Validation of Executable Application Models Mapped onto Network-on-Chip Platforms

Sanna Määttä*, Leandro Soares Indrusiak†, Luciano Ost†‡, Leandro Möller†,
Jari Nurmi*, Manfred Glesner† and Fernando Moraes‡
*Department of Computer Systems
Tampere University of Technology
P.O.Box 553, FIN-33101, Tampere, Finland
Email: [sanna.maatta, jari.nurmi]@tut.fi
†Institute of Microelectronic Systems
Technische Universität Darmstadt
Karlstrasse 15, 64283 Darmstadt, Germany
Email: [lsi, moller, glesner]@mes.tu-darmstadt.de
‡Catholic University of Rio Grande do Sul (PUCRS)
Av. Ipiranga, 6681 - P.32 - 90619-900
Porto Alegre, Brazil
Email: [ost, moraes]@inf.pucrs.br

Abstract— **Due to the increasing design size, complexity, and heterogeneity of today's embedded systems, designers need novel design methods in order to validate application-specific functionality together with different platform implementation alternatives. Ideally, this should happen at as early stage of the design process as possible, so that designers can explore the design space before they have to commit to specific processor architectures or custom hardware implementation. This paper takes advantage of the hierarchical design style and the support for heterogeneous Models of Computation (MoC) existing in actor-oriented frameworks and presents a methodology for modelling and validation of multiprocessor embedded systems. The proposed methodology is fully model-based, with different modelling styles for the application and the underlying implementation platform. In this paper we focus on the validation of applications modelled using Ptolemy II actors and UML sequence diagrams, mapped onto multiprocessor Network-on-Chip (NoC) platforms. We also present a case study, where one executable application model is mapped onto different NoC topologies, and show the simulation results for communication latency of each alternative.**

I. Introduction

The possibility to integrate hundreds of millions of transistors onto a single chip enables the design of such complicated systems that their design and verification is quite challenging. This leads to the need of system modelling, design, and verification at high abstraction levels, that is, at system level. Also the role of system specifications increases: system designers should be able to understand them unambiguously [1]. Even though the software community fights since early 90's whether the system specifications should be executable [2] or not [3], many software and hardware designers consider the executable specifications quite useful [1], [4], [5].

A high level model of a system is always a trade-off between design accuracy and abstraction. For instance, an executable specification often serves only as a behavioural reference of the system, without any accurate information about its performance or area consumption [5]. If a cycle accurate system model already exists, it is possible to annotate timing information extracted from the simulation of the Register Transfer Level (RTL) model into the high level model in order to increase the simulation accuracy.

Software designers often use programming languages such as C or C++ for the creation of executable specifications. Unfortunately, these languages do not apply to hardware design as such. They lack the ability of modelling timing as well as concurrent and reactive behaviour, which are nearly mandatory aspects in hardware design. SystemC seems adequate to transaction level and behavioural modelling of a system, since it supports concurrency, reactivity, and timed modelling of hardware. However, the heterogeneity and complexity of today's embedded systems require modelling with various Models of Computation (MoC) on multiple levels of abstraction [6], [7], which requires extensions to SystemC kernel [8], [9].

In this paper, we present a methodology for modelling and validating multiprocessor embedded systems using actor-orientation and UML. The methodology is fully model-based and advocates the separate modelling of application and the underlying implementation platform. A set of extensions to the actor-oriented framework Ptolemy II were needed in order to support modelling and joint validation of application and platform models under different constraints, mappings, and configurations.

The work this paper describes relies on actor-oriented design methods implemented within the Ptolemy II framework, which aims to facilitate heterogeneous design with multiple MoCs. We also take advantage of richer modelling constructs by integrating UML 2.0 notation within actor-oriented models. Even though UML does not provide execution semantics, our approach builds on previous work that allows designers to

978-1-4244-1995-1/08/$25.00 ©2008 IEEE.

assign specific MoCs to each UML diagram embedded within an actor-oriented model [10].

The rest of the paper is organized as follows: Section II describes related work, Section III covers high level modelling issues relevant to the scope of this paper, and Section IV details the NoC platform used in this paper. Section V explains how the application and platform models are constructed and jointly validated within Ptolemy II. Furthermore, Section VI presents such methodology within a case study. Finally, Section VII draws the conclusions.

II. Related Work

Using UML in high level modelling usually requires design automation through code generation. For example, Riccobone et al. present executable SystemC code generation from UML diagrams [11]. They describe a SoC design methodology and model a system with UML and then translate it into executable SystemC model. Further code generation issues are described in [12] and [13]. Arpinen et al. present a configurable multiprocessor platform for distributed execution of applications described in UML 2.0 [12]. They map the UML applications onto the platform by distributing WLAN medium access control protocol to multiple processor cores. They use design automation from UML to SoC implementation. In fact, to support automatic code generation, they use UML statechart diagrams and their formalism. Damaševičius et al. discuss how to apply design patterns in hardware design [13]. They propose a hardware design pattern, Wrapper, for adapting IP interface and behaviour to the context. To generate hardware patterns, they use a metaprogramming-based technique for code generation.

As far as we know, there are no other approaches of modelling and validating applications on a NoC platform that are fully model-based using actors and UML diagrams. However, other researchers have employed similar techniques or addressed similar problems, such as design space exploration of MPSoCs [14], embedded system mapping onto a NoC [15], application execution optimisation [16], or Quality-of-Service (QoS) guarantees of an application when running it on a platform [17].

Dumitrascu et al. present a flexible Multiprocessor System-on-Chip (MPSoC) platform for fast and accurate design space exploration [14]. Their aim is to evaluate the performance of a given application with different network configurations. Palma et al. concentrate on mapping of embedded systems onto a NoC in [15]. Their work highlights the importance of the network traffic modelling when optimising the platform and minimising for example the latency and power dissipation of complex SoCs. They present that design space exploration for finding an optimal platform for an application requires efficient mapping strategies and accurate application models.

Furthermore, Marcon et al. present a new model, which is based on communication dependence model for capturing also the dynamic behaviour of applications when trying to minimize their execution time and energy consumption [16]. Murali et al. present an approach of mapping cores onto NoC topologies considering the physical planning of NoCs and guaranteeing QOS at the same time [17].

III. High Level Modelling

In this section we present a few high level modelling concepts that are part of the foundations of the work this paper describes.

A. UML

UML is a standardized modelling and specification language for modelling not only applications but also business process and data structures [18]. UML offers various graphical notations for design specification, visualisation, and description of structural, behavioural, and physical characteristics of a system. UML supports object oriented system design and it is therefore popular among software designers. In fact, it is getting more acceptance also among the hardware and embedded system community [6].

UML 2.0 supports extension mechanisms, such as stereotypes, constraints, and tagged values, and it is possible to apply it into particular application and platform domains. However, UML lacks execution semantics, which leads to the use of UML as a system-level specification language for either static analysis or code generation methods [19].

Within this paper, we concentrate only on UML sequence diagrams. Fig. 1 shows different sequence diagrams related to an autonomous vehicle application, which we use as a case study later in this paper. The vertical lines correspond the lifelines of each instance, whereas the horizontal arrows connecting two lifelines represent the messages they exchange in the order they occur. That is, the lifelines also capture temporal behaviour. Such behaviour can be interpreted either as a partial or total order, which is enough for the definition of an untimed model of computation, as explored in [10].

B. Object-Oriented and Actor-Oriented Design

Object oriented design methods can be applied also in hardware design [20], as can be seen for instance in NoC modelling and simulation frameworks [21] and SoC design processes [22]. However, whereas object oriented design methods and languages lack concurrency, which inherently belongs to hardware, actor oriented design derives from mathematical model of concurrent computation [23] and later formal models of concurrency [24]. Actors are components, which communicate via channels using message passing instead of function calls like objects do. Actors contain a well-defined interface, which includes ports through which the actors can communicate [25]. However, it is the model of computation that defines whether each actor runs concurrently – conceptually or in its own thread – or if the whole model runs sequentially [7].

C. Ptolemy II Framework

As discussed earlier in this paper, the heterogeneity and complexity of embedded systems require the use of various MoCs for different subsystems [6], [7]. The Ptolemy II framework [26] supports both actor-oriented and hierarchical

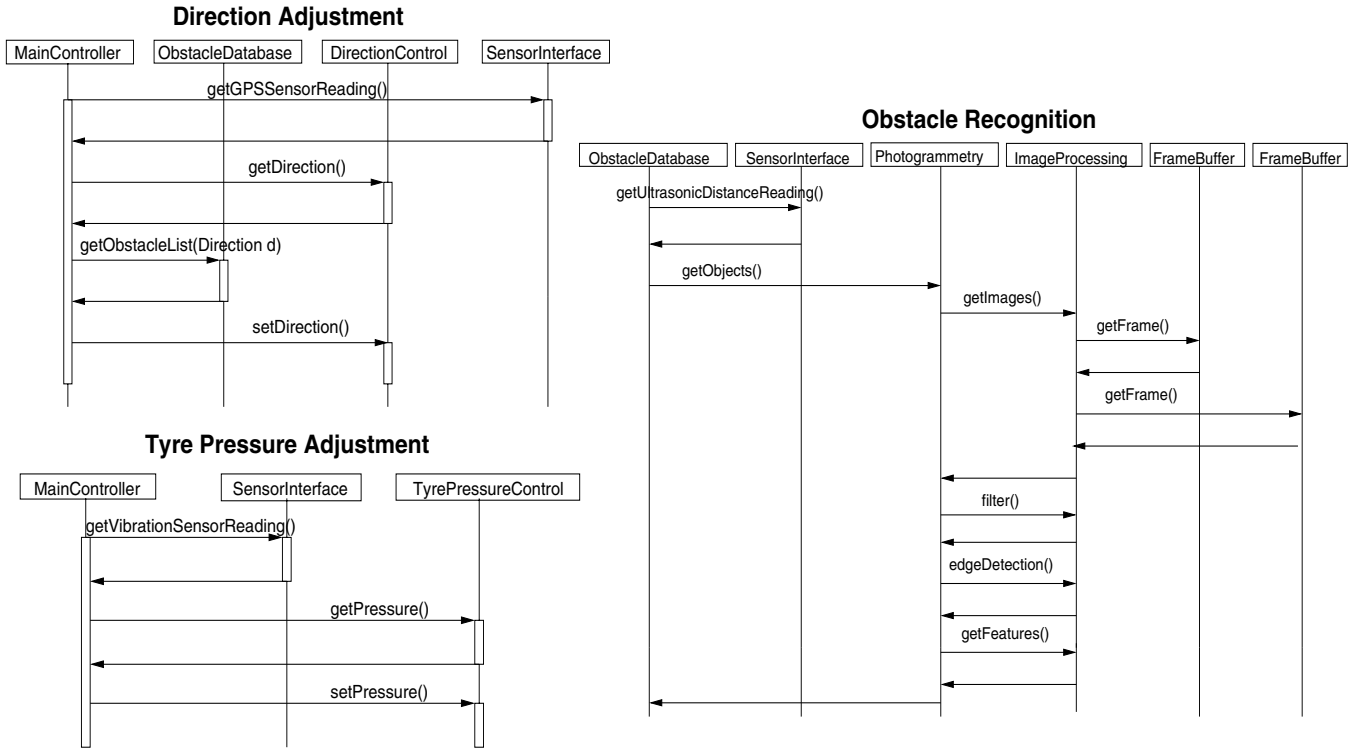

Fig. 1. UML sequence diagrams of an autonomous vehicle application

heterogeneous design with multiple MoCs for different subsystems. Ptolemy II includes various MoCs, such as Communicating Sequential Processes, Continuous Time, Discrete Event, Process Networks, and Synchronous Dataflow [7]. Moreover, through extensions to the Ptolemy II framework kernel, user interface, and director library, designers can also develop their own execution semantics for the needs of each subsystem [19].

Within Ptolemy II framework, actors can be either atomic or a composition of other actors. Actors are executable and contain three execution phases: setup, iteration, and wrapup. The setup phase contains both pre-initialization and initialization phases, which are executed before the iteration phase. The iteration includes prefire, fire, and postfire phases, during which the actor execution is performed. Finally, a wrapup phase ends the execution and releases all resources required during the execution.

An actor always belongs to a domain, which defines both a receiver and a director. A receiver can be for instance a FIFO queue or a mailbox and is present at each input port of every actor. A director is the element that controls the concurrent execution of all actors in the domain, defining precisely when they can communicate and update their internal state [7], [26].

IV. HERMES NETWORK-ON-CHIP

HERMES NoC is a packet-switched on-chip interconnection architecture, which consists of low area network switches. A switch is the basic element of the network and it can be connected to up to four other switches and a local IP core. The switches have been implemented to build 2D-mesh and related network topologies. Each switch contains routing controller logic and five bi-directional ports. The routing controller logic implements XY-routing strategy, arbitration logic, and a packet-switching algorithm with wormhole switching mode. The five ports establish the connection to four neighboring switches and to a local core. Each port uses input buffering for incoming packets. According to the wormhole switching strategy, packets are divided into entities called flits. The first and second flit of a packet form the header and contain the target switch of the packet and the size of the payload. Then the rest of the flits of a packet contain the payload. The switches are described using VHDL and prototyped on FPGA [27].

RENATO is an actor-oriented NoC model, implemented using Java within the Ptolemy II framework [28]. RENATO is a high-level model based on interactions, which derive from the RTL model of HERMES. Those interactions describe the behavioural patterns (that is, sequence diagrams) present on the RTL model and cover the whole functionality of a HERMES switch, from the arrival of a flit on the input buffer to the time when the flit is forwarded to the input buffer of the next switch.

Originally, RENATO was implemented without any timing information. In order to increase the model's accuracy, timing information extracted from the cycle-accurate HERMES model was annotated within RENATO's interactions. Additional details on RENATO's implementation and accuracy can be found in [28].

V. Joint Validation of Application and NoC Platform

A. Executable Application Models with Actors and Behavioural Patterns Defined by UML Sequence Diagrams

The approach this paper presents for executable application models using actors and UML sequence diagrams is based on work described in [10] and [19]. That work proposed the explicit definition of the communication among actors through behavioural patterns, which can be represented by UML diagrams. In order to make that kind of UML-based models executable, they were integrated within opaque composite actors in Ptolemy II. The upper part of Fig. 2 depicts such an application model.

According to [19], the following statements define the encapsulation of a sequence diagram within an actor in order to define a behavioural pattern:

- Like all opaque composite actors, also the ones encapsulating sequence diagrams (actors X and Y in Fig. 2) must have a director that defines the domain and must export ports to allow the communication with other actors.
- Each lifeline of a sequence diagram (named with Greek characters in Fig. 2) represents one participant of the pattern (actors named with letters A to G in Fig. 2).
- Each message between lifelines represent the communication between two participants of the pattern, which in turn is the communication between two actors of the application model. For each asynchronous message, an input and an output port are created on the opaque composite actor that encapsulates the diagram. For synchronous messages, two additional ports must be created to represent its return path (Fig. 2 shows only asynchronous messages).
- The director of the opaque composite actor ensures that the delivery of the messages follows the ordering, which is defined on the diagram, and the receivers of each port can be configured with the desired buffering behaviour (for instance bounded or unbounded FIFOs or a mailbox).

Thus, each of the actors A to G encapsulate some sequential computation that is a part of the designed application. They communicate with each other by exchanging tokens through input and output ports. Their communication must follow a behavioural pattern (defined by the encapsulated sequence diagrams). Director D1 defines the MoC that rules the concurrent execution of all actors of the application model, but since X and Y are opaque they must include their own directors (DX and DY). When D1 triggers the execution of X or Y (usually because other actors are communicating with them), their internal director determines the order of forwarding tokens from inputs to outputs according to the sequence diagram they encapsulate. Furthermore, every time X or Y are executed, they trigger the delivery of a message if that message has a token on their respective input port, and if all messages preceding it have already been previously triggered. It is possible to interpret the precedence order of messages in various ways and the application developer needs to choose the most suitable ones for his or her purposes [10].

B. Validation of the Application Model and the NoC Platform

According to the assumptions of [19] described in the previous subsection, an actor that encapsulates an UML sequence diagram is used to mediate the communication among other actors in an application model. The cost of the communication among actors is partially abstracted by that approach, as it only introduces delays on the communication if the tokens sent by the communicating actors arrive in an order that is not compatible with the provided sequence diagram. If all messages arrive at the predicted order, they are forwarded instantaneously and the communication has zero latency. In order to account for communication costs that are due to the communication media (which are likely to incur even if all messages are properly ordered), we need additional modelling constructs.

A first approach to model such communication costs is to annotate a delay for each message on the sequence diagram itself, in a similar way as it is proposed in the UML Profile for Schedulability, Performance, and Time Specification [29]. This kind of an approach could provide an initial verification of the communication behaviour and the designer can estimate the complete communication costs.

One of the goals this paper addresses is to achieve additional accuracy on the estimation of the communication costs for each behavioural pattern. Therefore, we propose a joint validation of application and platform. To do that, the hardware/software platform, which will be implemented to support the inter-actor communication, must be modelled in enough detail so that it can provide accurate figures for the communication latency. As stated before, this paper reports the use of such approach for the specific case of multiprocessor embedded systems based on NoCs, and thus the platform model must include all pertinent architectural details that affect the latency on such interconnects, including topology, routing algorithm, buffering strategy, arbitration, flow control, and switching scheme. The chosen platform model used on this paper is RENATO, an actor-oriented model of the HERMES NoC described briefly in Section IV. We used different network configurations for RENATO, such as 2x4, 3x3, 3x4 and 4x4 mesh topologies. The lower part of Fig. 2 represents the platform model.

In order to perform a joint validation of application and platform, in other words, to simulate each inter-actor communication of the application model on a platform, it is necessary to map the actors of the application model onto actors of the platform model. In this paper, the mapping process assigns the lifelines of the sequence diagrams of the application model (lifelines α to λ in Fig. 2) to processing elements attached to the NoC at the platform model (processing elements 1 to 9 in Fig. 2). A mapper, which is depicted between application and platform models in Fig.2 executes this mapping process.

For each message triggered within a behavioural pattern at the application level (for instance, the message sent by

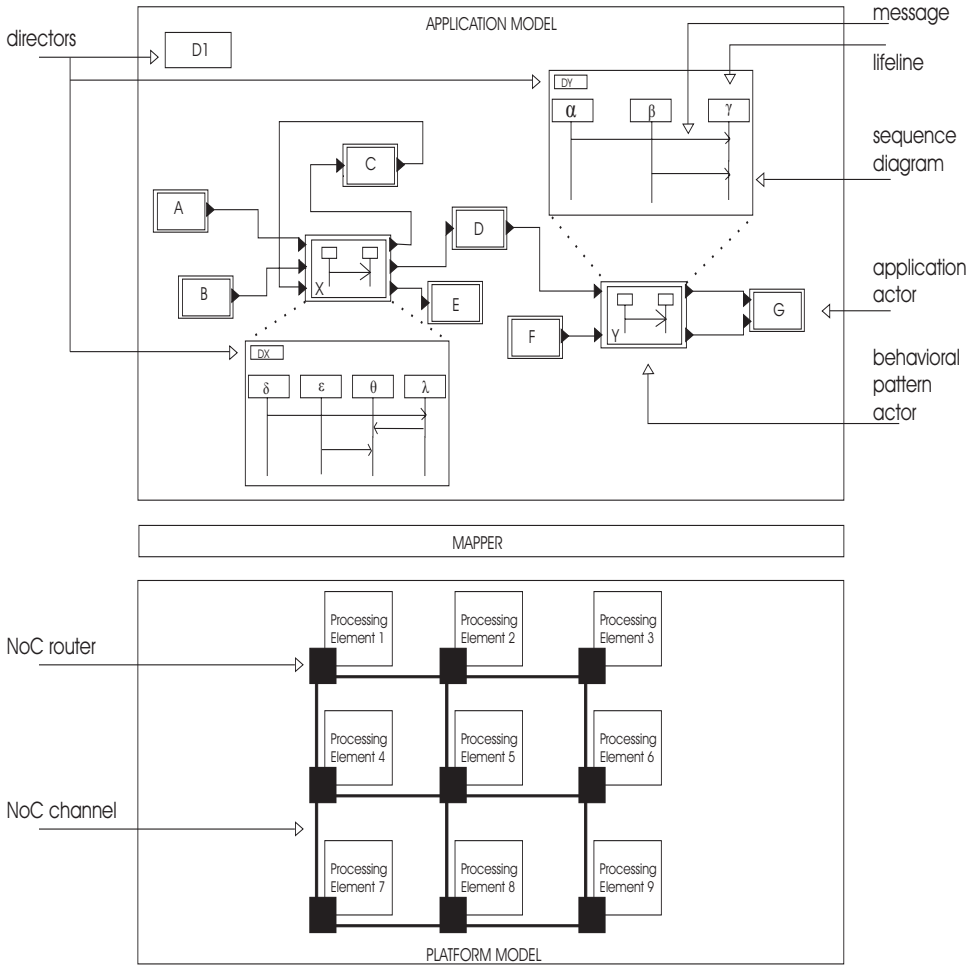

Fig. 2. Modelling constructs of the proposed methodology

lifeline α to lifeline γ within actor Y), the corresponding director (DY in this case) interrupts the delivery and notifies the mapper about the message. Since the mapper is responsible for assigning each lifeline to a platform processing element, it knows that for instance lifeline α is mapped to processor 2, whereas γ is mapped to 7. Once the mapper receives the information about the triggered message, it will command the processing element associated to the sender of the message (processor 2) to generate the corresponding traffic into the interconnect structure (in the case of RENATO, it must create a package with destination, size and payload and write this package flit by flit on the local input port of the corresponding router). Then the mapper waits until the processing element associated to the receiver of the message (processor 7) notifies the complete receipt of the package. Upon notification, the mapper calls back the director that triggered the message (DY) and informs that the message can now be delivered. After that, the director can forward the message to the output port of the behavioural pattern actor, and the message reaches its destination with the exact latency as it would take if the application was executed on top of the implementation platform.

As the mapper implementation has critical influence on the communication costs, the approach this paper presents considers the choice of a mapper as a design decision. Thus, the implemented approach is extensible, so that we can create a library of different mapping algorithms and heuristics. However, this is not within the scope of this paper. The current mapper implementation has no constraints (each application actor can be mapped to any processing element), produces mappings randomly, and supports only static mapping. These are not restrictions of the modelling approach, and future work will explore the possibility of different static and dynamic mapping heuristics, as well as the sharing of processing elements by multiple application actors.

VI. Case Study

In order to test the proposed modelling and validation methodology, we used Ptolemy II to model an application and map it onto a number of platform configurations within our case study. Fig. 4 shows how Ptolemy II models all three parts of the methodology that Fig. 2 depicts: application, mapper, and platform. A RENATO model configured to implement a 3x3 mesh network topology defines the platform. Actors

representing processing elements are attached to each node of the mesh (two actors per processing element, one producer and one consumer). Fig. 4 also illustrates some additional features of RENATO, such as the BufferScope on the lower right corner, graphically displaying the buffer occupation of each switch during the simulation. The mapper, which statically maps each lifeline to a processing element, is presented as a parameter of the model. Finally, actors encapsulating sequential computation and UML sequence diagrams on the upper right corner present the application model.

Fig. 1 details the behavioural patterns that compose the application model. They describe three different parts of the behaviour of an autonomous vehicle. The first pattern covers obstacle recognition. The modelled vehicle has two cameras that capture images on the direction the vehicle is moving. Such images are pre-processed and fed to a photogrammetry subsystem that is able to extract the coordinates of the obstacles ahead. The obstacles are fed to a database. The vehicle also has an ultrasonic sensor, which can estimate with more precision the distance of obstacles that are right in front of the vehicle. It uses such a sensor to finetune the entries on the obstacle database as the vehicle gets closer to them. The vehicle can also adjust its tyre pressure, as described in the second pattern. This is done according to the information from a vibration sensor, so that the vehicle can better adapt to the surface on which it is moving and minimize vibrations. Finally, the third pattern models the adjustment of the direction the vehicle is moving. This is done by analysing the vehicle's current position (read from a GPS sensor), its current direction and the results from a query to the obstacle database.

According to application-specific constraints, each pattern must be executed at a particular rate. For instance, the obstacle recognition pattern must be executed every 0,4 seconds, whereas the direction adjustment happens every second and the tyre pressure control every 2 seconds. For this case study, we fixed the operation frequency of the platform to 50 MHz and simulated the system for a period of 18 seconds. In order to accurately capture the behaviour of the interconnect, we used a cycle-accurate version of RENATO and simulated nine hundred million cycles. With this set up, the obstacle recognition pattern was executed 45 times, direction adjustment pattern 18 times and tyre pressure control pattern 9 times.

We measured the network latency for each message, that 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. Each message of the patterns corresponds to packets of different sizes, as a message carrying raw data from a camera to the image processing is much larger than, for instance, an ultrasonic sensor reading. Furthermore, we calculated the average latency of all packets in one run for each pattern. Table I illustrates the latency for each pattern using 2x4, 3x3, 3x4, and 4x4 mesh topologies. We used two different random mappings for the 3x3 mesh topology and one random mapping for each other topology.

Different topologies have an effect on each pattern. The

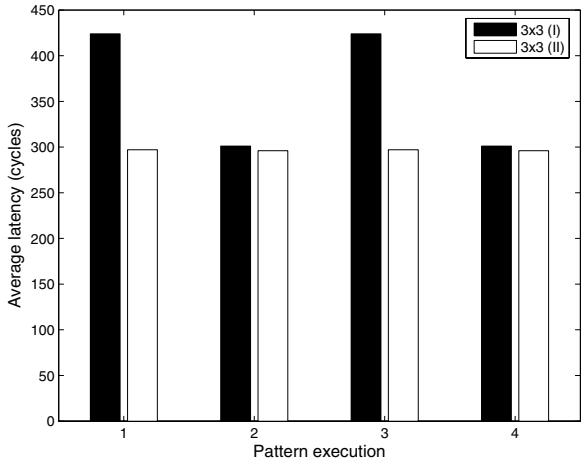

Fig. 3. Latencies of the direction adjustment pattern with different mappings onto a 3x3 mesh topology

TABLE I
NETWORK LATENCIES FOR EACH PATTERN WITH DIFFERENT NOC TOPOLOGY

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

obstacle recognition pattern carries most data, which explains its long latencies. However, the tyre pressure adjustment pattern is executed least frequently and most likely, its tokens need to wait longer for the routing since the two other patterns already occupy the routers at the same time. 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 happen to be mapped into neighbouring switches, which decreases the network latency by decreasing the length of the path from the traffic producer to the consumer. Fig. 3 illustrates the average latency of packets of the direction adjustment pattern at the four first run of it. The latency of the first mapping is either 424 or 301 and of the second mapping either 297 or 296. In case of mapping I, the longer latency of every second pattern (424) is caused by the simultaneous execution of the tyre pressure adjustment pattern, since those patterns use same processing elements and only one pattern at a time can have access to them while the other one must wait.

VII. CONCLUSIONS

This paper presented a methodology that supports a model-based joint validation of application and platform using UML and actor orientation. It focused on platforms that are based on NoC interconnects and provided means to model a given application and map it onto different configurations of the

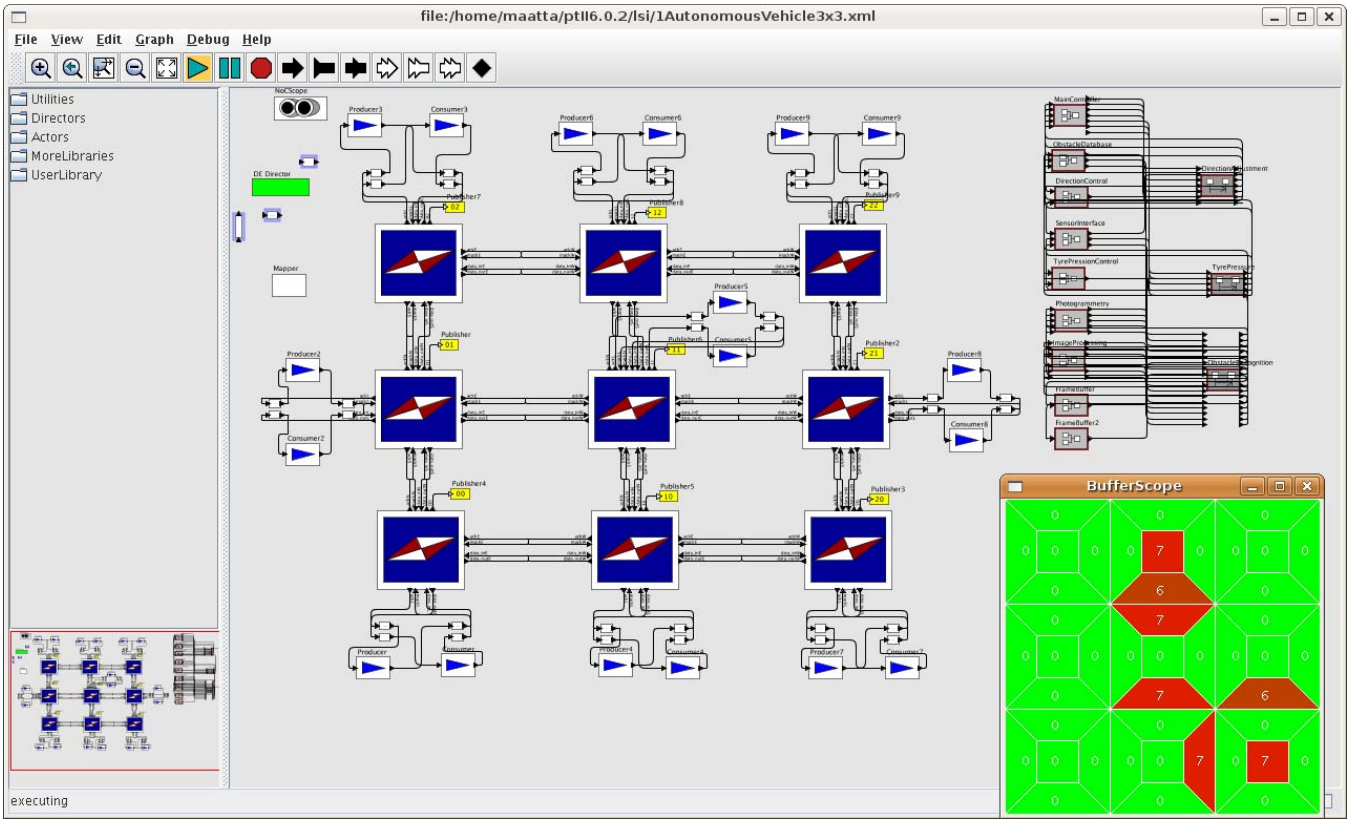

Fig. 4. A 3x3 NoC configuration with buffer occupation

platform, aiming at evaluating which is the most suitable one for the given constraints. The paper also presented a case study, which uses a customized version of the Ptolemy II framewok, including the modelling of an autonomous vehicle application, which was in turn mapped onto different configurations of the RENATO platform. The joint simulation of application and platform produced resulted for the communication latency, which can give designers an early yet accurate estimation of the communication costs. With such figures, designers can rule out platform configurations that are not likely to meet the application constraints and concentrate on those that have more promising results.

Future work includes improved application modelling through the support to UML 2.0 constructs for sequence diagrams and also improved mapping through the exploration of optimal mapping heuristics as well as dynamic mapping.

REFERENCES

[1] Gerlach J. and Rosentiel W., System level design using the SystemC modeling platform, In Proc. *3rd Workshop on System Design Automation*, Germany, March 2000, pp. 185-189.

[2] Fuchs N.E., Specifications are (preferably) executable *Software Engineering Journal*, vol 7, no. 5, 1992, pp. 323-334.

[3] Hayes I.J. and Jones C.B., *Specifications are not (necessarily) executable*, Technical Report 148, Key Centre for Software Technology, Department of Computer Science, University of Queensland, Australia, 1990

[4] Harel D., Biting the silver bullet: toward a brighter future for system development, in *Computer*, vol 25, no. 1, 1992, pp. 8-20.

[5] Virtanen, S., Truscan D., Määttä S., Westerlund T., Isoaho J., and Nurmi J., High-level simulation models, in *Processor Design: System-on-Chip Computing for ASICs and FPGAs*, Nurmi J., Ed. Kluwer Academic Publishers / Springer Publishers, 2007, ch. 18.

[6] Martin G., UML for embedded systems specification and design: motivation and overview in Proc. *Design, Automation and Test in Europe Conference and Exhibition*, France, March 2002, pp. 773-775.

[7] Eker J., Janneck J.W., Lee E.A., Liu J., Liu X., Ludvig J., Neundorffer S., Sachs S., Xiong Y., Taming heterogeneity - the Ptolemy approach, in Proc. *IEEE*, vol. 91, no. 1, 2003, pp. 127-144.

[8] Patel H.D. and Shukla S.K., *SystemC Kernel Extension for Heterogeneous System Modeling*, Kluwer Academic Publisher, 2004.

[9] Patel H.D. and Shukla S.K., Simulation Kernel for System Level Models: A SystemC Kernel for Synchronous Data Flow Models, in Proc. *IEEE Transactions in Computer-Aided Design*, vol, 24, no. 8, pp. 1261-1271.

[10] Indrusiak L.S. and Glesner M., Specification of alternative execution semantics of UML sequence diagrams within actor-oriented models, in Proc. *the 20th annual conference on Integrated circuits and systems design*, Brazil, September 2007, pp. 330-335.

[11] Riccobone E., Scandurra P., Rosti A., and Bocchio S., A SoC design methodology involving a UML 2.0 profile for SystemC, in Proc. *Conference on Design, Automation and Test in Europe*, Germany, March 2005, pp. 704-709.

[12] Arpinen T., Kukkala P., Salminen E., Hännikäinen M., and Hämäläinen T.D., Configurable multiprocessor platform with RTOS for distributed execution of UML 2.0 designed applications, in Proc. *Conference on Design, Automation and Test in Europe*, Germany 2006, pp. 1324-1329.

[13] Damaševičius R., Majauskas G., Štuikys V., Application of design patterns for hardware design, in Proc. *40th conference on Design automation*, Anaheim, USA, 2003, pp. 48-53.

[14] Dumitrascu F., Bacivarov I., Pieralisi L., Bonaciu M., and Jerraya A.A., Flexible MPSoC platform with fast interconnect exploration for optimal system performance for a specific application, in Proc. *Conference on Design, Automation, and Test in Europe, DATE '06*, Germany, March 2006, pp. 166-171.

[15] Palma J.C.S., Marcon C.A.M., Moraes F.G., Calazans N.L.V., Reis R.A.L., and Susin A.A., Mapping embedded systems onto NoCs: the traffic effect on dynamic energy estimation, in Proc. *18th Annual Symposium on Integrated Circuits and System Design*, Brazil 2005, pp. 196-201.

[16] Marcon C., Borin A., Susin A., Carro L., and Wagner F., Time and energy efficient mapping of embedded applications onto NoCs, in Proc. *2005 Conference on Asia South Pacific Design Automation*, China, 2005, pp. 33-38.

[17] Murali S., Benini L., and De Micheli G., Mapping and physical planning of networks-on-chip architectures with quality-of-service guarantees, in Proc. *2005 Conference on Asia South Pacific Design Automation*, China, 2005, pp. 27-32.

[18] Object Management Group UML http://www.uml.org/ (Referenced 02/2008).

[19] Indrusiak L.S., Thuy A., and Glesner M., Executable system-level specification models containing UML-based behavioral patterns, in Proc. *Design, Automation and Test in Europe Conference and Exhibition, 2007. DATE '07*, France, April 2007, pp. 1-6.

[20] Nebel W. and Schumacher G., Object-oriented hardware modelling - Where to apply and what are the objects? In Proc. *EURO-DAC with EURO-VHDL'96*, Switzerland, September 1996, pp. 428-433.

[21] Coppola M., Curaba S., Grammatikakis M.D., Maruccia G., and Papariello M., OCCN: A Network-on-Chip modeling and simulation framework, in Proc. *Conference on Design, Automation and Test in Europe*, February 2004, pp. 174-179.

[22] Zhu Q., Matsuda A., Kuwamura S., Nakata T., and Shoji M., An object-oriented design process for system-on-chip using UML, in Proc. *15th International Symposium on System Synthesis*, Japan 2002, pp. 249-254.

[23] Hewitt C., Bishop P., and Steiger R., A universal modular actor formalism for artificial intelligence, *IJCAI III*, Stanford, USA, August 1973, pp. 235-245.

[24] Agha G.A., *ACTORS: A model of concurrent computation in distributed systems*, The MIT Press Series in Articial Intelligence, MIT Press, Cambridge, 1986.

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

[26] Brooks C., Lee E.A., Liu X., Neuendorffer S., Zhao Y., Zheng H. (eds.), *Heterogeneous concurrent modeling and design in Java (Volume 1: Introduction to Ptolemy II)*, EECS Department, University of California, Berkeley, UCB/EECS-2007-7, January 2007.

[27] Moraes F., Calazans N., Mello A., Möller L., and Ost L, HERMES: an infrastructure for low area overhead packet-switching networks on chip, In *Integration VLSI Journal*, Vol. 38, Issue 1, October 2004, pp. 69-93.

[28] Indrusiak L.S., Ost, L., Möller L., Moraes, F., and Glesner M., Applying UML interactions and actor-oriented simulation to the design space exploration of network-on-chip interconnects in Proc. *IEEE Computer Society Annual Symposium on VLSI*, France, April 2008.

[29] Object Management Group UML Profile for Schedulability, Performance, and Time Specification. Version 1.1. 2005. http://www.omg.org/docs/formal/05-01-02.pdf (Referenced 02/2008).