A UML-based Environment for System Design Space Exploration

Ludovic Apvrille, Waseem Muhammad, Rabéa Ameur-Boulifa, Sophie Coudert and Renaud Pacalet
System-on-Chip laboratory (LabSoC), GET/ENST
2229, routes des Crêtes
BP 193
F-06904 Sophia-antipolis Cedex
Email: ludovic.apvrille@enst.fr

***Abstract*— The increasing complexity of System-on-Chip (SoC) and time-to-market constraints raise new methodological issues. To address these issues, this paper introduces a UML-based SoC modeling approach mixing simulation and formal verification techniques. A UML profile called DIPLODOCUS has been specified. Transformation rules were defined for generating from UML models either a SystemC model or a formal specification given in LOTOS. Thus, relying on SystemC or LOTOS tools the profile allows fast simulation or formal verification techniques to be used over the UML modeling. A toolkit supporting this profile has been implemented. The overall approach is experimented for the design of a telecommunication system.**

I. Introduction

Design space exploration is the process of analyzing various functionally equivalent implementation alternatives to select an optimal solution. In the traditional top down design approach, the designer starts with an informal specification and develops a reference model in some high-level language such as Matlab, C or C++. This model is then verified for functional correctness according to system's specification and is used to get rough estimates of its performance requirements. This initial step is followed by manual or semi-automatic generation of several alternative designs, which are subjected to a series of time-consuming and typically ad-hoc evaluations. Finally the most suitable design is chosen based on various metrics such as performance, cost, power, reliability, and flexibility. For modern embedded systems, it becomes more and more important to have efficient tools for system-level design space exploration, especially at an early design stage where the design space is very large. Unfortunately, most of the available design space exploration tools are not mature enough to satisfy all these requirements. They are either over-detailed or too superficial for exploration.

This paper proposes a methodology for modeling SoC applications at a high abstraction layer. The approach is centered around four principles: use of a high-level well-known language (UML), separation of application and architecture, data abstraction, and at last, use of simulation and static formal analysis techniques on abstract models. A UML profile called DIPLODOCUS is therefore introduced. A UML profile customizes the UML language for a given field of interest: in our case, the DIPLODOCUS[1] UML profile focuses on design space exploration with in mind the four points mentioned above. Its strength relies on transformation rules that make it possible to automatically transform DIPLODOCUS modelings either in SystemC [1], for simulation purpose, or in a LOTOS specification, a formal description technique standardized [2] and supported by formal validation tools [3].

The paper is organized as follows. Section 2 overviews related work. Section 3 presents the DIPLODOCUS profile: methodology, diagrams / operators, and TTool, a toolkit supporting this UML profile. Section 4 focuses on the semantics of the profile. A telecommunication system exemplifies our approach in section 5. At last, section 6 concludes the paper.

II. Related Work

A. Design Space Exploration

Contributions, such as [4]–[9] have already been proposed for addressing issues previously mentioned on design space exploration. They are commonly based on several points. First, they rely on high-level languages for separately describing functionalities and architecture. Second, they propose a way to map functions over hardware execution nodes. Third, they introduce simulation techniques to simulate the system built from the mapping of functions over hardware nodes. Their main drawback relies in the verification of the system since, most of the time, they only support slow-simulation techniques.

B. Use of UML and SystemC

Several previous research works address the use of UML for modeling mixed hardware and software systems. [10] proposes, instead of directly using the SystemC language, to rely on UML. Thus, they introduce an environment making it possible to model all SystemC components in UML, relying on the SPT profile defined at the OMG, and the profile defined in the RoseRT Toolkit. [11] introduces a UML2-based environment allowing the modeling of applications, architectures, and mapping of an application onto a given architecture. The application is described as a set of UML

[1]DIPLODOCUS stands for DesIgn sPace exLoration based on fOrmal Description teChniques, Uml and SystemC

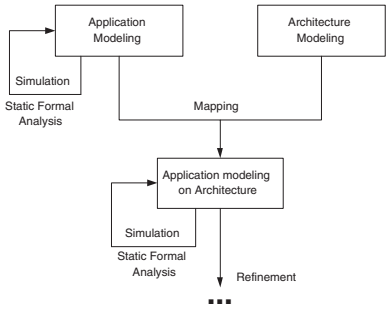


Fig. 1. Design methodology

active classes for which a behavior must be provided. The hardware platform is described using libraries of hardware components such as execution units (DSP, etc.), communication segments (with parameters such as data width, frequency, arbitration) and communication wrappers (with parameters such as address and buffer size). After the mapping step, code may be generated and simulated. At last, [12] relies on UML class and state diagrams, and on an abstract execution platform: UML diagrams may be translated to a bytecode file that can be executed over that execution platform. The latter has been implemented over a FPGA. Finally, and contrary to the three contributions mentioned above, the strength of our approach relies in abstractions, making it possible to perform fast simulations and apply static analysis techniques. Moreover, our approach has been implemented into a toolkit.

III. DIPLODOCUS: A UML PROFILE

A UML profile [13] selects a subset of the UML, extends this subset with elements such as stereotypes, and provides a methodology. Those points are explained in this section.

A. Methodology

The UML-based overall methodology we propose for system design space exploration phase is depicted in Fig. 1. It consists of 4 steps:

1) Abstract application modeling using two kinds of UML diagrams: a DIPLODOCUS class diagram modeling tasks, and activity diagrams for the behavior of those tasks. Simulation or static analysis can be performed from those diagrams.
2) Architecture modeling as a composition of instances of five generic components: CPU, bus, memory, hardware accelerator and input/output peripheral. These components are abstract and parameterized through a small set of simple parameters.
3) Mapping each task onto an execution node of the architecture.
4) Refining the application to go for the final implementation.

The contribution of this paper is focused on the first point: modeling the application and verifying this modeling with simulation and formal verification. At this abstraction level, we don't consider any architecture on which these applications are to be executed. Also at this level there is no difference between hardware and software tasks because no partitioning is defined yet. There is no data processing details inside the tasks. They are only control oriented without any notion of physical time. However operations within a task model are totally ordered and among a set of tasks, they are partially ordered.

B. Diagrams and operators

To model an application, with data abstraction in mind, DIPLODOCUS is based on two main characteristics:

- Functionalities are organized under the notion of tasks, communicating using three paradigms: *channels*, *events*, and *requests*.
- The behavior of each task must be described.

To define an appropriate UML profile for modeling these software and hardware tasks, we have relied on past work on the TURTLE UML profile [14], and more precisely on TURTLE designs. Thus *tasks are modeled using active UML classes, called dclasses, and communication between dclasses are modeled within a UML class diagram. The behavior of each task must be described using a DIPLODOCUS activity diagram*. A dclass is a stereotype of DIPLODOCUS. A dclass may define a set of attributes but, unlike regular UML classes, may not define operations. Tasks can be connected using three composition operators: *Channel*, *Event*, and *Request*. A composition operator is meant to attribute an association between dclasses: this provides associations with semantics. Only one composition operator may attribute an association, but several associations may link the same two dclasses.

Channels represent a way for modeling data exchange, without any knowledge about the implementation details of the underlying communication infrastructure. Therefore, in DIPLODOCUS, we specify channels for task-to-task communication instead of specifying the hardware models as in [4]. At mapping time, correspondence between channels and architecture components is set. Three types of channels have been defined: finite FIFO, infinite FIFO, and shared memory (read and write operations never block). Channels are also attributed with a size information.

Events are a mean for tasks to exchange signals. A task may wait for a signal, or send a signal to another task. Signals may convey values.

At last, *Requests* are a mean for tasks to spawn a new task. More precisely, when a task sends a request to another task, if the requested task is already running, the request is stored into a FIFO: the request is performed once other requests have been computed. Also, requests may convey values.

An example of DIPLODOCUS class diagram is provided in Fig. 4.

Apart from regular components of activity diagrams, we have added the following operators: write to channel, read from channel, send an event, wait for an event, send a request, loop structure, sequence, time interval. The semantics of these operators is provided afterward. Time intervals are used to represent the lower and upper bounds time taken by operations on integers, floats, or customized elements. Moreover, time

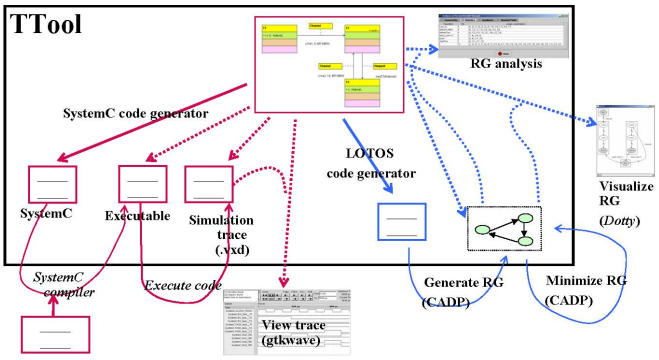


Fig. 2. TTool for DIPLODOCUS: main functionalities

intervals are particularly useful for modeling various possible execution paths in an abstract way. At application modeling level, there is no notion of physical time but these timing operations are used for simulation purpose. At SystemC simulation step, in order to visualize the interactions among tasks, we have assumed that each DIPLODOCUS operator takes one tick, where a tick can be defined as a simulation parameter (e.g 1 ns or 1 clock cycle) and can be changed accordingly.

C. TTool: a toolkit supporting DIPLODOCUS

TTool is a toolkit that has first been developed to support the TURTLE profile [15]. It is released as an open-source software that can be freely downloaded and used. The last beta version has been enhanced to support the DIPLODOCUS profile. Indeed, TTool enables a designer to create a DIPLODOCUS design composed of a dclass diagram plus one activity diagram per dclass. From a design, it is possible to generate SystemC [1], compile it and execute it. Execution traces may then be visualized using *gtkwave*. Also, from a design, LOTOS code may be generated and then, from a LOTOS specification, a reachability graph can be generated, using CADP [3] and further minimized, analyzed or visualized. It is important to note that the generation of SystemC or LOTOS code, and generation of simulation traces or reachability graphs is totally transparent to TTool's users i.e. absolutely no knowledge of SystemC or LOTOS is necessary to use the toolkit (see Fig. 2).

IV. Semantics of DIPLODOCUS

This section focuses on one of the two semantics defined in the DIPLODOCUS profile: the LOTOS semantics, used for formal validation purpose. While supported, the SystemC semantics used for simulation purpose is less relevant at this level. It is intended to be used after mapping on a physical architecture.

LOTOS [2] (Language of Temporal Ordering) is a formal specification language introduced for modeling temporal ordering between events. It is based on process algebra. Basically, a LOTOS specification structures a system with processes on which binary operators may be applied: parallel, synchronization on gates, sequence, preemption and choice. Also, complex data types may be defined in LOTOS.

A LOTOS specification is generated from DIPLODOCUS designs as follows:

- For each channel, event, or request, a particular LOTOS process is generated. For channels, this process relies, if necessary, on FIFO data types added to the LOTOS specification. Writing a message to a channel corresponds to a synchronization to the process managing the channel. Also, reading a message from a channel corresponds to a synchronization to the process managing the channel.
- For each dclass of the class diagram, a main process is generated. Its body corresponds to the translation of the activity diagram of that dclass. Several subprocesses may have to be defined, according to operators used in the activity diagram. Also, when the dclass is requested, a wait for event synchronization is added at the beginning of its main LOTOS process, and an infinite loop over this process is specified.

The table 3 summarizes, in a very basic way, the translation of some activity diagram operators, both in SystemC and LOTOS.

V. Case study: A telecommunication subsystem

WIDENS (Wireless Deployable Network System) is a European project for the implementation of ad-hoc mobile communication networks for public safety organizations. As a case study, we started from the specification of different services offered by the mobile terminals. These services are written in C and Matlab. Among these services, we have chosen OFDM modulation/demodulation at physical layer called the Broadcast Channel Encoder (BCH).
The modeling of BCH with DIPLODOCUS is composed of a class diagram with 12 dclasses connected throughout channels, events and requests. Five dclasses model the functional blocks, and some others are used for initialization of lookup tables and buffers at system startup. *BCH Encoder* is the main dclass which requests other dclasses using *REQ* in a sequence. Some dclasses further request others. An excerpt of this diagram is represented at Fig. 4. Also, Fig. 5 depicts the activity diagram of CRC_Table class. From this specification, we were able to generate SystemC code, and simulate it to obtain traces, and to generate a LOTOS specification, and obtain reachability graphs for subpart of the application.

VI. Conclusion and future work

The presented framework, while already usable, will be enhanced in several ways. The physical architecture modeling and the mapping will be tightly integrated. The simulation performances will be evaluated on real size systems and the simulation engine will be optimized. On the static analysis point of view, the tradeoff between the abstraction level and the combinatory explosion problem will be addressed. Last but not least a formal link between the fully functional model of the application and its DIPLODOCUS counterpart - where data processing is abstracted - will be added.

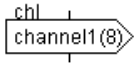
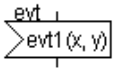
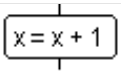
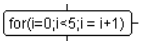
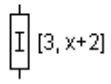
Icon	Description	SystemC	LOTOS
	Sending 8 samples in *channel1*	task.WR(8, channel1);	write__channel1!8
	Waiting for *evt1* with 2 params	task.WAIT(x, y, channel1);	wait__evt1?x?y
	Action state; here, increment x.	x = x + 1;	P[gates](.,x+1,.)
	Loop structure (loop on i)	for(i=0;i<5;i=i+1) { /* loop */} /* after loop*/;	process P[gates](.,0,.) $[i < 5] - >$ /* loop */ P[gates](.,i+1,.) $[not(i < 5)] - >$ /* after loop */
	Time interval between 3 and x+2	t = TML_tasks::myrand(3, x+2); task__T2.EXECI(t);	ignored (timing information are necessary for simulation only)

Fig. 3. Semantics of various operators of DIPLODOCUS activity diagrams

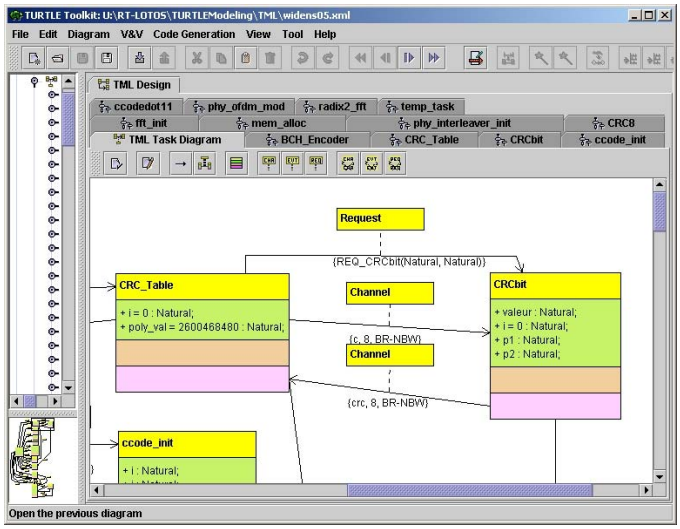


Fig. 4. WIDENS: class diagram made with TTool

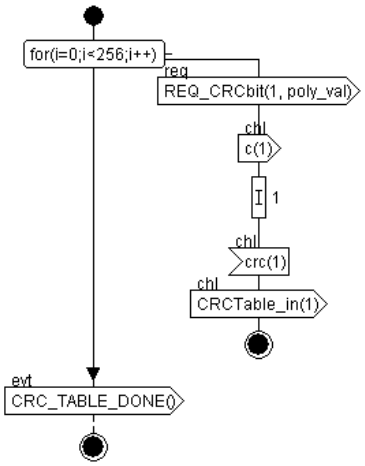


Fig. 5. WIDENS: activity diagram of CRC_Table task

REFERENCES

[1] "SystemC," in *http://www.systemc.org*.

[2] ISO-LOTOS, "A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour," in *Draft International Standard 8807, International Organization for Standardization - Information Processing Systems - Open Systems Interconnection*, Geneva, July 1987.

[3] V. T. (INRIA), "CADP Toolkit," in *http://www.inrialpes.fr/vasy/cadp*.

[4] A. Chatelain, G. Placido, L. R. A, Y. Mathys, and L. Lavagno, "High-level architectural co-simulation using Esterel and C," in *Proc. of IEEE/ACM symposium on Hardware/software codesign*, April 2001.

[5] M. Silbermintz, A. Sahar, L. Peled, M. Anschel, E. Watralov, H. Miller, and E. Weisberger, "SOC Modeling Methodology for Architectural Exploration and Software Development," in *Proc. of the IEEE International Conference on Electronics,Circuits and Systems*, Dec 2004.

[6] P. Lieverse, P. der Wolf, E. Deprettere, and K. Vissers, Eds., *A Methodology for Architecture Exploration of Heterogeneous Signal Processing Systems*, Oct 1999, siPS'99.

[7] J. Coffland and A. Pimentel, "A Software Framework for Efficient System-level Performance Evaluation of Embedded Systems," in *Proc. of the ACM Symposium on Applied Computing*, March 2003.

[8] A. Pimentel and C. Erbas, "An IDF-based Trace Transformation Method for Communication Refinement," in *Proc. of the ACM/IEEE Design Automation Conference*, June 2003.

[9] G. Vanmeerbeeck, P. Schaumont, S. Vernalde, M. Engels, and I. Bolsens, "Hardware/Software Partitioning of Embedded System in OCAPI-xl," in *Proc. of the ACM Hardware Software Codesign Conference*, 2001.

[10] A. Chureau, Y. Savaria, and E. M. Aboulhamid, "The role of model-level transactors and UML in functional prototyping of systems-on-chip: a software-radio application," in *Proc. of the Design, Automation and Test in Europe (DATE)*, June 2005, pp. 698–703.

[11] P. Kukkala, M. Hannikainen, and T. T.D.A. Hamalainen, "Performance Modeling and Reporting for the UML 2.0 Design of Embedded Systems," in *Proc. of the 2005 International Symposium on System-on-Chip)*, Nov 2005, pp. 50–53.

[12] T. Schattkowsky, W. Mueller, and A. Rettberg, "A model-based approach for executable specifications on reconfigurable hardware," in *Proc. of the Design, Automation and Test in Europe (DATE)*, Nov 2005, pp. 692–697.

[13] OMG, "UML 2.0 Superstructure Specification," in *http://www.omg.org/docs/ptc/03-08-02.pdf*, Geneva, 2003.

[14] L. Apvrille, J.-P. Courtiat, C. Lohr, and P. de Saqui-Sannes, "TURTLE: A Real-Time UML Profile Supported by a Formal Validation Toolkit," in *IEEE transactions on Software Engineering*, vol. 30, no. 7, Jul 2004, pp. 473–487.

[15] LabSoc, "The TURTLE Toolkit," in *http://labsoc.comelec.enst.fr/turtle/ttool.html*.