An Interactive System Level Simulation Environment for Systems-on-Chip

Daniel Knorreck, Ludovic Apvrille, Renaud Pacalet

System-on-Chip Laboratory (LabSoC)
Institut Telecom, Telecom ParisTech, LTCI CNRS
Route des Cretes BP 193, F-06904 Sophia Antipolis, France

Abstract: This article presents an interactive simulation environment for high level models intended for Design Space Exploration of Systems-On-Chip. The existing open source development environment TTool supports the MARTE compliant UML profile DIPLODOCUS and enables the designer to create, simulate and formally verify models. The goal is to obtain first performance estimations of the system intended for design while minimizing the modeling effort. The contribution outlined in this paper is an additional module providing means for controlling the simulation in real time by performing step wise execution, saving and restoring simulation states as well as animating UML models of the system. Moreover the paper elaborates on the integration of these new features into the existing framework consisting of a simulation engine on the one hand and a graphical user interface on the other hand.

Keywords: System-on-Chip, Design Space Exploration, UML, Simulation, Interactive, HW/SW partitioning, DIPLODOCUS

1. Introduction

A System-on-Chip can be defined as a set of communicating electronic components integrated into one single chip. The latter components are highly heterogeneous in nature: digital, analog and mixed signal components may be interconnected to make up complex systems ranging from mobile hand sets and set top boxes to automotive controllers and feedback control systems for rail cars. Due to recent advances in the field of semiconductor physics, higher and higher integration densities are achieved so that a given piece of silicon accommodates more and more transistors.

In order to make use of the available resources, the complexity of embedded systems and Systems-on-Chip has been increasing rapidly. On the one hand, users are demanding products exhibiting sophisticated features which are reliable, easy to use and affordable. On the other hand, the gap increases between integration and designer efficiency due to inadequate tools and methodologies. In addition to the increased demand of functionality, time-to market is an issue of great concern. Hence, developers are facing significant difficulties due to an exponentially raising complexity. It becomes more and more unlikely that an optimal design represents an intuitive solution, thus the experience of the designer may not lead him/her to optimal designs with respect to functional and non-functional requirements such as performance, size, energy consumption, reliability.

Thus, given a particular functionality and associated requirements, the design space is considered as representing all functionally equivalent implementation alternatives. Being almost infinitely large at the very beginning of the design flow, the design space should be gradually reduced during the design process by refining the model of the system. The analysis of systems at low abstraction levels exhibits a high degree of accuracy but comes with the downside of being demanding and slow. Traditional simulation techniques operating at register transfer level (RTL), instruction or transaction level are not appropriate for system level Design Space Exploration (DSE) for two reasons:

- Only a very limited number of implementation alternatives can be examined due to the high modeling effort and extensive simulation runtime.
- The lack of specification at early design stages may prohibit the construction of detailed models - even if the effort was acceptable.

Thus, abstractions are the key to success in performing System Level Design. In this context, we have previously introduced a UML-based environment named DIPLODOCUS. The strength of our approach relies on formal verification capabilities and fast simulation techniques ([2] [3]).
DIPLODOCUS design approach is based on the following fundamental principles:

- Use of a high level language (UML)
- Clear separation between application and architectural matters
- Data abstraction

- Use of fast simulation and formal static analysis techniques, both at application and mapping levels

The designer is supposed to model in an orthogonal fashion the application and the architecture of the targeted system. Thereafter, a mapping stage associates application and architectural components. The strength of our approach relies in simulation and formal proofs techniques that can be applied to modeled systems at all methodological stages. **UML application models can be simulated with respect to the underlying hardware,** as opposed to state of the art UML model simulators which operate on a purely functional level. **Feedback from the simulation** is directly **visualized** within the application model by for instance highlighting the currently executed operators. This feature paves the way for getting an intuitive insight into the behavior of application models on different hardware architectures. **Simulation states may be saved and restored** so as to explore several possible executions and **branches of non-deterministic decision operators can be explicitly selected** by the user. Also, our environment totally hides knowledge of simulation or formal proofs techniques: knowledge of our UML profile is the only asset for engineers. Today, this methodology is supported by an open-source toolkit named TTool [4].

The paper is organized in 6 sections. Section 2 surveys related work in the field of System Level Modeling and UML model simulation. Section 3 elaborates on the DIPLODOCUS methodology and its tooling. Section 4 details the simulation strategy which is embedded into the development environment TTool. Section 5 gives insights into the features and the technical realization of the interactive simulation environment. Section 6 finally concludes the paper and draws perspectives of future work.

2. Related Work

Some of the current state of the art UML modeling tools ([5], [6], [7], [8] amongst others) exhibit simulation capabilities. Simulations can only be performed based on purely functional models in an untimed fashion. Our interactive simulator however also accounts for architecture semantics like arbitration of shared resources, speed or data throughput of devices, etc. Furthermore, the execution behavior of models is tool dependent as the UML standard lacks an execution semantics. The DIPLODOCUS profile however fills that semantic gap and thus also paves the way for formal verification.

Related work in the field of system level modeling and simulation often suffers from one of the following problems: Off the shelves solutions like [10] and [9] mostly do not permit an orthogonalization of functionality and architecture. Detailed RTL models of HW components and the final software code must be at hand to perform co-simulation. For instance, Instruction Set Simulators are often used to estimate the impact on performance of software execution on a specific processor. Thus, only little abstraction may be applied to communication (SystemC TLM [11], etc) and computations. Some academic approaches enable the design of distinct models for architecture and application ([12], [13], [14], [15], [24]). In this case, the level of abstraction is often not pushed high enough to explore a representative subset of the HW/SW design space in a reasonable time. Sometimes application models do not exhibit data/functional abstractions. In other cases, the simulation strategy does not leverage abstractions and models have to be refined before being executable. For instance, [23] bears resemblance with our approach with respect to the modeling methodology where UML is applied for architecture, application and mapping models. As opposed to our framework, the focus is put on streaming applications exhibiting only occasional control messaging and branching. For this reason, the semantics of Kahn Process Networks has been adopted for application models. Simulation is carried out on SystemC TLM level, thus it does not leverage all abstraction applied at the modeling stage.

Analytical approaches like [21] and [16], [22] rely on the classical methods for real-time scheduling analysis to determine characteristics of distributed systems. The behavior of the environment is modeled by means of standard event arrival patterns including periodic and sporadic events with jitters or bursts. The main contribution is the extension of the scope of well-known scheduling theories for mono-processors. Event streams are propagated among resources of distributed systems in a way that each resource may be analyzed separately with classical algorithms. However, the applicability of scheduling theories requires the task model to be simplistic and thus it merely reflects best case and worst case execution times. Control flow within tasks cannot be considered at all. It may be tedious if not impossible to model tasks exhibiting a data dependent or irregular behavior.

The DIPLODOCUS environment however relies on data and functional abstractions to leverage fast simulation techniques. Nevertheless, the application model captures different control flow branches and explicitly models indeterminism. The latter property enables the developer to smoothly vary the coverage of the model during simulation. Explicit indeterminism also makes model amenable to both formal verification.

3. The DIPLODOCUS UML Profile and Tooling

DIPLODOCUS is a UML profile targeting the design of Systems-on-Chip at a high level of abstraction. A UML profile customizes UML [17] for a given domain, using UML extension capabilities. Furthermore, a UML profile commonly provides a methodology and is supported by a specific toolkit.

3.1 Methodology

DIPLODOCUS follows the Y-Chart methodology [26] and is therefore characterized by the following three-step design flow (see Figure 1):

- **Applications** are first described as a network of abstract communicating tasks using a UML class diagram. The latter represents the static view of the application. Each task behavior is described with a UML activity diagram.
- Targeted **architectures** are modeled independently from applications as a set of interconnected generic hardware nodes. The latter may be parametrized to exhibit a more specific behavior. UML nodes were defined to model HW elements (e.g. CPUs, buses, memories, hardware accelerators, bridges).
- A **mapping** process defines how application tasks can be bound to execution entities and also how abstract communications between tasks are assigned to communication and storage devices.

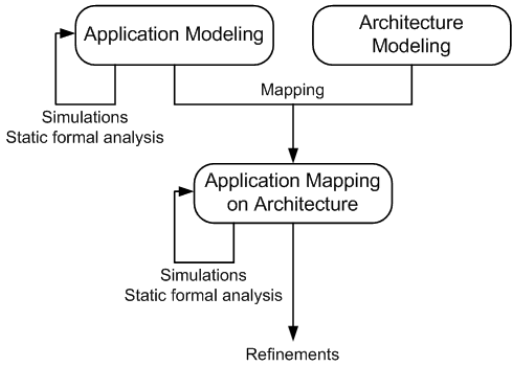


Figure 1: Global view of the Design Space Exploration methodology

Within a SoC design flow, Design Space Exploration is supposed to be carried out at a very early stage. Hence, the main DIPLODOCUS objective is to help designers to spot a suitable hardware architecture even if algorithmic details have not yet been stipulated thoroughly. To achieve this, DIPLODOCUS relies (i) on fast simulation and formal proof techniques, both at application and mapping level, and (ii) on application models clearly separated from architecture models (also referred to as Y-Chart approach [18]). Due to the high abstraction level of both application and architecture models, simulation speed can be increased significantly with regards to simulations usually performed at lower abstraction level (e.g. TLM level, RTL level, etc.). Additionally, efficient formal static analysis techniques may be applied before and after mapping [25].

3.2 Application Modeling

An application model is the description of functions to be performed by the targeted SoC. As it merely stipulates a partial ordering of actions, a refined execution semantics is introduced at the mapping stage. One has to bear in mind that, at application modeling level, computations and communication are accounted for by abstract cost operators. The time it takes to process the latter can only be resolved with the aid of parameters which are annotated to the architecture model. Abstract cost operators entail two kinds of abstractions which reflect the degree of uncertainty inherent to early design stages:

- **Data abstraction**: Only the amount of transferred data is taken into account, not the data itself.
- **Functional abstraction**: Algorithmic details are abstracted by means of their complexity operators.

As mentioned before, functions are modeled as a set of abstract tasks described within UML class diagrams. Task behavior is modeled using UML activity diagrams which are built upon the following operators: control flow and variable manipulation operators (loops, tests, assignments, etc.), communication operators (reading/writing abstract data samples in channels, sending/receiving events and requests), computational cost operators and delay operators. This section briefly describes a subset of the aforementioned operators as well as their semantics and provides definitions for Channels, Events and Requests:

Channels are characterized by a point-to-point unidirectional communication between two tasks. The following Channel types exist:

- Blocking Read/Blocking Write (BR-BW)
- Blocking Read/Non Blocking Write (BR-NBW)
- Non Blocking Read/Non Blocking Write (NBR-NBW)

Events are characterized by a point-to-point unidirectional asynchronous communication between two tasks. Events are stored in an intermediate FIFO between the sender and the receiver. This FIFO may

be finite or infinite. In case of an infinite FIFO, incoming events are never lost. When adding an event to a finite FIFO, the incoming event may be discarded or the oldest event may be dropped if the FIFO is full. Thus, a single element FIFO may be used to model hardware interrupts. In tasks, events can be sent (NOTIFY), received (WAIT) and tested for their presence (NOTIFIED).

Requests are characterized by a multi-point to one point unidirectional asynchronous communication between tasks. A unique infinite FIFO between senders and the receiver is used to store all incoming requests. Consequently, a request cannot be lost.

3.3 Architecture Modeling

A DIPLODOCUS architecture is built upon the following parameterized hardware nodes:

- **Computation nodes:** Typically, an abstract CPU model merges both the functionality of the hardware component and its Operating System. The behavior of a CPU model can be customized by the following parameters (amongst others): data size, pipeline size, cache miss ratio and scheduling algorithm.
- **Communication nodes**: A communication node is either a bus or a bridge. The bus model exhibits the following parameters: data size, latency and scheduling policy. Note that links which are established during the mapping stage are meant to interconnect a hardware node - except for buses - with a bus. A link may be annotated by a priority if the respective bus has a priority-based scheduling policy.
- **Storage nodes**: Memories are parametrized by two measures: latency and data size.

A DIPLODOCUS architecture is modeled in terms of a UML deployment diagram where DIPLODOCUS links and DIPLODOCUS nodes are depicted by their corresponding UML counterparts.

3.4 Mapping of Applications on Architectures

A DIPLODOCUS mapping is meant to describe the association of application elements - i.e. tasks, channels, requests and events - and hardware nodes. Thereby the following rules apply:

- Abstract tasks must be mapped onto exactly one computation node.
- Abstract communication entities must be mapped onto communication and storage nodes. For the time being, the simulation engine stipulates that a channel is mapped onto n buses, n-1 bridges and exactly one storage element. Furthermore, all connected communication links have to form a continuous path without loops. A future version of the simulator should support several memory elements per channel.

Depending on the mapping semantics, additional parameters may become necessary. For example, when mapping a task on a CPU node having a priority-based scheduling policy, task priorities have to be defined.

The mapping stage is carried out based on previously created DIPLODOCUS architecture

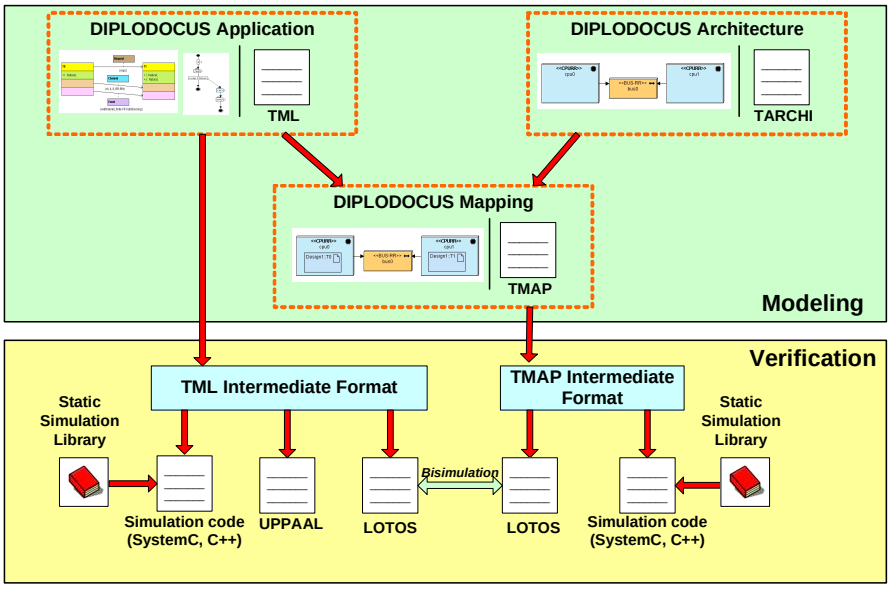


Figure 2: Modeling and verification capabilities of TTool

diagrams: symbols representing tasks and channels are simply bound to hardware components in a drag and drop fashion. An application model captures all possible interleavings of task activities: the mapping phase resolves those interleavings which are not related to non-deterministic operators (choices, time intervals, etc.). As a consequence, mapping models have less possible execution traces than application models

3.5 Toolkit

The DIPLODOCUS UML profile - including its methodology - has been implemented in TTool [4]. TTool is an open-source toolkit that supports several UML2/SysML profiles, including TURTLE [19] and DIPLODOCUS [20]. The main idea behind TTool is that all models may be formally verified or simulated. In practice, UML diagrams are first automatically translated into an intermediate specification expressed in a formal language, which serves as starting point for deriving formal specifications and simulation code (compare Figure 2). Based on the three building blocks (mapping, application, architecture), TTool automatically generates simulation code (C++) or formal specifications at the push of a button. The module which allows the designer to control the simulation in real time by performing step wise execution, saving and restoring simulation states as well as providing live feedback to UML diagrams will be elaborated in section 5.

4. The Simulation Strategy

Before going into details, it should be clearly emphasized what we understand by *fast simulation*. Our fast simulation approach is centered around two basic principles:

- A modeling methodology which allows for conceiving abstract application models by applying both data and functional abstractions.
- A simulation strategy which efficiently exploits these characteristics of the high level model. The granularity of the simulation thereby matches the granularity of the application model.

This implies that the gain in terms of simulation speed can hardly be expressed in cycles per second as it highly depends on the application model. Indeed, in theory the simulator could attain any ratio of cycles/second if transactions were sufficiently large. But in this case the respective model would be far from reflecting key characteristics of the real system for lack of detailedness. The trade-off between detailedness and simulation speed is still subject to our research and can be varied smoothly thanks to the simulation environment.

When experimenting with models such the one of an MPEG2 decoder, simulation speed increased by factor 30 as compared to a cycle-based SystemC simulator [1]. A car communication application where we experienced a simulation speed up to 30 times faster than real time may be also considered as a rough guide.

The simulator detailed in this paper is transaction-based. A transaction refers to a computation internal to a task, or a communication between tasks. Those transactions may obviously span up to hundreds of clock cycles. The duration of transactions is initially defined according to the application model, that is to say the maximum duration is given by the length of the corresponding operator within the task model. At simulation runtime, a transaction may have to be broken down into several chunks of smaller size, just because for example, a bus is not accessible and so the task is put on I/O wait on its CPU. Transaction cutting is more likely to happen if the amount of inter task communication is high and hence the need for synchronization arises. The aggregation of cycles may slightly impact simulation semantics when incorrect branch predictions arise on CPUs. The exact point in time of their occurrence cannot be resolved and thus the resulting delay is determined on a per-transaction base.

Unlike a conventional simulation strategy where all tasks run in lockstep, a local simulation clock is assigned to each active hardware component. The

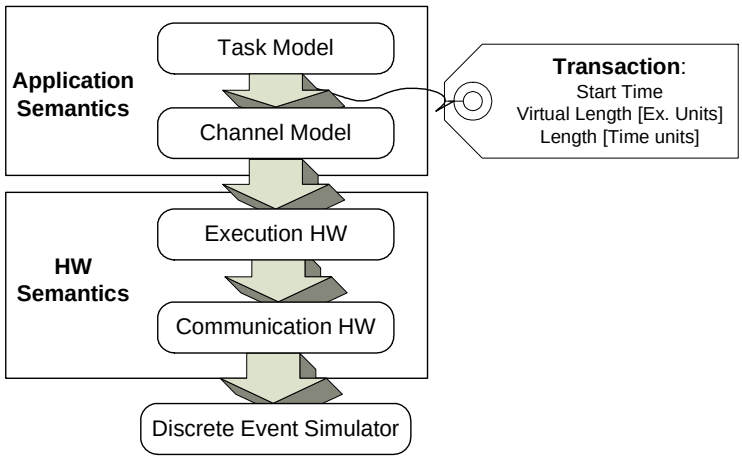


Figure 3: Layered Architecture of the Simulation Environment

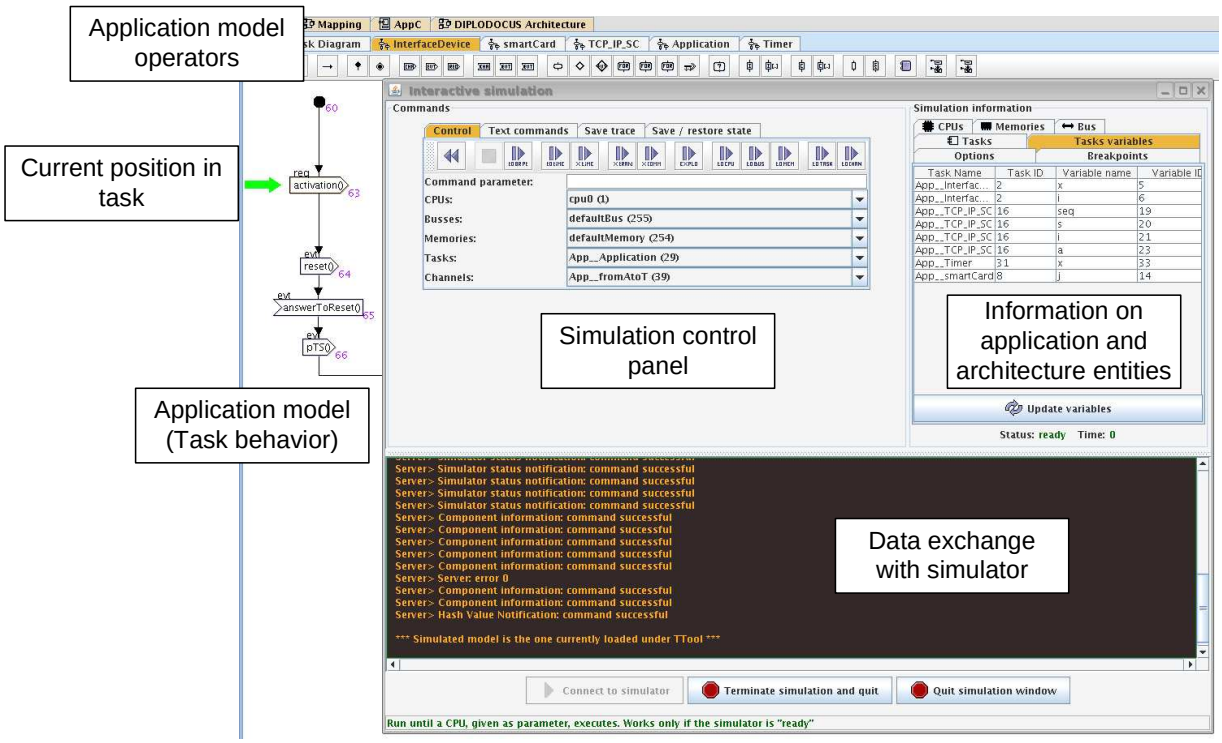


Figure 4: Graphical Interface of the Simulation Environment

simulation clock of a component is advanced upon reception of a transaction whose length is initially set to the cost of the corresponding operator defined within the application model. Thus, the simulation granularity automatically adapts to the application granularity. In case the application model is abstract in the sense of aggregating costs in only few operators, simulation performance increases significantly.

The transaction based simulation embraces mainly three layers (compare Figure 3) representing the semantics of the application model, the semantics of the underlying execution platform and finally a layer dedicated to the discrete event simulation itself. The topmost layer is subdivided into two parts: the first one accounting for the computational part of abstract tasks, the second one for the communication semantics of abstract channels. Based on the knowledge of their internal behavior, tasks are able to determine the next operation which has to be executed within their scope. The abstract cost of this command (in terms of computation or communication units) is subsequently encapsulated in a transaction data structure. The latter is tagged with the earliest possible start time (which corresponds to the finish time of previous transaction of the task). In case the system is dealing with communication or synchronization commands, transactions are forwarded to the dedicated layer for abstract channels. This layer takes into account constraints imposed by the channel semantics like the maximum number of samples to b4e stored, the blocking behavior, etc.

The hardware layer can be decomposed in the same manner as the model layer (as depicted in Figure 3): execution hardware elements (CPUs and hardware accelerators) and communication elements (bridges, buses, memories) alter the time stamp of transactions initially defined at the task layer. The basic idea is that this timing information is used to update the internal clock of hardware elements. Time stamps thus serve as means of synchronization among the local clocks of hardware elements. More precisely, on time management, CPUs have to recalculate the start time of transactions based on their internal schedule and they are able to convert the abstract length to time units. Moreover, a hardware component may delay transactions and modify their duration according to the execution time needed by that specific component. In so doing, the simulation algorithm accounts for the speed of CPUs, the data rate of buses, bus contention and other parameters characterizing the hardware configuration. If several devices are involved in the execution process, transactions are simply passed from one device to another while being modified accordingly. This procedure is for instance applied if an abstract channel at application level is mapped onto several buses and bridges.

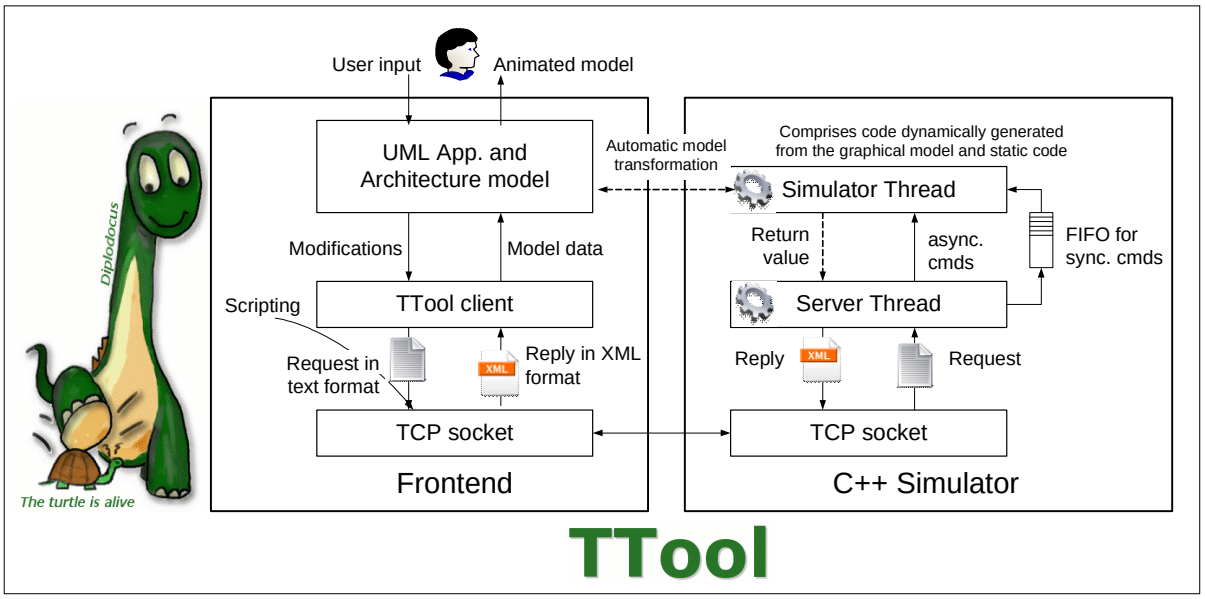


Figure 5: Interaction of the Frontend and the Simulator within the TTool Framework

The final values for start time and duration are hence determined incrementally by forwarding transactions to dedicated hardware nodes. Finally, the main scheduler acts as a discrete event simulator and assures the causality of the simulation. This is achieved by simply querying all CPUs for their next transaction and selecting for execution the transaction with the earliest finish time.

5. Interactive Simulation Environment

5.1 Usage and features

The above mentioned simulation environment has been enhanced and it henceforth allows for an interactive exploration of the application which is mapped onto a particular architecture. After having developed the static view of the application in terms of classes, the behavioral view, the architecture and the mapping, the developer first checks the syntax of the models. If the models comply to the constraints of the meta-model, the next stage is to generate the C++ counterpart of the graphical model. Once the sources have been compiled, the interactive simulation module is launched. All of the aforementioned stages are accomplished at the push of a button. No expertise in C++ programming, simulation or formal verification (in case the model should be verified) is required. The starting point for an interactive exploration is hence the window depicted in Figure 4.

The interface provides the following simulation commands:

- Different flavors of run commands: a given amount of transactions, commands or time units can be simulated...
- …likewise the simulation may be interrupted when a particular hardware element (CPU, bus, bridge, memory) or an application entity (channel) processes a transaction.
- Reset the simulation to the initial state.
- Save and restore the simulation state, especially useful when several branches of control flow are to be looked into.
- Simulation traces may be provided in several formats: the text based format is a simple listing of all transactions encountered on a hardware component This format enables the automatic evaluation of traces and the interchange of data with other applications. The VCD format is supported for the sake of compatibility with standard waveform viewers. The VCD output basically captures bus states (read, write, idle), task states (ready, running, blocked, terminated), CPU states (executing, idle, sleep mode). For debugging purposes of small designs, a user friendly HTML output may give an insight into the application's behavior. Transactions are represented on a time line for each hardware component and colored according to the task they belong to. Figure 6 and 7 show the available output formats.

- Breakpoints can be set graphically on any command within the UML activity diagram simply by selecting a dedicated option in the context menu. Two kinds of breakpoints are supported: conditional and unconditional breakpoints. Unconditional breakpoints stop the simulation whenever the respective command within the activity diagram is reached. Conditional breakpoints interrupt the simulation as soon as a condition is fulfilled which is a function of task variables.
- Furthermore, commands aiming at obtaining information about the simulation state and the state of application and hardware components. The user is not aware of the existence of these commands as they merely serve as a vehicle to convey information to the graphical user interface.
- Commands may also be supplied directly in text format, without using the graphical interface at all. Thus, scripts can be written to automate the simulation procedure.

TTool encompasses a graphical interface to direct the simulation (shown in Figure 4) and thus unburdens the user from familiarizing with a low-level simulation language. The feedback from the simulation engine is exploited by the graphical user interface and used to animate UML application diagrams. For instance, the current command of a task is highlighted so that the user is able to closely follow the simulation progress.

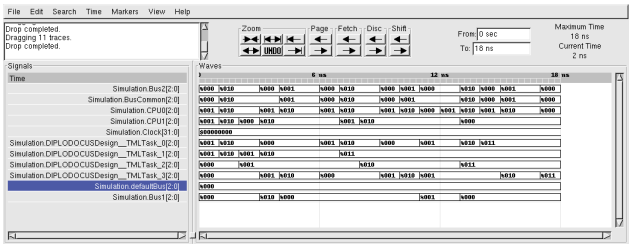


Figure 6: Traces in VCD Format

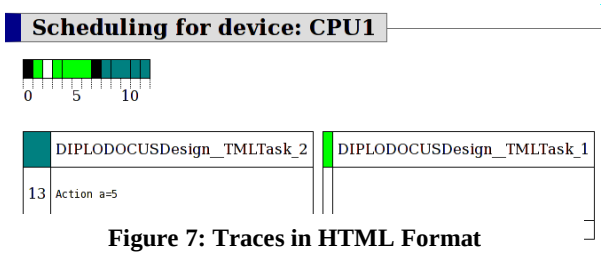
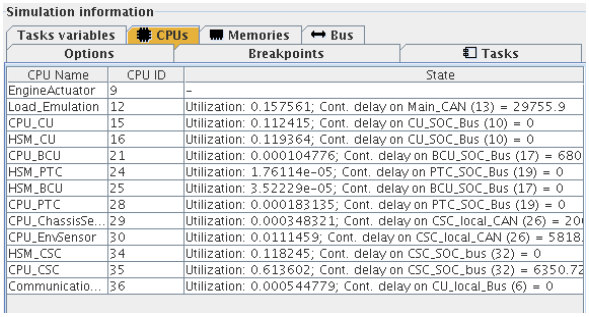


Figure 7: Traces in HTML Format

In addition to traces, simulation results comprise performance figures like the utilization of hardware elements, the contention delay for bus masters, the execution time of tasks, the average time a task gets blocked due to CPU contention, etc (compare Figure 8).



CPU Name	CPU ID	State
EngineActuator	9	-
Load_Emulation	12	Utilization: 0.157561; Cont. delay on Main_CAN (13) = 29755.9
CPU_CU	15	Utilization: 0.112415; Cont. delay on CU_SOC_Bus (10) = 0
HSM_CU	16	Utilization: 0.119364; Cont. delay on CU_SOC_Bus (10) = 0
CPU_BCU	21	Utilization: 0.000104776; Cont. delay on BCU_SOC_Bus (17) = 680
HSM_PTC	24	Utilization: 1.76114e-05; Cont. delay on PTC_SOC_Bus (19) = 0
HSM_BCU	25	Utilization: 3.52229e-05; Cont. delay on BCU_SOC_Bus (17) = 0
CPU_PTC	28	Utilization: 0.000183135; Cont. delay on PTC_SOC_Bus (19) = 0
CPU_ChassisSe...	29	Utilization: 0.000348321; Cont. delay on CSC_local_CAN (26) = 20
CPU_EnvSensor	30	Utilization: 0.0111459; Cont. delay on CSC_local_CAN (26) = 5818
HSM_CSC	34	Utilization: 0.118245; Cont. delay on CSC_SOC_bus (32) = 0
CPU_CSC	35	Utilization: 0.613602; Cont. delay on CSC_SOC_bus (32) = 6350.72
Communicatio...	36	Utilization: 0.000544779; Cont. delay on CU_local_Bus (6) = 0

Figure 8: Simulation Results

5.2 Technical Issues

As illustrated in figure 5, the simulator and the graphical user interface embedded in TTool are hosted in different processes communicating via a TCP connection. Therefore, the simulator and the graphical user interface can be run on different machines for the sake of performance. To get a better understanding of this interaction, let us now follow a user request which aims at defining a breakpoint. The user selects the option by clicking on the respective command within the UML activity diagram. In turn, the logic of the graphical user interface identifies the concerned command and signals a modification to the TTool client. The latter may perform additional checks and wraps information about the command (its ID,…) and the request into a message in text format. The message is sent over the network and received by the server thread of the simulator. The latter distinguishes so called synchronous and asynchronous requests. Asynchronous requests may be issued at any time and normally request information about the simulation without altering the simulation state. Asynchronous requests are handled in the scope of the server thread. Synchronous requests however directly impact the simulation state and must therefore be processed in order. Our breakpoint request is considered as such. Synchronous commands are carried out by the simulator thread which reads the FIFO entries one after another. In case of the breakpoint request, the simulator updates internal data structures accordingly and notifies the successful breakpoint insertion to the server. The server in turn encapsulates the reply into an XML message and sends it over the network. The TTool client subsequently interprets the message and informs the graphical user interface. The latter provides a feedback to the user indicating that the breakpoint has been set successfully.

As stated previously, conditional breakpoints are intended to stop the simulation as soon as a condition (a function of task variables) evaluates to true. To deal with this kind of breakpoints, the simulator has to generate a C++ routine first which is subsequently compiled and attached to the process in the form of a dynamic library. This procedure prevents the cumbersome and costly interpretation of conditions at simulation runtime.

5.3 Usage Scenario

As a simple example, let us consider an algorithm having two main branches which significantly differ in terms of execution time and resource usage in general. For the performance evaluation of a specific architecture, it would be crucial to try out both alternatives. Hence, the coverage of the simulation should be enhanced. As a first step, the designer could benefit from the various conditional run commands so as to get a more intuitive view of the behavior of the application and the interaction of hardware components. The next step could be to reset the simulation and to set a breakpoint on the branch command which is crucial for the continuation of the simulation. The simulation will stop at the previously defined choice command therefore allowing the user to specify which branch he means to explore. In combination with the feature of capturing simulation states, complex scenarios can be evaluated and meaningful traces be recorded. In our example, the user would certainly save the simulation state when reaching the choice command so that it can be restored to study other alternative executions.

6. Conclusions and perspectives

In conclusion it can be said that the contribution of this paper is on the one hand the extension of our simulation environment with a module providing an interactive control of the simulation procedure. On the other hand, the new simulation features have been tightly coupled to our integrated development environment TTool so that simulation progress is directly visualized within the UML diagrams representing the application model. Thus, a powerful toolbox is provided to the designer which is helpful when performing Design Space Exploration. It may alleviate considerably the process of

- Debugging applications
- Accessing intermediate simulation results
- Returning to previous system states
- Enhancing the coverage of the simulation by exploring several control flow branches.

Trading off accuracy against model complexity of hardware components will remain subject to our research. For example, instruction cache-misses and data cache-misses have been accounted for by static probabilities so far. Indeed, as algorithmic details are represented by symbolic instructions, the real code of the application is not available thus making state of the art cache models unsuited. Furthermore, the accuracy of bus and memory models shall be validated against a real embedded system. A fair comparison with a real implementation shall therefore reveal whether a set of parameters can be found to limit the inaccuracy to a reasonable percentage. To simplify the modeling of systems making extensive use of DMA engines, a specific UML stereotype could be introduced. This way, the designer would not have to model DMA transfers explicitly using a dedicated execution unit.

In addition to technical improvements of the simulator, future work will also include the automatic exploration of several alternative executions in order to enhance the simulation coverage. The exploration of some control flow branches could be privileged or abandoned based on certain criteria (CPU usage, resource contention, etc). When taking into account different executions, recurring system states should be tracked so as to be able to merge similar simulation runs.

Regarding formal verification, our environment will be enhanced with a refinement process from the application modeling step to the after-mapping step. The objective is to preserve properties proved at application level.

7. References

[1] Daniel Knorreck, Ludovic Apvrille, and Renaud Pacalet. *Fast simulation techniques for design space exploration*. In Objects, Components, Models and Patterns, volume 33 of Lecture Notes in Business Information Processing, pages 308-327. Springer Berlin Heidelberg, 2009

[2] M. Waseem, L. Apvrille, R. Ameur-Boulifa, S. Coudert, and R. Pacalet. *Abstract application modeling for system design space exploration*. Digital System Design: Architectures, Methods and Tools, 2006. DSD 2006. 9th EUROMICRO Conference on, pages 331-337, 0-0 2006

[3] L. Apvrille, W. Muhammad, R. Ameur-Boulifa, S. Coudert, and R. Pacalet. *A UML-based environment for system design space exploration. Electronics, Circuits and Systems*, 2006. ICECS '06. 13th IEEE International Conference on, pages 1272-1275, Dec. 2006.

[4] TTool, the Turtle Toolkit: http://labsoc.comelec.enst.fr/turtle

[5] Topcased. Topcased, www.topcased.org

[6] Tau. Tau, www-01.ibm.com/software/awdtools/tau

[7] Rhapsody. Rhapsody, www-01.ibm.com/software/awdtools/rhapsody

[8] Artisan. Artisan studio, www.artisansoftwaretools.com/products/artisan-studio

[9] Coware Virtual Platforms www.coware.com.

[10] Vast System Engineering Tools www.vastsystems.com.

[11] Members of the SystemC Verification Working Group. *SystemC Verification Standard Specification* Version 1.0e, www.systemc.org. 2003.

[12] Bastian Ristau, Torsten Limberg, and Gerhard Fettweis. *A mapping framework for guided design space exploration of heterogeneous mp-socs*. Design, Automation and Test in Europe, 2008. DATE '08, pages 780-783, March 2008.

[13] Jorgiano Vidal, Florent de Lamotte, Guy Gogniat, Philippe Soulard, and Jean-Philippe Diguet. *A co-design approach for embedded system modeling and code generation with uml and marte*. In Design, Automation & Test in Europe Conference & Exhibition, 2009. DATE '09., pages 226-231, April 2009.

[14] A. D. Pimentel and S. Polstra and F. Terpstra, *Towards efficient design space exploration of heterogeneous embedded media systems*, In Embedded Processor Design Challenges: Systems, Architectures, Modeling, and Simulation, pages 57-73, Springer LNCS

[15] A.D. Pimentel, C. Erbas, and S. Polstra. *A systematic approach to exploring embedded system architectures at multiple abstraction levels*. Computers, IEEE Transactions on, 55(2):99-112, Feb. 2006.

[16] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst. *System level performance analysis - the symta/s approach*. Computers and Digital Techniques, IEE Proceedings -, 152(2):148-166, Mar 2005.

[17] Object Management Group. UML 2.0 Superstructure Specification. 2003

[18] P. Lieverse, P. van der Wolf, E. Deprettere, and K. Vissers. *A methodology for architecture explorationof heterogeneous signal processing systems*. In Signal Processing Systems, 1999. SiPS 99. 1999 IEEE Workshop on, pages 181-190, 1999.

[19] L. Apvrille and J.-P. Courtiat and C. Lohr and P. de Saqui-Sannes, *TURTLE: A Real-Time UML Profile Supported by a Formal Validation Toolkit*, volume 30 of IEEE transactions on Software Engineering 2004, pages 473-487.

[20] L. Apvrille. *TTool for DIPLODOCUS: An Environment for Design Space Exploration*. In Proceedings of the 8th Annual International Conference on New Technologies of Distributed Systems (NOTERE'2008), Lyon, France, June 2008.

[21] S. Chakraborty, S. Kunzli, and L. Thiele. *A general framework for analysing system properties in platform-based embedded system designs*. In Design, Automation and Test in Europe Conference and Exhibition, 2003, pages 190-195, 2003.

[22] Arne Hamann, Marek Jersak, Kai Richter, and Rolf Ernst. *A framework for modular analysis and exploration of heterogeneous embedded systems*. Real-Time Syst., 33(1-3):101-137, 2006.

[23] Tero Arpinen, Erno Salminen, Timo Hämäläinen, and Marko Hännikäinen. *Performance evaluationof uml2-modeled embedded streaming applications with system-level simulation*. EURASIP Journal on Embedded Systems, 2009, March 2009.

[24] Jorgiano Vidal, Florent de Lamotte, Guy Gogniat, Philippe Soulard, and Jean-Philippe Diguet. *A co-design approach for embedded system modeling and code generation with uml and marte*. In Design, Automation & Test in Europe Conference & Exhibition, 2009. DATE '09., pages 226-231, April 2009.

[25] D. Knorreck, L. Apvrille, R. Pacalet Formal System-level Design Space Exploration. In Pro-ceedings of the 10th Annual International Conference on New Technologies of Distributed Systems (NOTERE'2010), Tozeur, Tunisia, June 2010 (to appear).

[26] P. Lieverse, P. van der Wolf, E. Deprettere, and K. Vissers. A methodology for architecture exploration of heterogeneous signal processing systems. In Signal Processing Systems, 1999. SiPS 99. 1999 IEEE Workshop on, pages 181-190, 1999.