Demonstration of a Coverage Driven Verification Environment for UML Models of Systems-on-Chip

Daniel Knorreck, Ludovic Apvrille, Renaud Pacalet
System-on-Chip Laboratory (LabSoC), Institut Telecom, Telecom ParisTech, LTCI CNRS
2229, Routes des Crêtes BP 193 F-06904 Sophia Antipolis, France
Email: {daniel.knorreck, ludovic.apvrille, renaud.pacalet}@telecom-paristech.fr

Abstract— **The DIPLODOCUS UML profile targets the partitioning of Systems-on-Chip early in the design flow, following the Y-Chart approach: application modeling, architecture modeling, mapping. DIPLODOCUS is supported by an open-source toolkit named TTool. One of the strengths of TTool is its capability to transform high level models into both simulation and formal models so as to obtain performance results and proofs of functional properties respectively. The version of TTool we presented at DATE 2010 featured two extreme cases: simulation and formal verification. The former produces only one possible system execution, the latter exhaustively explores the entire state space of the mapping model. Despite abstractions, formal verification of medium sized models pushes model-checkers to their limits. This year's demonstration presents a brand new strategy to enhance simulation with dynamic coverage of the state space, which may be guided interactively by the user or automatically conducted based on system requirements. For that purpose, we combine several techniques, including the static analysis of UML models, and also model checking techniques.**

I. Demonstration Description

A. Introduction

The demonstration presents TTool [1], a UML toolkit implementing several formal UML profiles, including DIPLODOCUS. DIPLODOCUS is a UML profile for the design and partitioning of Systems-on-Chip at a high level of abstraction. TTool provides diagramming facilities for DIPLODOCUS diagrams, as well as simulation and formal verification at the push of a button from DIPLODOCUS diagrams.
The increasing complexity of today's Systems-on-Chip (SoC) advocates for verification as early as possible in the design flow, even when low level models of those SoC are not available yet. Indeed, reconsidering design choices late in the design flow turns out to be extremely costly. Design Space Exploration at system level relies on high level models of the target system to identify the most suitable hardware/software platform complying to given constraints. To assess the satisfiability of both functional and non functional requirements (e.g., delay, throughput, latency), simulation [9] and static formal analysis techniques [8] have been introduced in the DIPLODOCUS UML profile: this profile, which is MARTE compliant, specifically targets the partitioning of Systems-on-Chip. DIPLODOCUS follows the Y-Chart approach [12], that is, its supports a three-stage methodology: application, architecture and mapping stages. Since formal verification on low-level models faces the state explosion problem, it is merely applied to subparts of the system where data has been abstracted to its mere presence or absence. While this limitation is partially resolved when raising the abstraction level, the state explosion issue still remains a major obstacle for the verification of medium sized system level models. Furthermore, representing high-level mapping models is extremely cumbersome in formal languages. Methods have been proposed to transform graphical high level models (UML, etc.) endowed with a formal semantics into a representation which can be analyzed by model checkers. Nonetheless, we experienced that the transformation of sophisticated mapping models results in complex syntactical structures, often pushing both UPPAAL [3] and CADP [5] model checkers to their limits. Moreover, even if the specification is accepted by the model-checker, system space coverage cannot be varied: formal verification is exhaustive by definition and model-checkers are conceived to explore all possible states. As a consequence, varying the coverage of the initial model requires to reconfigure the model transformation algorithm and to regenerate the formal model. However, from a designer's perspective, it would be desirable to generate an executable model once and to subsequently traverse an interesting fraction of its state space. Criteria for electing this subset may range from explicit selection by the user (e.g., to simulate a given ratio of various possible UML branches) to heuristics taking into account (non-) functional properties. In the latter case decisions concerning the coverage are made on the fly, at simulation runtime. The ultimate objective is to let the user **trade off simulation time against coverage of the UML model** (see Figure Figure 1).

In last year's demonstration, mapped models were assessed using formal verification and simulation (compare Figure 1, extreme cases). The former is not always well suited for the analysis of mapped models, as mentioned initially. The latter offers to the designer two options: either non-determinism in the application model is simply resolved using a random number generator, or different executions may be manually checked with interactive simulation capabilities (breakpoints, state saving, state recovery, etc). There was no automated exploration feature available covering an automatically or manually determined subset of the state space (see Figure 1, intermediate case): this limitation is

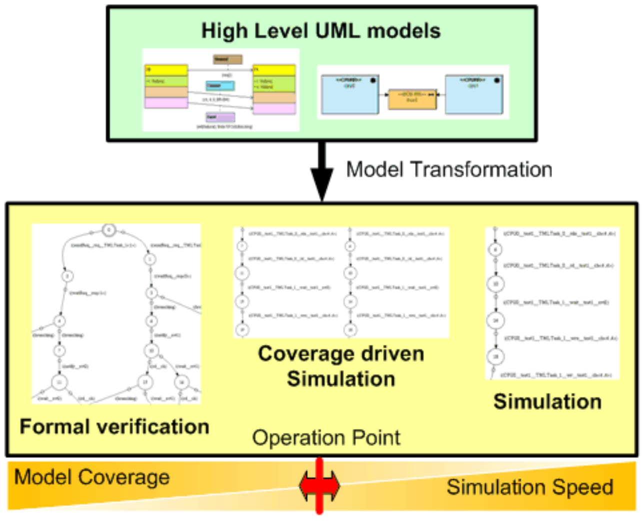


Fig. 1. Varying Model Coverage in DIPLODOCUS

addressed in the following. We propose a novel way to **enhance simulation coverage** of high level models based on model checking and static program analysis techniques. More specifically, DIPLODOCUS mapping models are statically analyzed to spot non-significant elements not being part of the state vector so as to speed up the identification of recurring system states.

The remainder of this description is structured in four sections: Section I-B provides an overview of the DIPLODOCUS UML profile. Section I-C presents the main technical elements of the demonstration. Then, the paper reviews relate works (section I-D) and draws perspectives (section I-F).

B. DIPLODOCUS

DIPLODOCUS is based on the following fundamental principles:

- Use of a **widely accepted** high level **language**: UML.
- Clear **separation of application and architecture**, enforcing the so called Y-Chart approach [12].
- **Data abstraction**: only the amount of data exchanged between functional entities is modeled. Data dependent decisions are abstracted and expressed in terms of non-deterministic and statistical operators.
- **Functional abstraction**: algorithms are described using **abstract cost operators**. The complexity of computations is thus taken into account without actually having to carry them out.

DIPLODOCUS design stages are (Figure 2) :

1) **Applications** are first described as a network of abstract communicating tasks using a UML class diagram. Each task behavior is expressed in terms of a UML activity diagram.
2) Targeted **architectures** are modeled independently from applications as a set of interconnected generic hardware nodes (e.g. CPUs, buses, memories, hardware accelerators, bridges), using UML deployment diagrams.
3) A **mapping** defines how application tasks are bound to execution entities, and communications between tasks are assigned to communication and storage nodes.

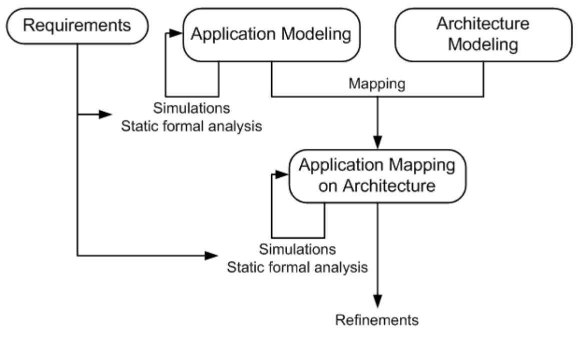


Fig. 2. Global view of our Design Space Exploration Approach

Within a SoC design flow, Design Space Exploration is carried out at a very early stage. Hence, the main DIPLODOCUS objective is to help designers to spot a suitable hardware / software architecture even if algorithmic details have not yet been stipulated thoroughly. To achieve this, DIPLODOCUS relies on fast simulation [9] and formal proof techniques [8], both at application and mapping level. 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, formal techniques may be applied before and after mapping.

C. Extending DIPLODOCUS for Coverage Driven Simulation

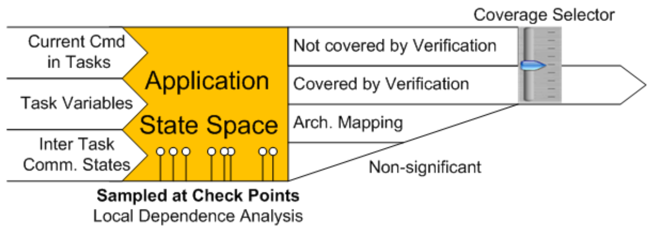


Fig. 3. State Space Exploration Concept

The state space of the application comprises all possible interleavings of task executions and is only constrained by inter-task synchronization. As depicted in Figure 3, the space is spanned by the actual command in a task, local task variables and the state of inter task communication primitives. The mapping of application tasks onto an architecture further constrains this space by introducing shared resources like processors, buses, etc (cf. Constraints arrow in Figure 3). However, as data dependent behavior is abstracted with non-deterministic operators, several execution traces are still possible. They may significantly differ in terms of non-functional properties like execution time, resource usage, etc. and in whether they satisfy functional requirements. It may therefore be important to explore more than one possible branch. Moreover, designer intuition, measurements taken during simulation or an analysis of formally expressed properties in conjunction with the application model could lead simulation into the right direction.

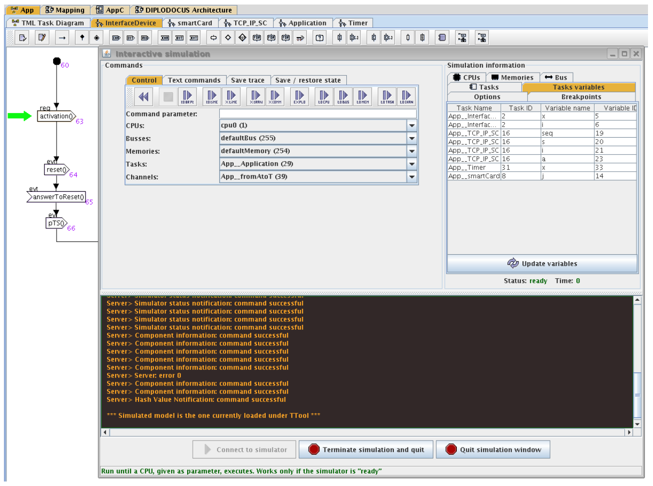
Our development environment TTool has recently been enhanced with techniques essential to the realization of the coverage selector shown in Figure 3. These techniques can be applied regardless of how the selector setting is actually determined. To prevent execution paths from being explored more than once, representations of encountered states have to be maintained, similar to what is done in Model Checkers like SPIN [7]. Then, the challenge is twofold: measures have to be taken to minimize the size of particular state vectors and the number of times the latter have to be stored and compared.
As explicit state representations are greedy in terms of memory, a thorough analysis of the application yields the minimal state information (see bold arrow in Figure 3). *Reaching Definition Analysis*, *Live Variable Analysis* and *Induction Variable Analysis* are commonly accomplished by compilers in the context of machine independent optimizations [2]. They reveal constant variables and variables whose content is not significant at a specific point in a task.
Moreover, *Local Input Dependence Analysis* detects so called *Check Points* in a task where traces are likely to branch or join, thereby giving directions on when to compare and store states (cf. sampling symbols in Figure 3).

D. Related work

There have been several efforts to extend the insight provided by simulation in order to address the changing behavior and demands of today's embedded applications. To this end, simulation techniques are often combined with formal methods. In the following, we roughly survey the landscape of related work in the field of high level modeling.
The common ground of all approaches is the attempt to examine a large fraction of possible system executions, optimally the whole state space whilst limiting verification run time and memory consumption. The listed work differs from ours in the way concerns (application, architecture) are separated [4] [6], the abstraction level of the task model is chosen [14] [15] [13], and the model of computation emphasizes data or control flow [11].

E. The Demonstration - TCP/IP Protocol Implementation of a Smart Card

In our demonstration, we illustrate the above described simulation and verification features in TTool (screenshot depicted in Figure 4) by means of the model of a TCP/IP Protocol implementation of a smart card. A smart card has



Command parameter:
CPUs: cpu0 (1)
Busses: defaultBus (255)
Memories: defaultMemory (254)
Tasks: App__Application (29)
Channels: App__fromAtoT (39)

Task Name	Task ID	Variable name	Variable ID
App__Interfac...	2	x	5
App__Interfac...	2	i	6
App__TCP_IP_SC	16	seq	19
App__TCP_IP_SC	16	s	20
App__TCP_IP_SC	16	i	21
App__TCP_IP_SC	16	a	23
App__Timer	31	x	33
App__smartCard	8	j	14

Status: ready Time: 0

```
Server> Simulator status notification: command successful
Server> Simulator status notification: command successful
Server> Simulator status notification: command successful
Server> Simulator status notification: command successful
Server> Component information: command successful
Server> Component information: command successful
Server> Component information: command successful
Server> Component information: command successful
Server> Component information: command successful
Server> Server: error 0
Server> Component information: command successful
Server> Component information: command successful
Server> Hash Value Notification: command successful

*** Simulated model is the one currently loaded under TTool ***
```

Run until a CPU, given as parameter, executes. Works only if the simulator is "ready"

Fig. 4. The TTool Development Environment

the size of a credit card and is equipped with a microchip that securely stores data mainly used for identification purposes. The data may be periodically refreshed in order to maintain or enhance the functionality of the card. Smart cards are commonly used for telephone calling, electronic cash payments, establishing identity when logging on to some online account or when demanding public health services, paying small amounts of money (bus, parking, subway fees, etc.). Smart Cards comprise several hardware components like a microprocessor and different kinds of (non-)volatile memories (ROM, EEPROM, RAM Flash). Most smart card systems adhere to the ISO-7816 standard which includes multiple parts defining for example physical characteristics, dimensions, involved protocols and other system properties. For the creation of our model, we mainly relied on the third part of the standard dealing with electronic signals and transmission protocols (ISO 7816-3).

1) Application modeling: The communication application has been decomposed into four DIPLODOCUS tasks corresponding to the main functional blocks. A task called *InterfaceDevice* represents the terminal the smart card communicates with, for instance the card reader at a cash desk. Another task (*SmartCard*) models the transmission protocol defined in ISO 7816-3. The *Application* task models a basic exemplary application which merely makes use of the basic TCP services: establishing a session, sending some application data and finally tearing down the connection. The *TCP* task accounts for the different phases of the TCP protocol like connection establishment, data transfer, connection termination. Last but not least, a *Timer* task may trigger time outs in the main *TCP* task.

2) Architecture and mapping: To demonstrate the applicability of our methodology, two candidate architectures are experimented with. A first basic mapping consists of one single CPU onto which all tasks are mapped. A second option is to map the *SmartCard*, *TCP* and *Application* tasks on one CPU named *MainCPU*, and to provide a dedicated Hardware Accelerator to the *Timer* and *InterfaceDevice* task

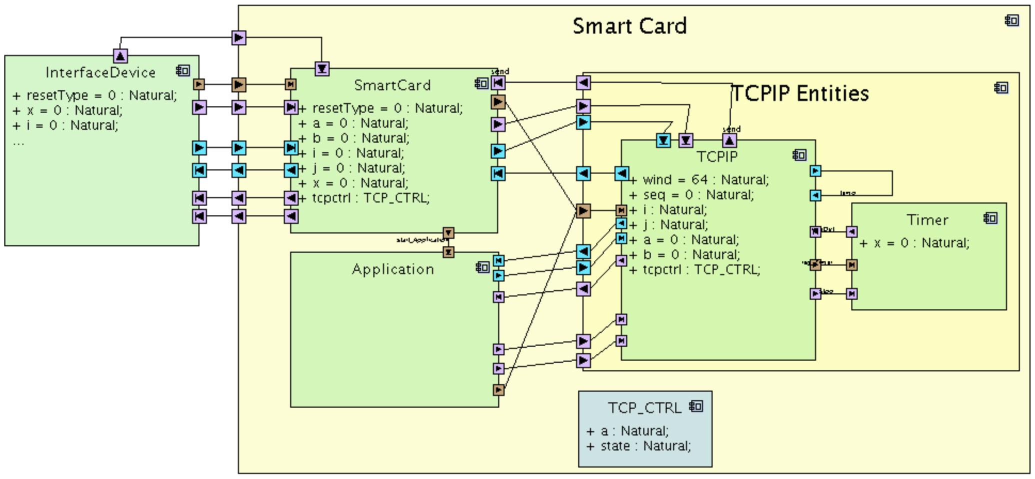


Fig. 5. Component based Diagram of the Smart Card Application

respectively. The three CPUs are connected via an on chip bus. In this second mapping, up to three tasks may execute concurrently and thus application level parallelism can be better exploited. Due to data and synchronization dependencies, further increasing the number of processing elements would not yield considerable performance improvements.

F. Conclusions and future directions

The demonstration highlights the combination of static analysis and model checking techniques to enhance the simulation coverage of UML models of SoC. This methodology yields a trade-off between exhaustive and costly formal verification and efficient simulation exhibiting a limited coverage. The two latter corner cases were already supported by the DIPLODOCUS methodology and presented at DATE 2010. As opposed to conventional UML frameworks, both simulation and formal verification outreach the functional level by considering constraints imposed by hardware architectures.

Our simulation framework has recently been enhanced with the suggested static analysis techniques. We will shortly integrate the automatic identification of recurring system states. In the medium term, the necessary coverage of the model providing a sufficient degree of confidence in simulation results should be automatically deduced from formally expressed requirements. For this purpose, an appropriate enhancement of SysML Parametric Diagrams has already been proposed [10].

References

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

[2] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: principles, techniques, and tools*. Pearson Education, Boston, MA, USA, 2007.

[3] J. Bengtsson and W. Yi. Timed automata: Semantics, algorithms and tools. In *Lecture Notes on Concurrency and Petri Nets*. W. Reisig and G. Rozenberg (eds.), LNCS 3098, Springer-Verlag, 2004.

[4] A. Bobrek, J.J. Pieper, J.E. Nelson, J.M. Paul, and D.E. Thomas. Modeling shared resource contention using a hybrid simulation/analytical approach. *Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings*, 2:1144–1149 Vol.2, Feb. 2004.

[5] Hubert Garavel, Frédéric Lang, Radu Mateescu, and Wendelin Serwe. CADP 2006: A Toolbox for the Construction and Analysis of Distributed Processes. In *Computer Aided Verification (CAV'2007)*, volume 4590, pages 158–163, Berlin Germany, 2007.

[6] Paula Herber, Florian Friedemann, and Sabine Glesner. Combining model checking and testing in a continuous hw/sw co-verification process. In Catherine Dubois, editor, *3rd International Conference on Tests and Proofs (TAP'09)*, volume LNCS, pages 121–136. Springer, 2009.

[7] G.J. Holzmann. The model checker spin. *Software Engineering, IEEE Transactions on*, 23(5):279–295, May 1997.

[8] D. Knorreck and L. Apvrille. Formal system-level design space exploration. In *Proceedings of the 10th Annual International Conference on New Technologies of Distributed Systems (NOTERE'2010)*, Tozeur, Tunisia, May 2010.

[9] 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.

[10] Daniel Knorreck, Ludovic Apvrille, and Pierre de Saqui-Sannes. Tepe: a sysml language for time-constrained property modeling and formal verification. In *Proceedings of the third IEEE International workshop UML and Formal Methods - ULM&FM'2010*. IEEE, November 2010.

[11] S. Kunzli, F. Poletti, L. Benini, and L. Thiele. Combining simulation and formal methods for system-level performance analysis. *Design, Automation and Test in Europe, 2006. DATE '06. Proceedings*, 1:1–6, March 2006.

[12] 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.

[13] Gabor Madl, Nikil Dutt, and Sherif Abdelwahed. Performance estimation of distributed real-time embedded systems by discrete event simulations. In *EMSOFT '07: Proceedings of the 7th ACM & IEEE international conference on Embedded software*, pages 183–192, New York, NY, USA, 2007. ACM.

[14] Peter van Stralen and Andy D. Pimentel. Trace-based scenario database for high-level simulation of multimedia mp-socs. 2010.

[15] A. Viehl, T. Schonwald, O. Bringmann, and W. Rosenstiel. Formal performance analysis and simulation of UML/SysML models for esl design. *Design, Automation and Test in Europe, 2006. DATE '06. Proceedings*, 1:1–6, March 2006.