Evaluating the impact of communication latency on applications running over on-chip multiprocessing platforms: a layered approach

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

[1] Real-Time Systems Group – Computer Science Department - University of York – United Kingdom
[2] Informatics Department - Catholic University of Rio Grande do Sul (PUCRS) – Porto Alegre – Brazil
[3] Department of Computer Systems – Tampere University of Technology – Finland
[4] Institute of Microelectronic Systems – Darmstadt University of Technology – Germany

***Abstract*—The evaluation of communication latency in multiprocessor platforms requires the profiling of the application, the description of the architecture of the platform and of the mapping of application tasks onto processing cores. In this paper, we describe a layered approach that allows application developers to obtain accurate figures for communication latency from an abstract model of the application functionality. A complete separation of concerns is a critical part of this approach, so the major contribution here is the definition of interfaces between different layers: an abstract application model, its executable counterpart, the mapping heuristic and the multiprocessor platform model. Case studies with a realistic application and a NoC-based multiprocessor platform show the potential of the proposed approach using two different system evaluation techniques: simulation and static analysis.**

Keywords-on-chip multiprocessing; ESL design; UML; actors.

I. INTRODUCTION

One of the major problems faced by embedded application developers is to check whether or not a given implementation meets its timing and performance constraints. For instance, unpredictable delays on memory access will cause image flickering on a video player, resulting in customer dissatisfaction. Even worse, the failure to meet constraints in hard real-time applications such as automotive systems or power plant controllers can have catastrophic consequences.

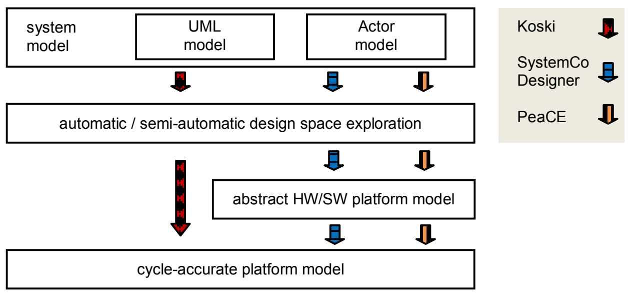
This problem is much worse when programming embedded multi-core platforms because of the increasing impact of inter-task communication latency, which in turn affects system timeliness and performance. The complexity of on-chip interconnect structures, often based on networks-on-chip (NoC) or hierarchical buses, prevents application developers from having a clear picture of the underlying sources of latency. This also restricts the use of their domain-specific knowledge to try out optimisations such as alternative task mapping and scheduling policies.

In this paper, we present an approach that supports application developers when evaluating the impact of on-chip interconnects on inter-task communication latency. Unlike similar techniques that deal with communication latency in a generic way, the proposed approach considers a model of each specific application and provides latency metrics for each individual inter-task communication instance. Such metrics can be as accurate as needed, taking into account low-level interconnect issues such as arbitration, link contention and buffering effects. Such effects, however, are hidden from developers by a well defined stack of abstraction layers that enforces complete separation of concerns between application and multiprocessor platform.

II. RELATED WORK

Taking into account the underlying design methodology, our approach can be compared to system-level design space exploration frameworks such as SystemCoDesigner [3], PeaCE [5] or Koski [4]. Just like those frameworks, our approach considers an abstract application model and supports designers on analysing and comparing different platforms that can efficiently execute that application. The application modelling in SystemCoDesigner and PeaCE is based on actor-orientation, while Koski uses UML 2.0. Our approach combines actors and UML, but we do not claim more expressiveness than the previous approaches. We claim, however, to be more flexible, convenient and thus having a lower adoption overhead by designers familiar with one or the other modelling approach.

Another crucial difference between the presented approach and the three forementioned frameworks is the fact that our focus is on layered models of communication-centric multiprocessor platforms. Instead of generating customised platforms out of the system-level model, our approach jointly executes application models and existing platform models, which are kept separated but feed each other with information as they execute, allowing for an accurate performance estimation. Platform models are based on predefined architectural templates that mimic well known communication-centric architectures such as AMBA AXI [10], HERMES [6] or QNoC [7]. We make such templates available at different levels of abstraction, so the profiling of the application and the parameterisation of the platform can be done step-wise using different models, allowing designers to choose between faster or more accurate validation as they follow the design flow.



system model
UML model
Actor model
Koski
SystemCo Designer
PeaCE
automatic / semi-automatic design space exploration
abstract HW/SW platform model
cycle-accurate platform model

Figure 1. Comparing the design flow of related frameworks.

III. Application Model

In the proposed approach, the highest abstraction layer comprehends the application model. At this level, developers specify an application by describing its components and communication channels in a platform-independent manner. As components, we mean individual atomic subsystems that perform sequential computation. Despite being more general than that, it can be used to model tasks and that is what we will do for the remainder of this paper. We do not advocate for any specific modelling methodology or notation and leave the choice open for each development team to choose their preferred ones. To benefit from our approach, though, the chosen notation should follow the guidelines listed below:

- for each component, developers should be able to specify its communication interface and whether it is active or passive. The communication interface defines how each component sends and receives data, and its active status denotes that it is able to send data without previously having received data. Within this paper, we will refer to the communication interface as input and output ports, but this is simply for convenience and for consistency with a well known notation, as we will see in the following paragraphs. However, the same concepts apply to the definition of communication interfaces based on other notations such as function calls, methods or send/receive primitives;

- for each active component, developers should be able to specify the activation policy (periodic or aperiodic) and respective parameters (period, list of activation time-stamps, probability distribution function, etc.) that would cause an output through each of its output ports;

- for each passive component, developers should be able to specify the combination of inputs that would cause an output through each of its output ports;

- for each communication channel, designers should be able to specify the component ports that are connected to it, which are the communication messages that are exchanged through it and what is the precedence relationship (if any) among the messages (denoted as a partial ordering);

- for each message, designers should be able to specify its data volume and priority.

There is nothing radically new in this application specification style. The same elements and attributes are found in well known system design methodologies and notations such as MARTE UML profile [1] or SysteMoC [2]. This means that developers are not expected to build an additional application model exclusively for the purpose of communication latency evaluation, since most (or all) information can be directly extracted from the development models, documentation and code that they are already producing.

In the case of UML/MARTE, Composite Structure Diagrams can be used to specify components, ports and communication channels. The definition of active/passive status and respective parameters can be done using MARTE stereotypes such as <<*RtUnit*>>, <<*PPUnit*>>, <<*SystemClock*>> and <<*TimedEvent*>>. Finally, attributes of communication messages such as data volume, precedence and priority can be described using Sequence Diagrams. Further details on the relationship between UML/MARTE and our application modelling guidelines can be found in [1], while [8] provides additional details on the use of sequence diagrams to denote precedence on communication channels.

Besides UML, we also performed tests with applications written using C++, Java and SystemC, and were able to successfully generate application models that comply with the forementioned guidelines.

IV. Executable Application Model

The application model described above provides a very abstract view of the application's behaviour, but it is powerful enough to provide insights on the potential concurrency among components and the functional dependencies that constrain it (i.e. precedence relations between communication messages). To formalise those insights within an unambiguous model of computation, our approach assigns execution semantics to the components described in Section III. We follow the principles of actor-orientation [9], so each component is considered to be an actor and is inherently concurrent to all the others. The synchronisation among actors also happens through message passing over channels and port-like interfaces, so it matches well the modelling constructs from Section III. However, channels in [9] immediately broadcast messages to all input ports connected to it and do not enforce or even allow the explicit definition of a precedence relation between them.

In order to fully support the modelling approach shown in Section III, we extended the actor-orientation principles from [9] in order to include the support for explicit definition of precedence between messages. This is a critical feature to support application models that include UML sequence diagrams or message sequence charts, which in turn are particularly useful when designing control-oriented applications or applications that require complex data exchange protocols. The details of the adopted extensions to actor-orientation are given in [8].

By adopting actor-oriented execution semantics, application models can be executed by an actor-oriented framework. Within this work, we use PtolemyII as the execution engine that allows us to observe the interactions among different application components (from now on referred as application actors) over time.

The execution of such an actor-oriented application model gives designers the chance to better understand the concurrent behaviour of the system they are designing, as well as the functional dependencies among its actors. It is even possible to extract some performance figures from that model. Such figures, however, assume the maximum possible parallelism among actors since the model does not account for resource contention: actors will always have the resources they need to compute and communicate, and will only wait in case there are functional dependencies to be resolved (e.g. waiting for a message signalising an event). While theoretically possible, a fully parallel solution is usually too expensive to be produced and virtually all implementations of industrial systems include some sort of resource sharing. To evaluate the actual impact of different resource sharing schemes in different implementations, we need models of the underlying platforms, detailed in the next section.

V. Evaluating Applications using Platform Models

A. *Design space delimitation*

Unlike the design exploration frameworks referenced in Section II, our approach does not generate a full-custom platform model out of the application model. Instead, it uses platform models based on predefined architectural templates, and evaluates their suitability to a particular application. This may be seen as an over-constraining of the design space, because the only solutions that are considered are those that can be achieved through the parameterisation of existing platform models. On the other hand, this is also a more realistic approach, since the design of electronic systems rarely starts from scratch and in almost every case a new system or product is built over legacy components or reusable intellectual property cores that are licensed from third party vendors. Thus, the seemingly over-constrained design space considered by our approach is simply a reflex of the current system design scenarios in the electronics industry, where only a portion of the design space is actually feasible when one considers the costs of design and fabrication.

For instance, one of the platform models that can be explored within our approach is a homogeneous multiprocessing platform based on MIPS processors interconnected with the HERMES NoC. This particular template can be parameterised regarding the number of processors, topology, maximum packet size, flit size, flow control scheme (hand shake vs. credit-based, number of virtual channels), routing algorithm and buffer sizes, resulting on a limited yet still large design space.

Due to the complete separation between application and platform models, designers do not have to restrict themselves to a single platform template, and they can successively evaluate the performance of an application running over different platforms. For instance, after trying to ensure an upper latency bound for a particular communication flow using the forementioned HERMES-based platform, a designer may want to simply substitute that platform model by another one based on QNoC, in order to explore the benefits of priority-based arbitration of virtual channels. This essentially means that designers can have access to an arbitrary number of disjoint design spaces, as long as they have access to their corresponding platform templates, potentially covering the complete design space that they can realistically fabricate.

B. *Joint Execution of Application and Platform Models*

No matter if a platform model represents different architectures or different customisations of a particular one, it will always have different ways to handle the sharing of resources by multiple application actors. The consequences of sharing resources such as processors, memory and communication channels must be taken into account when evaluating inter-task communication latency. Our approach evaluates such impact by jointly executing application and platform models. The interaction of both models, as they execute, allows for feeding realistic processing and communication load onto the platform resources, which in turn feedback to the application actors the performance impact caused by resource contention.

A typical interaction between an application model *AM1* and a platform model *PM1* is as follows: (i) during the execution of *AM1*, application actor *A* sends a message to actor *B* over channel *X*; (ii) channel *X* holds the message before delivering it to *B* and notifies the platform model about it; (iii) once *PM1* is notified by X of the message, it triggers processor P_A to send data packets over the on-chip interconnect to processor $\mathrm{P_B}$, simulating all relevant transmission, buffering and arbitration transactions associated to that data transfer; (iv) once all packets are received by $\mathrm{P_B}$, *PM1* informs channel *X* of the message delivery; (v) channel *X* releases the message, which then reaches actor *B*. On a realistic case, the platform model would be notified of each and every actor activation and communication. It will use its knowledge of platform-specific structure and functionality to determine which actors are competing for which platform resources, and evaluate the impact of such contention on the execution and communication latencies of all of them.

Based on the description of the interactions between application and platform models, one can see that our definition of platform models is quite loose. Any model that can keep track of the utilisation of communication and computation resources in a multiprocessor platform would suffice. This allows us to exploit multiple levels of abstraction, using a variety of platform models that can range from cycle-accurate HDL simulators to analytical solvers that estimate latency by calculating it rather than simulating the platform's functionality.

C. *Multi-abstraction Platform Models*

Taking advantage of the separation of concerns that resulted on the layered approach presented so far, designers can evaluate the performance of an application against different configurations of a platforms, or even completely different platforms. Based on the same principle, our approach also supports designers on the evaluation of applications against different views of the same platform configuration.

The use of multiple views is a common practice in electronic systems design, because its complexity requires designers to start the process at a high level of abstraction and refine such description towards an implementation. Our approach fully supports such process by allowing the application performance evaluation along the stack of abstraction layers. It allows for preliminary performance estimations based on early platform models even before the final implementation is available. It also allows the creation of simplified models of legacy platforms, allowing for faster (but less accurate) estimations during the first steps of the design space exploration.

D. Mapping Application Actors onto Platform Models

A complete separation between application and platform models has many advantages, as seen before, but it prevents the accurate evaluation of the system's performance through their joint execution. There are important interdependencies between those models, because the way application components are clustered and allocated to processors at the platform is critical to the communication latency and, as consequence, to the system's performance. Thus, the process of mapping application actors onto processors at the platform must be explicitly considered.

We then include another layer to our approach, a so-called mapper, which represents the application-platform mapping function and act as the only connecting point between application and platform models. As such, it plays a critical role in the joint execution of those two models: it will define which components of the platform model that are activated when there are actions notified by the application model. Referring back to the interaction between *AM1* and *PM1* described in the previous subsection, the mapper is responsible for identifying which, of all processors in *PM1*, are P_A and P_B once there is a message being sent from actor *A* to actor *B*.

It is important to notice that this new layer adds a new dimension to the design space exploration, as the choice of an application-platform mapping function can be considered a design decision. Thus, for a given pair of application-platform models, designers can choose among (and evaluate the impact of) different mapping heuristics. The case studies presented in Section VI were based on different mappers and show the impact of such design choices on the communication latency. However, the efficient exploration of this particular dimension of the design space, which can include a wide range of static and dynamic mapping heuristics, is fairly complex and is outside the scope of this paper.

E. Summary

The proposed approach to the design space exploration of multiprocessor systems was organised in layers with well defined interfaces, as shown in Figure 2. It allows for a separation of concerns that makes it easy to swap layers without requiring changes over the rest of the stack. For instance, a designer may swap between two slightly different configurations of a given platform (e.g. changing the routing algorithm of a NoC-based interconnect) and evaluate the impact of that change without a single modification to the application model or the mapper. Figure 3 shows all dimensions of the proposed design space organisation.

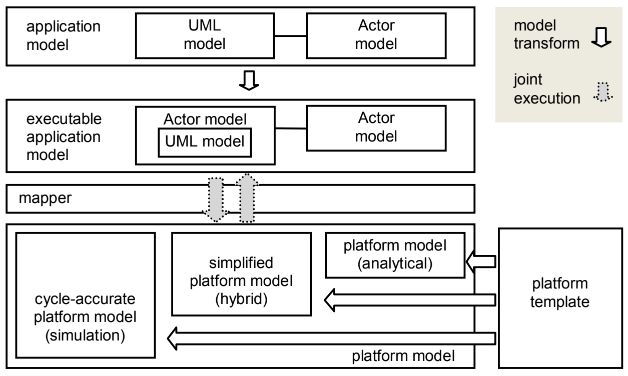


Figure 2. Conceptual layers of the proposed approach.

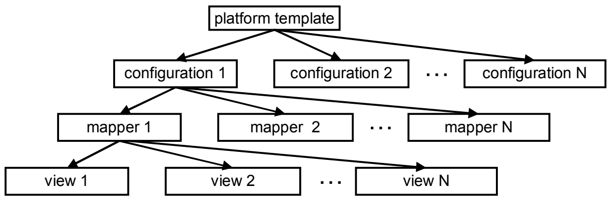


Figure 3. Multiple dimensions in the proposed design space organisation.

VI. Case Study

To demonstrate the potential of the approach presented here, we use a model of an autonomous vehicle as a case study. This is an embedded application which is likely to benefit from on-chip multiprocessor platforms, because of the high computing power required by its many subsystems and because its constraints on size and power consumption are better met by on-chip rather than on-board integration.

The autonomous vehicle we consider is designed to recognize an unknown space by populating a database of obstacles, obtained through stereo photogrammetry and ultrasonic sensors. Each of the two cameras used for the photogrammetric analysis feeds the system with a QVGA frame (320x240 resolution). After feature extraction of each of the images, photogrammetry estimates the distance from the vehicle to the obstacle, calculates the obstacle's absolute location (by using its own position obtained from a positioning sensor) and adds it to the database. Ultrasonic sensors also add obstacle information to the database, but since they have much shorter range, they are mainly used to refine the entries obtained with photogrammetry. The vehicle uses the obstacle database to guide its navigation process, which controls the speed and the direction of the vehicle. Furthermore, it also includes sensors to measure vibration. If the vehicle vibrates too much, it affects the quality of the camera images and may compromise photogrammetry. Therefore, the system controller is able to adjust the vehicle's tyre pressure so that it is suitable for the surfaces it moves on. Finally, a radio interface enables interaction with external entities. In this example, the radio interface is only used to command the capture of images. Figure 4 depicts five sequence diagrams that are part of the application model constructed for this case study.

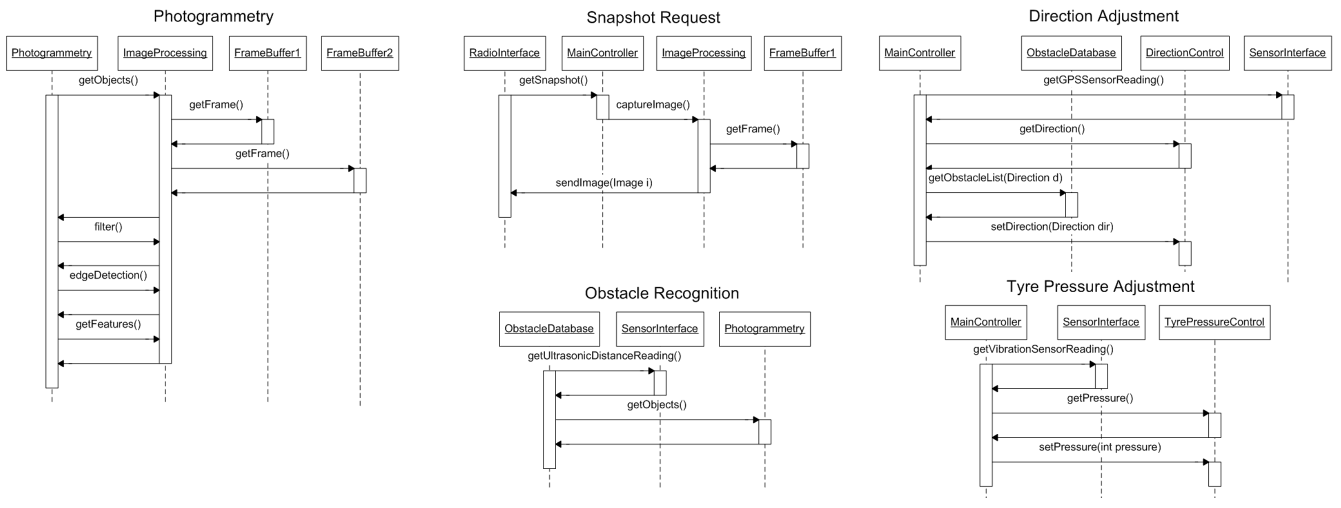


Figure 4. UML sequence diagramsshowing interactions between application actors and precedence relation between messages.

We use an extension of Ptolemy II to create an executable model of the application using actors and the UML diagrams from Figure 4 following the principles from [8]. To evaluate the communication latency impact of different multiprocessor platforms, we co-execute that application model with different platform models [11] built using Ptolemy II components, as depicted in Figure 5. We then implemented a number of heuristics that can map application actors onto platform components, completing the set of layers shown in Figure 2. That way, we can exemplify all different dimensions of the proposed design space organisation shown in Figure 3.

It is worth noticing that use of platform models built using Ptolemy II is not a requirement of this approach, and the use of RTL simulation models and even hardware-in-the-loop prototypes as platform models has already been successfully used to validate actor-based application models [12][13].

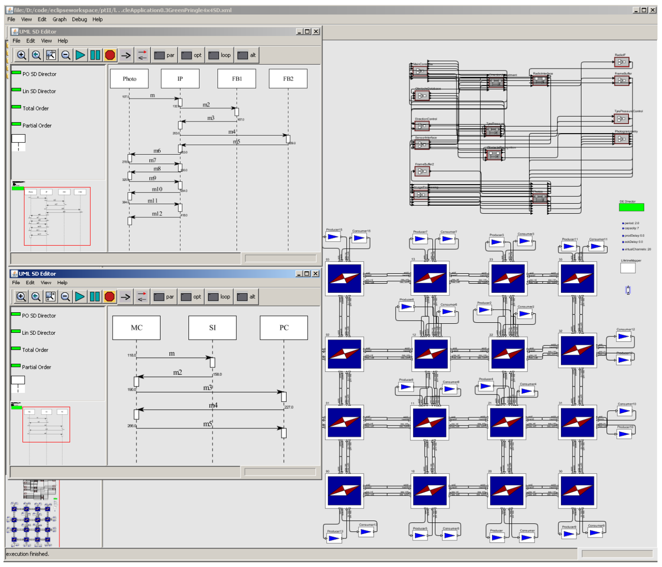


Figure 5. Joint execution of application and platform models in Ptolemy II.

As we intend to follow a realistic design process, we first explore a larger portion of the design space using fast abstract models and then analyse a few selected solutions using accurate models. The proposed approach can directly support such process without requiring change or recompilation of the application model, one of its major advantages in comparison with the work presented in Section II.

So the first step is to evaluate the autonomous vehicle application with different configurations of the platform and different mapping heuristics. The abstract platform model we used here mimics the HERMES platform, but it does not replicate its complete functionality and uses analytical components to estimate rather than simulate the impact of HERMES arbitration and flow control mechanisms. We consider two platform configurations, both with 2D mesh topology and XY routing, but differing on the mesh dimension (3x3 and 4x4). We then apply three different heuristics to map application actors onto both platform configurations. By jointly executing each application-platform pair, we obtained communication latency results for each instance of each message of the application. Figure 6 shows an aggregate of the obtained latency figures for each of the application's interactions (worst case latency of an interaction as the sum of the worst case latency of each of its messages).

To gain more confidence in the results produced by the abstract model, we must use less abstract views of the platform that represent its behaviour more precisely. We choose the first mapping and the 4x4 configuration, for its acceptable figures in all interactions and a low latency for tyre pressure adjustment (TPA). We repeat the process using two less abstract models of the HERMES platform. One of them is a hybrid model that is more accurate on estimating latencies, as it uses both simulation and analytical solutions [14], but it is slower than the abstract model. The other is a cycle-accurate model that fully simulates the platform and provides actual latencies in terms of clock cycles, but its execution is even slower.

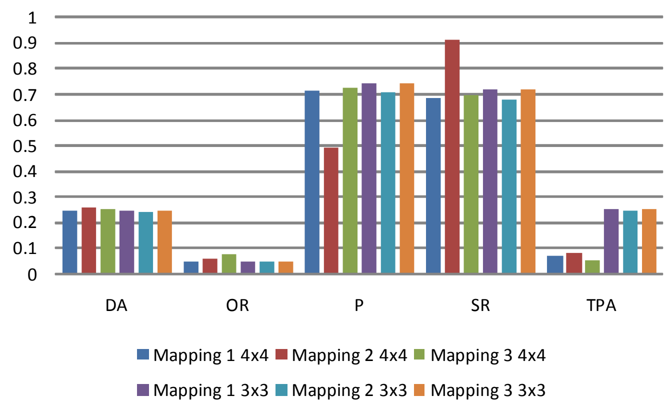


Figure 6. Obtained figures for worst case communication latency using different platform configurations and mapping heuristics (in milliseconds). DA- Direction Adjustment; OR – Obstacle Recognition; P – Photogrammetry; SR – Snapshot Request; TPA- Tyre Pressure Adjustment.

TABLE I. WORST CASE COMMUNICATION LATENCY RESULTS OBTAINED FROM DIFFERENT VIEWS OF THE 4X4 PLATFORM USING MAPPING 1

Interaction	**Communication Latency (ms)**		
	Abstract	***Hybrid***	***Cycle-accurate***
OR	0.04	0.09	0.12
TPA	0.07	0.09	0.14
DA	0.24	0.27	0.39
SR	0.68	0.94	1.35
P	0.71	0.99	1.41

Table I shows the latency figures obtained from the models providing different views of the platform. While there is a visible lack of accuracy between the cycle-accurate model and the simplified ones, it is clear that they show a consistent trend. This means that abstract models are valuable for fast comparative analysis, but accurate models are always needed for proper figures on system performance. This again shows that importance and relevance of the proposed approach, allowing for successive design space exploration across multiple abstraction layers.

VII. CONCLUSIONS AND FUTURE WORK

This work has proposed a layered approach to the evaluation of applications running over multiprocessor platforms. The separation of concerns that was achieved though layering allows for four different dimensions on the design space: platforms, configurations, mapping and views. All those dimensions can be explored without a single change on the application model using a novel evaluation approach based on the joint execution of application and platform models.

A case study with an autonomous vehicle application demonstrated the potential of the proposed approach, evaluating the impact of different platform configurations and mapping heuristics. The support for the evaluation of the system performance using different views of the platform at different levels of abstraction was also shown, allowing for a design flow that explores many alternatives using fast and abstract models, followed by the detailed exploration of selected solutions using accurate models. An even better solution would be to start the exploration with fully analytical models for worst case analysis similar to [15]. Such models are extremely fast and can provide upper bounds for communication latency, so we leave that for future work.

REFERENCES

[1] S. Maatta, L.S. Indrusiak, L. Ost, L. Moller, M. Glesner, F. Moraes, and J. Nurmi, “Characterising embedded applications using a UML profile,” Int Symposium on System-on-Chip, 2009. SOC 2009, pp. 172-175.

[2] C. Haubelt, J. Falk, J. Keinert, T. Schlichter, M. Streubühr, A. Deyhle, A. Hadert, and J. Teich, “A SystemC-based design methodology for digital signal processing systems,” EURASIP J. Embedded Syst., vol. 2007, n.1, 2007.

[3] J. Keinert, M. Streubuhr, T. Schlichter, J. Falk, J. Gladigau, C. Haubelt, J. Teich, and M. Meredith, “SystemCoDesigner - an automatic ESL synthesis approach by design space exploration and behavioral synthesis for streaming applications,” ACM Trans. Des. Autom. Electron. Syst., vol. 14, 2009, pp. 1-23.

[4] T. Kangas, P. Kukkala, H. Orsila, E. Salminen, M. Hännikäinen, T.D. Hämäläinen, J. Riihimäki, and K. Kuusilinna, “UML-based multiprocessor SoC design framework,” ACM Trans. Embed. Comput. Syst., vol. 5, 2006, pp. 281-320.

[5] S. Ha, S. Kim, C. Lee, Y. Yi, S. Kwon, and Y. Joo, “PeaCE: A hardware-software codesign environment for multimedia embedded systems,” ACM Trans. Des. Autom. Electron. Syst., vol. 12, 2007, pp. 1-25.

[6] F. Moraes, N. Calazans, A. Mello, L. Möller, and L. Ost, “HERMES: an infrastructure for low area overhead packet-switching networks on chip,” Integration, the VLSI Journal, vol. 38, Oct. 2004, pp. 69-93.

[7] E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny, “QNoC: QoS architecture and design process for network on chip,” Journal of Systems Architecture, vol. 50, Feb. 2004, pp. 105-128.

[8] L.S. Indrusiak, A. Thuy, and M. Glesner, “Executable system-level specification models containing UML-based behavioral patterns,” Design, Automation & Test in Europe Conference & Exhibition, 2007. DATE '07, 2007, pp. 301-306.

[9] J. Eker, J. Janneck, E. Lee, Jie Liu, Xiaojun Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Yuhong Xiong, “Taming heterogeneity - the Ptolemy approach,” Proceedings of the IEEE, vol. 91, 2003, pp. 127-144.

[10] M. Posner, D. Mossor, “Designing using the AMBA^{TM} 3 AXI^{TM} Protocol,” white paper, SynopsysInc, Apr. 2005.

[11] L.S. Indrusiak, L. Ost, L. Moller, F. Moraes, and M. Glesner, “Applying UML Interactions and Actor-Oriented Simulation to the Design Space Exploration of Network-on-Chip Interconnects,” IEEE Comp Society Annual Symposium on VLSI, 2008. ISVLSI '08, pp. 491-494.

[12] L.S. Indrusiak, R.B. Prudencio, and M. Glesner, “Modeling and prototyping of communication systems using Java: a case study,” IEEE Int Workshop on Rapid System Prototyping, 2005, pp. 225-231.

[13] L.S. Indrusiak, M. Glesner, and R. Reis, “On the Evolution of Remote Laboratories for Prototyping Digital Electronic Systems,” Industrial Electronics, IEEE Transactions on, vol. 54, 2007, pp. 3069-3077.

[14] L. Ost, F.G. Moraes, L. Moller, L.S. Indrusiak, M. Glesner, S. Maatta, and J. Nurmi, “A simplified executable model to evaluate latency and throughput of networks-on-chip,” Proc Symposium on Integrated Circuits and System Design, 2008. SBCCI 2008, pp. 170-175.

[15] Z. Shi and A. Burns, “Real-Time Communication Analysis for On-Chip Networks with Wormhole Switching,” ACM/IEEE Int Symposium on Networks-on-Chip, 2008. NOCS 2008, pp. 161-170.