TTool/DiplodocusDF: a UML Environment for Hardware/Software Co-Design of Data-Dominated Systems-on-Chip

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*—The development of new Systems on Chip commonly relies on previous products for whom, due to factors such as system complexities, time and cost constraints, little design space exploration can be performed. Hardware and software are typically composed as if they were separate components, whereas their interactions yield more than the sum of the two parts. In the scope of the demonstration, we present our enhanced version of TTool/DiplodocusDF, a UML model-driven engineering tool and methodology for the design of heterogeneous data processing systems. Our contributions enrich the modeling and design space exploration capabilities of TTool/DiplodocusDF to target complex transfer schemes and control information exchange at different abstraction levels. Our ameliorated methodology is applied to two signal processing applications, showing the analysis of novel interactions between typically conflicting aspects such as computations vs communications and dataflows vs controlflows.**

I. Introduction

Today's embedded Systems on Chip (SoC) are more and more realized as parallel systems (e.g., Multiprocessor Systems on Chip, MPSoCs) and/or distributed systems (e.g., Networks on Chip interconnect, NoCs). This trend constitutes a challenging scenario for both embedded systems designers and programmers. For instance, in data-dominated systems (i.e., signal, image, video processing) the storage and computational resources are usually pooled and used by a set of interconnected control processors.

The obstacles encountered when designing and/or programming these SoCs lay in managing the complexity of the hardware, the software and in understanding their interactions. Currently, embedded system applications are manually programmed by system experts in languages like C/C++, with little or no use of Electronic Design Automation (EDA) tools. New products are often based on existing solutions, where system experts have a solid understanding of the different hardware and software components and above all of their interactions. An example of the latter is a software command sent to configure a processing unit contending for a bus access with a data flow. Moreover, hard constraints imposed by development costs, time-to-market and cross-platform portability, leave little room to explore and implement new solutions. Consequently, a hot topic for the industrial and scientific communities is to find an efficient way to prototype such systems in order to explore hardware/software interactions before SoCs are programmed. In order to achieve this target, the most promising strategies are based on raising the level of abstraction at which SoCs are modeled to the Electronic System Level (ESL). Here, appropriate abstractions allow to increase comprehension about the system. Models of the latter are then combined with transformation engines and generators to synthesize artifacts such as application source code.

In the scope of ESL design, we propose a demonstration to illustrate our advancements applied to two signal processing applications running on an heterogeneous multiprocessor architecture with shared memory. Following the thread initiated with past presentations [1] and [2], this year's demonstration presents TTool/DiplodocusDF: a UML Model Driven Engineering (MDE) methodology for the design, verification and code generation of modern SoCs.

The demonstration shows the modeling, simulation, formal verification and code generation of an application mapped onto an architecture, with particular emphasis on hardware/software interactions, e.g., contentions. In terms of novelty we propose two main contributions. First, a novel modeling feature to explicitly capture complex transfer schemes for data and control information, such as those involving multiple intermediate bus, DMAs and memories. Secondly, the mapping of an application control needs onto the control resources (e.g., interrupt lines, CPUs) of the underlying architecture. In this way, we enhance the design space exploration of novel interactions for later simulation and code generation. For instance, the impact of routing a data transfer through a certain path as opposed to the real time constraints required by a task that consumes those data; the impact of contentions over the overall system's performance when a given unit is used to transfer both control and data items.

This abstract is organized as follows: Section II introduces the context of TTool/DiplodocusDF and our contributions to modeling and design space exploration of modern SoCs. Section III describes the demonstration's case study. Section IV discusses relevant works in the field of hardware/software co-design that are related to our research. Finally, the state and

directions of our future works are given in Section V.

II. The Context

A. The Methodology TTool/DiplodocusDF

TTool/DiplodocusDF [3], Fig. 1, is a UML MDE methodology for the design of heterogeneous dataflow applications for real time embedded systems, conceived with signal processing applications in mind. It stems from DIPLODOCUS, [4], a UML Model Driven Engineering methodology for hw/sw partitioning of Systems on Chip at **high abstraction level**, currently implemented by the free software TTool [5]. DIPLODOCUS is based on the following fundamental two principles:

- **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 static operators.
- **Functional abstraction:** algorithms are described using abstract cost operators. The complexity of computations is taken into account without having to actually execute them. Architectures are modeled as a set of generic hardware nodes (e.g. CPUs, memories, bus) that can be interconnected and parameterized.

The core strength of DIPLODOCUS is the automatic transformation of models for simulation and formal verification [6]. However, the DIPLODOCUS approach is too abstract to permit automatic code generation for modern SoCs as models lack the necessary expressiveness to face the complexity of architectures and applications.

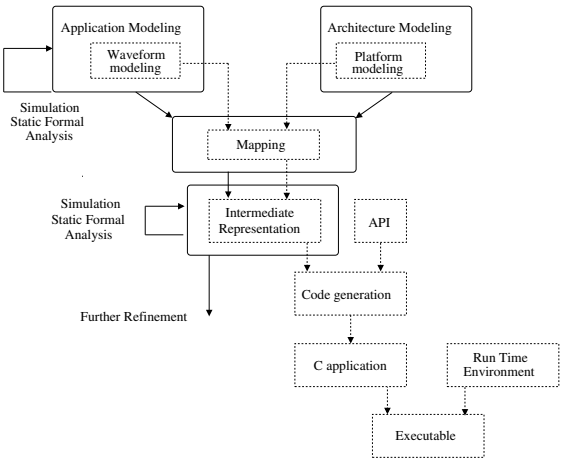

Fig. 1. The DIPLODOCUS (solid lines) and DiplodocusDF (dotted lines) methodologies

TTool/DiplodocusDF is a first attempt to fill the aforementioned gaps; it enriches DIPLODOCUS with the following extensions:

- **A dataflow semantics:** an application is modeled as a dataflow graph, where nodes represent tasks (processing, routing, addressing operations) and edges are used to carry data-blocks and the related control parameters (e.g., r/w memory addresses).
- **A specialization of the architecture language:** heterogeneous platforms are represented as a network of computation nodes (e.g., DSPs), storage nodes (e.g., memories) and transfer nodes (e.g., bus) interconnected by communication edges for data wiring.
- **An environment for automatic generation of executable code:** the description of an application mapped over a platform is translated in C-language code via an Intermediate Representation completed by the platform Application Programming Interface (API) and by a Run Time Environment for static scheduling computations.

B. Enhancing TTool/DiplodocusDF for Modern Systems on Chip

In order to address modern SoCs we need to supply for some modeling deficiencies of TTool/DiplodocusDF concerning the mapping of control and data:

1) **From the perspective of mapping data flows**, TTool/DiplodocusDF is not capable of describing complex transfer schemes such as those that can be implemented in highly parallel SoCs. Here multiple units (e.g., DMAs, Digital Signal Processors, DSPs) access shared memories and data can be transferred via multiple paths. Currently, only point-to-point transfers, corresponding to an edge between two nodes in the application graph, can be modeled. Consequently, an application graph must be manually adjusted with the injection of routing and addressing operations to meet the transfer capabilities of the hardware. This limits the cross-platform portability of applications (i.e., an application graph tailored to a given platform must be re-designed if a different architecture is targeted) and prevents an efficient usage of architecture resources (i.e., the best way to move data might not always be a point-to-point transfer).
2) **From the perspective of mapping control flows**, TTool/DiplodocusDF lacks the means to map control-related information from the application onto the architecture. Interactions between data and control flows, such as contention, thus cannot be represented and are hidden to the eyes of a developer. The mapping of an application is blind with respect to the real control capabilities of each hardware unit: two dependent tasks can be mapped to execution units that are not able to communicate control information.

We recently enhanced TTool/DiplodocusDF with the modeling features essential to tackle the above points:

1) To cope with the limitations in terms of the description of complex transfers, we have introduced **Communication Patterns** [7]. This novel modeling feature allows to explicitly describe complex data transfers schemes so as to decouple an application graph from the architecture constraints related to addressing. In a nutshell, a Communication Pattern is made up of a set of interconnected architecture units as well as the application source and destination of a transfer. By means of UML Sequence

and Activity diagrams, Fig. 3, 4, all the actions and the architecture units involved in a transfer are described and an order relation is provided for verification and code generation purposes.

2) **The description languages have been enriched**, as described in [8], **with** the expressiveness to address the **exchange of control information**. An application model now features control exchange primitives and nodes to gathercast and/or broadcast control information among tasks. Similarly, an architecture model is provided with dedicated nodes and links to describe the hardware control capabilities. The control primitives of an application graph can be mapped onto the control network of the hardware.

III. The Demonstration Case Study

The purpose of this demonstration is to show the methodology DiplodocusDF and the contributions we mentioned in Section II-B, within the framework of TTool [5].

The case study we chose for this demonstration belongs to the domain of Software Defined Radio (SDR) [9]. Here, adding reconfigurability has blurred the separation between hardware and software, therefore introducing new interactions as well as increasing the complexity of programming such systems. 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, all functionalities where previously implemented in hardware.

This section presents a Software Defined Radio system composed of a hardware architecture (Embb) and two software applications (High Order Cumulants, HOC and Welch Periodogram Detector, WPD) that are the subject of our demonstration.

A. The Applications

The two applications we chose for this demonstration are used in cognitive radio and operate on an input data stream that is processed to retrieve information about the usage of frequency spectrum.

1) High Order Cumulants (HOC): The first application, HOC is a classification algorithm (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 application graph for HOC is illustrated in Figure 2. The occupancy of a specific frequency range is determined by extracting a score out of the input data stream from the Source block. Such a stream is broadcast, FORK, and then processed by operations CWM1, CWM2 (Component-Wise Modulus) and CWS (Component-Wise Square). The so-computed classification score is then accumulated over a classification period by node ACC (ACCumulation) which in turn dispatches its result to the Sink. In the latter block, the spectrum occupancy is determined by comparing the accumulated scores with a pre-computed threshold. Sink also receives a copy of the unprocessed data directly from Source. Such a copy is not used to compute the classification score but is specially instantiated for the demonstration to illustrate how Communication Patterns are deployed to capture complex transfer schemes, Fig. 3 and Fig. 4. In Figure 2, data dependencies are represented by channels connected via blue ports to the graph nodes, while control dependencies are connected to via purple ports. Sink, CMW1, CWM2 and CWS control and configure the input data stream produced by Source. Moreover, Sink also control and configures the accumulation period of scores in ACC.

Fig. 3 illustrates the UML Activity Diagram for a Communication Pattern where three parallel data transfers take place. These transfers move data from Source to processing operations CWM1 (Source_to_CWM1), CWS (Source_to_CWS) and Sink (Source_to_Sink), via the channels ch1, ch2, ch3, ch8 and ch10 of Fig. 2. Fig. 4 details with a UML Sequence Diagram all the actors and the actions necessary to carry out the transfer Source_to_Sink. The latter is controlled and configured by Main CPU and executed by Bridge DMA. Data are transferred from the DSP memory to the Main Memory via Main Bus, Bridge and Crossbar.

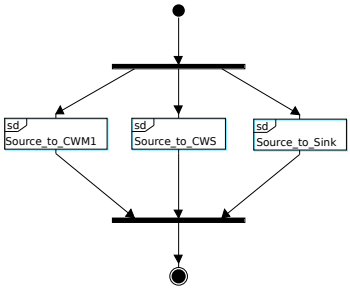

Fig. 3. The UML Activity Diagram for a Communication Pattern describing three parallel transfers from the graph of Fig. 2. Data is being transferred in parallel from the Source block to processing nodes CWM1, CWS and Sink.

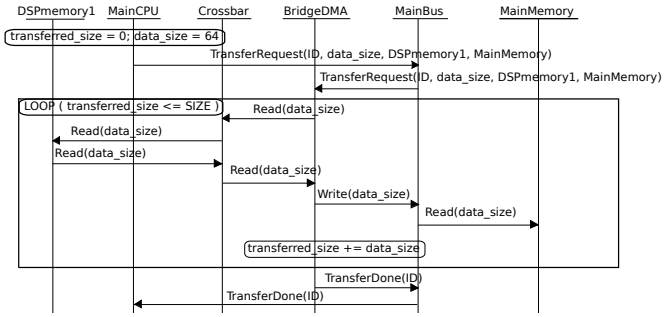

Fig. 4. The UML Sequence Diagram for transfer *Source_to_Sink* of Fig. 3. This example highlights the expressiveness of Communication Patterns to describe a transfer where data is routed through the whole system interconnect of Embb.

2) Welch Periodogram Detector (WPD): The second application we will show in this demonstration is a sensing algorithm, Welch Periodogram Detector (WPD) as implemented in [3], that is used for sensing the spectrum and detecting

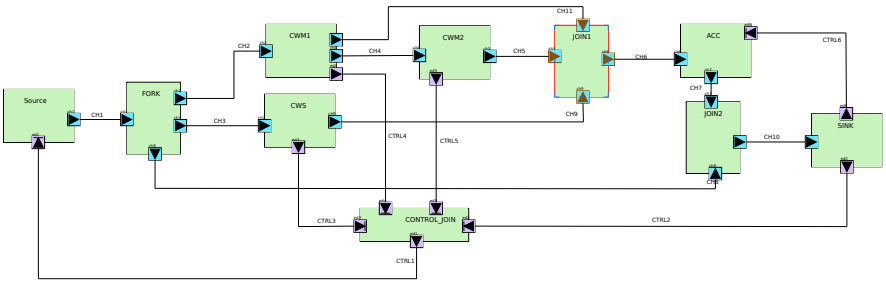

Fig. 2. The application graph for the High Order Cumulants application

when a given frequency band can be opportunistically used by a non-licensed user. Fig. 5 shows the application graph for WPD. Here, a Source node produces the input vectors whom frequency representation is computed via a Fast Fourier Transform by the node FFT. The output samples of the latter are then processed by Component-Wise Modulus (CWM) node. Next, two output vectors of node CWM are added by the Component-Wise Addition (CWA) node and the components of the resulting vector are summed by node SUM and collected by node Sink. As for the previous application, the processing operations of Sink, FFT, CMW and CWS control and configure the input data stream produced by Source.

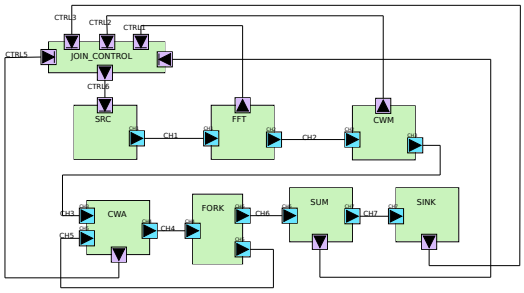

Fig. 5. The application graph for Welch Periodogram Detector algorithm

B. The Hardware Platform: Embb

The hardware architecture for this demonstration is Embb [11], Figure 6, a generic baseband architecture dedicated to SDR applications. Embb is composed of a Processing System, left-hand side of Figure 6, interconnected to a Control System, right-hand side of Figure 6. The Processing System operates on the raw samples that are transmitted or received via the Radio Frequency Interface (RFI), by executing signal processing operations 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 CPU (Central Processing Unit). The internal memory is 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 latter is shared with the Control System, which is also composed of a Main Memory and the Main CPU. The latter executes the control part of an SDR application: it manages data transfers, DSPs, the RFI and the External Environment Interface. Control information within Embb are exchanged either through a dedicated network of interrupt lines and interrupt controllers interconnecting DSPs with the Main CPU, or by means of control instructions passing through the system interconnect.

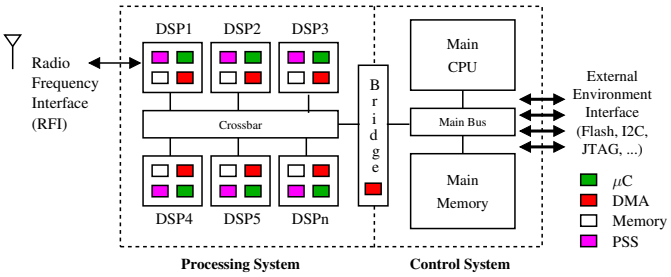

Fig. 6. The architecture of a generic Embb instance

C. Simulation and Formal Verification

Fast simulation and formal verification [12] can be conducted over TTool/DiplodocusDF's application models before and after mapping, as illustrated in Fig. 1. The simulation environment, which has been the subject of previous demonstrations [1], [2], allows for an interactive exploration of the application mapped onto a particular architecture via a revisited version of Discrete Event Model of Computation (MoC) based on *transactions*, a data structure that represents a computation or communication action involving one or more hardware components. The simulation speed directly matches the granularity of models and the simulator tool comes with a graphical interface which allows the user to easily interface with animated models whose execution can be customized by means of break-points, generation of execution traces, save/restore simulation states and other debug facilities. Fig. 7 shows a simulation window where the Welch Periodogram Detector application of Fig. 5 runs on an instance of Embb. Fig. 7 shows the load of each architecture component, defined as the occupation time with respect to the total simulation time. Thanks to these information, showing for instance that

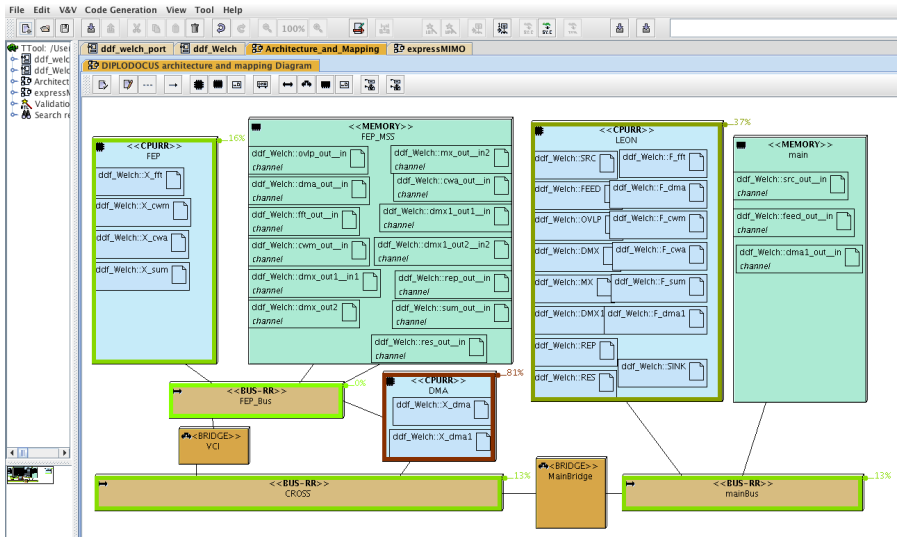

Fig. 7. Welch Periodogram Detector simulated on an instance of Embb composed of the Control System and the Processing System made up of a DSP, where all processing operations of Fig. 5 have been mapped.

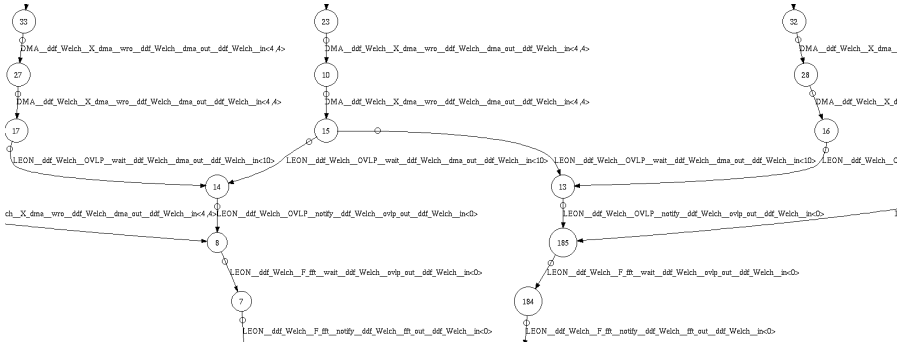

Fig. 8. A portion of the reachability graph for the WPD application running on the same instance of Embb as in Fig. 7. The reachability graph captures all possible simulation traces in the form of a graph. The latter is then used by TTool/DiplodocusDF for the model-checking of system properties.

the DSP's DMA is the most heavily loaded unit (81% of the total simulation time), a designer can easily identify a potential system bottleneck and explore the design space of the WPD simply by changing the application mapping, without re-designing the whole system.

Formal verification is made possible thanks to the high abstraction level of models (as described in Section II-A) as well as to their formal semantics. The latter is defined in the formal semantics of LOTOS and the timed automata underlying UPPAAL. TTool/DiplodocusDF formal verification facilities allow model-checking of system properties such as safety, schedulability and performance. Fig. 8 shows an excerpt of the reachability graph of the WPD application running on the same instance of Embb as the simulation of Fig. 7. The graph of Fig. 8 illustrates different traces merging together: in the upper part of Fig. 8, the reader can identify transactions caused by the DMA execution, whereas, in the lower part, transitions labeled with LEON are due to the execution of tasks on the Main CPU (a LEON SPARC-V8 microprocessor).

D. Code Generation

TTool/DiplodocusDF can automatically generate an almost complete software implementation of a dataflow application in C code. TTool/DiplodocusDF has been deployed to automatically synthesize the code for the WPD application of Fig. 5, which has successfully run on the FPGA-based prototyping board ExpressMIMO[]. Currently, the generated code is an almost complete version, in the sense that memory allocation and the related addressing parameters for DSPs are left to the designer. As summarized in Fig. 1, synthesis in TTool/DiplodocusDF takes place via the generation of an Intermediate Representation of an application mapped onto an architecture and enriched with the Application Programming Interface of the latter. The executable code is then produced by translating the Intermediate Representation into C code, which is in turn compiled and linked against the Run Time Environment in charge of scheduling the application.

IV. Related Work

Typically, ESL design tools are domain specific. Some of them they rely on formal Models of Computation (MoCs)

such as dataflow models, state machines or process networks. These MoCs are deployed to model both non-functional and functional requirements as input description to ESL synthesis tools. According to the novel classification proposed in [13], true ESL synthesis tools are the well-known Daedalus [14], [15], Koski [16], Metropolis [17], Ptolemy [18], PeaCE/HoPES [19], SCE [20] and SystemCoDesigner[21]. With respect to these works, TTool/DiplodocusDF targets the same hardware platforms, i.e., heterogeneous bus-based MPSoCs, and shares the same design principles in terms of providing a complete flow that can generate systems across hardware and software boundaries from an algorithmic specification.

The abovementioned MoCs are not the only solution to describe input requirements: UML has also gained attention in the domain of hardware/software co-design [22]. In terms of modeling the UML/MARTE [23] profile has been especially proposed for embedded system design. However, it lacks the notation to describe dataflow application in a pure abstract way and its procedural approach leads to models with only one centralized controller [3]. In terms of simulation, some of the state of the art UML modeling tools, such as Tau [24], TOPCASED [25], Papyrus [26], Artisan [27] and Rhapsody [28], only simulate purely functional models in an untimed fashion, whereas, TTool/DiplodocusDF also accounts for architecture constraints, e.g., arbitration of shared resources, speed and data throughput of components.

V. Conclusions and Future Directions

In conclusion, it can be said that we enriched TTool/DiplodocusDF with the expressiveness to describe control information exchange and complex communication transfer schemes. As a result of applying these contributions to TTool, our enriched TTool/DiplodocusDF is provided with the means to model and verify novel design space points and interactions. Controlflows, data transfers and interactions between the latter two, that before our contributions were invisible to the developer, can finally be taken into account. This early detection may considerably alleviate the process of:

- Debugging existing applications and systems
- Enhancing the analysis of hardware/software interactions early at design time
- Fast prototyping of applications for design space exploration on the real hardware

In future works, the investigation of new modeling techniques will remain subject to our research. In this direction, we will further study the application of Communication Patterns to recent communication architectures such as Networks on Chip. Parallel to this, a second direction of our research will target code generation from TTool/DiplodocusDF models [3]. We will re-visit the code generation phase of TTool/DiplodocusDF so as to include the automatic synthesis of code for data transfers. Additionally, we will also account for the control dependencies among nodes in the application graph, independently with respect to the dataflows. This will enable us to improve the existing Run-Time Environment of TTool/DiplodocusDF with dynamic scheduling and dynamic memory management, so as to make a given system reactive to the needs of its surrounding environment.

References

[1] D. Knorreck et al., "Demonstration of an Interactive System Level Simulation Environment for Systems-on-Chip", DATE'10 University Booth, Dresden, Germany, March 2010

[2] D. Knorreck et al., "Demonstration of a Coverage Driven Verification Environment for UML Models of Systems-on-Chip", DATE'11 University Booth, Grenoble, France, March 2011

[3] J. M. Gonzalez Pina, "Application Modeling and Software Architectures for the Software Defined Radio", Ph.D dissertation, Telecom ParisTech, May 2013.

[4] L. Apvrille et al., "A UML-based Environment for System Design Space Exploration", in *IEEE ICECS*, pp. 1272-1275, 2006.

[5] TTool, *http://ttool.telecom-paristech.fr/*, 2014.

[6] D. Knorreck, L. Apvrille and R. Pacalet, *Formal system-level design space exploration*, in Concurrency and Computation: Practice and Experience, vol. 25, no. 2, 2013, pp. 250-264.

[7] A. Enrici et al., "Communication Patterns: a Novel Modeling Approach for Software Defined Radio Systems", to be published in *IARIA COCORA 2014*, February 2014.

[8] A. Enrici et al., "Taming Control Exchange for Software Defined Radio in System Level Models", in 7^{th} Junior Research Workshop on Real-Time Computing, pp. 37-40, October 2013.

[9] J. Mitola III, "Cognitive Radio: An Integrated Agent Architecture for Software Defined Radio", Ph.D dissertation, Royal Institute of Technology (KTH), May 2000.

[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/, 2014.

[11] N. -ul. -I. Muhammad et al., "Flexible Baseband Architectures for Future Wireless Systems", in *EUROMICRO DSD*, pp. 39-46, 2008.

[12] D. Knorreck, "UML-based Design Space Exploration, Fast Simulation and Static Analysis", Ph.D dissertation, Telecom ParisTech, October 2011.

[13] 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.

[14] 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.

[15] H. Nikolov, "Daedalus: Toward composable multimedia MP-SoC design", in *DAC*, June 2008, pp. 574-579.

[16] 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.

[17] F. Balarin et al., "Metropolis: An integrated electronic system design environment", in *IEEE Computer*, vol. 36, no. 4, pp. 45-52, April 2003.

[18] The Ptolemy Project, *http://ptolemy.eecs.berkeley.edu*, 2014.

[19] 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.

[20] R. Dmer 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.

[21] 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.

[22] Y. Vanderperren et al., "UML for electronic systems design: a comprenhensive overview", in *Design Automation for Embedded Systems*, vol. 12, no. 4, pp. 261-292, 2008.

[23] The UML profile for MARTE, *http://www.omgmarte.org*, 2014.

[24] IBM Tau, http://www-03.ibm.com/software/products/en/ratitau, 2014.

[25] TOPCASED, http://www.topcased.org/, 2014.

[26] Papyrus, www.papyrusuml.org, 2014.

[27] Artisan Studio, http://www.atego.com/fr/products/artisan-studio/, 2014.

[28] IBM Rhapsody, http://www-03.ibm.com/software/products/en/ratirhapfami, 2014