The Ψ-chart Design Approach in TTool/DIPLODOCUS: a Framework for Hw/Sw Co-Design of Data-Dominated Systems-on-Chip

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

***Abstract*—In the scope of the DATE 2015 University Booth, we present our latest achievements for the system level design of parallel and distributed embedded systems. We propose a demonstration of a novel design approach, the Ψ-chart, in TTool/DIPLODOCUS, a UML/SysML framework for the design, validation and automatic code generation for data-dominated SoCs. The Ψ-chart is a design approach where communication patterns are designed with dedicated models, independently of a pair application-architecture, before mapping phase. It allows for a complete orthogonalization of concerns between the design of computations and communications, thus achieving faster Design Space Exploration, complete design portability as well as reduced design times and costs. The subject of our demonstration is the design of the physical layer (PHY) of the transmitter part of the Zigbee wireless standard (IEEE 802.15.4) mapped onto a MPSoC architecture with shared memory. Our demonstration will illustrate the full design of the Zigbee transmitter, from models to the automatic generation of the emulation code, via simulation and formal verification. We will validate our design by comparing the output samples produced by the emulation code, with a real implementation of the transmitter on a FPGA prototyping board.**

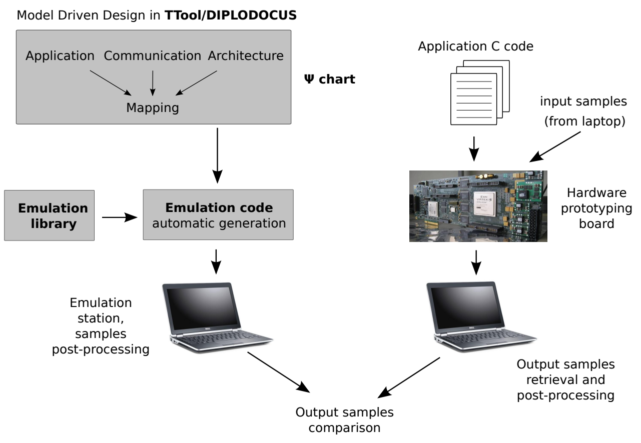
I. Introduction

To cope with conflicting needs such as performance and power consumption, today's embedded systems are more and more realized as parallel systems where the processing and the control are distributed over a network of interconnected subsystems (e.g., Multi-Processors Systems on Chip, MPSoCs, automotive and avionics systems).
In this context, traditional Model Driven Engineering (MDE) [1] design approaches, such as Kienhuis et al.'s Y-chart [2], separate the modeling of the system's functionality (i.e., *what* the system does, the application) from the modeling of the system's available resources (*who* does what, the architecture). Such separation of concerns results into an effective design methodology if there is a *match* between the semantics of the application Model of Computation (MoC) and the architecture MoC.
However, the advent of parallel and distributed systems with the complex communication services they offer, puts a heavy strain on design approaches based on the above-mentioned separation of concerns. In commonly used Models of Computations for system-level design (e.g., dataflow MoCs, Process Networks, UML), the semantics associated to the communication entities (i.e., relationships/ports) is sufficient to capture point-to-point logical communication channels. These channels naturally fit the communication services offered by centralized systems where transfers take place over simple point-to-point paths (e.g., processor-bus-memory). Nevertheless, it is not sufficient to capture the complex communication paths (e.g., hierarchical bus architectures with Direct Memory Access units, DMAs) and services (e.g. packet-based routing) available in the interconnects of modern Multi-Processors Systems-on-Chip. This lack of a match between application and architecture MoCs in terms of communications, greatly impacts the quality of a design (e.g., portability, scalability, re-use of models) and the overall development of a new products (e.g., time-to-market, design costs).
As a solution to this *communication mismatch*, we proposed in [3] the Ψ-chart approach. A novel MDE design approach where dedicated models are introduced to capture communication patterns independently of the application-architecture models, before mapping phase. To show the effectiveness of our novel approach we present in this demonstration the design of the physical layer of the Zigbee wireless standard (IEEE 802.15.4) mapped onto a multi-processor architecture with shared memory. Such a design is demonstrated in the frame of TTool/DIPLODOCUS [4], a UML/SysML Model Driven Engineering framework for the design, simulation, formal verification and code generation of data-dominated systems.
The rest of the paper is organized as follows. In Section II we provide the context of our demonstration by summarizing the Ψ-chart approach as well as its implementation in TTool-/DIPLODOCUS. Section III describes in further details the demonstration case study, Section IV discusses our contributions with respect to existing works for hardware/software co-design and communication infrastructure design. Section V concludes the paper.

II. The Context

In this section we first describe an overview of the demonstration. Next, we detail each sub-part in a dedicated sub-

section.
Fig. 1 left-hand side, shows the tool-chain of our Computer Aided Design approach. Here, the complete system is designed with the Ψ-chart design approach (subsection II-A), in the frame of the UML/SysML framework TTool/DIPLODOCUS (subsections II-B, II-C). Once the system under design meets the desired requirements (e.g., performance, functionality, throughput), TTool/DIPLODOCUS models are automatically transformed in order to produce the emulation code of the application (Zigbee TX physical layer). The emulation code is compiled and linked against the emulation library of Embb [15] (libembb). Embb (more in subsection III-B) is a multi-processor platform with shared memory that we have selected as a representative architecture of modern MPSoCs. The resulting binary file runs on a laptop (the emulation station) where output samples are collected and represented in the form of a graph that is compared with the outcome of the manual implementation (subsection II-D), right-side of Fig. 1.


Model Driven Design in TTool/DIPLODOCUS
Application Communication Architecture
Mapping
Ψ chart
Application C code
input samples (from laptop)
Emulation library
Emulation code automatic generation
Hardware prototyping board
Emulation station, samples post-processing
Output samples retrieval and post-processing
Output samples comparison

chch

Fig. 1. An overview of the proposed demonstration.

A. The Ψ-chart design approach

The Ψ-chart approach, Fig. 2, is a Model Driven Engineering approach for the design and Design Space Exploration of data-dominated applications for real-time embedded systems, e.g., signal, image and video processing systems. It stems from the Y-chart approach [2], currently the most dominant design approach for data-dominated embedded systems. As opposed to the Y-chart approach, the Ψ-chart approach separates the design of communications, from the design of an application-architecture pair. If designed withing the frame of the Y-chart approach, communications are typically described both in the application and in the architecture model. In the application model, communications are represented at a very high abstraction level in the form of logical dependencies between computations (e.g., channels, edges). In the architecture, communications are described in the form of resources (e.g. DMA engines, buses, bridges, shared memories) and in the services they offer (e.g., communication, storage, transfer). At mapping phase, it is frequent that the description of communications in

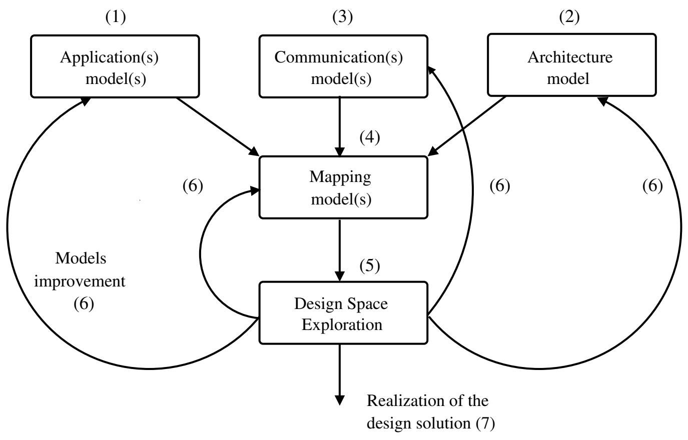


Fig. 2. The Ψ-chart approach for the design of parallel and distributed embedded systems.

the application (e.g., data channel with blocking read/write primitives) *does not match* the one present in the architecture (e.g., bus and DMA transactions from a source to a destination memory). This *communication mismatch* causes expensive re-designs with important consequences on the whole design methodology. In fact, models must be refined/modified/re-designed with additional information to properly describe how data are transferred. Consequently, at mapping phase, additional modeling is required. This results into communications being usually designed after a first mapping of the application onto the architecture. Such a solution makes the Design Space Exploration of communications dependent on the application mapping.
As depicted in Fig. 2, we solved the communication mismatch by introducing dedicated models for communications, before mapping phase. These models describe the behavior of communication protocols and standards currently used to transfer data in parallel and distributed embedded systems. On one hand, this is achieved by modeling the algorithm of a data transfer, independently of the simple point-to-point dependencies between computations of the application model. On the other hand, the units that are involved in a data transfer (e.g., memory, CPU, bus, DMA) are captured at a high abstraction level, in terms of generic parameterized entities (e.g., storage, controller, transfer components) that are independent of the specific units of a given platform. In order to map a pair application-communication models onto an architecture model, the Ψ-chart approach deploys the methodology depicted in Fig. 3. This methodology is based on refining models at four levels of abstraction, where each level is associated to the mapping of a given class of capabilities, namely *processing* (level L0), *storage* (level L1), *communication* (level L2) and *transfer* (level L3). At each level, the designer selects the desired architecture units and assigns a value to architecture-dependent variables (e.g., source and destination addresses of a data transfer).

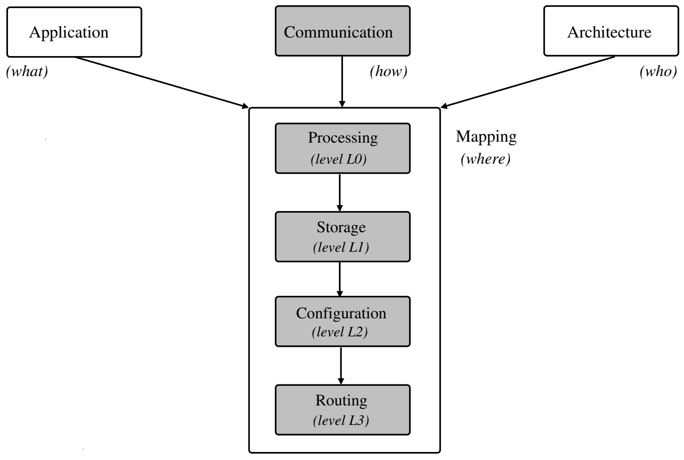


Fig. 3. The mapping methodology of the Ψ-chart (gray area).

B. The DIPLODOCUS profile

In DIPLODOCUS, an **application model** is a composition of SysML Block Definition and Block Instance diagrams, where the behavior of each block is described by a SysML Activity Diagram. An application model captures the functionalities of a data-processing algorithm, described as a set of blocks interconnected by data and control dependencies. Data dependencies are expressed by *ports* and *channels* to which blocking/non-blocking read() and write() primitives are associated. Control dependencies are expressed by *events* and *requests*. An event is similar to a channel but it is used to exchange control parameters to which natural or boolean values are assigned. Requests, in turn, are used to spawn the execution of a block. Similarly to an event, a request can carry control parameters to which a natural or a boolean value is assigned. DIPLODOCUS models an application at a level of abstraction that is explained by the two following principles:

- *Data abstraction:* only the amount of data exchanged through channels and ports between application blocks is modeled. Decisions that depend on the value of data are abstracted and expressed in terms of non-deterministic and static operators.
- *Functional abstraction:* algorithms are described using abstract cost operators. The complexity of processing data is expressed in terms of the number of operations. The complexity of algorithms is thus taken into account without having to actually execute them.

An **architecture model** is a UML Deployment Diagram composed of a set of generic interconnected units, e.g., bus, CPU, DMA. These units are characterized by performance parameters such as: the scheduling policy, the number of cores for a CPU, the arbitration policy and the size of data channels for a bus unit.

A **communication model** is called Communication Pattern (CP) and it is composed of UML Behavior Diagrams, namely Activity and Sequence Diagrams. The activities that compose a communication protocol or standard (e.g., to configure a data transfer, to execute a transfer cycle, to acknowledge the termination of a transfer) are captured by Activity Diagrams. On the other hand, the low level interactions (e.g., read/write bus transactions) among architecture components are described by Sequence Diagrams.

More in detail, a Communication Pattern is composed of a main Activity Diagram that describes the whole data-transfer algorithm. This main diagram then references either Sequence Diagrams or other Activity Diagrams.

A CP's Activity Diagram captures the structure and relations among sub-functions of a communication protocol. This diagram exposes the control relations and the parallelism among sub-functions. The latter are composed by means of operators to describe concurrency, sequencing, choice and iteration. These operators are governed by control variables that are declared as attributes of a Sequence Diagram's instance.

A CP's Sequence Diagram describes the way the architecture components interact in order to execute a sub-function (e.g., read/write bus transactions). Interactions are strictly ordered and are represented by the exchange of parameterized messages (e.g., Read(), Write(), TransferRequest()) between instances of a diagram. These messages abstract the functionalities of communication protocols' signals. The parameters of a message are the attributes associated to instances. In our current implementation, these attributes can be of type boolean or natural. Attributes are initialized or assigned a value only in Sequence Diagrams, whereas this value can only be read in Activity Diagrams. Action states can be deployed in a Communication Pattern's Sequence Diagrams and are typically used to describe time-related events, e.g., wait states.

In a **mapping model**, the functionality of application and communication models is bound to the resources of the architecture. A mapping model is created from an instance of the architecture model. UML artifacts are used to map the application blocks and events to specific architecture units (e.g., CPU, bus, memory). For the mapping of Communication Patterns a dedicated UML artifact is provided. The latter allows to map the instances of a Communication Pattern's Sequence Diagrams to specific units on the architecture Deployment Diagram. Additionally, it permits to associate to a Communication Pattern a (set of) channel(s) from the application model.

C. The TTool framework

TTool [4] is a Computer Aided Design framework that supports the edition, simulation and formal verification of UML and SysML diagrams for multiple profiles. In this subsection, we restrict our description of TTool to the facilities offered for the design of embedded systems with DIPLODOCUS.

1) Simulation and Formal Verification: Fast simulation and formal verification [5] can be conducted over DIPLODOCUS application models before and after mapping. The simulation environment, which has been the subject of previous demonstrations, [6] and [7], allows for an interactive exploration of the application mapped onto a particular architecture via a revisited version of Discrete Event Model of Computation

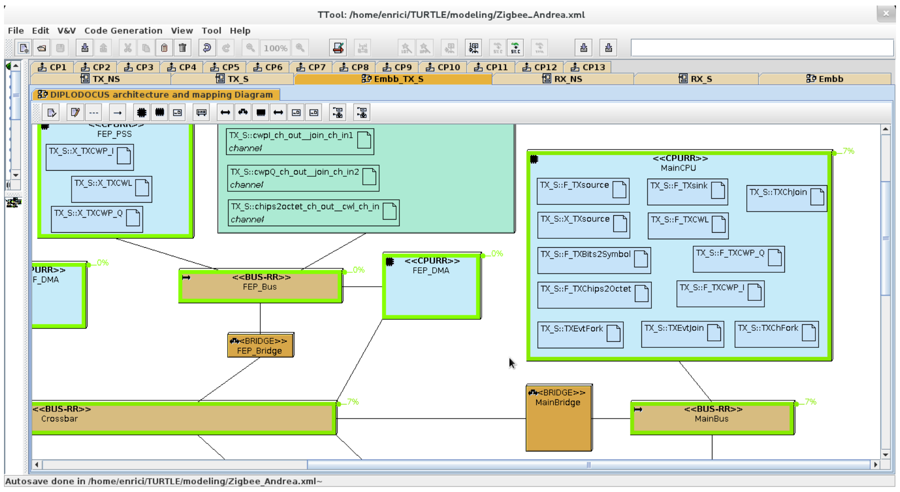

TTool: /home/enrici/TURTLE/modeling/Zigbee_Andrea.xml
File Edit V&V Code Generation View Tool Help
DIPLODOCUS architecture and mapping Diagram
<<CPURR>> MainCPU
<<BUS-RR>> FEP_Bus
<<BUS-RR>> Crossbar
<<BUS-RR>> MainBus
<<BRIDGE>> FEP_Bridge
<<BRIDGE>> MainBridge
<<CPURR>> FEP_DMA
Autosave done in /home/enrici/TURTLE/modeling/Zigbee_Andrea.xml~

Fig. 4. A portion of the Embb architecture model of Fig. 7, animated as a result of simulation. Simulation is done after mapping the application model of Fig. 6.

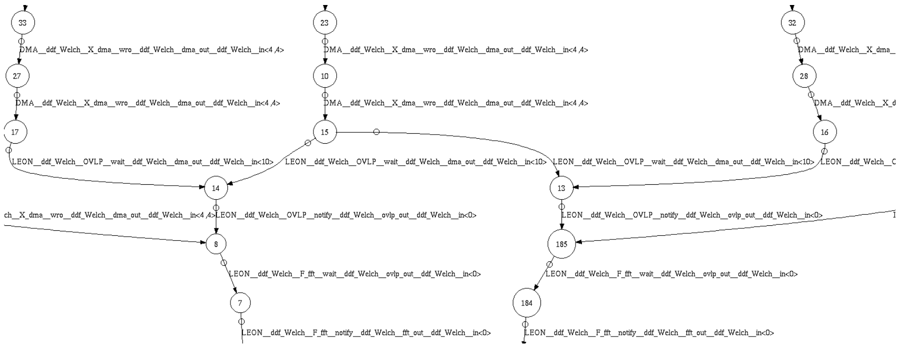


Fig. 5. A portion of the reachability graph for the Zigbee TX application running on the same instance of Embb as in Fig. 4. The reachability graph captures all possible simulation traces in the form of a graph. The latter is then used by TTool/DIPLODOCUS for the model-checking of system properties.

(MoC). This revisited version is 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. The simulator engine comes with a graphical interface which allows the user to easily interface with animated models. The execution of simulated models can be customized by means of break-points, generation of execution traces, save/restore simulation states and other debug facilities. Fig. 4 shows a simulation window where the Zigbee TX application model of Fig. 6 is mapped onto the instance of Embb of Fig. 7. Fig. 4 shows the load of each architecture component, defined as the occupation time with respect to the total simulation time. Thanks to these information, a designer can easily identify a potential system bottleneck and explore the design space of the application simply by changing the mapping, without re-designing the whole system. The simulator also provides some formal verification capabilities: the GUI is provided with a bar that allows the user to select a percentage of a mapping-model's state space to be traversed during simulation. This is based on model checking and static program analysis techniques. These are used to compare and merge logically equivalent execution paths and/or recurring system states. The objective is to allow for taking design decisions dynamically, at simulation run-time without regenerating the executable model. From a designer's perspective, it is in fact desirable to generate an executable model once and to subsequently traverse an interesting fraction of its state space. The design space may then be pruned with the aid of conventional coverage criteria (with respect to covered branches, statements, tasks, conditions,etc.), expert knowledge provided by the user (e.g. potentially critical parts of the control flow to the environment), or heuristics taking into account (non-) functional properties.

Apart from the formal verification facilities of the simulator, DIPLODOCUS also allows the model-checking of system properties such as liveness and reachability. Formal verification is performed on the application model before mapping thanks to a translation of DIPLODOCUS's concepts into the formal semantics of LOTOS and the timed automata underlying UPPAAL [8]. After mapping it is possible to perform formal verification via an in-house model checker. Fig. 5 shows an excerpt of the reachability graph of the Zigbee TX application running on the same instance of Embb as the simulation of Fig. 4. The graph of Fig. 5 illustrates different traces merging together: in the upper part of Fig. 5, the reader can identify transactions caused by the DMA execution, whereas, in the lower part, transitions labeled with MAINCPU are due to the execution of tasks on Embb's main CPU. A reachability graph may also be transformed into a Labeled Transition System. The latter can then be used as an input to CADP [9] that implements minimization techniques based on trace or observational equivalences.

2) Code Generation: TTool can automatically generate an almost complete software implementation of a dataflow application in C code, from DIPLODOCUS models. 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. Automatic code synthesis for emulation takes place via the generation of an Intermediate Representation (IR) of a mapping model that is enriched with the Application Programming Interface (API) of the target architecture. The latter is taken from `libembb`, a library intended for pure software emulation and algorithmic validation. It offers all the functionalities of the available Embb DSP units. The computations are bit accurate and applications built on top of libembb run on a regular desktop or laptop. The executable emulation code is then produced by translating the Intermediate Representation into C code, which is in turn compiled and linked against a Run Time Environment in charge of scheduling the application computations and data transfers.

D. The manual design approach

Fig. 1 right-hand side, shows how the Zigbee TX output samples are produced with a manual (traditional) design approach. In the figure, the application code is manually developed and executed on the prototyping board (a Zedboard [10]), where the instance of Embb modeled in Fig. 7 has been implemented. The application code runs on Embb's main CPU (an ARM processor), as part of a software stack composed of the operating system MutekH [11] and the operating system drivers specific for Embb's Processing SubSystems (PSS, see Section III). With respect to Fig. 7, the code running on the main CPU controls Embb's PSS to process the input samples and produce output samples that are stored in the General Purpose Control Processor main memory. From there, samples are then transferred via Ethernet to a laptop where they are plotted in the form of a graph. The same laptop is also used to dispatch the input samples to the protyping board.

III. The Demonstration Case Study

In line with the demonstration we presented in the previous edition of the University Booth 2014 [12], this year we present again a system from the field of the Software Defined Radio (SDR) [13].

A. The application

The application we model in this demonstration is the 2.4 GHz physical layer of the Zibgee transmitter as standardized by the IEEE 802.15.4 group [14]. Fig. 6 shows the TTool/DIPLODOCUS application model for the Zigbee transmitter. Here, the block labeled TX_source produces the data to be transmitted in the form of a flow of bits. These data are then converted to symbols by the TX_Symbol2ChipSeq block. In this block, we model the fact that each incoming 4-bits symbol is mapped to one of the 16 sequences of 32 chips as defined by the IEEE standard 802.15.4. The TX_Chips2Octet block, then transforms each incoming chip (bit) of a chip sequence into an unsigned 8-bits integer as expressed in equation 1:

$$\{\texttt{0;1}\} \rightarrow \{\texttt{0x00;0x01}\} \tag{1}$$

and separates the even-indexed chips that are used to modulate the in-phase (I) carrier component from the odd-indexed chips that are used to modulate the quadrature (Q) carrier component. The output is then transformed by means of a Component Wise Lookup (TX_CWL block) that maps unsigned 8-bits integers to signed 16 bits integers as expressed by equation 2:

$$\{\texttt{0x00;0x01}\} \rightarrow \{\texttt{0xffff;0x0001}\} \tag{2}$$

At this point, given the separation of the I and Q branches, their pulse shaping can be executed independently and the application graph exposes this parallelism by forking the output data of block CWL to two distinct Component Wise Product (CWP) blocks, TX_CWP_I for the I branch and TX_CWP_Q for the Q branch. These blocks multiply the input samples with a half-sine wave to realize the O-QPSK modulation. The quadrature shift between the I and Q branches is implemented by means of an offset between the addresses where samples corresponding to the two streams are stored in memory. The resulting frame of complex samples (16 bits for the real part and 16 bits for the imaginary part) is then collected by block TX_sink and transmitted over the air.
Each block of the model in Fig. 6 is composed of two tasks: one modeling the data-processing and one modeling the related control operations. By convention we name the data-processing tasks with a heading X that stands for eXecution and the control tasks with a heading F that stands for Firing.

B. The Hardware Platform: Embb

The target hardware architecture for our case study is Embb [15], a generic baseband architecture dedicated to signal processing applications.
Fig. 7a shows the UML Deployment Diagram of Embb's architecture, as modeled in TTool/DIPLODOCUS. Embb is composed of a Digital Signal Processing part (DSP part) and a general purpose control processor (the main CPU). In the DSP part, left-hand side of Fig. 7a, samples coming from the air are processed in parallel by a distributed set of Digital Signal Processing Units (DSPU1 through DSPUn) interconnected by a crossbar (Crossbar). Fig. 7b illustrates the internal architecture of a DSPU: each unit is equipped with a local micro-controller (μC) that allows to reduce interventions of the main

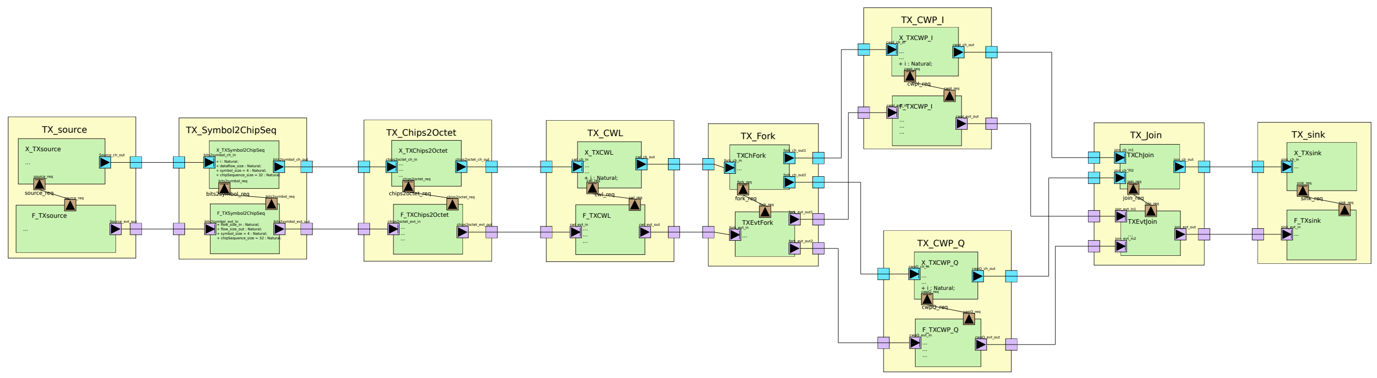


Fig. 6. The TTool/DIPLODOCUS model of the Zigbee transmitter corresponding to our implementation for Embb.

CPU, a Processing Sub-System (PSS) as computational unit and a Direct Memory Access controller (DMA) to transfer data in and out of the DSPU's local memory (the Memory Sub-System, MSS). The latter is mapped on the global address map of the main CPU and is accessible by the DMAs, the μCs and the system interconnect. The system interconnect permits exchanges of control and data items: it is composed of a crossbar, a bridge (Main Bridge) and a main bus (Main Bus). The system interconnect is shared between the DSP part and the main CPU, right-hand side of Fig. 7a, where the control operations of an application are executed. The main CPU is in charge of configuring and controlling the processing operations performed by the DSPUs and the data transfers. The main CPU disposes of a memory unit (MAINmemory) and a bus interconnect (MAINbus). The latter is linked to the DSP part via the Main Bridge.
Control information within Embb are exchanged either through a dedicated network of interrupt lines and interrupt controllers interconnecting the DSPUs with the main CPU, or by means of control instructions passing through the system interconnect. Such a control network is not represented in Fig. 7.

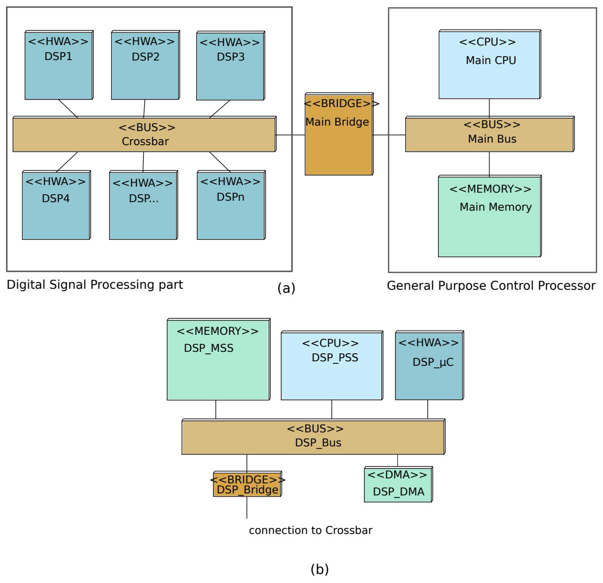


Fig. 7. The UML Deployment Diagrams of an architecture instance of Embb

C. Communications

Given the capabilities offered by the platform Embb, the Communication Patterns we will present represent DMA transfers as well as the non-cachable load/store operations that are executed by the PSS units and the main CPU in order to retrieve data from their local memories.
For the sake of simplicity, we illustrate here one pre-mapping Communication Pattern that models a DMA data-transfer where the transfer termination is signalled via polling mechanism. Fig. 8 shows the main Activity Diagram of this Communication Pattern, where we instantiated one Sequence Diagram, ConfigureTransfer, and two Activity Diagrams: TransferCycleAD and PollingCycleAD. First, the data transfer is configured (ConfigureTransfer), next the transfer of data (TransferCycleAD) is executed parallel to the polling cycle (PollingCycleAD).
Fig. 9 shows the Sequence Diagram ConfigureTransfer, where a generic (i.e., pre-mapping) CPU configures the DMA transfer via a set of parameters and the dedicated message TransferRequest(). Fig. 10 depicts the Activity Diagram TransferCycleAD. This model describes the data being iteratively transferred by the DMA controller (Sequence Diagram TransferCycleSD), that upon termination sets a flag to true (Sequence Diagram EnableFlag).
Fig. 11 shows the Activity Diagram PollingCycleAD, where Sequence Diagram PollingCycleSD models the iterative polling of the DMA controller. This scenario is depicted in Fig. 12 where each 10 microseconds the CPU polls the state of the data transfer via message PollingRequest() to the DMA controller until the flag `transferTerminated` is set to true (in diagram EnableFlag of Fig. 10).

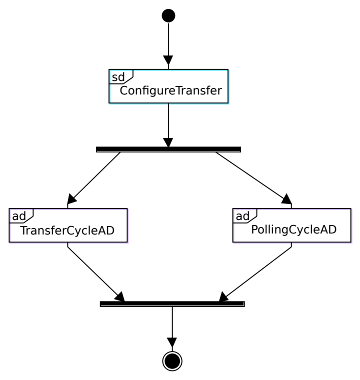


Fig. 8. The main Activity Diagram for a Communication Pattern modeling a DMA data transfer where polling is used to notify the transfer termination.

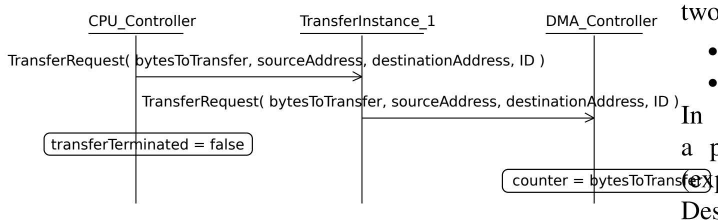

CPU_Controller
TransferInstance_1
DMA_Controller
TransferRequest(bytesToTransfer, sourceAddress, destinationAddress, ID)
TransferRequest(bytesToTransfer, sourceAddress, destinationAddress, ID)
transferTerminated = false
counter = bytesToTransfer

Fig. 9. The Sequence Diagram ConfigureTransfer of Fig. 8.

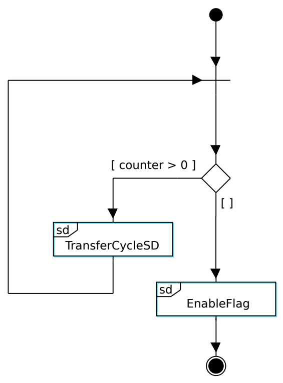


Fig. 10. The Activity Diagram TransferCycleAD of Fig. 8.

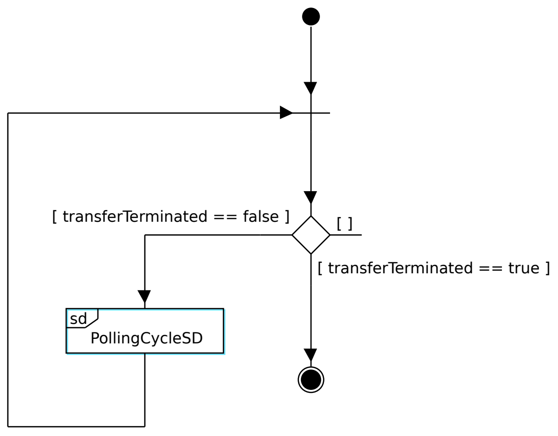

[transferTerminated == false]
[ ]
[transferTerminated == true]
sd
PollingCycleSD

Fig. 11. The Activity Diagram PollingCycleAD of Fig. 8.

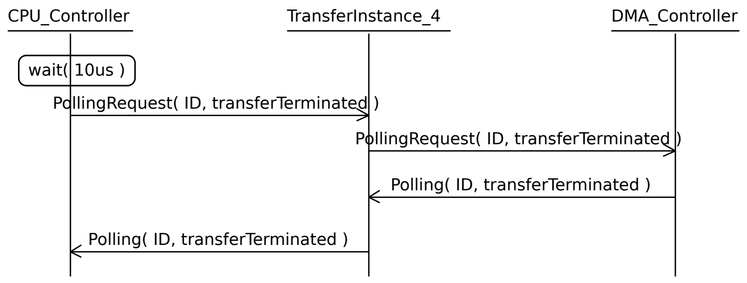

CPU_Controller
TransferInstance_4
DMA_Controller
wait(10us)
PollingRequest(ID, transferTerminated)
PollingRequest(ID, transferTerminated)
Polling(ID, transferTerminated)
Polling(ID, transferTerminated)

Fig. 12. The Sequence Diagram PollingCycleSD describing the message exchanges of the polling cycle of Fig. 11.

IV. Related Work

Our research works aim to fill the gap that exists between two distinct research axes:

- **hardware/software co-design**;
- **communication infrastructure design**;

In the field of **hardware/software co-design**, there exists a plethora of academic and commercial frameworks that (explicitly or implicitly) provide support for Y-chart-based Design Space Exploration (DSE) of embedded systems [16]-[30]. To the best of our knowledge, our work is the first to explicitly address the communication mismatch in the Y-chart approach. In the abovementioned works communications are still designed both in the application (e.g., dependencies between computations) and in the architecture (e.g., communication resources and services). Works from this domain, typically do not consider or neglect the communication infrastructure as explicitly as we do in the Ψ-chart approach, by means of our dedicated Communication Patterns.
More recently, contributions that target Networks-on-Chip ([31]-[34]), approach the **communication infrastructure design** with a more global perspective, taking into account the application design as well as DSE of the whole system (application-communications-architecture). The proposed solutions, nevertheless, are still strongly focused on modeling and evaluating communications rather than computations. With respect to the approach taken by the Ψ-chart, the communication infrastructure modeling and mapping are still dependent on the application mapping.

In the context of UML, the MARTE [35] profile shares many commonalities with our approach. In order to separate the modeling communications from a pair application-architecture, MARTE proposes Behavior Scenarios and Steps (Communication Steps). In contrast to our Communication Patterns, these assets are primarily designed for performance and timing analysis, rather than DSE. They intrinsically lack a separation between control aspects and message exchanges as we proposed in the Activity and Sequence diagrams of our Communication Patterns. Additionally, these modeling assets are not defined within the frame of systematic methodology for DSE.

V. Conclusions

In this paper we have described our proposed demonstration of the Ψ-chart, a novel design approach that combines hardware/software co-design with communication infrastructure design. In the Ψ-chart, dedicated models, called Communication Patterns, are used to describe the behavior of data transfers (communications), independently of the system's functionality (application) and resources (architecture). The contribution of our research works in terms of design quality, portability and models re-use will be demonstrated by means of a case study related to the design of the physical layer of the Zigbee communication protocol (IEEE 802.15.4) on a multi-processor heterogeneous platform with shared memory. The purpose of this case study is to show the complete design tool-chain, from models to the automatic generation of the emulation code. The output samples produced by the emulation code will be compared to a real-implementation of the Zigbee transmitter on a FPGA-based prototyping board, in order to prove the correctness of our CAD approach.

Acknowledgements

The research work leading to this paper has been supported by the French FUI project NETCOM under grant agreement n F1405046 U.

References

[1] D. C. Schmidt, *Model-Driven Engineering*, in IEEE Computer, vol. 39, no. 2, pp. 25-31, 2006.

[2] Kienhuis, B., Deprettere, E.F., van der Wolf, P., Vissers, K., "A Methodology to Design Programmable Embedded Systems - The Y-chart Approach", *Embedded Processor Design Challenges: Systems, Architectures, Modeling, and Simulation - SAMOS*, pp. 18-37, 2002.

[3] A. Enrici et al., "A UML Model-Driven Approach to Efficiently Allocate Complex Communication Schemes", *MODELS'14*, pp. 370-385, 2014.

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

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

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

[7] 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

[8] UPPAAL: `http://www.uppaal.org/`

[9] Construction and Analysis of Distributed Processes (CADP): `http://cadp.inria.fr/`

[10] Zedboard: `http://www.zedboard.org/`

[11] MutekH: `http://www.mutekh.org`

[12] A. Enrici, L. Apvrille, R. Pacalet, "TTool/DiplodocusDF: a UML Environment for Hardware/Software Co-Design of Data-Dominated Systems-on-Chip", Demonstration at DATE'2014 University Booth, Dresden, Germany, 2014.

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

[14] IEEE 802.15 Wireless Personal Area Networks (WPAN) Task Group 4 (TG4): `http://www.ieee802.org/15/pub/TG4.html`

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

[16] Daedalus Design Flow, `http://daedalus.liacs.nl/`

[17] Balarin, F., Watanabe, Y., Hsieh, H., Lavagno, L., Passerone, C., Sangiovanni-Vincentelli, A: Metropolis: An Integrated Electronic Design Environment. In: IEEE Computer 36, 4, 45-52, (2003).

[18] The Ptolemy Project: `http://ptolemy.eecs.berkeley.edu`

[19] Mohanty, S., Prasanna, V.K.: Rapid system-level performance evaluation and optimization for application mapping onto SoC architectures. In: IEEE International Conference ASIC/SOC, pp. 160-167, (2002)

[20] Thiele, L., Bacivarov, I., Haid, W., Huang, K.: Mapping Applications to Tiled Multiprocessor Embedded Systems. In: Proc. 7th Intl Conference on Application of Concurrency to System Design (ACSD 2007). IEEE Computer Society, pp. 29-40, (2007)

[21] Theelen, B., Florescu, O., Geilen, M., Huang, J., Putten, P. v., Voeten, J.: Software/Hardware Engineering with the Parallel Object-Oriented Specification Language. In: Proceedings of MEMOCODE. IEEE, pp. 139-148, (2007)

[22] https://www.symtavision.com

[23] Soonhoi, H., Sungchan, K., Choonseung, L., Youngmin, Y., Seongnam, K., Young-Pyo, J.: PeaCE: A Hardware-software Codesign Environment for Multimedia Embedded Systems. In: ACM Transactions on Design Automation of Electronic Systems, vol. 12, no. 3, pp. 24:1-24:25, (2008)

[24] Kwon, S., Kim, Y., Jeun, W.C., Ha, S., Paek, Y.: A retargetable parallel-programming framework for MPSoC. In: ACM Transactions on Design Automation of Electronic Systems, vol. 13, no. 3, pp. 39:1-39:18, (2008)

[25] Keinert, J., Streubühorbar, M., Schlichter, T., Falk, J., Gladigau, J., Haubelt, C., Teich, J., Meredith, M.: SystemCoDesigner - an automatic ESL synthesis approach by design space exploration and behavioral synthesis for streaming applications. In: ACM Transactions on Design Automation of Electronic Systems, vol. 14, no. 1, (2009)

[26] Kangas, T., Kukkala, P., Orsila, H., Salminen, E., Hännikäinen, M., Hämäläinen, T.D., Riihimäki, J., Kuusilinna, K.: UML-Based Multiprocessor SoC Design Framework. In: ACM Transactions on Embedded Computing Systems, vol. 5, no. 2, pp. 281-320, (2006)

[27] Abdoulaye, G., Le Beux, S., Piel, E., Ben Atitallah, R., Etien, A., Marquet, P., Dekeyser, J.L.: A Model Driven Design Framework for Massively Parallel Embedded Systems. In: ACM Transactions on Embedded Computing Systems, vol. 10, no. 4, pp. 39:1-39-36, (2011)

[28] Intel CoFluent: `http://www.intel.com/`

[29] MLDesign Technologies, MLDesigner: `http://www.mldesigner.com/`

[30] Posadas, H., Penil, P., Nicolas, A., Villar, E.: System synthesis from UML/MARTE models: The PHARAON approach. In: Electronic System Level Synthesis Conference (ESLsyn), pp. 1-8, (2013)

[31] Marculescu, R., Bogdan, P.: The Chip Is the Network: Toward a Science of Network-on-Chip Design. In: Foundations and Trends in Electronic Design Automation, vol. 2, no. 4, pp. 371-461, (2009)

[32] Popovici, K., Jerraya, A.: Flexible and abstract communication and interconnect modeling for MPSoC. In: Asia and South Pacific Design Automation Conference (ASP-DAC, pp. 143-148, (2009)

[33] Valinataj, M., Mohammadi, S., Plosila, J., Liljeberg, P., Tenhunen, H.: A reconfigurable and adaptive routing method for fault-tolerant mesh-based networks-on-chip. In: International Journal of Electronics and Communications, vol. 65, no. 7, pp. 630-640, (2011)

[34] Hollis, S.J., Jackson, C., Bogdan, P., Marculescu, R.: Exploiting Emergence in On-Chip Interconnects. In: IEEE Transactions on Computers, vol. 63, no. 3, pp. 570-582, (2014)

[35] The UML Profile for MARTE: Modeling and Analysis of Real-Time and Embedded Systems, `http://www.omgmarte.org/`