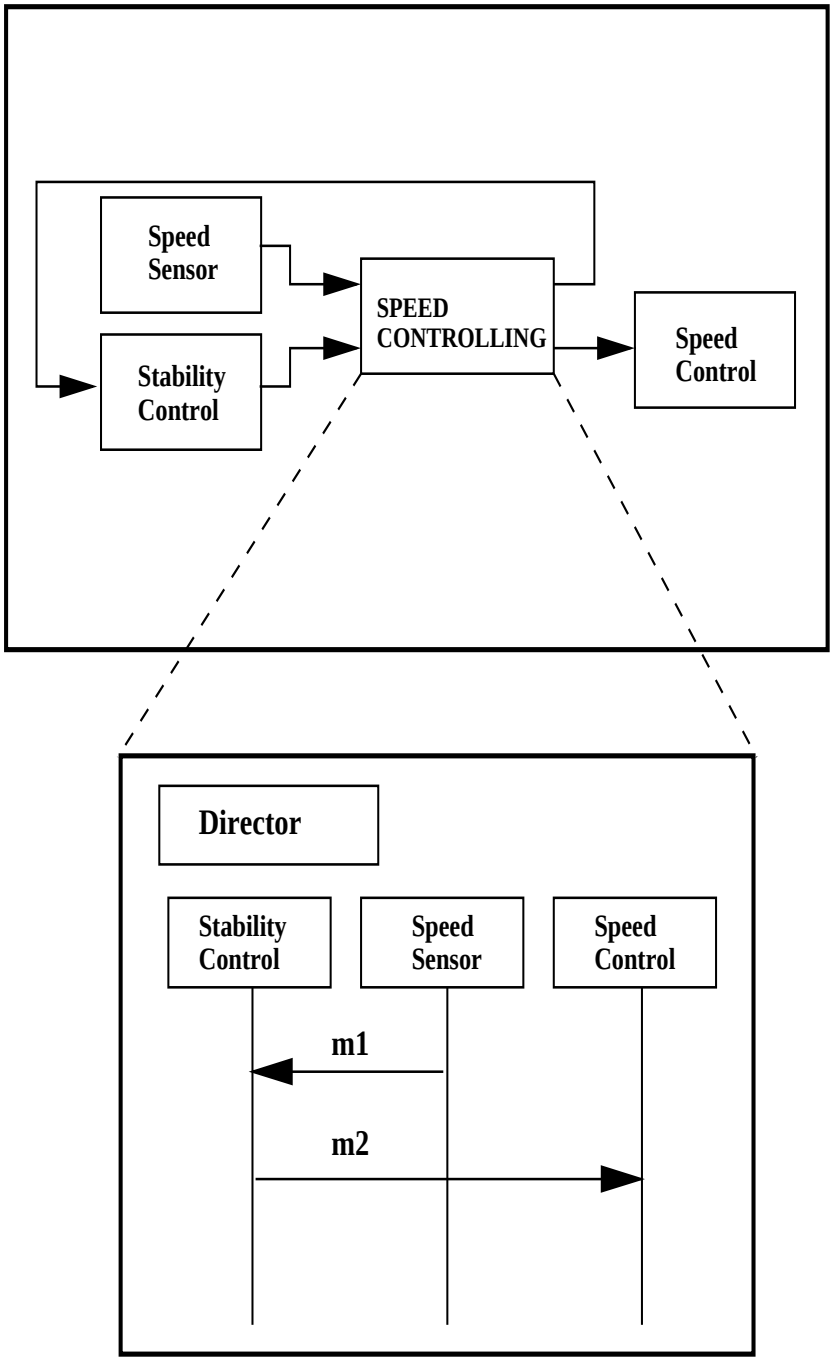

Fig. 12. *Speed controlling sequence diagram connected to application actors*

6.3 Platform Modelling

As defined in Chapter 4, a platform denotes a NoC based multiprocessing platform that consists of one of the models of a NoC interconnect and PEs connected to local ports of the NoC switches. The PEs described next were implemented within the contribution of this thesis.

The PEs are high level models of processors. Together with a model of a NoC interconnect they form a platform model. Each PE is attached to a NoC switch. The PEs actually consist of two actors: a producer and a consumer. The producer is sending packets, whereas the consumer is receiving them.

The PEs also include the network interface: they are responsible of creating packets, which carry the messages. The PEs buffer the messages until they are packetized and sent in case the PE is still executing a previous message. The PEs can also handle message priorities, executing higher priority messages before lower priority messages.

Due to the different packet forwarding methods of the different NoC models, different PEs were implemented. The RENATO PEs send packets flit by flit, whereas the JOSELITO PEs abstract the payload, thus sending only the packet header and trailer. The BOÇA PEs are sending packets flit by flit but the flits are not sent over the NoC but rather forwarded to the consumer without a delay.

In this thesis, the use of the MARTE profile and especially its RSM notation was expanded to cover also the visualisation of the platform model (even though the platform modelling is otherwise not a contribution of this thesis). Using the RSM notation the platform models can be described in a compact manner. This reduces the effort of drawing all NxN network nodes and processing elements, since all of them are similar. Figure 13 illustrates a NxN platform model, in which the NoC switches are arranged as a regular mesh topology.

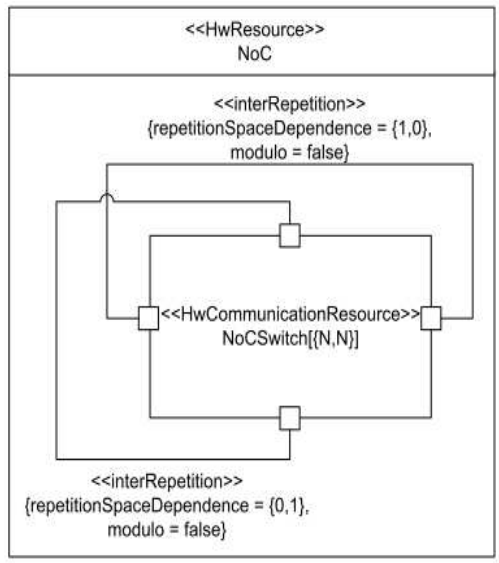

Fig. 13. *Platform description using the RSM notation*

6.4 Mapping Application Models on Platform Models

In order to establish a connection between the application model and the platform, a mapper needed to be implemented. A mapper maps each lifeline of the sequence diagrams on a PE of the platform. Since lifelines represent application actors, each actor is therefore mapped on a PE. Moreover, each application actor has tasks, which are then executed by the PE the actor is mapped on.

The mapping has a significant impact on the communication costs; therefore, the implementation of the mapper is flexible enabling different mapping algorithms and heuristics [66]. In the case studies presented in Chapter 7, usually only a random mapping strategy is used (that is, each lifeline is mapped randomly to one of the PEs and the mapping cannot be changed after it is done for one simulation). The purpose of the work this thesis describes was not to evaluate different mapping heuristics. Hence, Ost et al. have implemented several heuristics (greedy incremental, simulated annealing, and taboo search) in [83] and [84], which complement the author's work.

6.5 Enabling Joint Validation of Application and Platform Models

This subsection first presents some basic principles how the application and platform models can run in the same model. Then it describes the interaction between all the model elements in further detail.

6.5.1 Basic Principles

The following principles hold when encapsulating sequence diagrams inside Ptolemy II composite actors [48] [66]:

- The composite actors encapsulating sequence diagrams (actors SD1 and SD2 in Figure 14) must have a director (D2 and D3 in Figure 14) that defines the execution semantics of the sequence diagram.
- The director of the composite actor ensures that the delivery of the messages follows the ordering defined by the sequence diagram.
- The composite actors encapsulating sequence diagrams must have input and output ports to allow the communication with other actors.
- For each asynchronous message, one input and an output port are created to the composite actor that encapsulates a sequence diagram. For synchronous messages, two additional ports must be created to represent the return message.
- The receivers of each port can be configured with the desired buffering behaviour (for instance bounded or unbounded FIFO buffer or a mailbox).
- Each lifeline of a sequence diagram (lifelines L1 to L5 in Figure 14) represents one application actor (actors named with letters a to e in Figure 14).

- Each message (messages M1 to M6 in Figure 14) between lifelines represents the communication between two actors of the application model.

In order to jointly simulate the application and platform models, the application actors need to be mapped on the platform. In other words, the mapper assigns each lifeline of the sequence diagrams on a PE of the platform (PE1 to PE9 in Figure 14) [67]. Thus, each lifeline must be mapped on a PE; however, a PE may have more than one (or zero) lifelines mapped on it. For instance, considering the actors and PEs in Figure 14, the mapper maps each actor (actors a to e in Figure 14) to one of the processing elements (PE1 to PE9 in Figure 14) according to a given mapping heuristic (as explained in more detail in section 6.4).

6.5.2 Interaction Between the Different Elements of the System

Figure 15 illustrates the interaction between different elements of the system model. These interactions are described as a UML sequence diagram; however, this diagram is not encapsulated inside a composite actor and not executed like the sequence diagrams describing the application model. The sequence diagram in Figure 15 only depicts how the simulation of the system proceeds within the Ptolemy II framework.

The UML sequence diagrams describing the application model (see Figure 11) are first transformed into acyclic, directed message graphs. The SDDirector (from which the total and partial order directors inherit) calls the createGraph() method of all the total and partial order directors (depicted as the Director lifeline in Figure 15). The directors then pass this request to the precedence graph classes that create the precedence graphs of the messages according to their ordering in the sequence diagrams. Therefore, each total and partial order director is associated with one precedence graph. An example of a sequence diagram and its corresponding message graph can be seen in Figure 16.

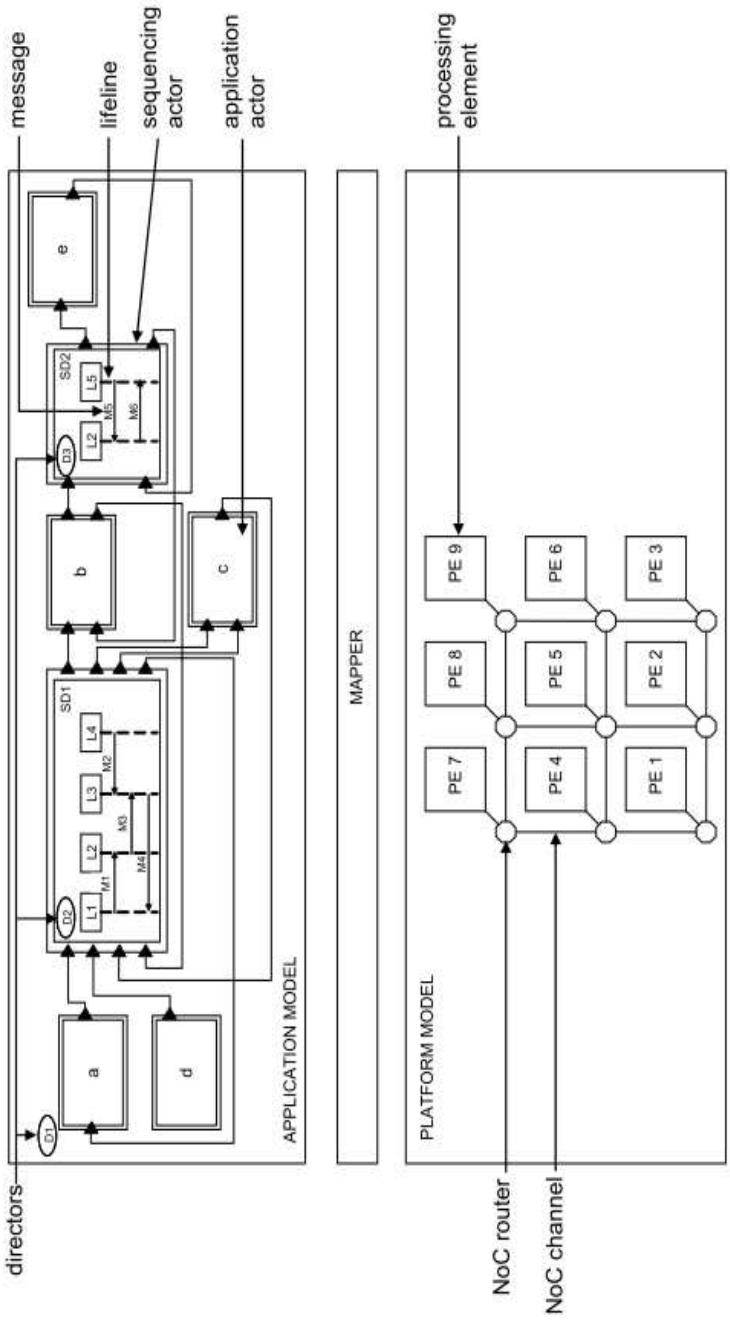

Fig. 14. *Application and platform models [67] ©IGI Global, 2010*

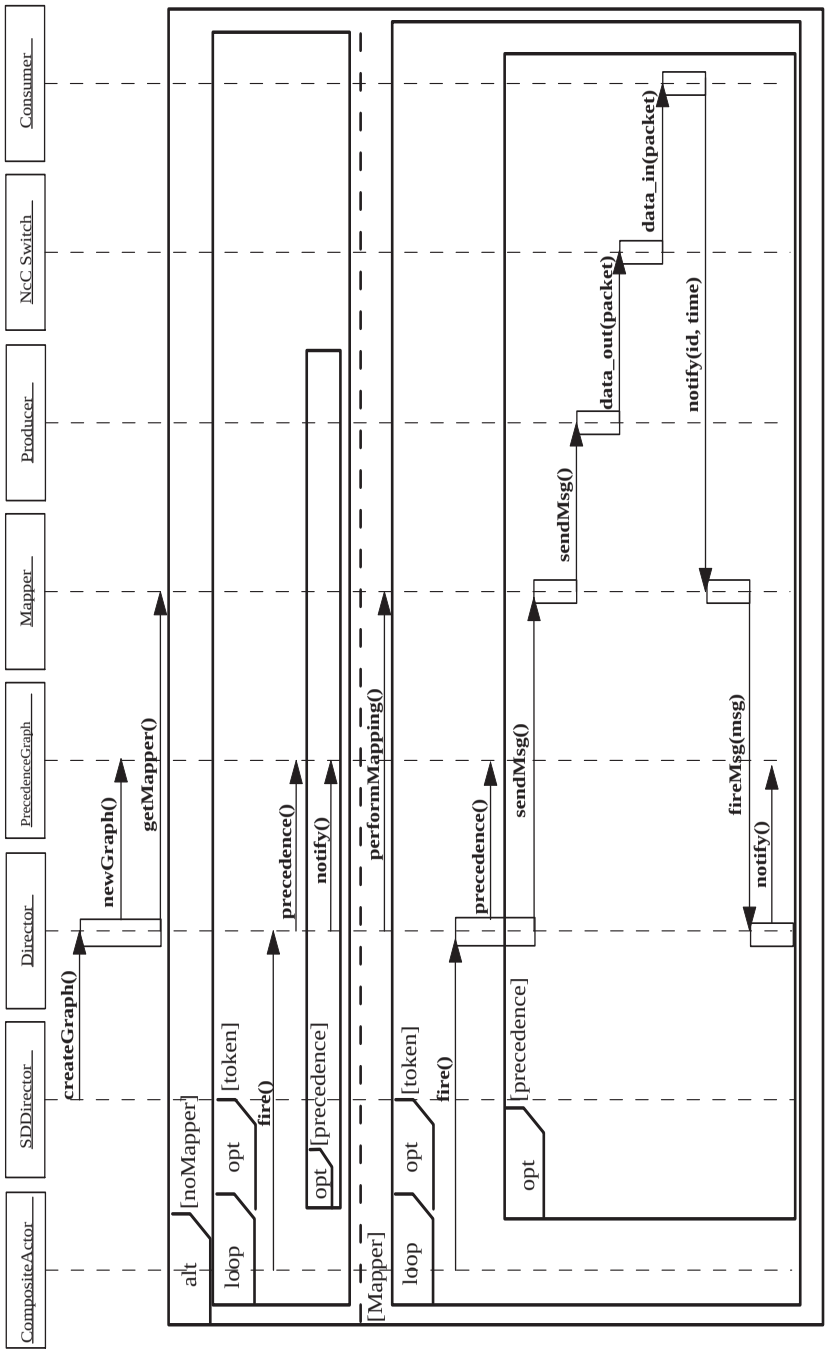

Fig. 15. *Sequence diagram of the application elements' interaction during simulation*

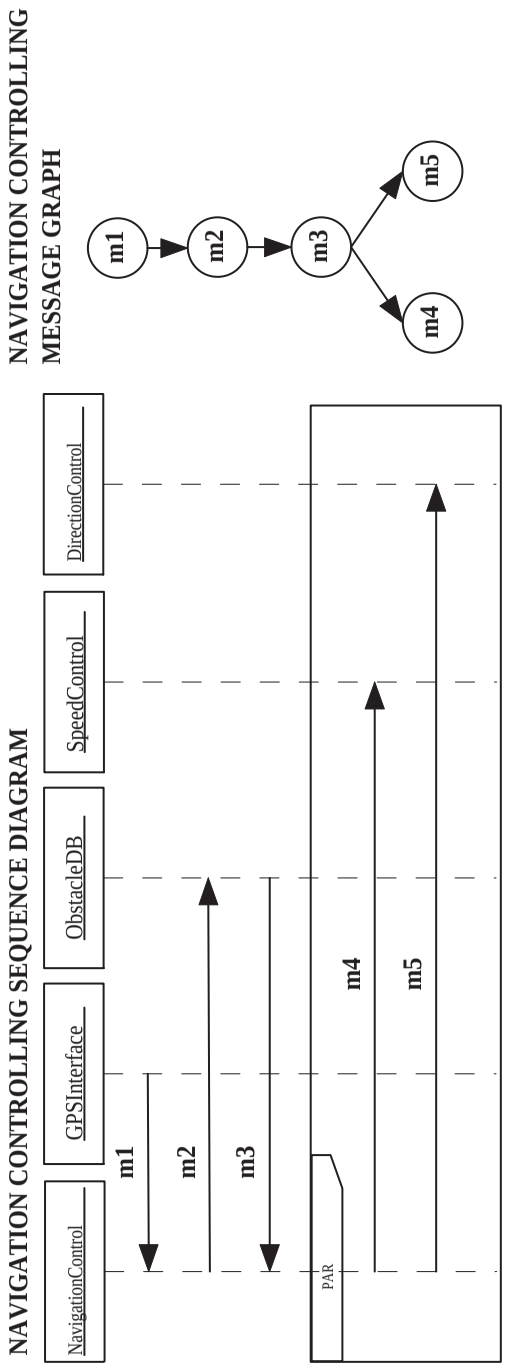

Fig. 16. *A sequence diagram with its corresponding message graph*

All total and partial order directors then find out whether the application is simulated as a platform independent stand-alone model or whether it is running on a platform. The method call getMapper() invoked by the director to its super class returns the mapper if it exists, otherwise it returns null. The mapper implements the mapping of application actors to the platform resources.

As depicted in Figure 15 if the noMapper condition is true, the execution of the application model follows the interactions described inside the uppert part of the alternative CF. Thus, the application is simulated as a stand-alone model without the platform. If there is a mapper, the application is simulated mapped on a platform model and the lower part of the alternative CF describes the interactions.

If there is no mapper, the composite actor, inside which an application sequence diagram is encapsulated, can fire the director using the fire() methods 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 using the precedence() method.

The precedence of a message is satisfied if the message has not been fired on that execution round and all its preceding messages have been fired on that round. Alternative, optional, loop, and parallel CFs set some restrictions for that rule. If there is an alternative CF, the precedence of that message is satisfied, whose alternative condition is true (only one condition can be true; thus, there can be two or more conditions. In case of only one condition, the optional CF can be used). 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 the message has been fired on that execution round as long as the loop condition is still true. If a message is inside a parallel CF, the precedence is satisfied even if preceding messages inside the same parallel CF have not been fired. Otherwise the precedence of the message is not satisfied and the token at the input port of the composite actor is stored until the message's precedence is satisfied.

For instance considering the sequence diagram and message graph in Figure 16, 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 meaning that the communication between actors can happen either simultaneously or in any order. 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 (new round(s) cannot overtake any of the older ones, though).

If the message's precedence is satisfied, the total or partial order director notifies the precedence graph using the notify() method to update the status of the message graph. In other words, the precedence graph changes the status of a node containing the message to be fired on that execution round. If the precedence was not satisfied, the notify() method is not called and only a new token at the input port of the composite actor can make the execution to proceed. This procedure is repeated in a loop until the simulation is stopped.

If the application is simulated together with the platform, the lower part of the alternative fragment in Figure 15 depicts the application simulation with the platform. First, the director calls the performMapping() method of the mapper to let it map application actors on the PEs. Then if the composite actor has an incoming token, it fires the total or partial order director (fire() method), which checks the message's precedence (precedence() method).

If the precedence is satisfied, the total or partial order director asks the mapper to communicate with the platform. The total and partial order directors are unaware of the mapping of application actors on the platform, therefore, they do not communicate with the platform directly. Since each lifeline represents an application actor, 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. For instance, considering the application actors and the PEs in Figure 14, if actor a is mapped on PE 1 and actor b on PE 5, a message between lifelines L1 and L2 causes PE 1 to send a packet containing the message to PE 5.

When the mapper calls the producer's sendMsg() method asking it to send a message, the 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 19) 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 through an output port data_out to the NoC that delivers the packet to the correct consumer.

When the consumer receives a packet through its input port data_in, it notifies the mapper that the packet is delivered correctly. It passes the packet id (that identifies each packet) and the receive time of the packet as parameters of the notify() method. Then the mapper notifies the total or partial order director about the received message (fireMsg() method). Finally, the director notifies the precedence graph to set status of a node containing the message into fired on that execution round (notify() method). This procedure described in the last 3 paragraphs is repeated in a loop until the simulation is stopped.

6.6 Creating Hierarchically Heterogeneous Application Models

The execution of the sequence diagrams encapsulated inside composite actors is enabled by the hierarchical modelling style of the Ptolemy II framework. It enables different execution semantics at different hierarchical levels as seen in Figure 17, which is a screen capture of the Vergil visual editor. The lowest window visible on the top left corner presents the top level of hierarchy, whereas the window on the right side presents the application actors and the composite actors encapsulating sequence diagrams that run under DE (thus, on the second level of hierarchy). The window on the bottom left corner presents the pressure controlling sequence diagram (the sequence diagrams are located at the third level of hierarchy).

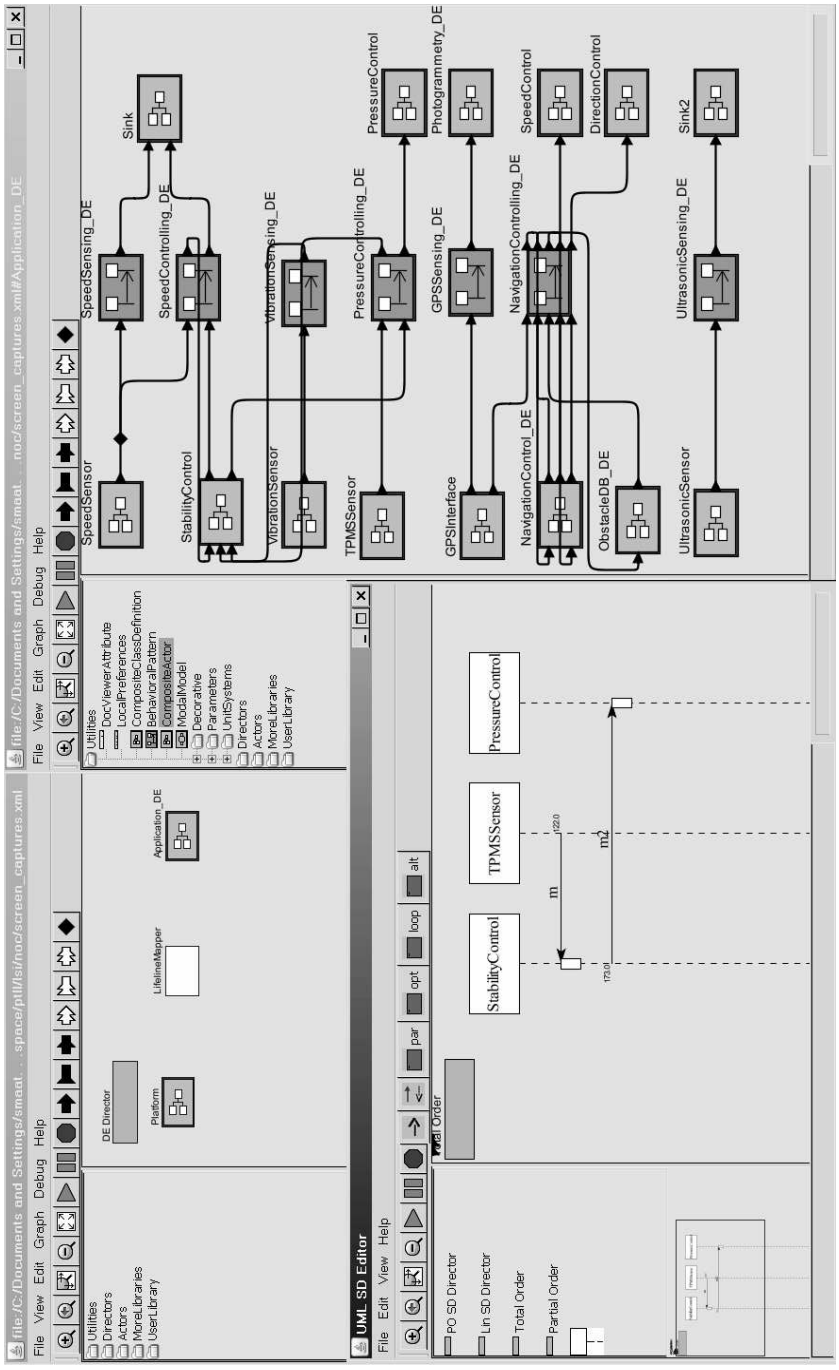

Fig. 17. *Hierarchical description style of the Ptolemy II framework*

The hierarchical description style is used further when creating heterogeneous application models using different MoCs for different parts of the application model. The total and partial order directors still execute the sequence diagrams, but the application actors are not necessarily executed under the DE MoC, as seen in Figure 18. The DE Director at the top level of hierarchy defines the MoC of the whole model to be DE unless otherwise defined at the lower levels of hierarchy. Therefore, the platform model runs under DE, which is typical for communication networks [15]. The application SDF actors run under the SDF MoC, application CT actors under the CT MoC, and application DE actors under the DE MoC. The mapper is an attribute and not an actor and is therefore not subject to the DE execution semantics; the mapper operates based on method calls by the total and partial order directors and PEs.

The different parts of the autonomous vehicle application model are divided to run under different MoCs: The controller logic controlling the vehicle's speed, stability, and tyre pressure is simulated under the DE MoC, whereas the application parts requiring signal processing are run under the SDF MoC. Moreover, the physical characteristics, such as speed, acceleration, and position of the vehicle are modelled in the CT domain [65].

The application actors and the composite actors containing the sequence diagrams are located at the second level of hierarchy with either the SDF or CT director (or no local director making the actors run under the DE MoC). The sequence diagrams are at the third level of hierarchy, as shown in Figure 18 [65].

The obstacle recognition sequence diagrams in Figure 11 run under SDF, whereas the speed controlling of the vehicle is modelled using the CT MoC. The rest of the sequence diagrams shown in Figure 11 run under DE [65].

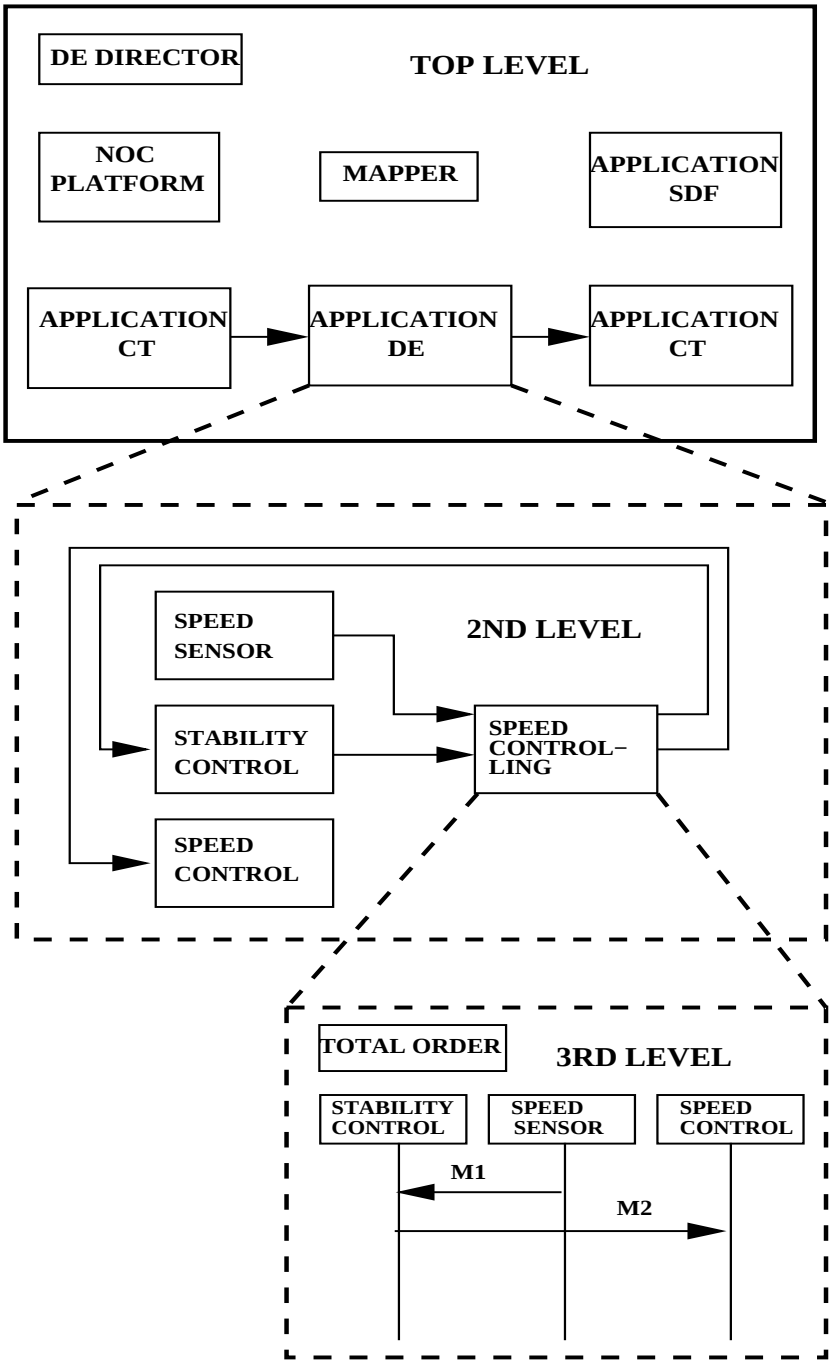

Fig. 18. *Hierarchical heterogeneity (Modified from [65] ©IEEE, 2010)*

6.7 UML Profiling

In order to extensively characterise the application model, the UML profile for MARTE with a few extensions are used [64]. For instance, using only UML it is impossible to define the firing periods of active actors, which is one reason the MARTE profile was needed. Furthermore, UML can handle message concurrency only if the messages are in the same sequence diagram, but using the firing periods, messages in different diagrams can be fired in parallel if they have the same firing period.

Various MARTE profile's stereotypes are used to describe the application and platform model elements. Table 1 lists the MARTE stereotypes and the additional ones necessary for UML sequence diagrams [64]. The *<<TimedEvent>>* stereotype describes an event whose occurrences are bound to clocks [78]. In the application model it defines the execution period of active actors, that is, how often they fire [64]. The real-time unit *<<RtUnit>>* resembles UML's active object (runs when needed and calls or delegates passive objects). A RtUnit owns one or more schedulable resources. The RtUnit's schedulable resource needs to invoce the services provided by the protected passive unit *<<PPUnit>>* [78]. In the application model RtUnit and PpUnit describe active and passive actors respectively [64]. The attributes of the RtUnit and PpUnit are so far disregarded, thus the stereotypes are otherwise used as defined in the MARTE profile's specification.

The *<<hwProcessor>>* and *<<hwCommunicationResource>>* stereotypes present a computing resource and communication resource respectively [78]. They are used for PEs and NoC switches, whereas the *<<Allocate>>* concept depicts the allocation of platform resources for application tasks [64] (the actual allocation (that is, the mapping process) is explained in further detail in 6.4). The attributes of the *<<hwProcessor>>* stereotype are so far disregarded, thus the stereotype is otherwise used as defined in the MARTE profile's specification.

The MARTE profile needed to be extended in order to extensively describe all

Table 1. *Stereotypes describing the application and platform elements (Extended and modified from [64] ©IEEE, 2009)*

	Stereotype	**Used for**
MARTE	*<<TimedEvent>>*	Firing active actors
	<<RtUnit>>	Active actors
	<<PPUnit>>	Passive actors
	<<hwProcessor>>	Processing Element
	<<hwCommunicationResource>>	NoC switch
	<<Allocate>>	Resource allocation
EXTENSION	*<<Sequence>>*	Sequence actors
	<<Message>>	Messages

the elements of the application model. For instance, MARTE neither defines a stereotype for a composite actor that encapsulates a UML sequence diagram nor a stereotype for the sequence diagram's messages.

Table 1 depicts the stereotypes *<<Sequence>>* and *<<Message>>*. The *<<Sequence>>* stereotype describes the composite actors encapsulating sequence diagrams (denoted as sequence actors in Table 1) that are semantically linked to the composite structure diagram connectors. The *<<Message>>* stereotype describes the sequence diagram messages with all their parameters as can be seen in Figure 19 (for the sake of clarity, only one active and passive actor and one sequence diagram are shown in the Figure and only 4 actors are allocated to the PEs) [64].

A composite structure diagram visualises the structure of the whole application model including all the application actors. It also describes, which actors are connected together and that all actors are mapped to the platform (that is, platform resources are allocated for the application tasks). Figure 19 depicts a composite structure diagram presenting the application actors. The

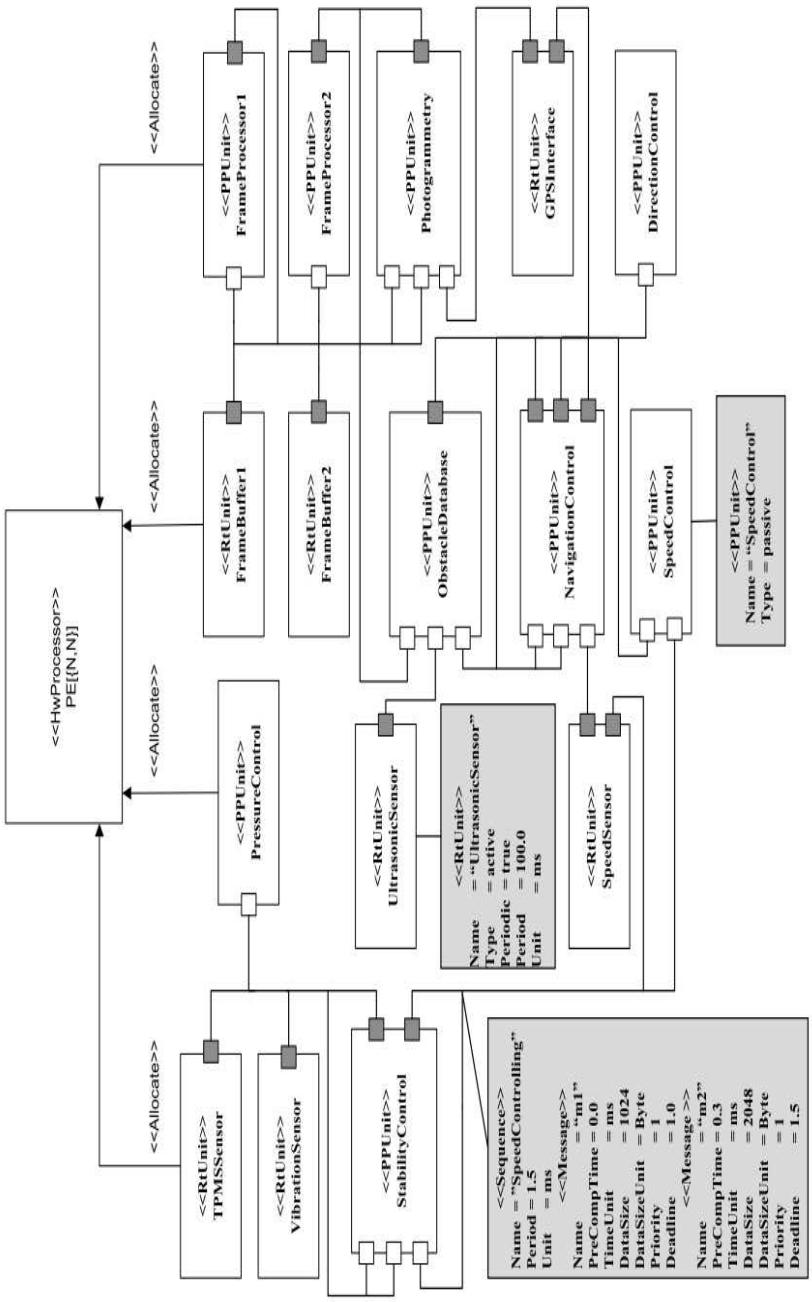

Fig. 19. *Composite structure diagram of the application model (Extended from [64] ©IEEE, 2009)*

Table 2. *Parameters for the UML sequence diagrams (Extended from [65] ©IEEE, 2010)*

Constraint	**Element**	**Used for**
Name	Application	Identification
Period	Active actor	Workload modelling
Time unit	Active actor	Workload modelling
Name	Message	Identification
Computation time	Message	Workload modelling
Time unit	Message	Workload modelling
Data size	Message	Workload modelling
Data size unit	Message	Workload modelling
Priority	Message	Scheduling and arbitration
Deadline	Message	Workload modelling

actors are depicted by big squares having a stereotype and the actors' names inside them and their input and output ports as small white and grey squares respectively [64].

Moreover, in Figure 19 the lines connecting the input and output ports correspond to the communication between the actors. All the actors that are connected to a contiguous set of line segments participate on a communication pattern. Hence, the ordering of the messages exchanged within each pattern is described by the corresponding sequence diagram [64].

Table 2 depicts the necessary parameters that characterise the application model. Application *name* identifies the application sequence diagram. Active actors need a *period* and *time unit* indicating how often they are executed, that is, how often they initiate communication. Messages' *name* identifies them and the *computation time* and *time unit* indicate the computation time

the corresponding task would need to execute on a processor. *Data size* and *data size unit* define the size of the packet that carries the message in the NoC. The *deadline* defines the longest time a message's computation and communication can take, whereas the *priority* can be adjusted higher for messages requiring faster access to platform resources [64].

7. CASE STUDIES

This Chapter presents various case studies that have been conducted using the proposed application modelling approach.

The direct validation and execution of the application models mapped on the platform models enable a fast and less error prone design flow. Moreover, the validation of the whole system early at the design flow reduces errors, speeds up the design flow, and decreases the time-to-market. The direct validation of executable application models mapped on the platform models is addressed in the case study of ”Joint Simulation of Application and Platform Models” presented in section 7.1.

As soon as the modelling approach evolved to be capable of jointly validating the application and platform models, the case study of ”Application Validation on Multi-Abstraction Platform Models” presented in section 7.2 was conducted in order to perform design space exploration and trade-off between simulation speed and accuracy.

Besides validating the correct behaviour of the application, the abstract models can be used also for early performance evaluation. However, at system level it is hard to get any accurate area or power consumption figures. The lack of early performance estimations complicate the selection of resources and may lead to over- or underestimations of the resources [59]. Hence, at system level it is possible to get early performance figures in terms of latency and throughput. Both computation and communication latencies are covered in the case study of ”Evaluating Communication and Computation Costs” presented in section 7.3

Within the case study of ”Modelling with Priorities and Timing Constraints”

presented in section 7.4 the application model is further parameterised showing the potential and flexibility of the application modelling approach.

The modelling of heterogeneous concurrent applications for embedded systems is challenging [71], as proved also by the case study of ”Simulating Heterogeneous System Models” presented in section 7.5. Furthermore, it is also difficult to find a programming model for heterogeneous multiprocessor platforms. The task of implementing heterogeneous applications on heterogeneous architectures requires the modelling of concurrency and using MoCs for formally capturing of the concurrent communication [71].

Following terminology is used from now on:

- Simulation means the act of executing a model in a simulator.
- Simulation time is the current time of the model. In the case studies presented in this Chapter the simulation time means the overall time that the simulation took.
- Wall clock time means the actual time.
- In the context of simulation, real-time means that the simulation time equals wall clock time. That is, the simulation runs real-time.
- Real-time system means a system whose correct behaviour depends not only the correct results of computation but also meeting the deadlines.

All case studied presented in this Chapter are simulated for nine hundred million clock ticks within the Ptolemy II framework. When the clock frequency is assumed to be 50 MHz, this corresponds 18 seconds of wall clock time. However, depending on the platform model, the simulation time could have been even up to 24 hours.

7.1 Case Study of Joint Simulation of Application and Platform Models

The following case study was presented in [66]. The purpose of the case study was to test the proposed modelling approach and to demonstrate its functionality as well as to evaluate the communication costs of packets using different platform configurations. Using the simulation results, platform configurations that are not likely to meet the application requirements can be ruled out and configurations having more promising results can be further optimised and evaluated.

Figure 20 shows the sequence diagrams of the autonomous vehicle application used in this case study. The application model consists of three sequence diagrams: direction adjustment, obstacle recognition, and tyre pressure adjustment.

Different configurations of the RENATO platform were used in order to explore the effect of the platform configuration on the communication latencies. The communication latency was extracted for each message of the sequence diagram when running the application on a platform arranged as either a 2x4, 3x3, 3x4, or 4x4 mesh topology. Furthermore, in order to also get preliminary measures of the effect of different mappings on the communication latencies, two different random mappings for the 3x3 mesh topology were used, whereas only one random mapping was used for all other configurations [66].

Each sequence diagram was executed at a particular rate, which was 0.4 seconds for the obstacle recognition, 1.0 seconds for the direction adjustment, and 2.0 seconds for the tyre pressure adjustment diagrams. The simulation had 50 MHz operation frequency and lasted 18 seconds of wall clock time. Thus, the obstacle recognition diagram was executed 45 times, direction adjustment diagram 18 times, and the tyre pressure adjustment diagram nine times [66].

The communication latency for every message of the sequence diagram was extracted. Furthermore, from the extracted communication latencies the aver-

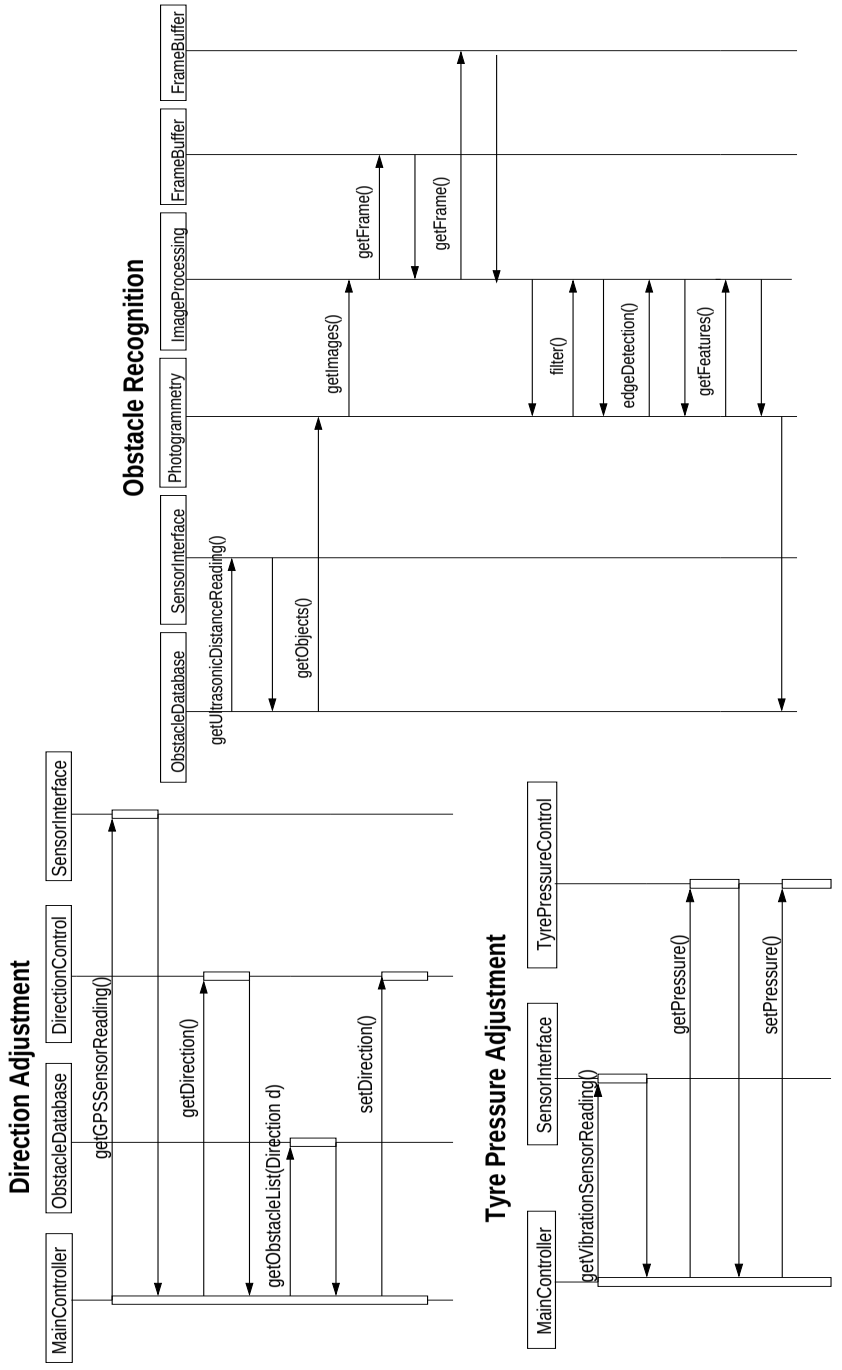

Fig. 20. *UML sequence diagrams of an autonomous vehicle application [66]* ©*IEEE, 2008*

Table 3. *Communication latency for each sequence diagram of the application model [66] ©IEEE, 2008*

Pattern	**2x4**	**3x3 (I)**	**3x3 (II)**	**3x4**	**4x4**
Obstacle Recognition	569,4	448,1	446,9	460,0	455,4
Direction Adjustment	323,5	362,4	296,8	372,5	311,6
Tyre Pressure Adjustment	429,0	515,4	500,0	410,2	517,4

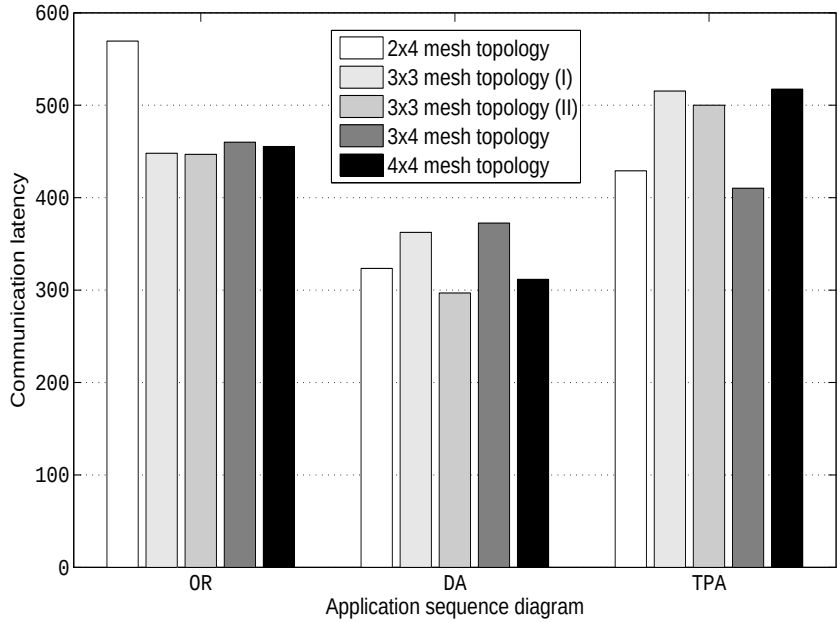

Fig. 21. *Communication latency of the sequence diagrams using different network configurations*

age latency of the execution rounds for each sequence diagram was calculated. Table 3 illustrates the average latencies for each sequence diagram using 2x4, 3x3, 3x4, and 4x4 mesh topologies. 3x3 (I) and 3x3 (II) stand for the two different mappings (the same notation is used in Figure 21) [66].

The obstacle recognition diagram's messages carry the largest amount of data

(because they transfer images), which explains the long latency. The tyre pressure adjustment diagram is executed least frequently and its messages need to wait longer for the routing since the messages of the two other diagrams already occupy the routers at the same time [66].

The two different mappings for 3x3 topology have the biggest effect on the direction adjustment pattern. In the mapping II, the lifelines of the diagram happened to be mapped into adjacent switches, which decreases the network latency by for instance decreasing the length of the path from the traffic producer to the consumer [66].

The simulation results are also presented in Figure 21. In the Figure, OR means the obstacle recognition sequence diagram, DA the direction adjustment diagram, and TPA the tyre pressure adjustment diagram, whereas 2x4, 3x3(I), 3x3(II), 3x4, and 4x4 indicate the size of the mesh topology (the I and II of the 3x3 configuration indicate the two different mappings). As can be seen from Figure 21, none of the configurations clearly outperforms the others.

The 3x3 configuration with routing II is the best for the obstacle recognition and direction adjustment diagrams, but the 2x4 and 3x4 topologies are clearly better for the tyre pressure adjustment. Smaller topologies cause more congestion to the NoC links, whereas bigger topologies increase the distance between the sender and the receiver. When 2 parameters (in this case the mapping and NoC size) change, it is impossible to say, whose effect is bigger on the communication latency.

7.2 Case Study of Application Validation on Multi-Abstraction Platform Models

The following case study was presented in [67]. The purpose of the case study was to successively map the application model onto various abstract platform models. Therefore, the usefulness of the more abstract platform models in

comparison with the less abstract platform models could be explored. The results obtained from the simulation of the same application model mapped on various platform models facilitate the system designers to choose and parameterise a platform model so that the platform model satisfies the application requirements.

The application model used in this case study is depicted in Figure 22. The application model is extended from the model presented in the case study in section 7.1. The extended model consists of five sequence diagrams: photogrammetry, snapshot request, obstacle recognition, direction adjustment, and tyre pressure adjustment.

The application model was simulated using two different random mappings on three different platform models, RENATO, JOSELITO, and BOÇA, all of them having a 4x4 mesh topology. In this simulation, the packet size was limited to 48 flits (each flit being 16 bits) and bigger packets were divided into multiple subpackets. This packet size is considered to be a good trade-off between the overhead the multiple packet headers generate and the occupation of platform resources, such as channels and buffers [67].

The communication latency was extracted for each message of the sequence diagrams when simulating with 50 MHz operation frequency for 18 seconds of wall clock time. The photogrammetry, obstacle recognition, and direction adjustment were executed once every two seconds, while tyre pressure adjustment and snapshot request were executed once a second [67].

Tables 4 and 5 present the simulation results. The RENATO model is back-annotated with the timing information from the HERMES model and is therefore used as a reference model in the simulations. BOÇA is the most abstract of the simulated models and is not back-annotated with the timing delays from the HERMES RTL model. Therefore, BOÇA has a significant error for the worst case latency, in average 46 per cent in comparison with RENATO [67].

Also JOSELITO has a high average latency error, 30 or 31 per cent in comparison with RENATO. However, both BOÇA and JOSELITO are useful models, since BOÇA simulated 402 and 492 faster and JOSELITO simulated 2.8 and

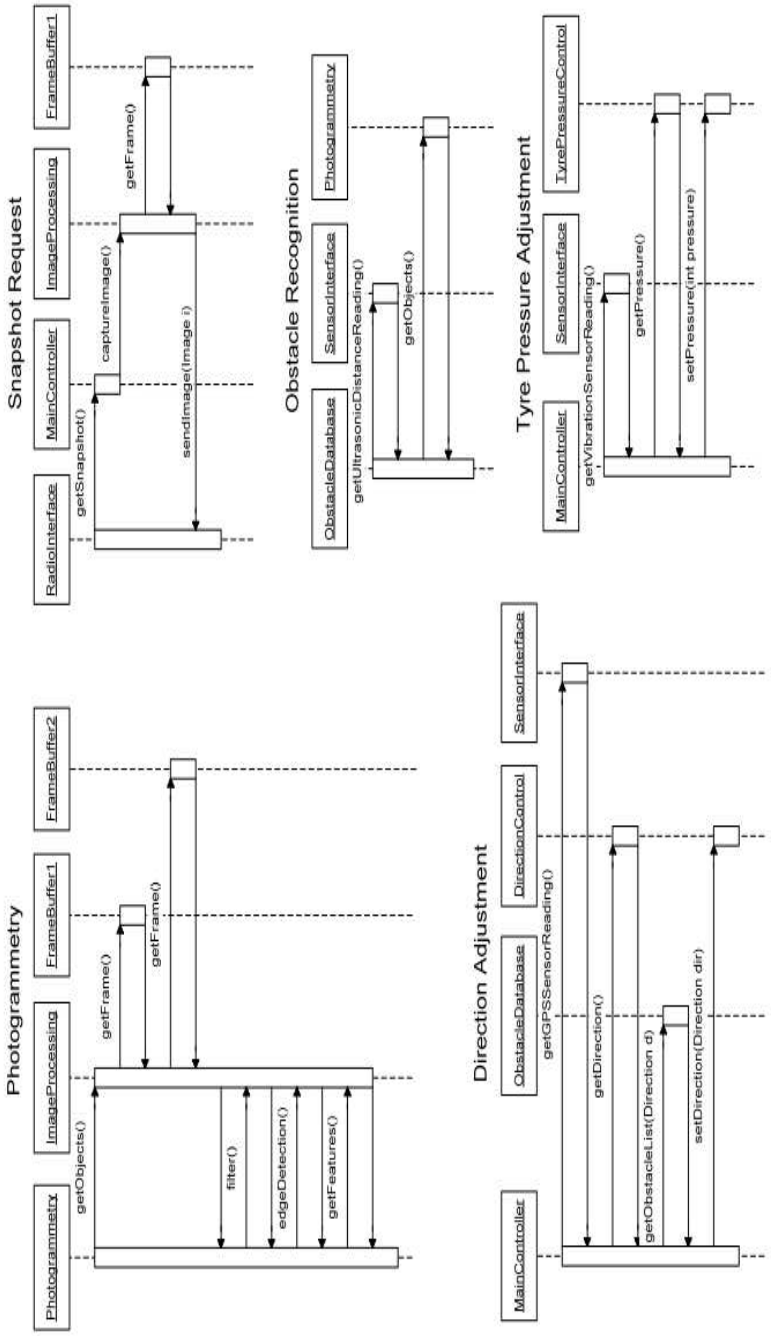

Fig. 22. *UML sequence diagrams of an autonomous vehicle application [67] ©IGI Global 2010*

Table 4. *Worst case latencies of each sequence diagram in milliseconds for mapping 1 (Modified from [67] ©IGI Global, 2010)*

	RENATO	**JOSELITO**	**BOÇA**
Direction Adjustment	0.39	0.27 (31%)	0.24 (38%)
Obstacle Recognition	0.12	0.09 (25%)	0.07 (42%)
Photogrammetry	1.41	0.99 (30%)	0.54 (62%)
Snapshot Request	1.35	0.94 (30%)	0.90 (33%)
Tyre Pressure Adjustment	0.14	0.09 (36%)	0.08 (55%)
Average	3.41	2.38 (30%)	1.83 (46%)

Table 5. *Worst case latencies of each sequence diagram in milliseconds for mapping 2 (Modified from [67] ©IGI Global, 2010)*

	RENATO	**JOSELITO**	**BOÇA**
Direction Adjustment	0.38	0.26 (32%)	0.25 (34%)
Obstacle Recognition	0.10	0.07 (30%)	0.07 (30%)
Photogrammetry	1.37	0.95 (31%)	0.52 (62%)
Snapshot Request	1.31	0.90 (31%)	0.89 (32%)
Tyre Pressure Adjustment	0.11	0.08 (27%)	0.05 (55%)
Average	3.27	2.26 (31%)	1.78 (46%)

3.0 times faster with the two different mappings than RENATO [67].

Figure 23 shows the worst case communication latency for each sequence diagram. The DA, OR, P, SR, and TPA stand for the sequence diagrams Direction Adjustment, Obstacle Recognition, Photogrammetry, Snapshot Request, and Tyre Pressure Adjustment. As can be seen from the results Figure 23 depicts, there is a clear difference between the worst case latencies of the sequence

diagrams using the RENATO platform in comparison with JOSELITO and BOÇA platforms. The bigger data size the packets are carrying, the bigger the difference in worst case communication latencies. The size of the photogrammetry sequence diagram's messages is bigger (they are carrying images) than of the other sequence diagrams. As the bigger packets are divided into smaller subpackets (having maximum size of 48 flits), bigger packets cause more traffic in the network and cause more difference in the worst case latencies of also JOSELITO and BOÇA. When simulating the sequence diagrams having small-sized messages (obstacle recognition and tyre pressure adjustment) the difference between the worst case latencies of the different platform models is close to negligible even though the error percentage figures might seem to illustrate otherwise in Figure 24.

As seen from Figure 24, the error percentage of BOÇA is much bigger than the error percentage of JOSELITO particularly in the photogrammetry sequence diagram. The more packets occupy the network, the faster the BOÇA model is, since no flits are actually routed in the network. The big difference of the error percentages of the tyre pressure adjustment diagrams can be explained with the relatively short simulation times of the diagram; therefore, even a small difference in the simulation times (see Tables 4 and 5) makes a huge difference in the error percentages.

Several observability and debugging features (referred as scopes) facilitated the analysis of the simulations. The scopes are actors that monitor the traffic in the NoC, collect data from the network, and display it graphically. The scopes are implemented to analyse buffer occupation (BufferScope), capture the input and output channel activity of the NoC routers (InputScope and OutputScope respectively), or analyse the power consumption of the routers (PowerScope). Moreover, the HotSpotScope detects blocked packets indicating network congestions, the EndToEndScope depicts which network nodes are communicating with each other, and the PointToPointScope illustrates the complete path the packets use in the NoC [67].

The scopes are graphical and the displays are updated as the simulation proceeds, resulting in full observability of the behaviour of the NoC models [67].

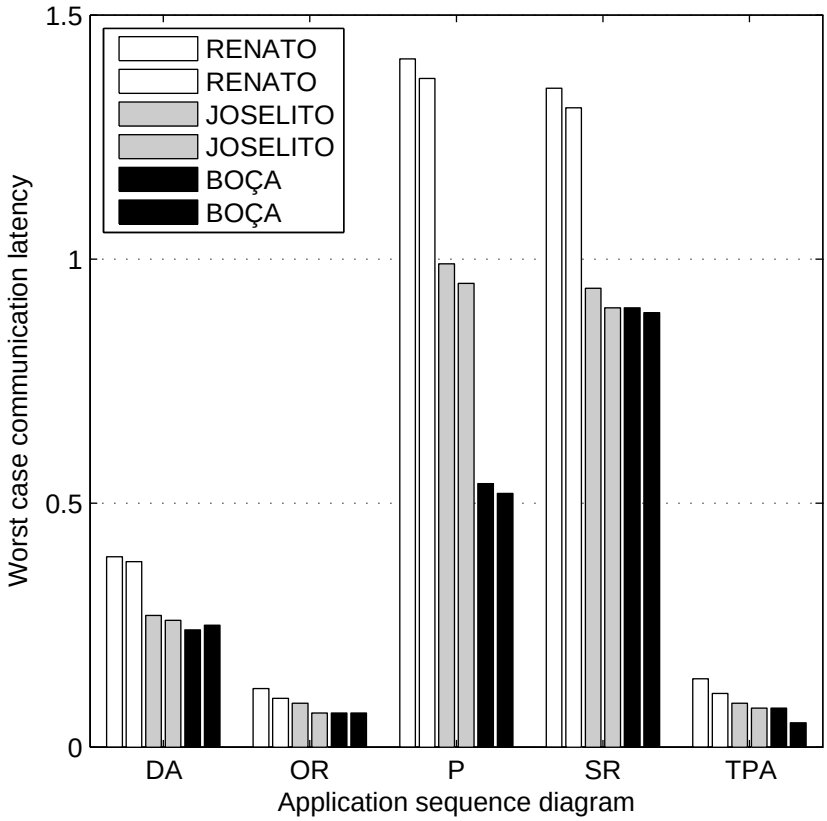

Fig. 23. *Worst case communication latency for each sequence diagram using 2 different random mappings for each NoC model*

However, simulating even more actors than what the application and platform models already include, slows down the simulation speed. Thus, the scopes can also be turned off. The scopes are presented in further detail in [73].

7.3 Case Study of Evaluating Communication and Computation Costs

The following case study was presented in [64], which also presented a set of modelling constructs that can extensively characterise the application model.

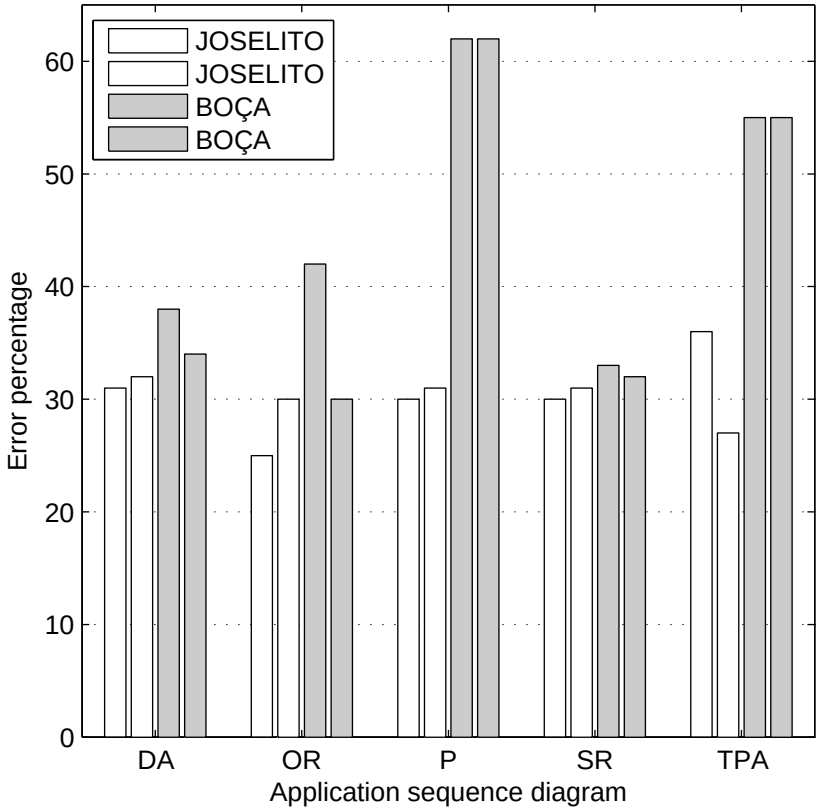

Fig. 24. *Latency error of JOSELITO and BOÇA in comparison with RENATO using two different mappings*

The set of constructs was organised as a UML profile (as presented at the end of Chapter 6). The purpose of the case study was to consider also the effect of computation latency of the platform; all the previous cases have dealt with only the communication latency. When both the communication and computation costs are considered, the evaluation of whether the platform models satisfy the application requirements is more extensive.

This case study uses the most evolved autonomous vehicle application model (see Figure 11). The autonomous vehicle application sequence diagrams presented in the previous two Chapters are further divided into nine sequence diagrams: navigation controlling, pressure controlling, speed controlling, ob-

stacle recognition 1 and 2, ultrasonic sensing, GPS sensing, speed sensing, and vibration sensing.

The platform used for the simulation was the JOSELITO model having 3x3, 3x4, and 4x4 mesh topologies and two different mappings. The first mapping was totally random and the other mapped those actors that communicate the most with each other onto adjacent processing elements (referred as static mapping) [64].

The simulation was using 50 MHz operation frequency and lasted 18 seconds of wall clock time [64]. The pressure controlling, navigation controlling, and GPS sensing sequence diagrams were executed twice a second; ultrasonic sensing, vibration sensing, speed sensing, and speed controlling diagrams 10 times a second; and both the obstacle recognition diagrams 25 times a second.

Both the communication and computation latencies were extracted as well as the worst case execution latency for each message of the sequence diagrams [64]. Table 6 depicts the average latency for critical communications and for all communications as well as average latencies for critical tasks and all tasks using two different mappings (Random and Static). The parts of the vehicle that control its speed and direction are considered as critical (the navigation controlling, obstacle recognition, speed controlling, and speed sensing sequence diagrams in Figure 11) [64].

As can be seen from Figure 25, the critical communication has remarkably shorter latency in comparison with all communication. On the contrary, the critical computation has much longer latency than all computation when using the smaller topologies.

In bigger topologies task execution is divided among more PEs thus causing shorter delays. The scheduling strategy was nonpre-emptive and all the messages had the same priority. Consequently, if a noncritical task has already reserved a PEs resources, a critical task needs to wait. This is one reason for longer latencies of the critical tasks in comparison with all tasks.

Another reason for the longer average latencies of the critical tasks is the long communication and computation latencies of the obstacle recognition

Table 6. *Average latency of critical and all communication and computation in milliseconds [64] ©IEEE, 2009*

Topology		**Random**	**Static**
3x3	Critical communication	0,077	0,075
	All communication	0,507	0,510
	Critical tasks	0,390	0,245
	All tasks	0,063	0,191
3x4	Critical communication	0,097	0,075
	All communication	0,491	0,509
	Critical tasks	0,199	0,005
	All tasks	0,084	0,027
4x4	Critical communication	0,074	0,074
	All communication	0,480	0,510
	Critical tasks	0,002	0,005
	All tasks	0,003	0,005

diagrams' messages. The critical messages having long latencies also delays other critical messages' execution; in bigger network topologies it is less likely that many critical tasks are mapped onto the same PEs.

In this simulation case, the 4x4 network configurations perform the best, since the computation delays are minimal in comparison with the smaller configurations.

In some cases the random mapping slightly outperforms the static mapping as can be seen from Tables 6 and 7 as well as from Figure 25. In those cases the actors that communicate the most with each other were randomly mapped onto the same PE. This minimises the communication costs thus increasing the computation latency. Therefore, choosing the right mapping is a trade-off between communication and computation costs [64].

Table 7 shows the worst case latency of each message of the sequence diagrams. The mapping or the topology has only a minor effect on the worst case latency of each message and none of the configurations outperforms the

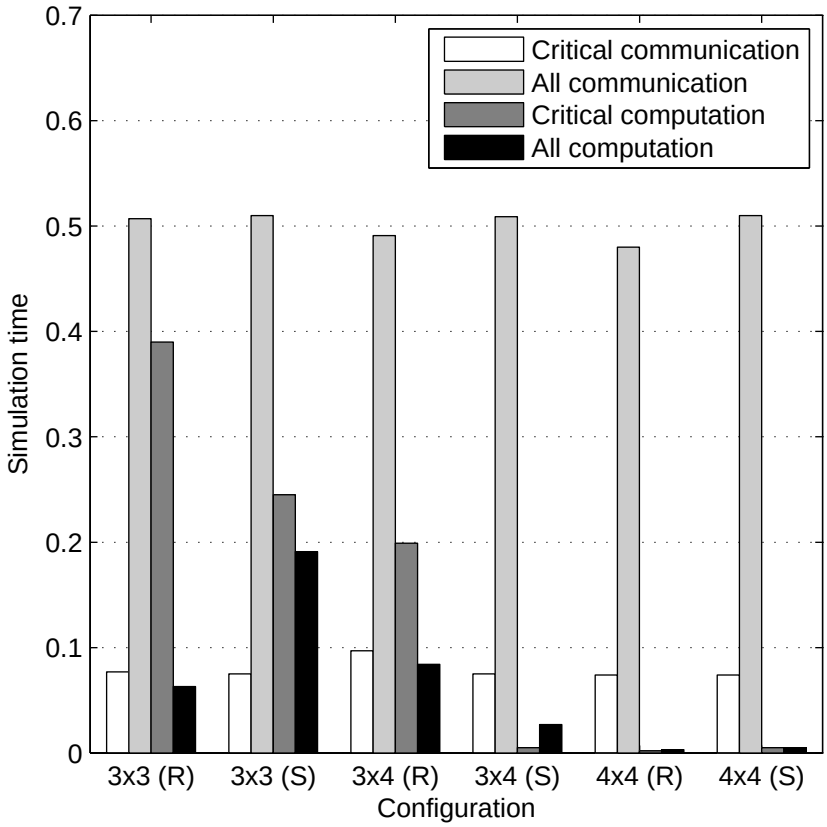

Fig. 25. *Average latency of critical and all communication and computation using different platform configurations and random (R) and static (S) mapping*

others. Messages M9 and M12 carry the picture frames from the vehicle's cameras causing longer latencies of the messages due to their bigger size [64].

7.4 Case Study of Modelling with Priorities and Timing Constraints

The following case study is previously unpublished. The purpose of the case study was to extend the application modelling approach with different scheduling policies and timing constraints, such as priority based scheduling

Table 7. *Worst case latency for each message for all configurations in milliseconds [64] ©IEEE, 2009*

	3x3		3x4		4x4	
Message	**Random**	**Static**	**Random**	**Static**	**Random**	**Static**
M1	0,027	0,027	0,037	0,028	0,027	0,027
M2	0,014	0,013	0,014	0,013	0,014	0,014
M3	0,109	0,112	0,107	0,112	0,106	0,112
M4	0,027	0,027	0,027	0,027	0,025	0,027
M5	0,106	0,062	0,053	0,062	0,053	0,060
M6	0,424	0,423	0,424	0,424	0,424	0,424
M7	0,027	0,027	0,027	0,027	0,027	0,027
M8	0,014	0,014	0,014	0,014	0,014	0,014
M9	4,096	3,968	3,968	3,968	3,776	3,968
M10	0,212	0,379	0,213	0,380	0,212	0,379
M11	0,096	0,053	0,054	0,053	0,051	0,053
M12	4,096	3,968	3,968	3,968	3,968	3,968
M13	0,213	0,379	0,212	0,379	0,212	0,379
M14	0,096	0,053	0,054	0,053	0,054	0,053
M15	0,054	0,076	0,054	0,053	0,064	0,074
M16	0,054	0,054	0,053	0,054	0,054	0,054
M17	0,011	0,012	0,011	0,011	0,012	0,017
M18	0,021	0,021	0,021	0,021	0,021	0,021
M19	0,029	0,038	0,025	0,024	0,014	0,038

and deadlines. Moreover, the application model was extended with several new sequence diagrams increasing the number of simulated application actors and workload on platform resources.

In this case study the autonomous vehicle application (depicted in Figure 11) was simulated with Video Object Plane Decoder (VOPD), High-Definition Television (HDTV), and Moving Picture Experts Group (MPEG) 4 decoder. The platform model was JOSELITO having 3x3, 4x4, 5x5, and 6x6 mesh topologies.

For reference, the application model was first simulated without priority-

Table 8. *Percentage of messages of each platform configuration violating timing constraints*

Scheduling	**3x3**	**4x4**	**5x5**	**6x6**
First in first served	48.9	45.0	37.8	39.4
Priority based	46.7	46.2	37.2	39.4

based scheduling using the first-in-first-served scheduling policy. Then higher priority was set to the critical application messages. The messages used for controlling the vehicle's speed or direction were considered as critical. Both scheduling policies were nonpre-emptive. The same static mapping was used for both scheduling policies.

The simulation was using 50 MHz operation frequency and lasted 18 seconds of wall clock time. The pressure controlling, navigation controlling, and GPS sensing sequence diagrams were executed twice a second; ultrasonic sensing, vibration sensing, speed sensing, and speed controlling diagrams 10 times a second; and both the obstacle recognition diagrams 25 times a second. The HDTV sequence diagram was executed once in every 1.5 seconds and the VOPD and MPEG diagrams once in every 1.2 seconds.

This case study showed how many timing constraint violations happen for critical messages using different scheduling policies and different platform configurations. A timing constraint violation means that a message exceeds its deadline. The deadlines are set as parameters of the messages, as shown in Table 2 in Chapter 6.

Table 8 depicts how many per cent of the critical messages violate the timing constraints and Table 9 shows the average violation time in milliseconds for all topologies.

As can be seen from the Tables and also from Figures 26 and 27, usually a bigger network results in less and shorter violations (with a few exceptions). Especially, the 5x5 configuration performs clearly better than the 3x3 and 4x4

Table 9. *Average timing constraint violation of each platform configuration in milliseconds*

Scheduling	**3x3**	**4x4**	**5x5**	**6x6**
First in first served	0.78	0.44	0.15	0.38
Priority based	0.82	0.58	0.16	0.38

configurations and slightly better than the 6x6 configuration. Bigger topologies have more PEs and fewer tasks are executed on the same PE. However, bigger topologies result in longer communication latency due to longer distance between the packets' sources and destinations.

The priority based scheduling does not outperform the first-in-first-served scheduling in this simulation case. This is due to the small number of tasks mapped onto a same PE and different firing periods of the sequence diagrams' active actors. Therefore, more than one message is seldom processed at the same time by each PE and the priority based scheduling does not improve the simulation results much.

7.5 Case Study of Simulating Heterogeneous System Models

The following case study was presented in [65]. The purpose of the case study was to build heterogeneous application models in order to respond to the need of heterogeneity of today's embedded systems. The refinement of the abstract, high level model containing various MoCs to a lower level implementation, where only the DE MoC is used is out of the scope of this thesis.

Within this case study, four different cases were simulated using the same application model, which is presented in Figure 11. The first case is a reference case, a homogeneous, nonhierarchical model using only the DE MoC (that is, all application actors are at the top level of hierarchy and the sequence diagrams at the second level of hierarchy). The second case is a homogeneous

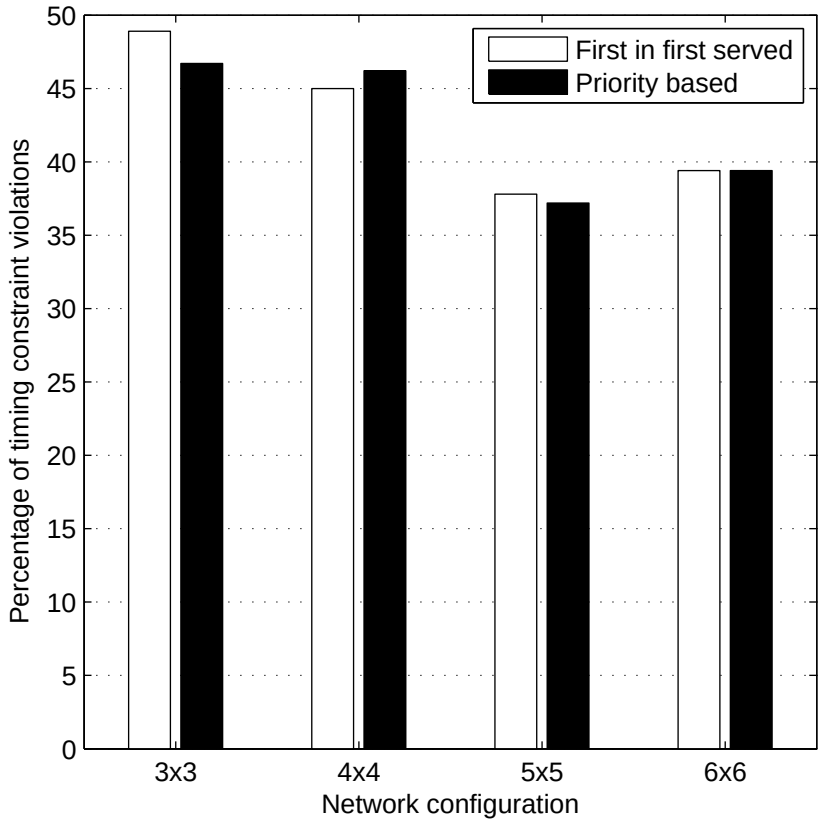

Fig. 26. *Percentage of critical messages violating timing constraints*

but hierarchical model using only the DE MoC (that is, the application actors are at the second level of hierarchy and the sequence diagrams at the third level of hierarchy). The third case is a hybrid model using the DE and SDF MoCs. The fourth case is a hybrid, mixed-signal model using CT, SDF, and DE MoCs (depicted in Figure 18). These cases show how much the artificial layer of hierarchy affects the simulation time and whether the use of multiple MoCs is beneficial [65].

Figure 28 depicts the simulation setup for connecting the CT and DE actors. A CT subsystem models the speed of the vehicle (the setup was inspired by an example of a car tracking application [92]), a periodic sampler compo-

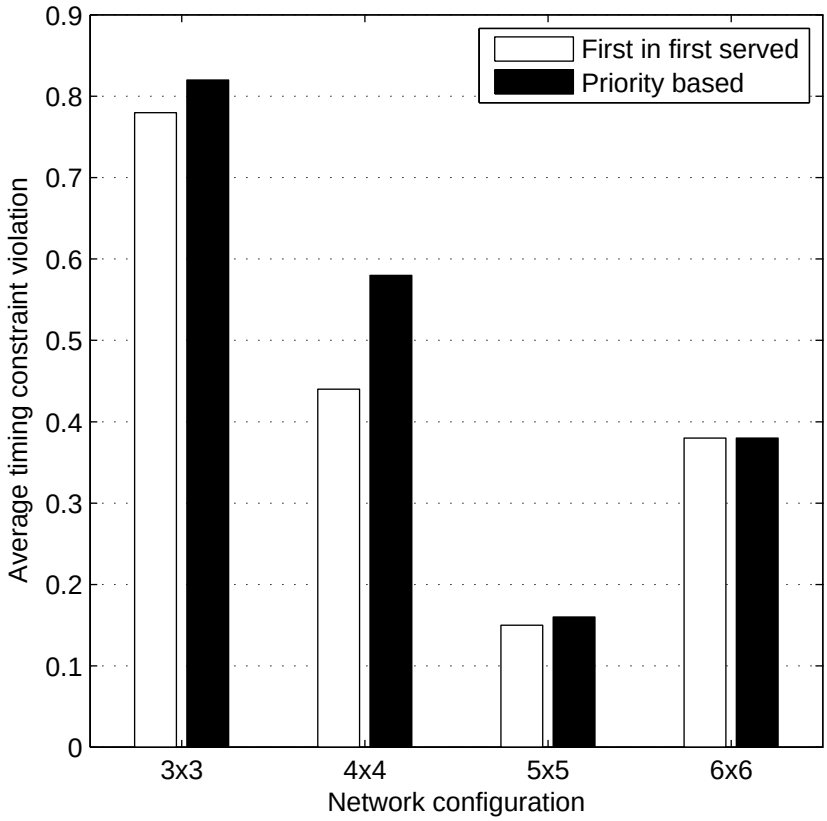

Fig. 27. *Average timing constraint violation in milliseconds*

nent converts the continuous signal into a discrete one and that discrete value controls the speed of the vehicle (the control logic is modelled in the DE domain). After the speed has been changed, the discrete value is converted back into a continuous signal using a zero order hold component. Then the speed, acceleration, and position of the vehicle can be monitored using timed plotters [65].

The overall simulation time of each of the model were compared to the reference model (which is the nonhierarchical homogeneous model). The second case, which is the hierarchical homogeneous model is 0.989 per cent slower and the third case, the hierarchical hybrid model is 1.34 per cent slower than

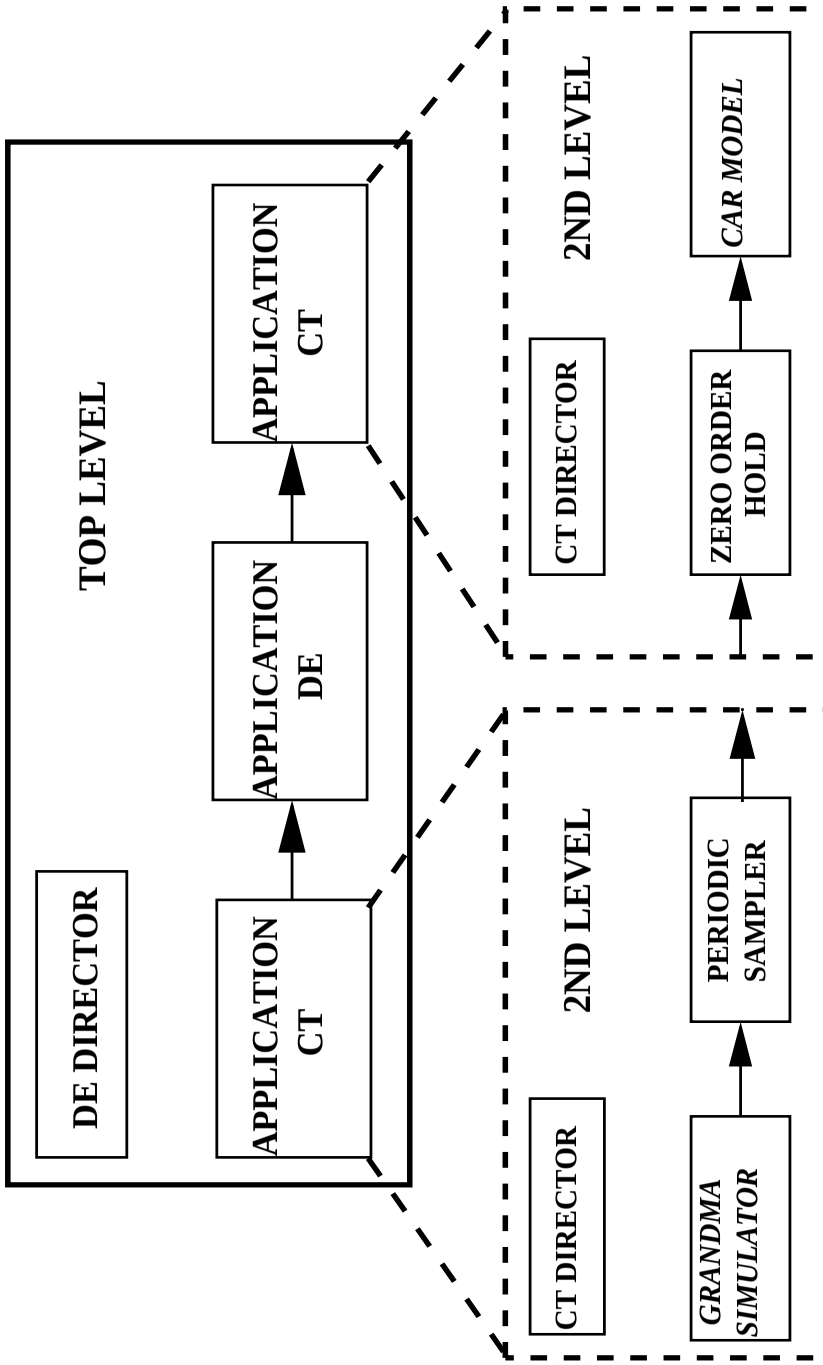

Fig. 28. *Heterogeneous application model (Modified from [65] ©IEEE, 2010)*

the reference model [65].

Interestingly, the hierarchical hybrid model is 0.347 per cent slower than the hierarchical homogeneous model (meaning that the static scheduling does not speed-up the simulation). Nevertheless, in an approximately 24 hour simulation (wall clock time), the simulation speed difference is negligible (only about 5 minutes) [65].

The fourth case, which is the mixed-signal model using DE, SDF, and CT MoCs was not able to simulate until the end due to the restricted size of Java heap space [65].

The extra layer of hierarchy when building a heterogeneous application model has a negligible impact on the simulation time. Moreover, even though the static scheduling of SDF was not advantageous regarding the simulation time, selecting a feasible MoC has still several other advantages, as discussed earlier in Chapter 2 [65].

7.6 Discussion

Embedded system designers need methods to validate application-specific functionality together with different platform configurations. In an ideal case, this should happen at as early stage of the design process as possible, so that designers can explore the design space before committing to specific processor architectures or custom hardware implementation [66].

The first case study presented in this Chapter demonstrated the proposed modelling approach, in which the application and platform can be validated together in the same model. This addresses the problem of separated HW and SW design flows in embedded system design, which is often caused by the use of different languages for HW and SW modelling.

The Ptolemy II framework enables the implementation of execution semantics for UML sequence diagrams. This makes it possible to model the application

and platform using different modelling styles (UML for the application and Java for the platform), yet still simulate the whole system in the same model.

The design space of a multiprocessing system based on NoC interconnects is huge. Considering the interconnect structure only, the design space includes for instance buffering and flow control mechanisms, network topology, packet structure and size, buffer size, as well as routing and arbitration algorithms. Therefore, abstract platform models that simulate fast should be used at the early stages of the design flow in order to perform rough performance evaluations and to rule out poorly performing platform configurations and to delimit the design space. More accurate models can then be used for fine-tuning platform parameters and for choosing the best mapping.

The application modelling approach can handle the successive refinement of platform models modelled at different level of accuracy. The second case study presented in this Chapter demonstrated the joint validation of one application model successively mapped on three different platform models. This case study also facilitates the trading-off between the model accuracy, observability, and simulation speed. The successive mapping of applications on different platform models is an effortless plug-and-play operation. The platform templates as well as different mapping heuristics can be chosen from a library.

Performing the second case study, the application simulation on the RENATO platform took over 20 hours (and the simulation time corresponds only 18 seconds of wall clock time); therefore, achieving over 400 times faster simulation when using the BOÇA platform model is worth sacrificing a little accuracy, especially when the correct behaviour of the system is not sacrificed. Thus, in order to simulate the system and explore its design space within a reasonable time, it is necessary to either strictly delimit the design space or use more abstract executable models.

Until regarding also the computation latency caused by the application execution on processors, the best mapping of the application model on the platform would be to map all application actors on a single PE. This would minimise

the communication latency. The third case study of this Chapter considered also the impact of computation latency, which means that minimising communication latency maximises the computation latency. This motivates the evaluation of different mapping heuristics (which is out of the scope of this thesis).

Also, the third and fourth case studies presented in this Chapter demonstrated how important it would be to have enough platform resources: it is not tolerable that in some cases almost half of the critical messages exceeded their deadlines. Nevertheless, it is not enough that a platform barely supports just the current application (and as the case study showed, if even that), it should also support the future evolutions of the application [56]. Therefore, the platform performance should not be let to restrict the application design and future evolution. Instead, applications that require more powerful platforms will set the requirements for the future platform development.

The last case study presented in this Chapter demonstrated that even if the Ptolemy II framework suits well for heterogeneous, multi-MoC modelling, it is still not easy to fully benefit from its capabilities. The particular application model does not benefit from for example the static scheduling of SDF regarding to simulation time. But then again, the artificial layer of hierarchy caused by the hierarchical heterogeneity did have only a negligible adverse impact on the simulation time.

The last case study also showed the limitations of the Ptolemy II framework. The simulation of hundreds of actors using different MoCs slows down the simulation engine significantly or even prevents the simulation of complex, mixed-signal models.

The results obtained in the case studies presented in sections 7.1 – 7.4 cannot be really compared to any other approach, since there is no similar work using this kind of application modelling approach. Academic approaches often rely on presenting the application models as task graphs as seen for instance in [52], which not only may lead to ambiguous application description but also requires code generation or manual transformation into an executable form.

Industrial tools are not capable of simulating UML sequence diagrams and as not being open source, it would be hard or impossible to extend them with the execution semantics for UML diagrams. Using code generation from UML diagrams to an executable language could be one possible approach to make the results comparable to the results achieved using the approach presented in this thesis.

Hence, the purpose of the case studies has been to demonstrate the usefulness of this system modelling approach. The approach is flexible and extensible: it enables the use different platform models, different application models, different mappings, and different parameters for the application models. Important model characteristics can be added, the approach evolved from being capable of capturing only the communication latency to be capable of depict also the computation latency, timing constraints, and priorities.

This system modelling approach is used as a part of a model-based design flow, as described in further detail in [83] and [84]. Moreover, [47] describes how to obtain accurate communication latency figures using this approach. Even though this approach is based on validation by simulation, any formal description of the application model is not excluded, as seen in [67].

8. CONCLUSIONS

This thesis presented an approach for application modelling and joint validation with on-chip multiprocessing platform models using actor orientation and UML within the Ptolemy II framework. The approach enables design space exploration as well as the extraction of performance figures in terms of for instance communication and computation latency.

Raising the level of abstraction and creating executable system models using UML and actor orientation can be considered useful in many ways. First, model-based design especially at the system level increases design productivity and facilitates the comprehending of complex systems. Second, UML is widely understood by SW and HW designers. Thus, the increasing proportion of embedded SW encourages the use of approaches usually associated with SW engineering. Third, actor orientation enables the use of MoCs in actor interaction and is therefore suitable for heterogeneous embedded system design. Finally, executable models can be validated by simulation. Thus, the behaviour of the system is easier to understand by simulating a model than by reading a written description of it.

8.1 Future Development

Executable models enable the system validation by simulation. However, this approach has several shortcomings: Simulation is much slower than the actual design, even though the simulatable model is a trade-off between simulation speed and system accuracy. Moreover, all possible cases cannot be simulated

anyway. Therefore, the application modelling approach presented in this thesis could benefit also from formal verification.

The application mapping on the platform was only static (and random). The implementation of different mapping heuristics as well as dynamic mapping is left as future work.

The Ptolemy project and the Ptolemy II framework have really solid ground. Moreover, the framework is easy to learn and use, above all due to the visual editor (Vergil), which allows the creation of models by simply dragging and dropping components on the workspace; this kind of user interfaces are completely missing from system design languages such as SystemC or System Verilog. However, the underlying JVM restricts the speed and available memory of the simulation. Therefore, it would be beneficial to run the models in an environment without the virtual machine layer.

BIBLIOGRAPHY

[1] *System Verilog 3.1a Language Reference Manual*, Accellera, 2004.

[2] *Verilog-AMS: Language Reference Manual Version 2.3.1*, Accellera, 2009.

[3] G. A. Agha, *ACTORS: A Model of Concurrent Computation in Distributed Systems*. Cambridge: MIT Press, 1986.

[4] S. S. Alhir, *UML in a Nutshell*. O'Reilly & Associates, Inc., 1998.

[5] T. Arpinen, M. Setälä, P. Kukkala, E. Salminen, M. Hännikäinen, and T. D. Hämäläinen, "Modeling embedded software platforms with a UML profile," in *Forum on Specification and Design Languages*, 2007, pp. 237–242.

[6] A. Bakshi, V. K. Prasanna, and A. Ledeczi, "MILAN: A model based integrated simulation framework for design of embedded systems," in *ACM SIGPLAN Workshop*, 2001, pp. 82–93.

[7] F. Balarin, M. Chiodo, P. Giusto, H. Hsieh, A. Jurecska, L. Lavagno, C. Passerone, A. Sangiovanni-Vincentelli, E. Sentovich, K. Suzuki, and B. Tabbara, *Hardware-Software Co-Design of Embedded Systems – The Polis Approach*. Kluwer Academic Publishers, 1997.

[8] F. Balarin, Y. Watanabe, H. Hsieh, L. Lavagno, C. Passerone, and A. Sangiovanni-Vincentelli, "Metropolis: an integrated electronic system design environment," *Computer*, vol. 36, no. 4, pp. 45–52, 2003.

[9] L. Benini and G. de Micheli, "System-level power optimization techniques and tools," *ACM Transaction on Design Automation of Electronic Systems*, vol. 5, no. 2, pp. 115–192, 2000.

[10] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*. Addison Wesley, 1999.

[11] P. Boulet, P. Marquet, E. Piel, and J. Taillard, "Repetitive allocation modeling with MARTE," in *Forum on Specification and Design Languages*, 2007, pp. 280–285.

[12] L. Brisolara, M. E. Kreutz, and L. Carro, "UML as front-end language for embedded systems design," in *Behavioral Modeling for Embedded Systems and Technologies: Applications for Design and Implementation*, L. Gomes and J. M. Fernandes, Eds. IGI Global, 2009, pp. 1–23.

[13] C. Brooks, E. A. Lee, X. Liu, S. Neuendorffer, Y. Zhao, and H. Zheng, "Heterogeneous Concurrent Modeling and Design in Java (Volume 1: Introduction in Ptolemy II)," Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, Tech. Rep., 2008. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-28.html

[14] ——, "Heterogeneous Concurrent Modeling and Design in Java (Volume 2: Ptolemy II Software Architecture)," Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, Tech. Rep., 2008. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-29.html

[15] ——, "Heterogeneous Concurrent Modeling and Design in Java (Volume 3: Ptolemy II Domains)," Department of Electrical Engineering and Computer Sciences, University of

California at Berkeley, Tech. Rep., 2008. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-37.html

[16] J. R. Burch, R. Passerone, and A. L. Sangiovanni-Vincentelli, "Using multiple levels of abstraction in embedded software design," in *EMSOFT*, 2001, pp. 324–343.

[17] A. Cuccuru, J.-L. Dekeyser, P. Marquet, and P. Boulet, "Towards UML 2 extensions for compact modeling of regular complex topologies," in *Conference on Model Driven Engineering Languages and Systems*, 2005, pp. 445–459.

[18] W. J. Dally and B. Towles, "Route packets, not wires: On-chip interconnection networks," in *Proc. Conference on Design Automation*, June 2001, pp. 684–689.

[19] ——, *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers, Inc., 2004.

[20] M. Damm, J. Haase, C. Grimm, F. Herrera, and E. Villar, "Bridging MoCs in SystemC specifications of heterogeneous systems," *EURASIP Journal on Embedded Systems*, vol. 2008, 2008, 16 pages.

[21] Ptolemy II Home Page. Department of Electrical Engineering and Computer Sciences, University of California at Berkeley. [Online]. Available: http://ptolemy.berkeley.edu/ptolemyII/

[22] R. Dömer, A. Gerstlauer, and D. Gajski, "SpecC Language Reference Manual Version 2.0," SpeC Technonology Open Consortium, Tech. Rep., 1990. [Online]. Available: http://www.cecs.uci.edu/specc/

[23] The Eclipse Foundation open source community website. Eclipse. [Online]. Available: http://www.eclipse.org

[24] S. Edwards, L. Lavagno, E. A. Lee, and A. Sangiovanni-Vincentelli, "Design of embedded systems: Formal models, validation, and synthesis," *IEEE*, vol. 85, no. 3, pp. 366–390, 1997.

[25] L. Eggermont, Ed., *Embedded Systems Roadmap 2002*. Technology Foundations (STW), 2002.

[26] J. Eker, J. W. Janneck, E. A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong, "Taming heterogeneity – the Ptolemy approach," *IEEE*, vol. 91, no. 1, pp. 127–144, 2003.

[27] J. Falk, C. Haubelt, and J. Teich, "Efficient representation and simulation of model-based designs in SystemC," in *Proc. International Forum on Specification and Design Languages*, September 2006, pp. 129–134.

[28] A. Ferrari and A. Sangiovanni-Vincentelli, "System design – traditional concepts and new paradigms," in *International Conference on Computer Design*, 1999, pp. 2–12.

[29] D. D. Gajski, S. Abdi, A. Gerstlauer, and G. Schirner, *Embedded System Design – Modeling, Synthesis and Verification*. Springer, 2009.

[30] D. D. Gajski, J. Zhu, R. Dömer, A. Gerstlauer, and S. Zhao, *SpecC: Specification Language and Methodology*. Kluwer Academic Publishers, 2000.

[31] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley, 1995.

[32] B. Graaf, M. Lormans, and H. Toetenel, "Embedded software engineering: the state of the practice," *IEEE Software*, vol. 20, no. 6, pp. 61–69, 2003.

[33] C. Grimm, A. Jantsch, S. Shukla, and E. Villar, "C-based design of heterogeneous embedded systems," *EURASIP Journal on Embedded Systems*, vol. 2008, 2008, 2 pages.

[34] T. Grötker, S. Liao, G. Martin, and S. Swan, *System Design with SystemC*. Kluwer Academic Publishers, 2002.

[35] D. Harel and P. S. Thiagarajan, "Message sequence charts," in *UML for Real: Design of Embedded Real-Time Systems*, L. Lavagno, G. Martin, and B. Selic, Eds. Kluwer Academic Publisher, 2003, pp. 77–105.

[36] F. Herrera and E. Villar, "A framework for heterogeneous specification and design of electronic embedded systems in SystemC," *ACM Transactions on Design Automation of Electronic Systems*, vol. 12, no. 3, pp. 1–31, 2007.

[37] C. Hewitt, P. Bishop, and R. Steiger, "A universal modular ACTOR formalism for artificial intelligence," in *Proc. of the 3rd international joint conference on Artificial intelligence*, 1973, pp. 235–245.

[38] W. Hwu, K. Keutzer, and T. G. Mattson, "The concurrency challenge," *IEEE Design & Test of Computers*, vol. 25, no. 4, pp. 312–320, 2008.

[39] *IEEE Standard 1076.1: VHDL Analog and Mixed-Signal Extensions*, IEEE, 1999.

[40] *IEEE Standard for VHDL Register Transfer Level (RTL) Synthesis*, IEEE Computer Society, 2004.

[41] *IEEE Standard SystemC® Language Reference Manual*, IEEE Computer Society, 2006. [Online]. Available: http://standards.ieee.org/getieee/1666/download/1666-2005.pdf

[42] *IEEE Standard for System Verilog – Unified Hardware Design, Specification, and Verification Language*, IEEE Computer Society, 2009.

[43] L. S. Indrusiak and O. M. dos Santos, “Fast and accurate transaction-level model of a wormhole network-on-chip with priority preemptive virtual channel arbitration,” in *Design, Automation & Test in Europe*, March 2011.

[44] L. S. Indrusiak and M. Glesner, “SoC specification using UML and actor-oriented modeling,” in *International Baltic Electronics Conferenc*, October 2006, pp. 1–6.

[45] ——, “Specification of alternative execution semantics of UML sequence diagrams within actor-oriented models,” in *The 20th Annual Symposium on Integrated Circuits and Systems Design*, September 2007, pp. 330–335.

[46] L. S. Indrusiak, L. Ost, L. Möller, F. Moraes, and M. Glesner, “Applying UML interactions and actor-oriented simulation to the design space exploration of networks-on-chip interconnects,” in *IEEE Computer Society Annual Symposium on VLSI*, April 2008, pp. 491–494.

[47] L. S. Indrusiak, L. Ost, F. G. Moraes, S. Määttä, J. Nurmi, L. Möller, and M. Glesner, “Evaluating the impact of communication latency on applications running over on-chip multiprocessing platform,” in *International Conference on Industrial Electronics*, July 2010, pp. 148–153.

[48] L. S. Indrusiak, A. Thuy, and M. Glesner, “Executable system-level specification models containing UML-based behavioral patterns,” in *Design, Automation & Test in Europe*, April 2007, pp. 1–6.

[49] 2004 update Design. International Technology Roadmap for Semiconductors. [Online]. Available: http://www.itrs.net/Links/2004Update/2004_01_Design.pdf

[50] 2009 edition Design. International Technology Roadmap for Semiconductors. [Online]. Available: http://www.itrs.net/Links/2009ITRS/2009Chapters_2009Tables/2009_Design.pdf

[51] 2009 edition Modeling & simulation. International Technology Roadmap for Semiconductors. [Online]. Available: http://www.itrs.net/Links/2009ITRS/2009Chapters_2009Tables/2009_Modeling.pdf

[52] A. Jalabert, S. Murali, L. Benini, and G. D. Micheli, "xpipesCompiler: A tool for instantiating application specific networks on chip," in *Design, Automation and Test in Europe*, 2004, pp. 884–889.

[53] A. Jantsch, *Modeling Embedded Systems and SoCs – Concurrency and Timing in Models of Computation*. Morgan Kaufman Publishers, 2003.

[54] A. Jantsch and I. Sander, "Models of computation and languages for embedded system design," *IEE Proceedings of Computers and Digital Techniques*, vol. 152, no. 2, pp. 114–129, 2005.

[55] A. Jantsch and H. Tenhunen, Eds., *Networks on Chip*. Kluwer Academic Publishers, 2003.

[56] K. Keutzer, S. Malik, R. Newton, J. Rabaey, and A. Sangiovanni-Vincentelli, "System level design: orthogonalization of concerns and platform-based design," *IEEE Transactions on Computer-Aided Design of Circuits and Systems*, vol. 19, no. 12, pp. 1523–1543, 2000.

[57] A. Kienhuis, "Design Space Exploration of Stream-based Dataflow Architectures: Methods and Tools," Ph.D. dissertation, Delft University of Technology, Delft, The Netherlands, 1999.

[58] B. Kienhuis, E. Deprettere, K. Vissers, and P. van der Wolf, "An approach for quantitative analysis of application-specific dataflow architectures," in *Proc. IEEE International Conference on Application-Specific Systems*, July 1997, pp. 338–349.

[59] T. Kogel, R. Leupers, and H. Meyr, *Integrated System-Level Modeling of Networks-on-Chip Enabled Multiprocessor Platforms.* Springer, 2006.

[60] P. Kukkala, J. Riihimäki, M. Hännikäinen, T. D. Hämäläinen, and K. Kronlöf, "UML 2.0 profile for embedded system design," in *Design, Automation and Test in Europe*, 2005, pp. 710–715.

[61] E. A. Lee, S. Neundorffer, and M. J. Wirthlin, "Actor-oriented design of embedded hardware and software systems," *Journal of Circuits, Systems, and Computers*, vol. 12, no. 3, pp. 231–260, 2003.

[62] E. A. Lee and A. Sangiovanni-Vincentelli, "A framework for comparing models of computation," *IEEE Transaction on Computer Aided Design of Integrated Circuits and Systems*, vol. 17, no. 12, pp. 1217–1229, 1998.

[63] P. Lieverse, P. van der Wolf, E. Deprettere, and K. Vissers, "A methodology for architecture exploration of heterogeneous signal processing systems," in *Proc. IEEE Workshop on Signal Processing Systems*, 1999, pp. 181–190.

[64] S. Määttä, L. S. Indrusiak, L. Ost, L. Möller, M. Glesner, F. G. Moraes, and J. Nurmi, "Characterising embedded applications using a UML profile," in *International Symposium on System-on-Chip*, October 2009, pp. 172–175.

[65] ——, "A case study of hierarchically heterogeneous application modelling using UML and Ptolemy II," in *International Symposium on System-on-Chip*, September 2010, pp. 68–71.

[66] S. Määttä, L. S. Indrusiak, L. Ost, L. Möller, J. Nurmi, M. Glesner, and F. Moraes, "Validation of executable application models mapped onto network-on-chip platforms," in *3rd International Symposium on Industrial Embedded Systems*, June 2008, pp. 118–125.

[67] S. Määttä, L. Möller, L. S. Indrusiak, L. Ost, M. Glesner, J. Nurmi, and F. Moraes, "Joint validation of application models and multi-abstraction network-on-chip platforms," *International Journal of Em-*

bedded and Real-Time Communication Systems, vol. 1, no. 1, pp. 85–100, 2010.

[68] G. Martin, "UML for embedded systems specification and design: motivation and overview," in *Proc. Conference on Design, Automation and Test in Europe*, 2002, pp. 773–775.

[69] P. Marwedel, *Embedded System Design*. Springer, 2006.

[70] Model-Based Design – MATLAB & Simulink Solutions. The MathWorks. [Online]. Available: http://www.mathworks.com/model-based-design/

[71] A. Mihal and K. Keutzer, "Mapping concurrent applications onto architectural platforms," in *Networks on Chip*, A. Jantsch and H. Tenhunen, Eds. Kluwer Academic Publishers, 2003, pp. 39–59.

[72] A. Mihal, C. Kulkarni, M. Moskewicz, M. Tsai, N. Shah, S. Weber, Y. Jin, K. Keutzer, K. Vissers, C. Sauer, and S. Malik, "Developing architectural platforms: a disciplined approach," *IEEE Design & Test of Computers*, vol. 19, no. 6, pp. 6–16, 2002.

[73] L. Möller, L. S. Indrusiak, and M. Glesner, "NoCScope: a graphical interface to improve networks-on-chip monitoring and design space exploration," in *Design and Test Workshop*, 2009, pp. 1–6.

[74] F. Moraes, N. Calazans, A. Mello, L. Möller, and L. Ost, "HERMES: an infrastructure for low area overhead packet-switching networks on chip," *Integration VLSI Journal*, vol. 38, no. 1, pp. 69–93, 2004.

[75] MDA. Object Management Group. [Online]. Available: http://www.omg.org/mda/

[76] OMG SysML. Object Management Group. [Online]. Available: http://www.omgsysml.org/

[77] UML. Object Management Group. [Online]. Available: http://www.uml.org/

[78] A UML profile for MARTE: Modeling and analysis of real-time embedded systems. Object Management Group. [Online]. Available: http://www.omgmarte.org/Documents/Specifications/08-06-09.pdf

[79] UML profile for schedulability, performance, and time. Object Management Group. [Online]. Available: http://www.omg.org/technology/documents/formal/schedulability.htm

[80] UML profile specifications. Object Management Group. [Online]. Available: http://www.omg.org/technology/documents/profile_catalog.htm

[81] K. Olukotun and L. Hammond, "The future of microprocessors," *Queue*, vol. 3, no. 7, pp. 26–29, 2005.

[82] L. Ost, "Abstract Models of NoC-Based MPSoCs for Design Space Exploration," Ph.D. dissertation, Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre, Brazil, 2010.

[83] L. Ost, G. Guindani, L. S. Indrusiak, S. Määttä, and F. Moraes, "Using abstract power estimation models for design space exploration in NoC-based MPSoCs," *IEEE Design & Test*, vol. 28, pp. 16–29, 2011.

[84] L. Ost, L. S. Indrusiak, S. Määttä, M. Mandelli, J. Nurmi, and F. G. Moraes, "Model-based design flow for NoC-based MPSoCs," in *IEEE International Conference on Electronics, Circuits and Systems*, December 2010.

[85] L. Ost, L. Möller, L. S. Indrusiak, F. Moraes, S. Määttä, J. Nurmi, and M. Glesner, "A simplified executable model to evaluate latency and throughput of networks-on-chip," in *21st Symposium on Integrated Circuits and System Design*, September 2008, pp. 170–175.

[86] H. Patel and S. Shukla, *SystemC Kernel Extension for Heterogeneous System Modeling - A Framework for Multi-MoC Modeling & Simulation*. Kluwer Academic Publishers, 2004.

[87] H. D. Patel and S. K. Shukla, "Towards a heterogeneous simulation kernel for system-level models: a SystemC kernel for synchronous data flow models," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 8, pp. 1261–1271, 2005.

[88] D. A. Patterson and J. L. Hennessy, *Computer Organization & Design: The Hardware/Software Interface*, 2nd ed. Morgan Kaufmann Publishers, Inc., 1998.

[89] A. D. Pimentel, C. Erbas, and S. Polstra, "A systematic approach to exploring embedded system architectures at multiple abstraction levels," *IEEE Transactions on Computers*, vol. 55, no. 2, pp. 99–112, 2006.

[90] A. D. Pimentel, L. O. Hertzberger, P. Lieverse, P. van der Wolf, and E. F. Deprettere, "Exploring embedded-systems architectures with Artemis," *Computer*, vol. 34, no. 11, pp. 57–63, 2001.

[91] J. Plantin and E. Stoy, "Aspects of system-level design," in *Proc. Workshop on Hardware/Software Codesign*, 1999, pp. 209–210.

[92] Ptolemy II CT domain. Ptolemy Project, UC Berkeley, EECS. [Online]. Available: http://ptolemy.berkeley.edu/ptolemyII/ptIIlatest/ptII/ptolemy/domains/ct/doc/index.htm

[93] J. M. Rabaey, A. Chandrakasan, and B. Nikolic, *Digital Integrated Circuits – A Design Perspective*, 2nd ed. Prentice Hall / Pearson Education International, 2003.

[94] ForSyDe: Formal System Design. Royal Institute of Technology, School of Information and Communication Technology. [Online]. Available: http://www.ict.kth.se/forsyde/

[95] I. Sander and A. Jantsch, "System modeling and transformational design refinement in ForSyDe," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, no. 1, pp. 17–32, 2004.

[96] A. Sangiovanni-Vincentelli, "Quo vadis SLD: reasoning about trends and challenges of system-level design," *Proceedings of the IEEE*, vol. 95, no. 3, pp. 467–506, March 2007. [Online]. Available: http://chess.eecs.berkeley.edu/pubs/263.html

[97] A. Sangiovanni-Vincentelli and G. Martin, "Platform-based design and software design methodology for embedded systems," *IEEE Design & Test*, vol. 18, no. 6, pp. 23–33, 2001.

[98] B. Selic, "Models, software models and UML," in *UML for Real: Design of Embedded Real-Time Systems*, L. Lavagno, G. Martin, and B. Selic, Eds. Kluwer Academic Publisher, 2003, pp. 1–16.

[99] S. K. Shukla, G. Smith, and C. Pixley, "Guest editor's introduction: The true state of the art of ESL design," *IEEE Design & Test of Computers*, vol. 23, no. 5, pp. 335–337, 2006.

[100] J. Sztipanovits and G. Karsai, "Model-integrated computing," *IEEE Computer*, vol. 30, no. 4, pp. 110–111, 1997.

[101] Metropolis: Design Environment for Heterogeneous Systems. University of California, Berkeley, The Donald O. Pederson Center for Electronic Systems Design. [Online]. Available: http://embedded.eecs.berkeley.edu/metropolis/

[102] A. Vachoux, C. Grimm, and K. Einwich, "Towards analog and mixed-signal SoC design with SystemC," in *Proc. Workshop on Electronic Design, Test and Applications*, January 2004, pp. 97–102.