DiplodocusDF: Analyzing Hardware/Software Interactions with a Dinosaur

Andrea ENRICI, Ludovic APVRILLE and Renaud PACALET
Institut Mines-Telecom
Telecom ParisTech, CNRS/LTCI
Campus SophiaTech, 450 Route des Chappes, 06410 Biot, FRANCE
Email: {andrea.enrici, ludovic.apvrille, renaud.pacalet}@telecom-paristech.fr

***Abstract*—Development of new products commonly relies on existing designs where hardware and software are modeled separately and their interactions investigated late at verification time. Composing hardware with software yields more than the sum of the two parts and solid design dedicates a great deal of time and money to understand their interactions. This paper presents a novel modeling methodology based on DiplodocusDF, a UML Model-Driven Engineering approach for the design of heterogeneous data processing systems. Our methodology separately models conflicting aspects such as communications versus computations, dataflow versus controlflow and allows for their impact on the overall system's performance to be separately analyzed at various abstraction levels. Our methodology is applied to a case study which shows how it is possible to study interactions parallel to design, thus greatly reducing time-to-market of new products and comprehension of existing designs.**

I. Introduction

The manner in which hardware and software interact is a key issue in the quest for increased performance that drives the development of new embedded systems. On one hand, technological advances in hardware make possible the development of devices with increasing computing power and complex architectures. Synergistic developments in mathematics and software engineering provide new methods for implementing evolving algorithms (e.g., 3G, 4G wireless technologies). On the other hand, interactions between hardware architectures and software applications are poorly understood, in spite of their fundamental role in determining the success of new products.

One way to investigate interactions is to leverage debugging techniques such as hand-written test programs, inspection with oscilloscopes and logic analyzers, simulators, emulators. Nevertheless, those solutions can only be applied to existing designs and the know-how required to deploy them is a major obstacle for new members of a design team to rapidly and efficiently apply them. A different approach is, instead, to apply modeling techniques either to existing designs via reverse engineering or to new products via design space exploration of models from existing designs. Modeling strategies are usually classified according to their abstraction level. At the lowest abstraction level, Register-Transfer Level (RTL), hardware description languages, e.g., Verilog, VHDL, are directly used to model the architecture. At the next abstraction level, Transaction Level (TL), languages such as SystemC and SpecC are deployed to create libraries of building blocks that are instantiated and interconnected to generate an architecture model. Approaches at RTL and TL provide accurate simulation results thanks to detail of architecture models but often result to be application-specific and thus limited in terms of portability. Moreover, a solid understanding of how models are built and work complicates the analysis of hw/sw interactions for users who are not system experts. The above limitations, thus call for raising the level of abstraction to Electronic System Level (ESL) where "appropriate abstractions are used to increase comprehension about the system and to enhance the probability of a successful implementation of functionality in a cost-effective manner using generic architecture and abstract application models" [1]. Matlab/Simulink is an example of a widely used tool for such system level design and analysis. Working at ESL normally simplifies models, augments their portability and allows for the deployment of verification techniques such as simulation and formal verification. Typical methodologies, however, commit the analysis of hw/sw interactions to verification phase only. This results into increased costs and analysis time: the information about interactions that was contained in models at design time is lost and must be re-extracted by interpreting verification results.

This paper enriches DiplodocusDF [2], a UML Model-Driven Engineering methodology for the design of heterogeneous data processing systems. We propose a novel modeling approach to analyze hw/sw interactions, early at design time where conflicting aspects such as computations vs communications, dataflows vs controlflows, are modeled by separate features at different abstraction levels. We apply our enhanced DiplodocusDF to a case study taken from the domain of signal processing, showing the benefits of our solutions in terms of reduced design time, enhanced design quality and understanding of existing designs.

The next section depicts the context of DiplodocusDF. Section III introduces the case study: an heterogeneous multiprocessor architecture with shared memory running an application for signal processing. Section IV depicts the principles of our approach to analyze hardware/software interactions. Such principles are applied in Section V to the case study. Related works in the field of hardware/software co-design are illustrated in Section VI. Conclusions along with the state and directions of our works are given in Section VII.

II. The Context: DiplodocusDF

DiplodocusDF is the result of a continuous workflow that has its roots in DIPLODOCUS [3]. The latter is a UML Model Driven Engineering methodology for hardware/software partitioning of Systems on Chip at high abstraction level, implemented by the free software TTool [4]. DIPLODOCUS targets control-oriented systems and it is based on the Y-Chart approach [5]: the application and the architecture models are independently developed and then merged at mapping step where each element of the application is assigned to an element of the architecture. The core strength of DIPLODOCUS is the automatic transformation of models for simulation and formal verification [6]. The simulation environment allows for an interactive exploration of the application mapped onto a particular architecture via a revisited version of Discrete Event Model of Computation (MoC). Formal verification can be performed on the application model before and after mapping thanks to a translation of DIPLODOCUS's concepts into the formal semantics of LOTOS and the timed automata underlying UPPAAL [7]. DIPLODOCUS' formal verification allows model-checking of system properties such as safety, schedulability and performance.
DiplodocusDF stems from DIPLODOCUS, Figure 1, and extends the latter to target the design of heterogeneous dataflow applications for real-time processing systems. DiplodocusDF bridges the gap between these systems and DIPLODOCUS by supplying for DIPLODOCUS' abstract approach and its lack of expressiveness to generate executable code for data-dominated systems. The contributions of DiplodocusDF are new modeling capabilities (e.g., dataflow semantics) for application and architecture descriptions and a code generation environment to synthesize executable C-code from an application model. As Figure 1 shows, formal verification and simulation in DiplodocusDF can be equally performed as in DIPLODOCUS.

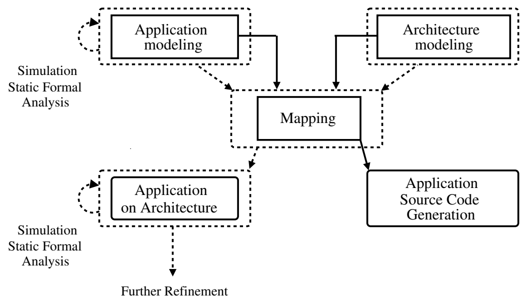

Fig. 1. The DIPLODOCUS (dotted lines) and DiplodocusDF (solid lines) methodologies.

III. A Case Study

A domain where adding reconfigurability has blurred the separation between hardware and software and therefore has introduced new interactions is Software Defined Radio (SDR) [8]. SDRs are complex telecommunication systems where some or all of the physical layer functions are implemented in software. This has made the signal processing of radio equipments software-reconfigurable, whereas, previously, all functionalities where implemented in hardware. By presenting an SDR architecture and application, this section illustrates how the increased flexibility of SDR systems comes at the price of new and complex hw/sw interactions.

A. The Architecture: Embb

The authors of [9] propose Embb, Figure 2, a generic baseband architecture dedicated to SDR applications. Embb is composed of a *Processing System*, left-hand side of Figure 2, interconnected to a *Control System*, right-hand side of Figure 2. The *Processing System* operates on the raw samples that are transmitted or received via the *Radio Frequency Interface* (RFI) by executing signal processing operations, e.g., Fast Fourier Transform, on a set of interconnected Digital Signal Processor (DSP) units. The latter are equipped with a hardware accelerator (Processing SubSystem, PSS), a DMA, a microcontroller (μC) and an internal memory, mapped on the global address map of the main processor within the *Control System*. According to the type of each DSP, the internal memory of the latter may be further divided in independent banks, accessible by the system interconnect, DMAs and μCs. The system interconnect permits exchanges of control and data items; it is composed of a *Crossbar*, a *Bridge* and a *Main Bus*. The *Control System*, instead, is composed of a main memory and a main processor that executes the control part of an SDR application: it manages data transfers, DSPs, their μCs, the interface with the air (RFI) and the External Environment Interface. Control information within Embb are exchanged either through a dedicated network of interrupt lines and controllers interconnecting PSSs, DMAs and μCs with the main CPU, or by means of control instructions passing through the system interconnect. In the latter case, the Bridge and its DMA are subject to contention between controlflows and dataflows.

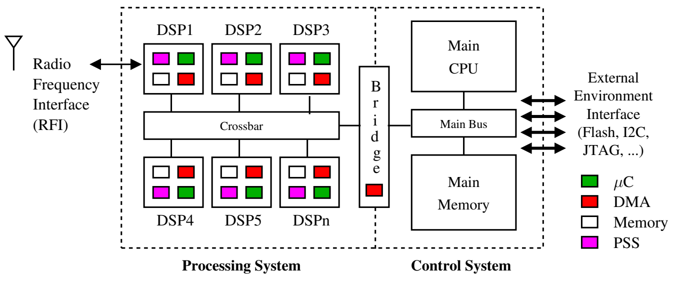

Fig. 2. The architecture of an instance of Embb.

B. The Application: High Order Cumulants

The application we chose for this paper is a classification algorithm, High Order Cumulants (HOC) as implemented in [10], that is used in cognitive radio by a transmitter to sense the spectrum and detect if another user is currently transmitting in the same frequency range. The algorithm operates on

segments of the input stream that are independently processed to extract a score. The occupancy of a specific frequency range is determined by accumulating scores over a given classification period and by comparing the accumulated scores with a pre-computed threshold. The application graph for the implementation described in [10] is illustrated in Figure 4.
Executing HOC on Embb is not trivial because of the pooling of memory and DSPs, which implies the transfer of dense flows of data and control information within and across the Processing and Control Systems. These flows interact via dependencies (i.e., data transfers and processings must be configured by control instructions) and contention (i.e., data transfers and control instructions compete for the Bridge). Currently, a truly efficient analysis of these interactions can only be efficiently performed by a platform expert.

IV. A Methodology for the Analysis of Hardware/Software Interactions

By looking at modern computing devices from an ESL perspective, it is easy to see that hardware and software depend upon each other structure in order to function. An architecture is designed to meet requirements of a specific application domain (e.g., signal processing). Similarly, an application is written with the capabilities of a given architecture in mind (e.g., multiprocessor). Structure can be broadly defined as a composition of parts (e.g., DSPs within an architecture, threads within an application) that carries information about how those parts relate and interact, e.g., how DSPs are interconnected, how threads exchange information. The modeling phase of our enhanced DiplodocusDF is based on the above principles, Figure 3. By modeling and mapping the structure of a pair application-architecture, DiplodocusDF allows to analyze hw/sw interactions such as: contentions between transfers of data and control items as well as dependencies between data transfer and processing operations. Models that are created with DiplodocusDF are then used for improving design quality (i.e., enhanced design space exploration), testing and debugging (i.e., code generation and rapid prototyping on the real hardware) as well as verification (i.e., model-checking and simulation).

DiplodocusDF differentiates between an application and an architecture according to the definition that the former imposes a workload on the latter. From the specifications of an existing design, the structure of the hardware architecture and of the software application are extracted, upper part of Figure 3. Models for the hardware and the software are composed by instantiating and interconnecting building blocks taken from an application and an architecture libraries (*Composition*, Figure 3). In order to make the analysis of interactions effective, we create the application and architecture models by describing aspects that may potentially interact, with separate modeling features. On the application side, computations and control operations are described in a graph as different vertexes interconnected by separate control and data edges. Exchanges of data and control items, represented by the graph edges, are described with **Communication Patterns (CP)** [11],

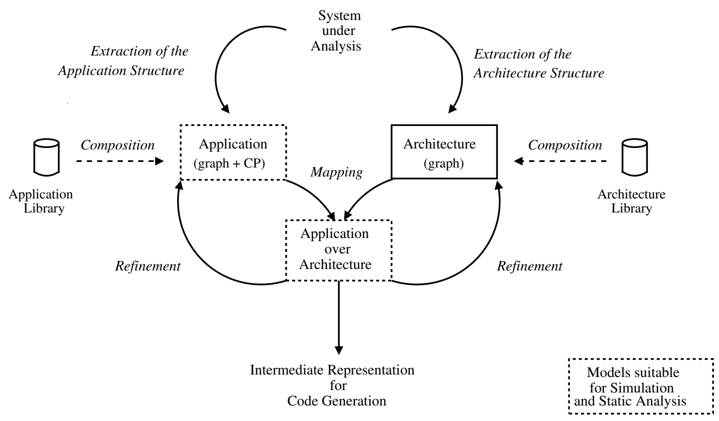

Fig. 3. Analyzing hardware/software interactions with DiplodocusDF.

independently with respect to the computations and the control operations of the graph. On the architecture side, the latter is described as a graph whose building blocks are separately classified according to their functionality: computation (e.g., DSPs), storage (e.g., memories), control (e.g., CPUs) and communication (e.g., bus). Such elements are interconnected by separate data and control paths via independent edges.
Once the application model (graph and Communication Patterns) is available, it is mapped (*Mapping*, Figure 3) onto an architecture graph that is built throughout four hierarchical levels of abstraction. These levels are defined according to the functionalities that an architecture offers to an application: computations (*level L0*), storage (*level L1*), control (*level L2*) and communications (*level L3*). At each level the architecture model is expanded and refined (*Refinement*, Figure 3) by exposing the hardware parallelism: if a given functionality is implemented by an independent component, then a building block for such a component is instantiated and interconnected to the rest of the graph. For instance at storage level L1, for each memory that can be independently accessed, a memory unit is instantiated and interconnected. Such a hierarchy, based on the functionalities of components, allows our approach to be ported to architectures from different domains. Moreover, the impact of each class of components on the design space can be separately evaluated via formal verification and simulation at each level of abstraction, provided a default mapping is given for the application when not all the architecture parallelism has been exposed.
In the next section, we apply the above principles to the case study previously introduced in Section II.

V. The Methodology Applied to the Case Study

The structure of the High Order Cumulants algorithm is illustrated in Figure 4. Here, the *Source* vertex splits the input data stream into segments that are broadcast via the *Fork* node to processing operations *CWM1,2* (Component-Wise Modulus) and *CWS* (Component-Wise Square). These operations process the input segments to compute the classification score that is accumulated over a classification period by the *ACC* vertex which dispatches its result to the *Sink*. The latter compares the accumulated scores with a pre-computed threshold and also receives a copy of the input segments. Such a copy is not used to compute the classification score but is rather instantiated to illustrate how Communication Patterns can be deployed to capture complex transfer schemes. In Figure 4, data dependencies are represented by solid edges, while control dependencies are pictured as dotted edges. The latter are labeled with the control primitives being exchanged: *start()* and *stop()* are sent from *CWM1,2*, *CWS* and *Sink* to control the *Source* input stream. Control primitive *set_period()* is sent by *Sink* to *ACC* to configure the length of the classification period.

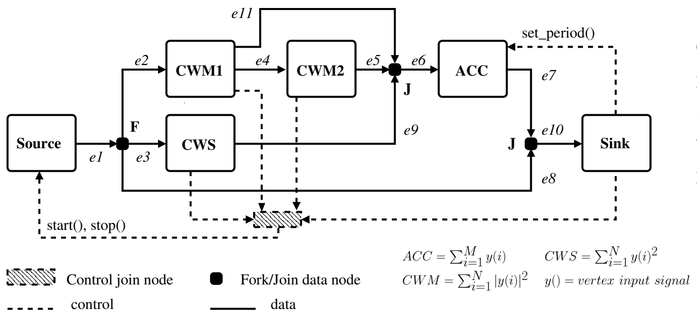

Fig. 4. The application graph for HOC algorithm.

The application workload in terms of communications is described by CPs associated to graph edges. An interesting case is represented by edges *e1,2,3,8,10* in Figure 4, where a dataflow needs to be transferred from a memory location accessible to *Source*, to a location reachable by operations *CWS, CWM1, Sink*. This behavior is captured by the abstract CP of Figure 5, independently with respect to the architecture or the operations it serves. The architecture of Embb is initially

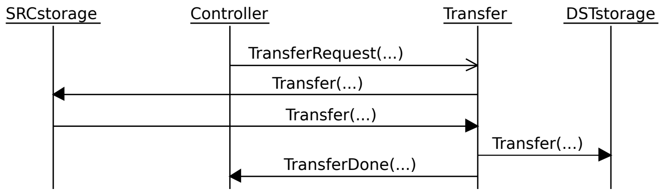

Fig. 5. The Communication Pattern for edges *e1,2,3,8,10*.

modeled at high abstraction level as illustrated in the left-hand side of Figure 6: the memory space of the platform is modeled by a memory unit (*Storage System*) interconnected via a bus (*System Interconnect*) to a DSP for the platform's processing part (*Processing System*) and to a CPU for control operations (*Control System*).

At level L1, right-hand side of Figure 6, the architecture parallelism in terms of computations is exposed and two DSP units replace the *Processing System*. The *FEP* (Front End Processor) for vector and scalar processing and the *ADAIFEM* (Analog-to-Digital-InterFace) for receiving input streams from the air, are instantiated and interconnected. Consequently, we map processing operations: *CWM1,2*, *CWS* and *ACC* to *FEP*, the *Source* to *ADAIFEM* and the *Sink* to the *Control System*.

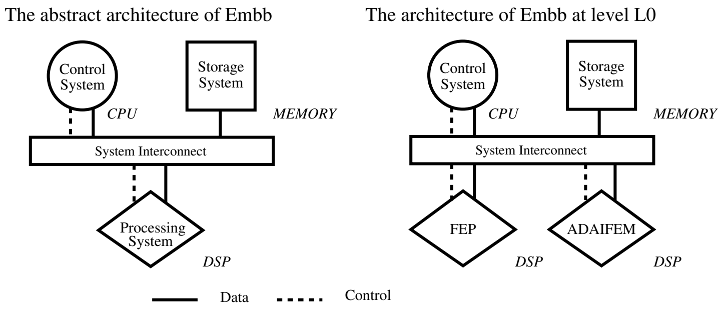

Fig. 6. The Embb architecture at the most abstract levels.

Clearly, *CWM1* and *CWS* will have to be executed sequentially in spite of the parallelism present in the application graph: already at the very first step of the approach, an important interaction that constraints the scheduling of all HOC operations is exposed.

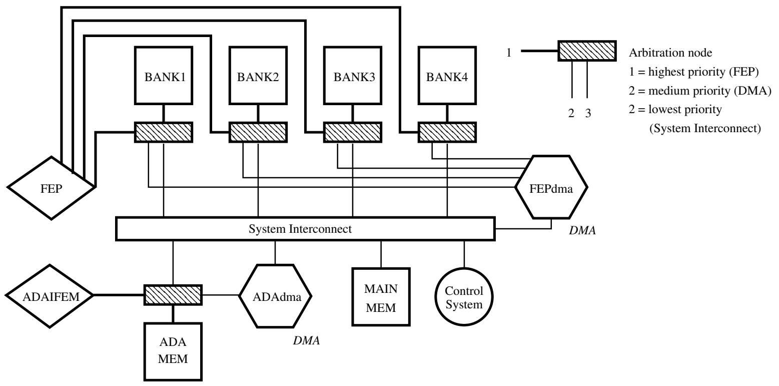

Fig. 7. The architecture of Embb after the storage refinement of level L1. Only the dataflow links are shown.

At level L1 in Figure 7, the *Storage System* of Figure 6 is split: the FEP's internal memory, made up of four banks, is modeled by four memory units. Embb's main memory (*MAIN MEM*) and ADAIFEM's memory (*ADA MEM*) are instantiated as one memory unit each. The parallelism in terms of access to memories is also modeled in Figure 7 by *Arbitration nodes* and the instantiation of a DMA for each DSP unit (*ADAdma*, *FEPdma*). From the point of view of communications, the CP of Figure 5 must now be re-described, to account for the mapping at level L0 and for the new information about storage units. *CWS, CWM1, Sink* have been mapped to units that can access different memories, so the initial transfer of Figure 5 is refined with a first *Transfer* to serve *CWM1, CWS* and a *Copy* to serve *Sink*, Figure 8. Without Communication Patterns, as it was the case in DiplodocusDF before our contribution, we would have re-designed from scratch the application graph in order to accommodate for the copy operation. Moreover, without a hierarchical approach, the need for such a re-design would have emerged only after design completion. Thus, at this level we get an understanding of the complexity and role played by data transfers in Embb, transfers which are as vital as computations. What we thought to be able to describe with a single scheme is in fact much more complicated that will be totally unveiled at level L3.

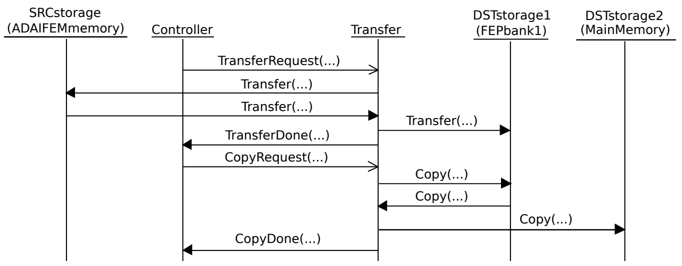

Fig. 8. The Communication Pattern of Figure 5 at L1.

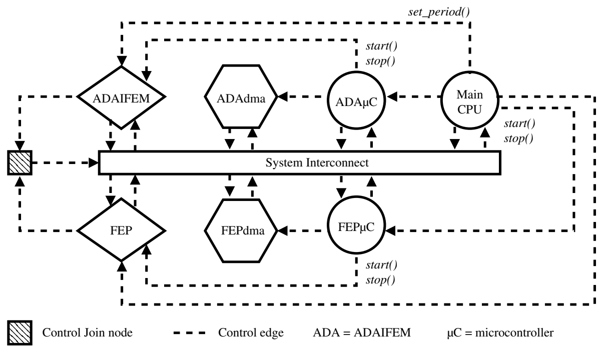

Fig. 9. The architecture of Embb after the control refinement at level L2. Only the controlflow links are shown.

At level L2, the study of interactions continues by exposing the control parallelism of Embb: *Control Block* of Figure 6 is split in three independent CPU units. The latter model Embb's *Main CPU* and the microcontrollers of FEP and ADAIFEM (*FEPμC*, *ADAμC*). The control network of the platform is also exposed by interconnecting the units with dedicated control nodes and edges. As illustrated in Figure 9, *Main CPU* is able to either exchange control information with the μCs via dedicated control edges (interrupt lines) or via the *System Interconnect* (control instructions). *FEP* and the *ADAIFEM* can be driven either by the *Main CPU* or by the μCs. The *Control join node* models the interrupt controller that gathers control information from the DSPs to *Main CPU* via the *System interconnect*. From the mapping point of view, the CP's *Controller* of Figures 5, 8 is mapped to *mainCPU* and control primitives *start(), stop(), set_period()* are mapped to control edges as depicted in Figure 9. We now understand that controlflows associated to the primitives *start()* and *stop()* may result into conflicts with dataflows from memories, due to the pooling of the *System Interconnect*.

At level L3, the architecture refinement is completed by exposing the parallelism in terms of communications: the *System Interconnect* bus is substituted by the *mainBus* and *Crossbar*, interconnected by the *Bridge*, similarly to what shown in Figure 2. This stage unmasks the *Bridge* as the system bottleneck in terms of communications due the contention between data and control flows, as mentioned in the introduction to Embb of Section III. The analysis of interactions is then completed by mapping the actors of our Communication Pattern: we choose to map the *Transfer* actor of Figures 5, 8 to a communication network made up of *BridgeDMA*, *Crossbar*, and *mainBus*. Such a mapping is displayed in Figure 10, for the *Copy* that serves *Sink*.

Although not all graphs and CPs are shown here due to lack of space, we understood the key system interactions in terms of computations, storage, control and communications by means of design only, without deploying simulation or formal verification. These techniques can be used in DiplodocusDF once design and analysis of interactions have been completed, to extract numerical results for comparison and design space exploration.

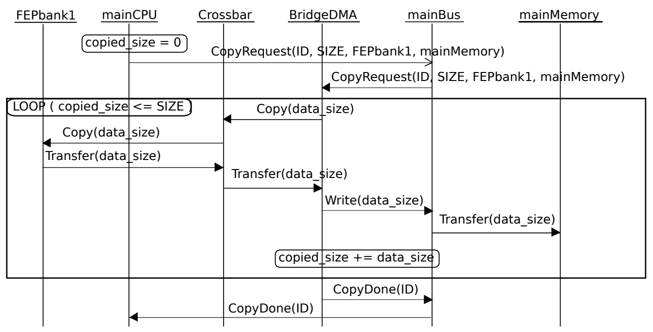

Fig. 10. The CP of Figure 8 for the *CopyRequest* at L3.

VI. Related Work

According to the novel classification scheme for ESL methodologies proposed in [12], true ESL synthesis tools are the well-known Daedalus [13], [14], Koski [15], Metropolis [16], Ptolemy [17], PeaCE/HoPES [18], SCE [19] and SystemCoDesigner[20]. With respect to these works, DiplodocusDF targets the same hardware platforms, i.e., heterogeneous bus-based multiprocessor platforms, but does not allow the complete generation of a system, as it does not include the synthesis of an hardware implementation. Since the purpose of the above works is to provide the highest possible degree of automation, the study of hw/sw interactions is delayed to verification phase as most of the design exploration steps are carried out within dedicated tools. The hands-on approach of DiplodocusDF, that consists in manually composing and mapping a pair application-architecture, directly impacts the user with the system complexities thus unveiling most of the hardware/software interactions earlier at design time.

Companies such as Xilinx and Altera provide design tool chains that attempt to generate efficient implementations starting from descriptions higher than the Register Transfer Level of abstraction. The required input specifications are still so detailed that understanding interactions is time consuming and can be conducted only by system experts. In contrast, DiplodocusDF raises the design to an even higher level of abstraction allowing the study of hw/sw interactions in a short amount of time and with little expertise in simulation and

formal verification.
With respect to DiplodocusDF component for DSE (i.e., the simulator), there exist a number of UML architectural exploration environments ([21], [22], [23], [24], [25]) that offer performance evaluation by supporting the mapping of an application onto an architecture. In comparison to the simulation of purely functional models offered by these tools, the simulator of DiplodocusDF also accounts for architecture constraints, e.g., arbitration of shared resources, speed and data throughput of components.

VII. Conclusions

In this paper we have presented a novel modeling approach to explore hardware/software interactions based on enhancing DiplodocusDF, a methodology for the design of heterogeneous data-processing systems. Our solution is based on separately taking into account conflicting aspects of the application (e.g., computations vs communications, dataflows vs controlflows) and on mapping them onto a progressively refined architecture whose parallelism is exposed in terms of four functional levels: computational, storage, control and communication. The core strength of our approach is to enable the analysis of hw/sw interactions early at design time while actually designing the system itself. Thanks to the framework from previous works on DIPLODOCUS and DiplodocusDF, our models can be verified with simulation or formal verification. We have successfully applied our methodology to an instance of a baseband architecture and a signal processing application, showing how we are able to capture and analyze interactions as the design unfolds. The major benefits of our approach are: (1) the analysis of hw/sw interactions early at design time, that reduces time-to-market of new products by shortening the post-design verification phase, (2) the improvement of design quality and of the understanding of existing designs. When modeling in our enhanced DiplodocusDF, the application and the architecture models are completely disjoint thanks to the separation of concerns introduced by Communication Patterns and by separate modeling facilities for data and control aspects. The design space exploration does not require to re-arrange or re-design the entire system models, but rather to tune single features (e.g., modify the mapping of a Communication Pattern). Improved design quality and understanding of existing designs are given by the novel modeling features themselves (e.g., Communication Patterns, control exchange primitives) coupled with the methodology. The former provide the expressive power to represent complex communication schemes, while the latter allows to unveil the interactions with computations (e.g., contention between control and data flows) thus introducing new points in the design space corresponding to novel interactions. Although the modeling described here is applied to a case study specific for telecommunication systems, our approach can be ported to other heterogeneous data-processing architectures and applications, e.g., image and video processing.
Main research directions for the future are re-visiting the code generation phase of DiplodocusDF so as to translate the information captured at design time into an implementation code capable to adapt to the environment needs with dynamic scheduling and memory management.

VIII. Acknowledgement

The research leading to these results has been conducted in the framework of the Celtic-Plus project SPECTRA (CP07-013) which has been partially funded by the French ”Direction générale de la compétitivité, de l’industrie et des services (DGCIS)”, the ”Ministry of Finance and Economy / Business Development Agency” of the Monaco Principality and the ”AVANZA2” framework of Spanish Industry, Tourism and Commerce Ministry (Ministerio de Industria, Turismo y Comercio).

References

[1] M. Grant et al., ”ESL Design and Verification: A Prescription for Electronic System Level Methodology”, Morgan Kaufmann USA, 2007.
[2] J. M. Gonzalez Pina, ”Application Modeling and Software Architectures for the Software Defined Radio”, Ph.D dissertation, Telecom ParisTech, May 2013.
[3] L. Apvrille et al., ”A UML-based Environment for System Design Space Exploration”, in *IEEE ICECS*, pp. 1272-1275, 2006.
[4] TTool, *http://ttool.telecom-paristech.fr/*.
[5] B. Kienhuis et al., ”A Methodology to Design Programmable Embedded Systems - The Y-Chart Approach”, in *IEEE SAMOS*, pp. 18-37, 2002.
[6] D. Knorreck, ”UML-based Design Space Exploration, Fast Simulation and Static Analysis”, Ph.D dissertation, Telecom ParisTech, October 2011.
[7] UPPAAL, *http://www.uppaal.org/*.
[8] J. Mitola III, ”Cognitive Radio: An Integrated Agent Architecture for Software Defined Radio”, Ph.D dissertation, Royal Institute of Technology (KTH), May 2000.
[9] N. -ul. -I. Muhammad et al., ”Flexible Baseband Architectures for Future Wireless Systems”, in *EUROMICRO DSD*, pp. 39-46, 2008.
[10] SACRA, Spectrum and Energy efficiency through multi-band Cognitive Radio, *D6.3, Report on the Implementation of selected algorithms*, http://www.ict-sacra.eu/public_deliverables/.
[11] A. Enrici et al., ”Communication Patterns: a Novel Modeling Approach for Software Defined Radio Systems”, to be published in *IARIA COCORA 2014*.
[12] A. Gerstlauer et al., ”Electronic System-Level Synthesis Methodologies”, in *IEEE Transanctions on Computer-Aided Designed of Integrated Circuits and Systems*, vol. 28, no. 10, October 2009.
[13] M. Thompson, et al., ”A Framework for rapid system-level exploration, synthesis and programming for multimedia MP-SoCs”, in *CODES+ISSS*, 2007, pp. 9-14.
[14] H. Nikolov, ”Daedalus: Toward composable multimedia MP-SoC design”, in *DAC*, June 2008, pp. 574-579.
[15] T. Kangas et al., ”UML-based multiprocessor SoC design framework”, in *ACM Trans. Embed. Comput. Syst.*, vol. 5, no. 2, pp. 281-320, May 2006.
[16] F. Balarin et al., ”Metropolis: An integrated electronic system design environment”, in *IEEE Computer*, vol. 36, no. 4, pp. 45-52, April 2003.
[17] The Ptolemy Project, *http://ptolemy.eecs.berkeley.edu*, 2014.
[18] S. Ha et al., ”PeaCE: A hardware-software codesign environment for multimedia embedded systems”, in *ACM Trans. Des. Autom. Electron. Syst.*, vol. 12, no. 3, pp. 1-25, August 2007.
[19] R. Dömer et al., ”System-on-chip environment: A SpecC-based framework for heterogeneous MPSoC design”, in *EURASIP J. Embed. Syst.*, vol. 2008, no. 3, pp. 1-13, January 2008.
[20] J. Keinert et al., ”SystemCoDesigner - An automatic ESL synthesis approach by design space exploration and behavioral synthesis for streaming applications”, in *ACM Trans. Des. Autom. Electron. Syst.*, vol. 14, no. 1, pp. 1-23, January 2009.
[21] IBM Tau, http://www-03.ibm.com/software/products/en/ratitau, 2014.
[22] TOPCASED, http://www.topcased.org/, 2014.
[23] Papyrus, www.papyrusuml.org, 2014.
[24] Artisan Studio, http://www.atego.com/fr/products/artisan-studio/, 2014.
[25] IBM Rhapsody, http://www-03.ibm.com/software/products/en/ratirhapfami, 2014