SAME 2010 Forum

Session 3

Partitioning of in-vehicle Systems-on-Chip: a Methodology based on DIPLODOCUS

Daniel Knorreck, Ludovic Apvrille, Renaud Pacalet,
{daniel.knorreck, ludovic.apvrille, renaud.pacalet}@telecom-paristech.fr
Institut Telecom, Telecom ParisTech, CNRS LTCI
Route des Cretes BP 193, F-06904 Sophia Antipolis, France

Abstract

This article highlights a methodology yielding a HW/SW partitioning of in-vehicle Systems-on-Chip. The methodology is based on a MARTE compliant UML profile called DIPLODOCUS. The starting points are separate UML application and architecture models derived from an informal, textual specification. The information available at this early design stage is efficiently leveraged to automatically generate executable models. Simulation reveals first performance results without the designer having to write any single line of code. The performance figures guide the designer towards the partitioning which best fits to given non-functional requirements. Abstractions inherent to application and architecture models allow for fast simulations. The methodology has been successfully applied to in-vehicle Systems-on-Chip.

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.

The complexity of embedded systems and Systems-on-Chip has been increasing rapidly not only in the field of consumer electronics. Also embedded in-vehicle systems now comprise up to 70 ECUS (Electronic Control Units) executing security related functionality as well as sophisticated control and convenience features [5]. On the one hand, the virtues of integration should be leveraged to improve security, safety and the user experience. On the other hand, the gap increases between integration and designer efficiency due to inadequate tools and methodologies. It becomes more and more unlikely that an optimal design represents an intuitive solution.

Thus, given a particular functionality and associated requirements, the design space is considered as representing all functionally equivalent implementation alternatives. 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 Design Space Exploration (DSE) at system scope 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 DSE. In this context, we have previously introduced a UML-based environment named DIPLODOCUS. The strength of our approach relies on formal verification capabilities [2] [3] and fast simulation techniques [1]. In this article, the applicability of the methodology is demonstrated with the aid of a case study in the automotive domain. An insight into the involved models, the design stages and finally the obtained simulation results is provided.

The paper is structured as follows: Section 2 surveys the DIPLODOCUS methodology which was followed to carry out the partitioning of an in-vehicle System-on-Chip. Thereafter, Section 3 puts the methodology into practice by showing how requirements translate into a UML model with formal semantics and how it is leveraged to obtain insightful simulation results. Section 4 is positions the DIPLODOCUS approach in the landscape of related work in the field of System Level Design Space

Exploration. Section 5 concludes this paper and provides perspectives on future work.

2 The DIPLODOCUS methodology

DIPLODOCUS design approach (cf. Figure 1) is based on the following fundamental principles:

- Use of a **high level language** (UML)
- Clear **separation between application and architecture,** DIPLODOCUS thus enforces the so called Y-Chart approach [6]
- **Data abstraction**: only the amount of data exchanged between functional entities is modeled
- **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.
- Use of **fast simulation** and **formal static analysis** techniques, both at application and mapping level

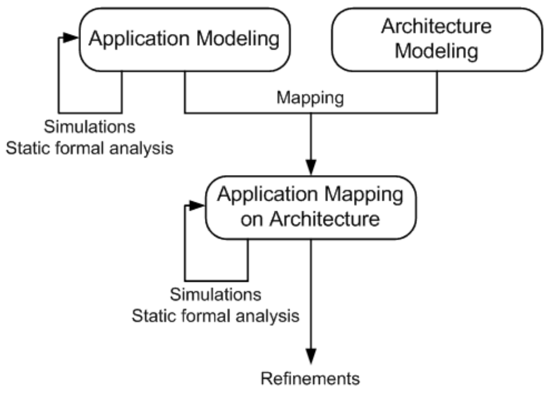

Figure 1: DIPLODOCUS Methodology

The designer is supposed to model in an orthogonal fashion the application and the architecture of the targeted system.

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**. The DIPLODOCUS methodology provides synchronization and data transfer primitives, where data flow and control flow are not strictly separated to simplify matters. Events are a means of inter-task synchronization and come with finite and infinite FIFO semantics. The reception of an event is always blocking, its notification may or may not be blocking. Channels are useful when it comes to modeling extensive data transfers. Several semantics are provided: Blocking Read/Blocking Write, Blocking Read/Non Blocking Write, Non Blocking Read/Non Blocking Write.

Each **task behavior** is described with a UML activity diagram. The latter is composed of 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 physical delay operators.

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. execution nodes like CPUs and hardware accelerators, communication nodes like buses and bridges as well as memory nodes).

A **mapping** process defines how application tasks can be bound to execution nodes and also how abstract communications between tasks are assigned to communication and storage nodes.

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 at a purely functional level. The 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 tool comprises features for the creation of models, the automatic transformation to representations suited for simulation and formal verification as well as the generation of simulation traces (cf. Figure 2).

3 From specification to simulation

3.1 Context and considered use case

In the remainder of this article, the model of an application running in an automotive environment is considered. It has been taken from a case study which was carried out in the scope of the European EVITA [5] project. The objective of this projects is to analyze, to design, to verify, and to prototype a modular, (cost-)efficient security solution for automotive on-board embedded systems. A major concern is to protect sensitive data within such systems against compromise and, in doing so, to enable secure communication inside cars and between cars and infrastructure.

The use case covered in this paper deals with the flashing procedure of the internal software of an ECU (Electronic Control Unit). When a car owner takes his car to a service station, the motor mechanic may start a diagnosis session which also allows to update the firmware of ECUs. The specification comprises mainly two elements: first, an informal textual description of the use case is provided. The latter identifies the reaction of the system to external stimuli and moreover explains the context in which the use case may be encountered. Second, the text is accompanied by a semi-formal table whose columns are standardized for all documented use cases. Each row stands for a communication or computation transaction between or within ECUs respectively. A transaction is characterized by an initiating actor, a receiver in case of communications, a type (communication/algorithm), a short description, the length of a message and a remark on the control flow (repetitions, parallelism) to overcome the strictly sequential nature of the table.

3.2 Application model

Given a specification as described in section 3.1, the developer first has to **draw clear boundaries between** pieces of information concerning the **application** and the **architecture**. The notion of "actor" as referred to in the specification combines a behavior and an underlying execution hardware. The first design step consequently consists in **identifying the set of behaviors an actor may exhibit.** Figure 2 depicts a subset of the behaviors - namely tasks - which have been identified for the considered use case. The *OutsideWorld* task accounts for the stimuli generated externally to the system. *DiagReqManagement* responds to external stimuli and dispatches requests to dedicated tasks not shown in the figure. *CipherFunctions1* is in charge of ciphering and deciphering messages sent by DiagReqManagement. At the end of this design stage, our UML class diagram has been populated with tasks deduced from the specification.

After the functional analysis, **emphasis is now placed on the the communication behavior** of the previously identified tasks. The designer is faced with the question which of the available communication primitives best fits to the specification and allows the simulator to determine the expected metrics. As a rule of thumb, events transfer control information and no data and the opposite case is handled by non blocking write-non blocking read channels. The other channel types are in between these extreme cases. In Figure 2 for example, the communication between Task *OutsideWorld* and Task *DiagManagement* relies on channels as the amount of exchanged data is significant. Events were used to synchronize *DiagReqManagement* and *CipherFunctions1* because merely control information is exchanged. The events signal for instance the presence of data. Hence, after having accomplished this stage, the class diagram has been enriched with communication links between tasks.

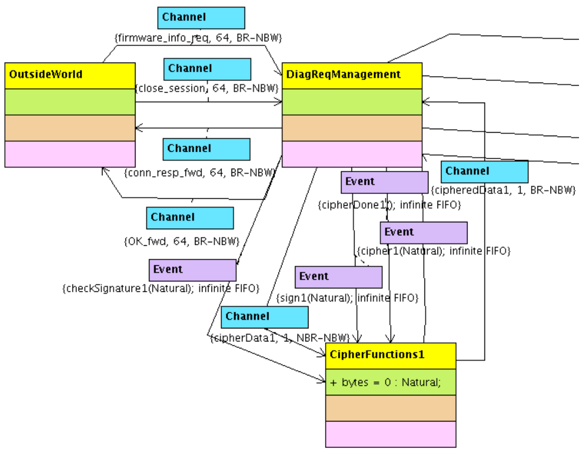
Channel
{firmware_info_req, 64, BR-NBW}
OutsideWorld
Channel
{close_session, 64, BR-NBW}
DiagReqManagement
Channel
{conn_resp_fwd, 64, BR-NBW}
Event
{cipherDone1(); infinite FIFO}
Channel
{cipheredData1, 1, BR-NBW}
Channel
{OK_fwd, 64, BR-NBW}
Event
{cipher1(Natural); infinite FIFO}
Event
{checkSignature1(Natural); infinite FIFO}
Event
{sign1(Natural); infinite FIFO}
Channel
{cipherData1, 1, NBR-NBW}
CipherFunctions1
+ bytes = 0 : Natural;

Figure 2: Excerpt of the Class Diagram

The next steps towards an executable model is to **detail the behavior of each task** by means of a UML activity diagram. The global system behavior defined in the specification must therefore be broken down into suitable local behavior descriptions for each task. Figure 3 shows the Activity Diagram of the *CipherFunctions1* Task. After having received one of the three events *cipher2*, *sign2*, or *checkSignature2*, the respective functionality is carried out. For that purpose, we make use of a Read command in order to fetch the data to be processed. Thereafter, the computational complexity is calculated as a function of the amount of data to process and it is modeled in terms of an abstract cost operator. Finally, the produced data is written to a channel leading back to the *DiagReqManagement* Task. Complexity estimations may be obtained in one of the following ways: by (1) deducing the number of operations from an algorithmic description or another suitable source, (2) by extracting data from measurements or traces of similar existing systems, or (3) by inferring the amount of operations from low level models, for instance source code. For our model, we relied on given execution frequencies which came with the specification and on qualified guesses of designers experienced in that field.

At this early design stage, algorithms often lack a detailed definition. For that reason, cost operators are at first used at a very coarse grained level (cf. Figure 3) and stand for high level activities like "Check message for correctness", "Driving power reduction strategy", etc. As the specification evolves, the model may be refined so as to reflect different executions of this algorithm. This is made possible by control flow operators like loops and conditions. It should be reemphasized that a data dependent behavior of the application has to be expressed in terms of random operators. That is, a stochastic model of data hazards has to be embedded into the application model. However, this effort is not particular to our methodology, neither it is to a high level of abstraction. Whenever a system is loaded with data dependent tasks, the designer is obliged to come up with a statistical model of the data to be processed. Only with that model, it is possible to avoid over-dimensioning the system for the worst case. The statistical model gives the designer the confidence that an architectural trade-off delivers an acceptable performance with a defined likelihood.

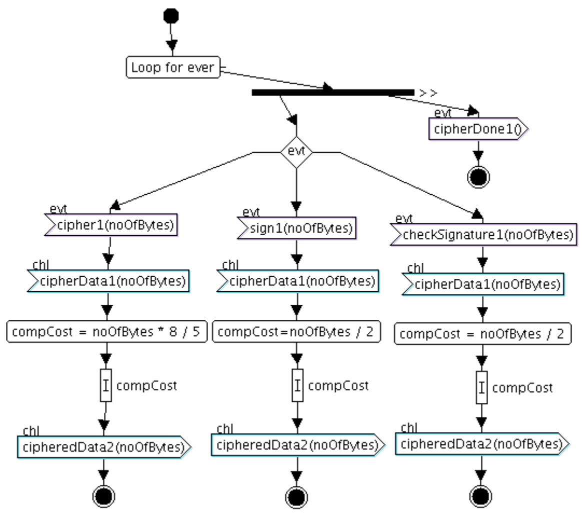

Figure 3: Activity Diagram of Task CipherFunctions1

3.3 Architecture model

An excerpt of the the architecture model of the ECU Flashing use case is depicted din Figure 4. An ECU called CU (Communication Unit) manages the communication with the outside word. It is connected to the main CAN bus of the vehicle and contains a SoC consisting of a CPU, a local memory, a Hardware Security Module (HSM) and the internal bus CU_SOC_Bus. The HSM accommodates the task *CipherFunctions1* and the task D*iagReqManagement* is mapped onto the CPU. Three of the channels depicted in Figure 2 are associated with the local memory of the SoC. The ECU to be flashed is omitted in Figure 4 .

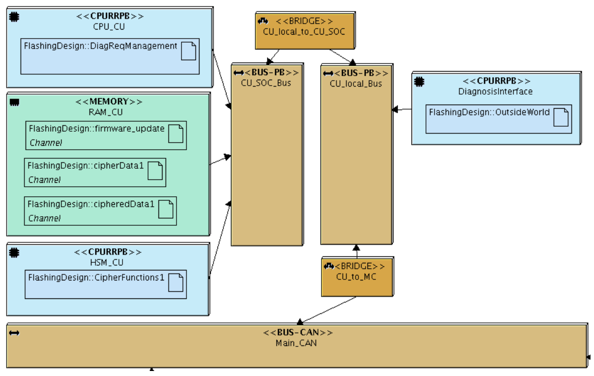

Figure 4: Excerpt of the Architecture Diagram

3.4 Experimental results

Key performance figures of the abstract model are obtained by simulation just at the push of a button. The UML model is automatically translated into executable C++ code without intervention of the user. So far, the utilization of HW components, the average contention delay for tasks (when sharing CPUs) and CPUs (when sharing buses) as well as the execution time of tasks can be determined. More sophisticated metrics like end-to-end delays, etc. could be easily added thanks to dynamic listeners which can be plugged into the simulation environment. In addition to that, formalized properties may be checked on the fly at simulation time.
In the context of the EVITA project, the topology of the CAN network and the positioning of ECUs is constrained by the specification of the car architecture and the use case respectively. The objective of the partitioning stage is to compare the deployment of security related functionality in software and in hardware. Therefore, two different architectures were assessed for the ECU Flashing use case. The first one is partially illustrated in Figure 4, the second one does not comprise a separate HSM module so that the ciphering is accomplished by the internal CPU of ECUs. More concretely, this means that *CipherFunctions1* is mapped onto CPU_CU as well.
For the first configuration, the simulation revealed utilization factors of 63% for CPU_CU, 68% for CU_SOC_Bus and a factor

close to 0% for HSM_CU as ciphering algorithms are applied to small messages. Being utilized at 97%, the internal bus of the ECU to be flashed was identified as a bottleneck. This load is caused by the transfer of the software update which transits the bus twice (in decrypted and encrypted state).
For the second configuration, the load of CPU_CU does not increase as the workload imposed by ciphering algorithms is neglectable. However, the utilization of the internal CPU of the ECU to be flashed increases by more than three times, which might not be tolerable.

4 Related work

Some of the current state of the art UML modeling tools ([7], [8], [9], [10] 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 [11] 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 [16], etc) and computations. Some academic approaches enable the design of distinct models for architecture and application ([12], [13], [14], [15], [17]). 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, [18] 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. The initial model has to be refined before being simulated as the simulation is carried out at SystemC TLM level.

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. This property enables the developer to easily vary the coverage of the simulation. Explicit indeterminism also alleviates the state explosion by abstracting decision making processes. This in turn makes the model amenable to formal verification.

5 Conclusions

This paper presented the DIPLODOCUS methodology which defines a modeling framework for Systems-on-Chip at a high level of abstraction. All involved design stages are supported by the modeling tool called TTool, which also enables the simulation and formal verification of UML models at the push of a button. The methodology was explained and illustrated with a case study of an in vehicle embedded system. As it has been shown, the methodology finally yields insightful key performance figures characterizing the system intended for design. Based on the obtained performance, the designer can consider revising the mapping, the architecture, or application. Important design choices are thus made early in the design flow before much time and money have been spent on the development of RTL or transaction level models. Last but not least, the methodology allows developers to trade off performance against cost.

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 extent.

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] http://www.evita-project.org/

[6] 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.

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

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

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

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

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

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

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

[17] 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.

[18] 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.

About the Authors

Daniel Knorreck obtained his diploma degree in Electrical Engineering from the University of Stuttgart and Telecom ParisTech in 2008. He is currently doing a PhD at the "Systems-on-Chip" laboratory of Telecom ParisTech at Sophia-Antipolis. His research interests focus on the modeling and simulation of embedded systems and control engineering.

Ludovic Apvrille obtained his M.Sc. in Computer Science from ENSEIRB and ISAE/ENSICA in 1997 and 1998, respectively. Then, he completed a Ph.D. at LAAS-CNRS, Toulouse, France, in the research group Software and Tools for Communication, in collaboration with ISAE and Thales Alenia Space. After a postdoctoral term at Concordia University (Canada), Electrical & Computer Engineering department, he joined LabSoc in 2003 as an assistant professor at Telecom ParisTech, in the Communication and Electronics department. His research interests focus on tools and methods for the modeling, simulation and formal verification of embedded systems and System-on-Chip.

Renaud Pacalet received his M.S. from Telecom ParisTech in 1988. He currently leads the "Systems on Chip'" laboratory of Telecom ParisTech at Sophia-Antipolis. His research fields are the specification, modeling, design and verification of integrated systems and their security (side-channels, memory buses privacy and integrity, formal verification of embedded software).