

Layered UML Workload and SystemC Platform Models for Performance Simulation

Jari Kreku, Yang Qu, Juha-Pekka Soininen,
Kari Tiensyrja
VTT Technical Research Centre of Finland
Kaitovayla 1, 90570 Oulu, Finland

Abstract

Future mobile devices will be based on heterogeneous multiprocessing platforms accommodating several currently stand-alone applications. Increasing complexity of both application and platform development requires coordinated separation of concerns so that interoperability can be preserved. Application designer needs abstract platform models to check rapidly whether a new feature or application is feasible on a platform and how it will impact on the performance of other coexisting applications. Platform designer needs abstract application models for defining platform computation and communication capacities.

We propose a layered UML workload and SystemC platform modelling approach that allows application and platform to be modelled at several levels of abstraction to enable early performance evaluation of the resulting system. Platform services are presented to workload models through APIs that allow a Y-chart-like specify-explore-refine performance modelling and simulation. The approach has been experimented by applying it to MPEG-4 encoder, Quake2 3D game and MP3 decoder case studies that validate the approach.

1 Introduction

Digital convergence is changing rapidly the landscape of designing future mobile devices. On one hand, devices accommodate a large number of currently separate applications, some of which run independently while others interact concurrently. On the other hand, devices combine technologies in different ways resulting in heterogeneous multiprocessors with tens of processing elements interconnected with network communication architecture. As a result, complexity will increase by orders of magnitude on both application and platform development sides. Because of deeper integration through technology, there are needs to exchange more information between sides, e.g. an application designer needs to know whether a new application or feature is

feasible on a platform and a platform designer needs means to rapidly analyse performance impacts of a new requested application or feature.

The goal of the work is to develop a model-based approach for system-level design and performance evaluation of future real-time embedded systems. The emphasis is on the raising of the modelling abstraction of both the application development and platform development, and on making them interoperable by applying the principles of the OMG's Model Driven Architecture [MDA03]. The approach aims to improve design productivity by reducing complexity, increase predictability by allowing early validation of performance, and optimise costs and risks by providing justified information for decision-making.

The co-modelling approach follows the specify-explore-refine paradigm applying the principles of the so called Y-chart model [BSC⁺97], i.e. by mapping a model of application onto a model of platform and by executing the analysis of the resulting allocated model. In the recent years the use of UML [UML] has become common in the application modelling, while SystemC [Gro02] is being used for hardware oriented architectural models at higher level of abstraction, i.e. for the models of platform resources, such as processors. Mapping between UML use case and workload models and the SystemC platform models is proposed to be defined as transformation rules to enable at least semi-automatic generation of simulation models for system-level performance evaluation.

Traditional system modelling approaches have trusted on hierarchical containment that has a number of excellent properties related to topological organization, encapsulation of functionality and simplification through abstraction as long as presented information is of static nature. Future heterogeneous multiprocessing systems will, however, contain application software that will be scheduled dynamically [PTC05]. Design layering is an ancient recipe for addressing dynamism. Each layer is a logical collection of functions, i.e. services, and data objects that support the higher layers of the system. Each layer may contain private state and may have ability to schedule its services among multiple requestors. Services are globalized so that more than one function at a higher-level may use a service. A layered model is not complete without some representation from all the design layers, since each layer contains elements that are not present in the other layers [Sel05].

Several proposals for combining UML with the platform-based design emerged in the beginning of this decade, e.g. layering of embedded system platforms in UML [CSSV⁺02], linking UML with SystemC platform descriptions [GE02] and generating SystemC code from UML specifications [BNS02]. SystemC-based approach for linking different models of computation for performance simulation of heterogeneous systems was proposed in [PHS⁺04]. Needs for more formal definition of abstractions and transformations between them

in practical application of MDA and UML were analysed in [Oli05]. Recently, several proposals of linking UML to SystemC [RSRB05] or multi-processor platforms [KRH⁺05] have been presented.

The presented work defines a number of model abstraction layers for describing application workloads in UML and platform services in SystemC to enable early system-level performance modelling and evaluation through transaction-level simulation. The approach has been partly validated with an MPEG-4 encoder design case that used on one hand the UML/SystemC modelling approach, and was on the other implemented on an OMAP platform for comparison. Two other design cases have been done, however they were modelled fully in SystemC.

The rest of the paper is structured as follows: Chapter 2 describes the general modelling approach, Chapter 3 considers the hierarchical structure of workload models and some methods for obtaining load information, Chapter 4 presents the platform model layers, Chapter 5 describes experimental case studies, and finally, Chapter 6 presents some concluding remarks and planned future work.

2 Modelling approach

Figure 1 presents our modelling approach according to the Y-chart model. The purpose of workload modelling is to illustrate the load an application causes to a hardware platform when executed. Workload models are non-functional in the sense that they do not perform the calculations or operations of the original application, e.g. a video encoder model is not able to encode video. Workload modelling enables performance evaluation already in the early phases of the design process, because the models do not require that the applications are finalised. Workload modelling also enhances simulation speed as the functionality is not simulated and models can typically be easily modified to quickly evaluate various versions of use cases.

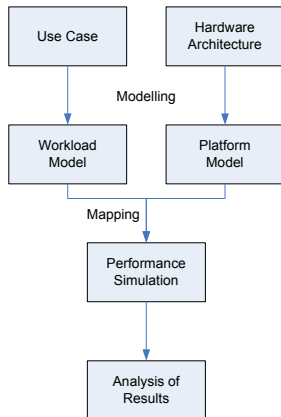


Figure 1: The mapping based approach to performance simulation.

Platform modelling comprises the description of both hardware and platform software (middleware) components and interconnections that are useful for performance simulation. Similar to workload modeling, platform modelling considers hierarchical and repetitive structures to exploit topology and parallelism. The resulting models will provide interfaces, through which the workload models can use the resources and services provided by the platform.

In our approach the workload models are created with UML or SystemC, while the platform models are based on SystemC only. If the workload modelling is done with a UML tool, the models have to be transformed into SystemC using e.g. code generation. After mapping the workloads to the platform, the models can be combined for transaction-level performance simulation in SystemC. Based on the simulation results, we can analyse e.g. processor utilisation, bus or memory traffic and execution time.

3 Workload Model

Workload modelling layers: Workload models have a hierarchical structure, where main workload model W divides into application workloads $A_i, 1 \leq i \leq n$ for different processing units of the physical architecture model:

$$W = \{C_a, A_1, A_2, \dots, A_n\}, \quad (1)$$

where C_a denotes the common control between the workloads, which takes care of the concurrent execution of loads mapped to different processors, and n is the number of application workloads under the main workload.

Each application workload A_i is constructed of one or more processes P_i :

$$A_i = \{C_p, P_1, P_2, \dots, P_n\}, \quad (2)$$

where C_p corresponds to the control between the processes. The structure of the main workload model and the application workloads is depicted in the UML diagram of figure 2. The application and process control are shown as classes in the diagram; however, they may be implemented using e.g. standard C++ control structures in SystemC based workload models.

The processes are comprised of function workloads F_i :

$$P_i = \{C_f, F_1, F_2, \dots, F_n\}, \quad (3)$$

where C_f is control and describes the relations of the functions, e.g. branches and loops. Process workload models can also be statistical, in which case the model will describe the total number of different types of load primitives and the control is a statistical distribution for the primitives (figure 3). This is beneficial in case the chosen workload modelling method is not accurate enough so that functions could be modeled in detail, or

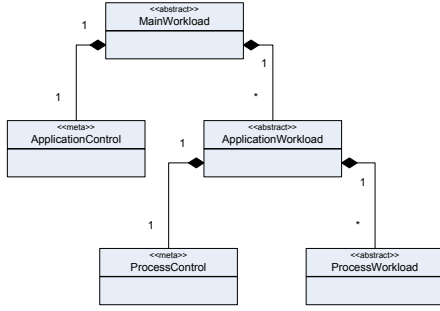


Figure 2: The hierarchical structure of the main and application workloads.

if less important workloads are modeled less accurately for reducing the modelling effort.

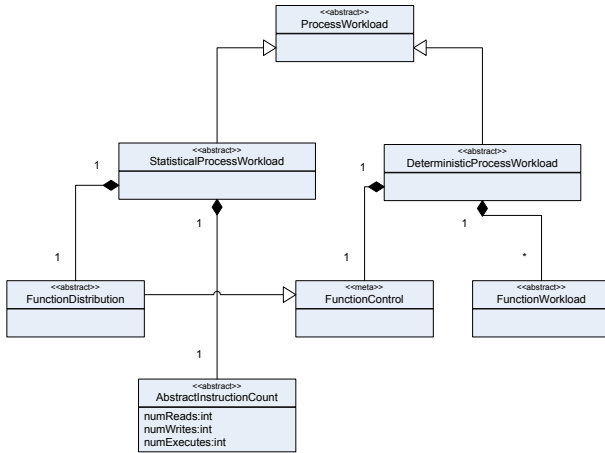


Figure 3: The process (and function) workloads can be statistical or deterministic models.

Function workload models are basically control flow graphs

$$F_i = (V, G), \quad (4)$$

where nodes $v_i \in V$ are basic blocks and arcs $g_i \in G$ are branches. Basic blocks are ordered sets of load primitives, e.g. abstract instructions, which are used for load characterization. For example, these abstract instructions could be read and write for modelling memory accesses and execute for modelling data processing. Furthermore, the function workloads can be statistical the same way and for the same reasons as process workloads. Workload models using deterministic process models but statistical function models are more accurate than those using statistical process models, but less accurate than models that are deterministic down to basic block level.

Further one or two workload model layers, namely a distributed application and/or networked application layer, could be considered beside those shown above. This addition would help in modelling of complex distributed workloads that are typical in e.g. telecom systems.

Obtaining load information: There are a number of methods for obtaining the load information for the workload models from different sources. In the analytical method, the number of memory accessing and data processing operations is estimated from the algorithm description or other suitable source. This is most suitable for simple DSP like applications (e.g. static data flow models), where the control flows are simple enough so that such estimations can be made with reasonable accuracy. However, it is often very difficult to get a realistic distribution for the operations with real-life applications.

Another alternative is source code based modeling, which relies on a compiler tool chain for producing the required information for the workload models or for mapping execution models on SystemC TLM or ISS architecture models. The source code of the application or of a very similar application has to be available and it has to be mature enough that it can be compiled at least in a workstation environment if not in the target architecture. In this method, a profiler is used to obtain control information for the workload models and to limit the modeling work to the most loading functions. The compiler front-end can be used to produce the load primitives for the models.

The third method for obtaining the load information is trace or measurement based approach. In this case, an existing execution trace or measured load information of a similar application in a resembling architecture is used as input. Multiple independent traces or measurements can be combined to model complex use cases. This is a rapid way for producing workload models but also a bit limited due to the prerequisites.

4 Platform Model

The platform model is an abstracted hierarchical representation of the actual platform architecture. It contains timing information along with structural and behavioral aspects. The platform model is composed of three layers: component layer, HW architecture layer, and platform architecture layer (figure 4). Higher layers that describe more complicated system, e.g. distributed computers and networked platforms, could be built on top of low-level layers, but these are not detailed further in this document.

Each layer has its own services, which are abstraction views of the architecture models. They describe the platform behaviors and related attributes, e.g. performance, but hide other details. Services in the HW architecture layer and platform architecture layer can be invoked by workload models. High-level services are built on low-level services, and they can also use the services at the same level. Each service might have many different implementations. This makes the design space exploration process easier, because replacing components or platforms by others could be easily done as long as they implement the same services.

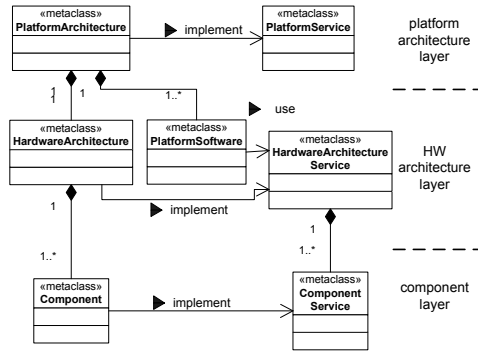


Figure 4: The three layers of the platform side.

Component layer: This layer consists of processing (e.g. processors, DSPs, dedicated hardware and reconfigurable logic), storage, and interconnection (e.g. bus and network structure) elements. An element must implement one or more types of component-layer services. For example, a DMA controller should implement both the master services and the slave services. In addition, some elements need to implement services that are not explicitly defined in component-layer services, e.g. a bus shall support arbitration and a network shall support routing. The component-layer services are the primitive services, based on which top-level services are built.

Models of component-layer elements are organized in a library. The components in the library have parameterizable UML models, so it is easier to import the platform model into the UML environment of workload models. These parameters are component specific. For example, an embedded reconfigurable core should include following parameters: reconfiguration latency, power consumption, and size of reconfigurable logic.

Since the parameterized description does not describe the functionality of hardware, the description can not be functionally simulated. Instead, the simulation models are conducted from the parameterized model through transformations or library instantiations. Either way, parameterized templates of simulation models are needed.

In the library, each component-layer element has a corresponding transaction-level SystemC model with compatible interfaces to OCP-IP TL channels for simulation. Using standard interface can make the system integration phase easier, which happens in the mapping stage where the workload model is available and mapping decision is done. Fast performance evaluation needs high abstraction-level models. Therefore, units should be modeled at OCP-IP transaction-level layer 2, namely timed, but not cycle-accurate.

HW architecture layer: The HW architecture layer is built on top of the component layer. It describes the components of the system and how they are connected. The services used at this layer include the abstract instructions, e.g. `execute()`, `read()` and

`write()`, and hide all the interconnection services.

The model can be presented as a composition of structure diagrams that instantiates the elements taken from the library. The load of the application is executed on processing elements. The communication network connects the processing elements with each other. The processing elements are connected to the communication network via interfaces.

Platform architecture layer: The platform architecture layer is built on top of the HW architecture layer by incorporating platform software, e.g. OS, and serves as the portals that link the workload models and the platforms in the mapping process. Platform-layer services consist of three parts, service declaration, instantiation information and timing information. The service declaration describes the functionalities that the platforms can provide. Because a platform can provide the same service with quite different manners, the instantiation information describes how a service is instantiated in a platform. The timing information is a tagged value of a particular instance of the service, and it gives preliminary performance estimate. The value is meant for early system analysis, and more accurate value must be collected from the system simulation results.

Instead of trying to build up the complete set of services, we consider only the application domain-specific core services. The platform-layer services are divided into several categories with each category matching one application domain, e.g. video processing, audio processing and encryption/decryption. The OS system-call services are in an individual domain, and as mentioned earlier they can also be invoked by other services at the same level. A number of platform-layer services are defined for each domain, and more could be added if necessary.

5 Experimental case studies

The proposed workload and platform model layers have been examined in a number of case studies. The cases have focused on specific parts of the modelling approach, e.g. on certain workload and platform layers or on different methods for obtaining load information.

5.1 Case 1: MPEG-4

The application, function and load primitive layers of the workload modelling hierarchy have been experimentally applied to an MPEG-4 encoder, which was partitioned into the ARM and DSP processors of the OMAP architecture [KES05]. Four variations of the encoding use case were developed, where certain algorithms of the encoder were accelerated by the DSP, while the ARM was doing most of the work. The workload models were created with a UML tool and transformed to SystemC so that they could be combined with the SystemC-based platform model.

Table 1: OMAP utilisation with two alternative MPEG-4 partitionings.

Algorithms accelerated by the DSP	Fps	MCU load	DSP load
DCT, SAD	16.5	100%	27%
DCT	13.8	100%	11%

In the platform side, the component and hardware architecture layers were used together with higher-abstraction level platform service models of the hardware accelerators available in the OMAP system. The model was developed using the OSCI SystemC library and OCP-IP protocol models.

For validation purposes the same MPEG-4 encoder was ported to the Spectrum Digital’s OMAP Starter Kit (OSK). The operating system in the OSK was Linux. The encoder was executed in the OSK using the four partitioning alternatives and its performance, e.g. achieved frame rate and processor utilisation, was monitored (figure 5, table 1). Finally, the simulated and monitored results were compared and the average error was found to be only 12%.

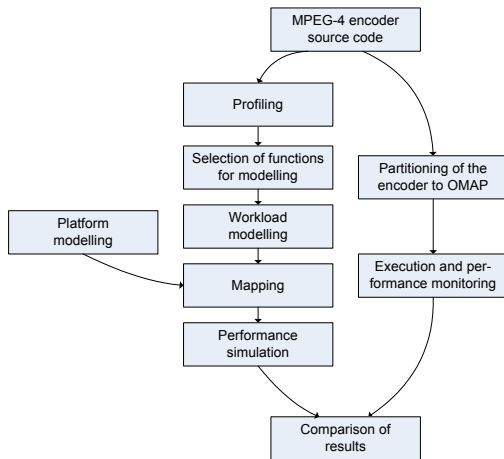


Figure 5: The modelling and validation flow for the MPEG-4 encoding.

5.2 Case 2: Quake2 3D game

Approximately the same workload model hierarchy up to the application level was evaluated in [KPKS04] using a source code based modelling approach. However, specially-crafted workload subsets were used instead of the process layer in this case. The purpose of the subsets was to take the effect of the operating system scheduler into account by manually collecting related function models into the subsets so that the length of each subset would correspond to the length of the scheduler time slice.

In this case, the Quake 2 3D game and a prototype future platform were modelled using SystemC. Quake

was partitioned for the multiprocessor architecture in such a way, that the first CPU was executing game logic, the second was dedicated to audio and the last one was processing the 3D graphics. Again, there were four variations of the use case consisting of two display resolutions and two frame rates. The modelling approach enabled rapid evaluation of the four cases and the obtained simulation results could easily have been used to feasibility evaluation. For example, we were able to estimate that, with a 320 x 200 pixel display resolution and a frame rate of 30 fps, the graphics processor will be at the limit of its performance (table 2).

Table 2: Quake 2 PU utilisation with different frame rates.

Fps	PU1	PU2	PU3
30	57.8%	34.7%	97.1%
15	44.0%	29.9%	48.1%

5.3 Case 3: MP3

Furthermore, the same platform model layers as in the previous two cases have been used in combination with the statistical, measurement-based workload modelling approach [KKS04]. Two rather complex use cases consisting of several, both parallel and sequential applications were modelled, including e.g. data communication (bluetooth download, GPRS connection), text messaging (both SMS and MMS) and video capture. In addition, an MP3 decoder was modelled from the source code using the deterministic approach.

The OMAP platform model used in this case was the same SystemC-based model as in the case 1. All workload models for the case were also created with SystemC. The simulation results were compared to measurements obtained by running the same applications in a prototype product based on the OMAP architecture and Symbian operating system; the worst case error was about 25%. The average error (with original unadjusted models) was about 15%. A sample of the results for this case is shown in table 3.

Table 3: OMAP processor utilisation with case 3.

Use case	MCU	DSP
MP3 playback	19%	0%
Complex	43%	9%

6 Conclusions

A layered workload and platform model structure for early system-level performance evaluation was presented. On the workload modelling side we covered application, process, function and basic block layers, whereas on the platform side component, hardware architecture and platform architecture layers were discussed. The modelling approach has been validated in

three separate cases, where the average and maximum errors between simulated and monitored results have been about 15% and 25% respectively.

In the future, we plan to work on model transformations, e.g. improving the SystemC code generation from UML. The approach should be expanded, so that other criteria besides performance, like power consumption, can be evaluated. Future work will also include a real-scale case study to further validate the approach.

7 Acknowledgements

This work is supported by Tekes (Finnish Funding Agency for Technology and Innovation) and VTT under EUREKA/ITEA contract 04006 MARTES. We would also like to thank Matti Eteläperä, Tarja Kauppi and Jani Penttilä for their valuable contributions.

References

- [BNS02] F. Bruschi, E. Di Nitto, and D. Sciuto. SystemC code generation from UML models. In *Forum on Specification & Design Languages*, Marseille, France, September 2002.
- [BSC⁺97] F. Balarin, E. Sentovich, M. Chiodo, P. Giusto, H Hsieh, B. Tabbara, A. Jureska, L. Lavagno, C. Passerone, K. Suzuki, and A. Sangiovanni-Vincentelli. *Hardware-Software Co-Design of Embedded Systems — The POLIS Approach*. Kluwer Academic Publishers, 1997. 297 p.
- [CSSV⁺02] R. Chen, M. Sgroi, A. Sangiovanni-Vincentelli, J. Rabaey, L. Lavagno, and G. Martin. Embedded system design: Using UML and platforms. In *Forum on Specification & Design Languages*, Marseille, France, September 2002.
- [GE02] P. Green and M. Edwards. Platform modelling with UML and SystemC. In *Forum on Specification & Design Languages*, Marseille, France, September 2002.
- [Gro02] T. Groetker. *System Design with SystemC*. Kluwer Academic Publishers, 2002. 240 p.
- [KES05] J. Kreku, M. Eteläperä, and J-P. Soininen. Exploitation of UML 2.0-based platform service model and SystemC workload simulation in MPEG-4 partitioning. In *International Symposium on System-on-Chip Proceedings*, pages 167–170, 2005.
- [KKS04] J. Kreku, T. Kauppi, and J-P. Soininen. Evaluation of platform architecture performance using abstract instruction-level workload models. In *International Symposium on System-on-Chip Proceedings*, pages 43–48, 2004.
- [KPKS04] J. Kreku, J. Penttilä, J. Kangas, and J-P. Soininen. Workload simulation method for evaluation of application feasibility in a mobile multiprocessor platform. In *Proceedings of the Euromicro Symposium on Digital System Design*, pages 532–539, 2004.
- [KRH⁺05] P. Kukkala, J. Riihimäki, M. Hännikäinen, T. D. Hämmäläinen, and K. Kronlöf. UML 2.0 profile for embedded system design. In *Proceedings of the Design, Automation and Test in Europe Conference*, volume 2, pages 710–715, Munich, Germany, March 2005.
- [MDA03] MDA guide version 1.0.1, June 2003. Document number: omg/2003-06-01. Available at <http://www.omg.org/mda/>.
- [Oli05] I. Oliver. Applying UML and MDA to real systems design. In *Proceedings of the Design Automation and Test in Europe Conference*, Munich, Germany, March 2005.
- [PHS⁺04] H. Posadas, F. Herrera, P. Sánchez, E. Villar, and F. Blasco. System-level performance analysis in SystemC. In *Proceedings of the Design Automation and Test in Europe Conference*, Paris, France, February 2004.
- [PTC05] J. M. Paul, D. E. Thomas, and A. S. Cassidy. High-level modeling and simulation of single-chip programmable heterogeneous multiprocessors. *ACM Transactions on Design Automation of Electronic Systems*, 10(3):431–461, July 2005.
- [RSRB05] E. Riccobene, P. Scandurra, A. Rosti, and S. Bocchio. A SoC design methodology involving a UML 2.0 profile for SystemC. In *Proceedings of the Design Automation and Test in Europe Conference*, pages 704–709, Munich, Germany, March 2005.
- [Sel05] B. Selic. On software platforms, their modeling with UML 2, and platform-independent design. In *Proceedings of the 8th International Symposium on Object-Oriented Real-Time Distributed Computing*, pages 15–21, May 2005.
- [UML] <http://www.omg.org/uml/>.