High-Level soc modeling and performance estimation applied to a multi-core implementation of LTE enodeb physical layer

Chafic Jaber

ParisTech
INSTITUT DES SCIENCES ET TECHNOLOGIES
PARIS INSTITUTE OF TECHNOLOGY

TELECOM ParisTech

2012-ENST-00xx

EDITE - ED 130

Doctorat ParisTech

T H È S E

pour obtenir le grade de docteur délivré par

TELECOM ParisTech

Spécialité « Electronique et Communications »

présentée et soutenue publiquement par

Chafic JABER

le 27 Septembre 2011

Titre

Modélisation de haut niveau d'abstraction de systèmes intégrés et estimation de performances. Application à une implémentation multi-processeurs de la couche physique d'une station de base LTE

Directeur de thèse : **Renaud PACALET**
Co-encadrement de la thèse : **Ludovic APVRILLE, Amer BAGHDADI**

Jury

M. Michel AUGUIN, Direteur de recherche, LEAT	Président
M. Robert DE SIMONE, Directeur de recherche, AOSTE, INRIA	Rapporteur
M. Bertrand GRANADO, Professeur des Universités, ETIS, INRIA	Rapporteur
M. Martin BEUTTNER, Manager, Freescale Semiconductors	Encadrant

TELECOM ParisTech
école de l'Institut Télécom - membre de ParisTech

T H È S E

Abstract

The impressive technical and technological advances in both fields of semi-conductors and software engineering enabled modern System-on-Chip *"SoC"* to host complex and interdependent applications. These advances are coupled with higher systems complexity and heterogeneity. Thus, forcing designers to re-evaluate their design methodologies and to raise the level of abstraction to the system level targeting the co-design of the entire SoC rather than just individual components.

The objective of this Thesis work is to provide the system designer with means (on the methodology and tools levels) to estimate system's performances and evaluate rapidly and very early the design decisions in the SoC design flow.

Our work provides contributions in two main aspects: (1) **On the Conceptual Level**, we defined (using the UML meta-modeling concepts) modeling concepts to estimate shared resources impact on system's overall performances, by introducing the *"virtual node"* concept for scheduling and shared resources access control. Furthermore, we introduced the *"Communication Pattern"* concept for modeling the interaction between architecture elements to ensure the orthogonalization of computation and communication concerns. (2) **On the Simulation Level**: A SystemC simulator was written to simulate the UML models. Simulation is done at a high level of abstraction and runs faster than real time execution.

The usability and capabilities of our approach are shown with an industrial use case. We modeled a Freescale multi-core DSP platform for LTE base stations (LTE stands for Long Term Evolution is the 4G standard for cellular networks). The comparison of modeling results with the real implementation proved the accuracy of our approach.

Key themes: System level design and modeling, UML for embedded systems, Resources management and sharing, Communication modeling, Performance estimation, Telecommunication systems

Abstract

Les impressionnantes avancées techniques et technologiques dans les deux domaines des semiconducteurs et de l'ingénieurie logicielle ont permis aux Système sur puces (System-on-Chip *"SoC"*) d'intégrer des applications complexes et interdépendantes. Ces progrès vont de pair avec la complexité accrue des systèmes et de leur hétérogénéité. Ainsi, les concepteurs ont été forcé à réévaluer leurs méthodes de conception et d'élever le niveau d'abstraction au niveau système en ciblant la conception de l'ensemble du SoC plutôt que des composants individuels.

L'objectif de ce travail de thèse est de fournir aux concepteurs systèmes les moyens nécessaires (au niveau méthodologique et au niveau outils) pour estimer les performances du système et évaluer rapidement les décisions de conception, idéalement trés tôt dans le flot de conception.

Notre contribution portera sur deux aspects principaux: (1) **L'aspect conceptuel**: où nous avons défini (en utilisant les concepts de méta-modélisation UML) des concepts de modélisation permettant d'étudier l'effet de la gestion et du partage des ressources sur les performances globales du système (les *"noeuds virtuels"*) . En outre, nous avons introduit le concept de *"Patron de communication"* pour la modélisation de l'interaction entre les éléments d'architecture afin d'assurer l'orthogonalisation des concepts de l'exécution et de la communication. (2)**L'aspect simulation**: Un simulateur en SystemC a été développé pour simuler les modèles UML proposés. La simulation est faite à haut niveau d'abstraction et elle est plus rapide que l'exécution en temps réel.

L'approche proposée a été appliquée pour la modélisation de la couche physique du protocole de télécommunications mobile de 4ème génération (LTE, Long Term Evolution) sur un DSP muli-core produit par Freescale. Les résultats ont été validés en les comparant avec l'implémentation réelle.

Thèmes clés: Modélisation et conception au niveau système, UML pour les systèmes embarqués, la gestion et le partage des ressources, la modélisation de la communication, l'estimation des performances, les systèmes de télécommunication

Acknowledgements

Over the last years that I have been working towards my PhD degree, I had the opportunity to meet and co-operate with many bright people. I am very indebted to these people, without their support and encouragement I would not be able to make my accomplishments come true. their enthusiasm was overwhelming. It is difficult to overstate my gratitude to my supervisors Renaud PACALET, Ludovic Apvrille and Amer Baghdadi. I would like to thank them for guiding and inspiring me and for showing me what research is about. Thank you for being so open-minded, for your friendship, your constant motivation and for sharing valuable technical insights for my PhD.

I would like to express my special gratitude to Robert de Simone and bertrand Granado, for showing keen interest in my work and for taking their valuable time to evaluate my work and providing their feedback, remarks and possible enhancements. Many thanks to Michel Auguin for presiding the jury.

I am also thankful to Martin Beuttner from Freescale, for giving me a great opportunity to integrate his team. This project would have been very difficult without his engagement. A special thank I need to give to the members of his group: Jean-Paul, Peirre, Vincent, Samuel and Arnaud, who since my arrival provided me with support, details and critical review of my work.

This thesis benefited from continuous support of Andreas Kanstein and Christopher Yasko, who even after leaving Freescale, continued to help me. I also would like to thank ”les muchacho(a)s” at LabSoC: Jair, Hocine, Feriel, Daniel and Gabriel who contributed significantly in making so much nice the PhD adventure. My most warm thanks go for my ”French” friends: Omar, Alexandre, Mohamad, Sami, Maha, Céline, Ihab, Aref, Bachar, Antoinette, Siwar, Chamesddine, Mustapha, Kawthar, Hadi, Sébastien, Carina; who were alawys by my side. A special thank goes to my ”french-familly” Roseline and Bruno Vidal for receiving me in their house and introducing me to ”la France profonde”.

Lastly, I would like to thank my family for all their love, encouragement and support. For my parents (Khoder and Nawal) who believed in me and

supported me in all my pursuits, thank you for giving me the opportunity to be all that I am capable of being. For my brothers: Ali, Nizar and Bassel and my sister Lara, for my grand-mother Hayyat (Oum Ali) and my aunt Rabha, Thank you for your caring and support, you are my family wherever I go.

Chafic JABER
Paris, September 2011

To my familly
To all who are dear to me

Contents

List of Figures

Chapter 1

Introduction

1.1 Thesis Context

The development and evolution of modern System-on-Chip *"SoC"* can be characterized by its main target: miniaturization. Highly advanced features are associated with targets in terms of performance, energy consumption, security, viability, and many other. All of these features are mainly enabled by the impressive technical and technological advances in both fields of semi-conductors and software engineering.

On the other side, the development cycle becomes shorter to challenge the competition. Furthermore, to differentiate their product, industrials are integrating more and more functionalities and hence increasing the design complexity. For instance, a modern mobile phone, in addition to the telecommunication standard, integrates a camera, a music player, a GPS, Internet browsing tools and many other applications.

System's higher complexity [40], [53] forced designer to re-evaluate their design methodologies. Initially, design was performed at transistor level, and then it evolved to the logic gate level and later to register transfer level (RTL). Tools developed at the RTL level enabled designers to verify the system's behavior. However, the increasingly growing complexity and heterogeneity of electronics embedded system push designer to raise the level of design abstraction to the system level that target the co-design of the entire SoC, rather than just individual components.

In this context, many methodologies and approaches [28] are targeting system level design. These methodologies provide designers with means to model the system and then analyze and optimize it. System analysis could target various domains such as functional verification, performance estimation, and power estimation. In addition, higher levels of abstractions enable fast system evaluations and performance analysis. However, the first challenge remains to choose the appropriate level of abstraction that preserves the accuracy of results.

As most recent Systems-on-Chip (SoC) are meant to host complex and interdepen-

dent applications at low cost (area, power) to the end user, the number of resources must be minimized while increasing their utilization by sharing them between multiple applications. Such shared resources have a strong impact on SoC performance because of the contention they typically induce. Indeed, end-to-end performance of a given application is the sum of the time needed to execute that application with the total contention delay of all involved shared resources. Shared resources contentions, communication delays and many other factors play a role in the overall system performances. Thus, identifying the main factors that influence the system's performance and how to model them constitute another challenge in performance analysis.

Once the performance metrics (such as latency, throughput, resource utilization, etc.) identified and modeled, the designer can perform multiple system's evaluation (through simulation or other analysis mechanisms) to explore various combinations of architecture (number of computation units, communication topology, etc.) and application (number of tasks, parallelism between tasks, etc.). Ideally, s/he wants to verify if the system's performances satisfy the design requirement. Thus, rapidly extracting the performance metric needed to perform the design analysis is a primordial step in the system design. The system designer, can then choose among different possible solution the one that is most adapted. Hence, a third challenge that needs to be considered is how to analyze the extracted performance metrics of the modeled system to verify its satisfaction to design requirements.

Thesis Objective. *The objective of this Thesis work is to provide the system designer with means (on the methodology and tools levels) to estimate system's performances and evaluate rapidly and very early the design decisions in the SoC design flow. Specification of the abstraction level and methods for the analysis of abstract models is the key for a successful system level design methodology. Modeling at a high level of abstraction eases the system-level design decisions making process, as the designer does not need to have detailed expertise on lower level software, hardware, and design tools.*

This thesis work is done in the framework of the DIPLODOCUS methodology developed at Telecom ParisTech. DIPLODOCUS defines a UML profile [76] targeting design space exploration at a high level of abstraction. The following section describes briefly the contributions of our thesis.

1.2 Thesis Contributions

Towards the above mentioned objective, following are the proposed contributions of this thesis, classified into three levels:

- **Contribution on the Conceptual Level**: Definition (using the UML meta-

modeling concepts) of modeling concepts to take in consideration:

  - the **shared resources** impact on system's overall performances, by defining the *"virtual node"* concept for scheduling and shared resources access control.
  - the orthogonalization of computation and communication concerns by defining the *Communication Pattern* concept for modeling the interaction between architecture elements.

- **Contribution on the Simulation Level**: A SystemC simulator was written to simulate the UML models. Simulation is done at a high level of abstraction and runs faster than real time execution.

- **Contribution on the Experiment Level**: The proposed approach was applied on an industrial case study. A muli-core Freescale DSP platform for LTE base stations was modeled and validated by comparison to the real platform to estimate its accuracy and speed

1.3 Thesis Layout

This thesis report is divided into six chapters as follows:

- **Chapter 2** provides an overview of the System-on-Chip design domain and the increasing design complexity. It describes as well the typical SoC design flow, that starts from the client specification and through various refinements results in a functional SoC product. The flow covers both application and architecture parallel development and their integration and validation.

- **Chapter 3** presents the system level design domain concepts and objectives and provides a survey of existing methodology. In a second part, this chapter describes the DIPLODOCUS methodology on which our research work is based and compares it and the extensions added during this thesis with the state of the art.

- **Chapter 4** illustrates the DIPLODOCUS modeling of the application and the architecture and presents the *Communication patterns* for communication modeling. It provides the UML meta-models for the different modeling concepts.

- **Chapter 5** presents the mapping process proposed to execute the application on top of the architecture. It defines the concept of *virtual node* for shared resources access control and how it could be used to develop useful access policies for SoCs design. The mapping includes as well an allocation step that defines how

the architecture resources (computation, communication and storage resources) are allocated to application artifacts. An example is used to illustrate the different steps of the mapping process.

- **Chapter 6** describes the simulation environment developed and used to simulate the DIPLODOCUS models and to extract performance metrics that will help designers to make early design decisions.

- In **Chapter 7** the methodology developed is used to model an industrial system, the LTE uplink physical layer that will execute on a multi-core platform. The simulation results are compared with the real implementation.

Part I

PhD Domain Overview and State of the Art

Chapter 2

System-on-Chip Design and Application Domains

This PhD work is done with the aim to develop a high level modeling methodology for System-on-Chip (SoC) performance estimation. This chapter provides a global view of the domain of the research work. It shows the complexity of the design of a SoC, by using the example of the LTE (Long Term Evolution) standard for cellular networks and an example of a real chipset designed for its implementation. Then, the second part discuss the SoC design flow and steps and how this PhD work fits into it.

2.1 Introduction

Systems-on-Chips are omnipresent in the modern daily life and environment, like Telecommunication systems and devices, automotive, avionics, multimedia devices, etc. This evolution comes in pair with an evolution in the embedded hardware. In fact, the computation power is growing rapidly following the Moore's law. More sophisticated and complex applications are introducing new functionalities and trying to cover new markets, while increasing the complexity of the embedded hardware. Hence, The design complexity of modern embedded systems is exploding due to the market demand for more, increasingly complex features, and shorter time-to-market.

The objective of this chapter is two-fold. The first objective is to illustrate the complexity of modern SoCs through an example on a wireless telecommunication standard (LTE), and its implementation in an industrial context. The second objective is to present the conventional SoC Design flow and situate the topic of this PhD work.

This chapter is organized as follow. Section 2.2 presents the LTE wireless communication protocol, with special focus on the throughput increase introduced in this protocol and the inherent complexity needed to support this feature. Then section 2.3 takes the example of a SoC that supports the LTE standard and shows its main design

characteristics. The complexity of design of such systems, and the design challenges that it imposes are presented in section 2.4. The section 2.5 describes the main steps of the conventional SoC design flow that, through multiple iterations and refinement, permits the creation of a SoC from the initial client specifications.

2.2 LTE Standard for Mobile Communication: Higher Complexity for better quality of services

In the last two decades, the world wide cellular network utilization has evolved from voice oriented to data oriented networks. In fact, Figure 2.1 compares the rapid growth in wireless data traffic compared to voice traffic across multiple operators in the United States [83]. This evolution is basically a structural one that affected the network infrastructure and came with new standards for supporting more quality of services.

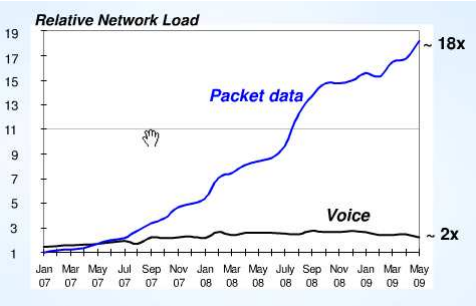

Figure 2.1: Evolution of data versus voice transfer in US cellular networks

The mobile communication systems have evolved from 1st generation, which are analog cellular systems and limited to voice transfer, through 2nd generation-known as digital narrowband that introduced data transfer to cellular networks, to 3rd generation (3G), with higher bandwidth radio interface and throughput. The latest commercialized wireless network technologies, called High Speed Packet Access (HSPA)[32], is currently deployed around the world. The evolution of 3G networks is under implementation and will be ready to be deployed in the following years. The standardization work of the Long Term Evolution (LTE) [33], started from 2004, is close to completion. LTE provides higher data rate compared to previous standards, with the peak of more than 300 Megabits per second in downlink and 50 Megabits per second in uplink. Besides, it provides lower user costs, better spectrum efficiency, smaller latency and many other advanced features. Figure 2.2 shows the evolution, over the last decade, of throughput of the cellular network standards [10]. The throughput is increasing from less than one Mbps in 2003 to more than 300 Mbps as expected by 2012. This dramatic

growth of throughput is mainly driven by the increasing demand for Internet connectivity on mobile devices. Mobile Users are now expecting to have the same quality of service (high throughput, stable connection, low latency, etc.) while accessing to their requested multimedia contents, from anywhere and even when they are on the move.

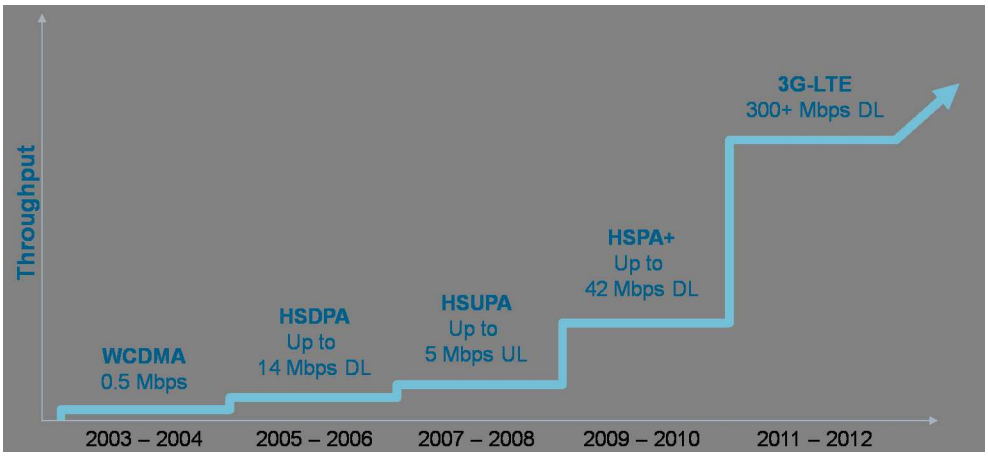

Figure 2.2: Evolution of cellular network's throughput and protocols

The evolution of cellular networks is becoming a cyclic process: Higher throughput enables sophisticated mobile applications, and the later are requesting even more higher throughput. The user's experience and satisfaction depends on the quality of services provided by the operator and by the quality of the mobile device s/he is using. Hence the network infrastructure and the design of mobile devices become more complex . So, The higher throughput comes with an increasing complexity of the devices (software and hardware) that will implement the standards. In order to achieve the required performances, complex heterogeneous architectures are used, including many processors, many hardware accelerated parts, specific communication infrastructure and various input/output interfaces.

Another factor of complexity is that LTE itself was meant as an evolution and not a revolution of the existing standards. It will co-exist and inter-work with the 3G system. Thus, the LTE network base stations will be even more complex as they will support as well the previous standards. In fact, multi-standard platforms are currently developed by the principal platform companies (like Alcatel [12], Ericson [31], Huawei [44] and others).

This increase in complexity is common between the different embedded application domains, and the SoC implementation of all these applications share the following trends:

1. New features and value added services, lead to the increase of processing and communication requirements.

2. The standards are introduced more rapidly and become more sophisticated. This calls for high flexibility of the SoC implementation to meet the resulting time-in-market severe constraints.

3. The reuse of developed components is increased, to shorten time to market and reduce production cost.

4. For mobile devices and sensors, reduced size and power efficiency become prevailing cost factor and product differentiation.

2.3 A SoC example for LTE implementation

Industrial solutions for the embedded application reflect the application complexity. Furthermore, **heterogeneous** multi-core SoC platforms are generally believed to meet the above mentioned conflicting performance, flexibility and energy efficiency requirements of demanding embedded applications. The heterogeneity of SoC implementations is driven by the heterogeneity of the embedded applications, where each part of the application has an inherent suitable implementation technology. Figure 2.3 depicts the architecture of a DSP SoC, MSC8156 [73], produced by Freescale Semiconductor [72], that targets among others the wireless broadband domain. This DSP is currently used in industrial implementations of the base stations of the LTE standard developped by companies like Alcatel. This SoC has been used to test and validate the methodology developed in this thesis. More details on its modeling and performance estimation of the implementation of the LTE physical layer are provided in Chapter 7.

The MSC8156 DSP delivers a high level of performance and integration. It combines together six DSP cores, each running at up to 1 GHz, a hardware accelerator (maple-B) that supports hardware acceleration for Turbo [55] and Viterbi [81] channel decoding, in addition to DFT/iDFT and FFT/iFFT algorithms (these algorithms require high computation performances). It includes as well a high-performance internal RISC-based (QUICC Engine) subsystem that supports multiple networking protocols to guarantee reliable data transport over packet networks.

The MSC8156 embeds an on-chip memory (M3) of 1 MByte, and two interfaces for off-chip DDR memories. In addition, each DSP core has its own internal L1 and L2 caches. It supports a variety of advanced, high-speed interface types, including two "RapidIO" interfaces, two Gigabit Ethernet interfaces for network communications, a PCI Express controller, two DDR controllers for high-speed industry standard memory interface. The set of all these integrated components are connected through a Chip-Level Arbitration and Switching Fabric (the CLASS), that ensures non-blocking, fully pipelined and low latency communication scenarios. The system is doted as well by two DMA blocks to transfer the large quantity of processed data. To have an idea on the data transfer performed and the complexity of the data processing, recall that this

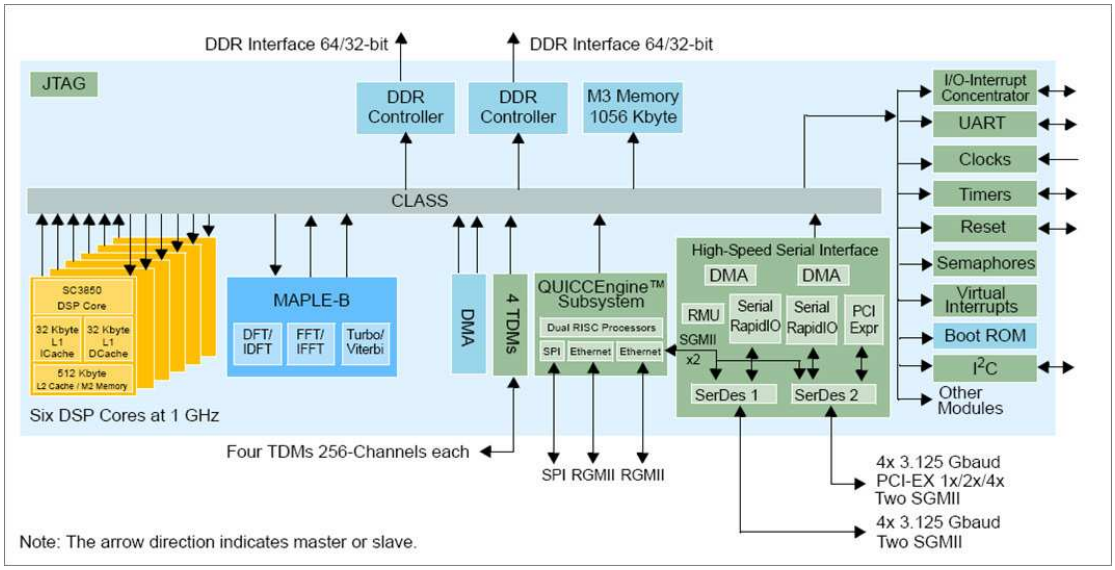

Figure 2.3: The Freescale DSP based SoC for LTE physical layer

system is used to implement a part of the LTE standard, and it guarantees throughput of more than 100 Mbps.

2.4 System on Chip Design Complexity

A System on Chip (SoC), like the one presented in the previous section, integrates software and hardware jointly and is specifically designed to provide given functionalities to the surrounding environment [40], [53]. The system must be able to continuously react to stimuli in the desired way, i.e., within bounds on timing, power consumption, and cost. As introduced in the previous sections, such systems are highly complex and heterogeneous. In fact, the complexity of system design is dramatically increasing in three dimensions:

1. Heterogeneity and complexity of the hardware architecture: New design technologies and higher integration density result in increasing the computational power of modern SoCs which enables sophisticated functions to be embedded in ever-smaller devices. A modern SoC may include general purpose processors (GPP), Application-Specific Instruction-set Processors (ASIP), different communication topologies, complex memory hierarchy, different Input/Output devices ... The Freescale DSP MSC8156 presented in section 2.3 is a good example of such architecture.

2. Heterogeneity and complexity of the embedded software: As the computational

power of SoCs grows, more advanced functionalities are introduced. Software is taking the lion's share of the implementation budgets and cost. For instance, in a cell phone, more than one million lines of code are the standard today [69]. In contrast with traditional software systems where the abstraction process leaves out all the physical aspects of the systems as only the functional aspects of the code matter, the embedded software is at least loosely coupled to the hardware architecture which limits the code reuse when the system specifications evolve. Furthermore, heterogeneous applications share the same hardware architecture: a cell phone device must be able to handle simultaneous voice and data calls, while also handling complex imaging tasks like image or video capture. This heterogeneity increases the complexity of resource sharing and design optimization.

3. Integration complexity: During the integration phase, software and hardware components are integrated to create the SoC. As these components are usually developed by different teams and sometimes in different countries (and/or by different companies), this phase is of extreme complexity [41]. Verification and re-engineering work done during this phase is time consuming and expensive.

2.5 System on Chip design flow

A System-on-Chip design is an iterative process, that aims the implementation of a product based on the client specifications. A System-on-Chip design flow is a succession of refinements and optimizations steps to accomplish the system design. Each step of the flow consists of modeling and/or implementation, verification and integration of hardware and software. The ideal design starts with a high level description of the functionality and the architecture. The designer is expecting for the following from SoC design flow:

1. Reuse of the existing code and models developed for previous products to fasten the design cycle. One should note that usually new products are an evolution of existing product and rarely a revolution. Hence, design reuse can lead for considerable cost reducing factor.

2. Identify design decisions early in the design flow. Late design changes are expensive and time consuming, thus finding the bottlenecks and estimating the system's performances at early stages reduce the design's cost and increase the productivity.

3. Ensure a functioning product, that satisfies the client requirements. Thus, the adopted verification and validation process should be correct and accurate in order to lead to a correct product.

The design specification at each design step must cover the specification of the functionality and the architecture of the system. Currently, there is no standard language or format for design specification. In addition to specification, each level of the SoC design flow should define verification and validation methodologies to verify the satisfaction of the design with the requirements.

Specification Languages

Many programming languages and modeling paradigms are used to specify the functionality of a SoC. While embedded software is usually written in C, the functionality could be described with a different paradigm: it could be directly written in C or specified using synchronous languages such as Esterel [21], Lustre/SCADE [39], Signal [19], from which some tools can automatically generate C code and formally verify the system. More recently UML is proposed to enable the specification of very complex applications by providing a large set of language constructs. In addition UML has the advantage of being implementation independent [58][56].

For hardware design, system level design relates to any level of abstraction that is above the register-transfer-level (RTL). Transaction-level modeling (TLM) [37], behavioral, algorithmic, and functional modeling are terms often used to indicate higher levels of abstraction in hardware design. Due to its popularity and efficiency, the C programming language and its derived languages are gaining in the market. Many approaches and tools propose synthesis of C code into RTL code; Catapult C (Mentor) [38], Handel-C [26] are well known examples of these approaches and tools. In addition new approaches are using UML for modeling of SoC hardware and software. Simulation code, formal verification specification or synthesis code may be generated afterwards from the high level UML models; Gaspard2 [18][56], Koski [47], and DIPLODOCUS methodology described and extended in this thesis [14][45] and many other approaches has adopted this approach.

Ideally a specification language spans many abstraction levels so that it can be used throughout the design process.

Verification and Validation

Verification and Validation is a phase of system development, that is performed at the different levels of abstraction and where software and hardware are analyzed to verify that they satisfy the desired properties. The most common techniques for design validation and verification are simulation and formal verification. While simulation permits the evaluation of large and complex systems, formal verification is a process for checking whether a system satisfies a given property under all possible inputs, and is applied to safety-critical subsystems to ensure correctness.

2.5.1 SoC Design flow steps

Figure 2.4 shows a typical SoC design flow. It starts from the client specification and through various refinements its result is a SoC product, ideally ready for the market. From the client specifications the experimented designers will gather design requirements and define a first draft of the system specification. Then, through three main steps, the design will evolve from the client specification to a system level model than through virtual prototyping a more thorough analysis is done to finally give the final product with the prototyping phase. In each step software and hardware are ideally developed/modeled in parallel and an integration step permits to evaluate the system design progress and satisfaction to design requirements.

2.5.2 System Level Design

Approaches to raise the level of abstraction in SoC design are called "System-Level Design Methodologies". Their objective is to help designers to take and validate design decisions at early stages of system design. They permit designers to model, simulate, explore, verify and refine a system design. Some frameworks further provide a design flow by integrating a set of refinements to transform a system-level model to an implementation. Ptolemy [23], Artemis [66], CoFluent Studio [29], Metropolis [16], Koski [47], Design Space Explorer (DSX) [57], Platform Architect (CoWare) [77], SoC Designer (ARM - Carbon Design Systems) [25] and many others are well known examples of system-level design frameworks and tools.

Most of these approaches adopt a clear separation between the application and the architecture modeling. Thus, a mapping phase is needed to bind both models and to define the execution of the application on the architecture. After Mapping, the system is evaluated to test if it meets to design requirements. This phase is called: Design Space Exploration. Its objective is to find an optimal model that fits the requirement. Design choices like how many cores are needed or how much on-chip and off-chip memories are required are ideally taken at this stage.

In figure 2.4, the system level design is the first step after the specification of the system based on the client specifications. Application and architecture are modeled first, then their mapping and the design space exploration phase. Based on the system validation and test during this exploration, the designer can modify her/his models.

2.5.3 Virtual prototyping

Virtual prototyping is the second main step in the process of product development. Its main objective is to validate a design before committing to making a physical prototype. In fact, the design process for SoC-based platforms suffers from productivity issues due to the effort needed for verifying and validating the system. System veri-

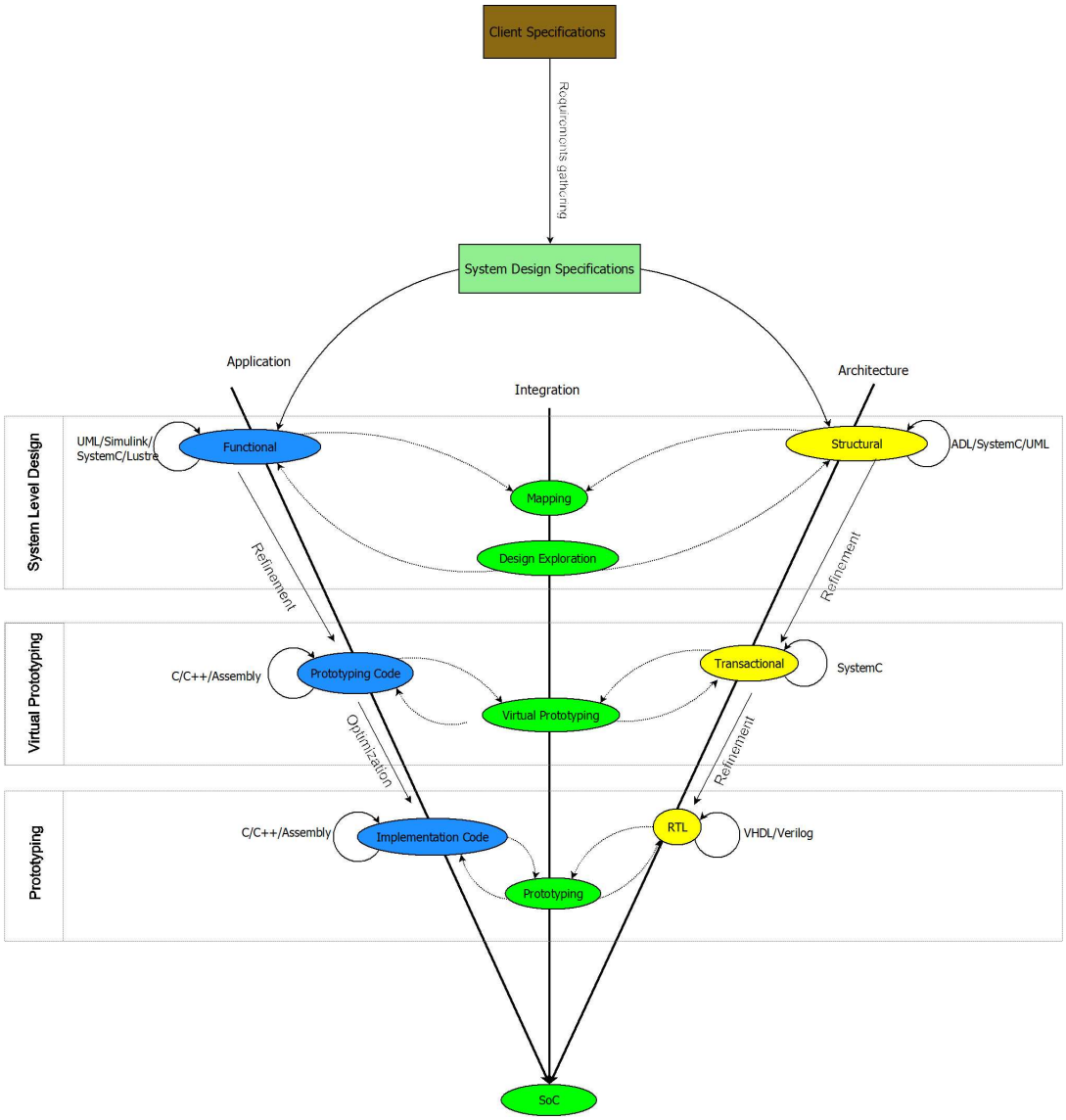

Figure 2.4: SoC Design Flow

fication is typically carried out at an intermediate to lower level of abstraction: prototype software implementations with transaction-level or register-transfer-level hardware models. It is very expensive and time consuming to create these prototype system models, and thus it is also expensive to revise system architecture decisions. Furthermore, because of higher system integration (and thus complexity), the design risk and the associated verification cost are higher. Rapid virtual prototyping of the system can verify the system in a real environment and identify potential implementation bottlenecks, which could not be easily identified in the previous step. A working virtual prototype can demonstrate to clients the feasibility and show possible technology evolutions, thus significantly shortening the time to market.

Transaction level model (TLM [37]) is one of the most well-known models that can serve as a virtual prototype. This step comes just after the system level design step. The code is more mature and the designer can make more accurate analysis. Even though, the simulation time is clearly higher than at system level design.

2.5.4 Prototyping

Prototyping corresponds to the integration of the optimized software during the previous design step on the final product hardware or on FPGA based platform that corresponds to actual hardware. Final verification are executed on the design and new validation tests prepare for the final step of the SoC design corresponding to the creation of hardware masks that will be the base of the SoC silicon production. At this step of the design flow, it is very expensive to make design changes. Hence, the vast majority of tools and approaches are trying to validate the system at earlier steps.

2.6 Summary

The approach to cope with the increasing design complexity of SoCs is to raise the level of abstraction, moving the development process from lines-of-code to coarser-grained architectural elements. Modeling accurately the functionality and the candidate system architectures at an early stage is becoming essential for a successful system design. In addition, higher levels of abstractions enable fast system evaluations and performance analysis. However, the challenge remains to choose the appropriate level of abstraction that preserves the accuracy of results.

The system Level Design is a promising domain, that is used widely nowadays. Many approaches are trying to take benefit from the system level to take early design decisions. This PhD work fits into the system level design domain and has as a main objective to estimate SoC's performance at early design stages. The next chapter will review the state of the art of system level design and presents the contributions of this thesis.

Chapter 3

System Level Design: Models, Methodologies and Trends

The previous chapter showed the design complexity of modern Systems on a Chip (SoC) and the design flow used for their design, validation and implementation. Design at system level proved to be a good solution to reduce complexity and early performance analysis based on abstract models has already been demonstrated to increase design efficiency [50; 67; 82]. System-level design frameworks are commonly based on models meant to describe functions to be implemented by a set of candidate hardware architectures.

The objective of this thesis is to develop a methodology for system level modeling and early performance analysis of SoCs. Our approach is based on the DIPLODOCUS UML profile. The main contributions of this thesis are a set of extensions for DIPLODOCUS to model shared resources and ensure orthogonalization of the communication and computation. A SystemC based simulator is developed as well to simulate and estimate models performances. After describing the concepts and objectives of system level design, this chapter will position the proposed approach (DIPLODOCUS plus the extensions) wrt to existing approaches.

3.1 Introduction

3.2 System Level Design: Concepts and Objectives

"System Level Design" refers to large set of design methodologies [36] that target the design of the entire SoC, rather than just individual components. Such methodologies provide designers with means to model the system (its functionality and architecture), and then analyze and optimize it. It is worth noting that the system level design is a large domain and various industrial and academic approaches exist to tackle different

aspects of design (functional verification, performance estimation, power estimation, etc.). However all these approaches share some basic concepts: all of them target modeling at a high level of abstraction, they adopt to some extent the separation (orthogonalization) of concerns to increase the modularity and re-usability and to reduce analysis time. Furthermore, all of them are equipped with a design space exploration process to analyze and to optimize the system level models

3.2.1 Modeling and abstraction

System's complexity and heterogeneity are influencing the development of system level design methodologies. As shown in section 2.4 of the previous chapter, the increasing complexity can be identified on the application level, the platform level, as well as on their integration. The ever increasing number of transistors on a die helps by increasing the computation power and memory space. However, more advanced methodologies on system level (application and platform) are required to better take advantage of the added silicon. The design shift from super-scalar mono-processors to multi-cores platforms (in the telecommunication, multimedia, networking and other application domains) is confirming this trend for enhancing the utilization of the platform. To cope with this increasing complexity a system level design approach proceeds by modeling and abstraction of the system in order to analyze it.

A **model** by definition, is a " representation of some phenomenon of the real world made in order to facilitate an understanding of its workings. A model is a simplified and generalized version of real events, from which the incidental detail, or 'noise' has been removed" [71].

An **Abstraction(s)** enables designers to understand (analyze) complex domains of concern, like programs, software systems, and their application domains, which contain a plethora of detail. It is a process of separating essentials from details, and focusing on the former while ignoring, "abstracting away", the latter. Creating abstract models help designer to better understand the system under design by reducing complexity. Abstraction can be seen as the process of hiding aspects of the design in order to increase the ability of the designer to only consider those which help to develop a design at that particular stage. Examples of abstraction could be a set of transistors being abstracted as logic gates, a set of bus transactions being reduced to an IP interface, or a set of processor instructions (add, divide ...) being represented by their computation complexity. For instance the DIPLODOCUS profile made a choice of data abstraction to focus more on the control behavior of the system and to identify the system execution to better estimate its performance.

Higher levels of abstraction allow designer to make early design decisions, as the changes performed at these levels affect dramatically the overall design. However, the designer has less insight into how precisely and accurately the changes brought about this change. Thus, a system level methodology is the first step in the design flow and

is always followed by more optimizations and refinements of systems models.

3.2.2 Separation of concerns

The abstraction done at the system level design allows for a designer to think of traditional software and hardware aspects of the design separately. The Algorithmic part is decoupled from the architectural elements which implement them. The **Y chart** introduced in [50; 51] is widely adopted by system level methodologies. It consists of modeling separately the application (what the system should do) and the architecture (how it can do it), and then integrating both during a mapping phase. For instance, a signal processing algorithm (FFT, Viterbi, ...) itself is well defined functionally, and some Matlab/Simulink models may exist to simulate this functionality. However, the architecture that will implement this algorithm may be DSP-based or a dedicated hardware block. In the mapping phase the architectural element (the DSP or the Hardware block) is allocated to the application. As the application and architecture concerns are separated, models are reusable and so a designer can easily evaluate candidate architectures using the same application model. This separation also permits the exploration of mapping of two different applications on a given architecture during first stages of projects.

Furthermore, Keutzer et al. [50] extended the separation of concerns principal to include the separation of computation and communication concerns. In other words, the "how" the system computes should be separate from the "how" system's components interact.

The following subsection will describe the *Design Space Exploration* used for analyzing and optimizing the system level models.

3.2.3 Design Space Exploration

"Design Space Exploration" (DSE) is the process of looking at a variety of designs and using the results of simulation, verification, or other analysis methods to make decisions regarding which design should be selected. These modifications that can be made to a design to optimize its performance, and to observe potential design issues that may have been overlooked during specification. The DSE process is done prior to the development of implementation prototypes. It relies on the system level of abstraction, where the models' modification cost is smaller than on a low level of abstraction. In fact, abstract models can be quickly changed in terms of structure, behavior, design parameters and components to test new system configurations. These higher-level abstractions enable a faster exploration of a larger design-space.

In general, the exploration process is a multi-objective one [48]. The list of supported objectives includes the SoC size, power consumption, execution speed, throughput and many others. After a first analysis step the designer can optimize the system

model to satisfy one (or more) objective. To do so, s/he can modify the application models (algorithmic decisions, task partitioning), the architecture models (number of computation resources, memory hierarchy, communication infrastructure), and the mapping decisions (tasks' memory map, execution allocation of the computation architecture to the tasks, the scheduling decisions and choices). The consequent complexity and size of the design space prevent an exhaustive search to find the optimal solution satisfying all of the objectives. Hence, establishing a priority of the design objectives is required, and the design exploration process tries to find one possible (not necessarily the optimal) solution for the exploration problem with the specified priorities.

In the system level design literature, two explorations methods are widely used: analytical and simulation-based methods. In general, analytical methods are used to prune the vast design space rapidly but with very coarse models while simulation-based methods explore in more details a part of the design space. Simulation models are well suited to identifying characteristics of a dynamic system that are hardly predictable. They can be used to explore the task scheduling (especially with interruption), communication scheduling and dynamic input models. Analytical methods fit well for application with limited dynamic behavior. Some approaches combine both methods to exploit their best properties. In both types of methods, a major aspect of the design space exploration process of a system level design methodology is the time required to define the design objective and the time required to evaluate (explore) the system model to validate its satisfaction to the objective.

3.3 System Level Specification Languages

In a system level design methodology, the choice of a suitable specification language is primordial. Ideally, the language should support different abstraction levels and capture the concurrency and timing of a SoC (the so called expressiveness of the language). Furthermore, it should be supported by easy-to-use tools (compilers, modeling tools, ...).

Various specification languages exists and often more than one language are used in a SoC system design process. For instance, the embedded software is mainly written in C and assembly, while VHDL[6] and Verilog[1] are largely used to describe the hardware specification at register transfer level (prototyping phase as described in the section 2.5 of the previous chapter). At the system level of abstraction the objective is more to first model the system (software and hardware), then in a next step to generate the code of it. The trend is to use the same language for the application and the architecture as the designer can easily master one language and take profit of already known languages (such as C/C++, UML ...). The following the next subsections will present some of the most used specification languages for software and hardware modeling.

3.3.1 C/C++ based design languages

C/C++ are popular for software development, and are familiar to an important part of designers and engineers. Thus, many initiatives want to take advantage of this popularity to introduce new system design languages with ideally a fast learning curve for new designers/engineers. In this scope, the two major initiatives are SystemC that is based on C++, and SpecC that is based on C. In both cases, the language is enriched with concurrency and timing constructs to better model SoCs.

SystemC is a language proposed by the Open SystemC Initiative (OSCI) [43], and had a large industrial and academic support. The language is standardized by the IEEE standard 1666-2005 [4]. SystemC is used to describe the hierarchical structure and the behavior of complex embedded systems. It can be used to describe the system at different levels of abstraction. Using SystemC, a system can be described at functional level, architectural level, and implementation level. SystemC is a C++ library that provides the constructs needed to express hardware timing, concurrency, and reactive behavior that are missing in standard C++. The OSCI simulation kernel (free download) is used to simulate the SystemC models. Being an extension of C++, SystemC profits from all the legacy compilers and tools developed for this language.

SpecC [34; 59] is a superset of the ANSI C programming language. It is developed at the University of California, Irvine, to address system level design. The SpecC language and the underlying methodology were created by Rainer Domer and Daniel Gajski. Similarly to SystemC, it enriches the C language with a system design library addressing the timing and concurrency aspects. SpecC defines different levels of abstractions to take in consideration the timing representation of the computation and the communication.

3.3.2 Synchronous Languages

Synchronous languages [20] target the design of reactive systems. These systems are interacting permanently with their environment. Their execution is a set of atomic "instantaneous" reactions to inputs of the environment. A synchronous model is deterministic, in contrast with the non-determinism of the classical sequential programming and concurrent formalism such as Petri-nets. Thus, Synchronous languages are used for development of safety-critical systems in avionics and industrial control. The synchrony assumption of the synchronous models makes them particularly interesting for hardware design. In fact, it is identical with the zero-delay model of electronic circuits. Therefore, most synchronous languages provide automatic translation/generation in VHDL. Esterel[21] and Lustre/SCADE [39] are well known synchronous languages.

Lustre/SCADE targets the dataflow programming. It is formally defined and continuously developed since 1984. A LUSTRE program is called a node. It is declared with two sets of parameters: inputs and outputs, and the body of the node defines

the functional relationship between inputs and outputs. Variables in Lustre are data streams. Calculations and data exchange between nodes are instantaneous. In 1993, Lustre became a part of the industrial environment SCADE developed nowadays by Esterel Technologies. One of Lustre/SCADE success stories is its adoption for modeling safety critical parts of the A380 Airbus plane (the world passengers largest air plane).

3.3.3 UML: Unified Modeling Language

The Unified Modeling Language (UML) [76], initially targeted complex software modeling and was for example used for modeling banking systems. UML 2.0 (the actual version of the UML standard) is a graphical specification language for object modeling that captures the abstract system specifications. The main feature of UML is its genericity, thus it is not limited to a specific domain. In fact the UML profiling mechanism permits the extension of UML for modeling a specific domain. UML can be considered as an emerging specification language for embedded systems, and many UML profiles (some of these profiles are described in section 3.5.2) target the embedded systems design specification. An important step for the UML adoption in embedded systems design was the standardization of the Marte UML profile targeting embedded and real time systems. Unfortunately, UML is still lacking full-interoperability between tools. This limits the exchange of models between different companies or teams and reduces the opportunity to build libraries of UML models as it is the case in C++, Matlab and VHDL

3.3.4 Matlab/Simulink

Matlab is a mathematical environment used for algorithm development. Control Theory and digital signal processing are two main domains where Matlab is used to model the underlined algorithms. These later can be simulated using the Simulink tool that provides flexible simulation capabilities and a wide range of tools for visualizing results. Matlab/Simulink is largely used in industry and a large components legacy is available. Thus, building new models is made simpler due to increased re-use. Both continues time and discrete time models can be simulated using Matlab. In addition, it is sometimes possible to generate C implementation of Matlab models. However, this code does not take into consideration complex architectures with complex communication and memories infrastructure. The Matlab simulation is mainly useful to study the modeled system at the algorithmic level.

3.3.5 Discussion

All the above presented specification languages target system level design. However, some of them are textual while the others are graphic, some target reactive safety critical systems while others target signal processing systems, some rely on a formal semantics while others are simulation based. A system level design **methodology** should be generic and ideally target different design domains. Thus it could relies on one or more specification languages. UML is generic and extensible (through the profiling mechanism) to target different domains. In fact, it is common nowadays, to find approaches [18; 47; 82] where the system's specification is done in UML and where the performance analysis step uses a model translation/transformation into SystemC or other simulation languages. This later approach is adopted for the work presented in this thesis. It is possible as well to generate different simulation/analysis codes from UML such as Matlab, and SCADE

3.4 Survey of some Existing System Level Design Methodologies

The System Level design domain is a large and promoting domain. Several Industrial and academic approaches are proposed and developed. Examples of such design frameworks and languages are Ptolemy II [23], Metropolis [17] , Koski [47] and MESH [22]. This section describes, briefly, some of these system level design methodologies. The objective is not to provide an exhaustive list but rather to introduce the relevant ones with this thesis work.

Metropolis [17] is based on a meta-model that supports multiple types of model of computation and that can be extended to support others. It specifies applications and architectures separately and the association (mapping) between them. The basic modeling elements in Metropolis are processes, ports, media, quantity manager and state media. Processes represent the system's functionality and communicate through ports. The ports are interfaced using media, and the quantity manager define the scheduling scenarios. The Metropolis environment provides techniques and tools for formal verification and for simulation of developed models.

Ptolemy II [23] is a meta-modeling framework, implemented in JAVA, for heterogeneous and hierarchical designs of embedded systems. It focuses on simulation and the interaction between different models of computation. It uses tokens as the underlying communication mechanism. Directors regulate how actors execute in the design and how tokens are used to communicate between them. This mechanism allows different models of computation to be constructed within Ptolemy. Hierarchical

composition is used to handle heterogeneity. Each level in a hierarchy has a director that organizes the bring of the actors at that level. Ptolemy does not focus on function-architecture separation and mapping.

Koski design flow [47] provides a single framework for modeling of applications, automatic architectural design space exploration, automatic synthesis, programming, and prototyping of selected MPSoCs. Koski's design flow starts with the capturing, in UML, of requirements for an application and architecture, including design constraints, such as the overall maximum cost. The environment is doted with a UML interface that handles the transformation of application and architecture models to an abstracted model for fast architecture exploration. Particularly, the application model is transformed to an abstract process network model. In addition, the UML interface can back-annotate the UML design with performance information obtained from lower-level simulations. The design space exploration process is composed of a static (analytical) and a dynamic (simulation based) steps. Once an "optimal" system is defined, the last step of the Koski framework is to automatically generate the low-level code (software C code and hardware VHDL code).

MESH [22] stands for Modeling Environment for Software and Hardware. It separates the design into three parts: the software layer (corresponding to functional model), the hardware resource layer (corresponding to the architecture model) and the scheduling layer between them (corresponding to mapping). Each layer provides a set of services to the layer next above. The scheduling layer allocate an execution resource to a software thread. Software threads are annotated with time budgets for the corresponding hardware elements. This timing cost is extracted beforehand by estimation or profiling.

Gaspard2 [18] is an environment providing designers with the following means: a formalism for the description of embedded systems at a high abstraction level (based on the Marte UML profile), a methodology covering all system design steps, and a tool-set that supports the entire design activity. The model of computation adopted by Gaspard 2 offers a very suitable way to express and manage the potential parallelism in a system. Different automatic refinements steps from the high level of abstraction towards the lower levels are defined: for simulation at different levels with SystemC, for hardware synthesis with VHDL and for formal validation with synchronous languages.

Syndex [52] is a system-level computer Aided Design (CAD) software that enables to quickly explore the solution space to extract an efficient solution and generate a real time distributed executive without deadlock for multi-components architectures. It is based on the AAA, Algorithm Architecture Adequation, methodology that aims to find the best matching between an algorithm and an architecture, while satisfying design

constraints. AAA is based on graphical models that model both the potential parallelism of the algorithm and the available parallelism in the architecture components. The implementation consists in distributing and scheduling the algorithm data flow graph on the architecture multicomponent while satisfying the real-time constraints. This is formalized in terms of graphic transformations. Heuristics are used to optimize real time performances and resource allocation for embedded real-time applications on the SOC.

At an industrial level, there are as well many approaches for system level design and virtual prototyping. Hereafter a brief description of two of these industrial approaches: Panama [75] and CoFluent [29].

Panama[75] is an in-house Freescale development, with the main objective of enabling early analysis when some information is not yet available; namely lack of software applications and stable hardware component definitions. It targets to reach the desired accuracy while maintaining a good simulation speed. The application is specified using the TML (Task Modeling Language) language that contains abstract constructs to express an application flow with regards to its use of hardware resources. The architecture is specified in SystemC at the cycle accurate level. After a first analysis using the TML model, the modeling could be enriched using simulation traces to increase the accuracy. It is worth noting that Panama does not describe the application and architecture in an orthogonal fashion. In fact the task modeling language includes details of the underlying architecture which mixes the architecture and application models.

Cofluent tools [29] are based on the MCSE [64] methodology developed by the reasearch team of professor Jean-Paul Calvez. It provides a graphical description for the application and the architecture that resembles to UML and recentely a UML profile for CoFluent has been defined [30]. An internal tool enables the automatic generation of SystemC code from the graphical description in order to analyze the behavior. The MCSE methodology covers the design flow from requirements collection up to the prototyping.

Discussion

System Level Design methodologies represent a large umbrella of tools and approaches. The set of previously described approaches are few examples of existing well-known approaches. In fact, even back in 2006, D. Densmore et al [28] counted a 90 different academic and industrial system level design tool and/or methodology. However, these tools are sharing some common trends and similarities:

- Separation of concerns: a vast majority of these tools separate application and

architecture concerns, and the separation of communication and computation is becoming more popular

- Usage of the same specification language to specify the architecture and the application. Usually this language should rely on a clear definition (meta-model) of the modeled concepts. For instance Metropolis relies on a formal meta-model and Gaspard2 and Koski rely on well defined UML profiles. UML is getting the hype in this domain, especially that its semantic is generic and independent from any particular implementation
- Increasing interest in formal verification techniques integration to design flows
- Code Generation for simulation and analysis from the high level models. SystemC is largely the most preferred language due to its standardization and adoption in industry.

3.5 DIPLODOCUS and Extensions: Yet Another System Level Design Methodology for Early Design Analysis

The objective of this PhD work is to develop a methodology for performance estimation of complex SoCs at a high level of abstraction and in early design stages; ultimately, when software and hardware are not yet defined. Thorough architecture performance exploration cannot be performed with cycle and bit accurate system models, because they are too detailed and thus very expensive to be built and require an equally advanced software implementation. A general solution is to use abstract system models for both application and architecture. The challenge remains to preserve the accuracy of the performed estimations. Raising the level of abstraction will reduce the time needed for developing models, thus reducing the effort before the first performance estimations. The proposed approach extends the DIPLODOCUS UML profile to model shared resources and communication architecture interactions. Performance analysis and design exploration process are performed by using the SystemC based simulator developed to simulate the SystemC code generated from the UML models.

3.5.1 Modeling Approach

Figure A.1 depicts our approach, the red boxes represent our contribution to the DIPLODOCUS methodology. The numbers between brackets represent the section or chapter that will describe the specific part of the approach. The “Application Modeling” package specifies how the functionality of the system is captured. It provides

modeling constructs for modeling the structure (using a simple task model or a hierarchical component model) and the behavior (using the UML activity diagram) of the application. The "Architecture Modeling" package specifies the architectural resources that will be used to execute the application and the communication schemes between these resources through the "Communication Interaction Modeling" Package that defines all the possible communication patterns in a specific architecture. During the system modeling phase, application and architecture concerns are clearly separated, thus a mapping phase is needed.

The "Mapping Package" defines how the application constructs are mapped and executed on the architecture resources. It clearly defines:

- How the shared resources will be used and specifically how scheduling decisions are taken (the "Shared resources Modeling" Package)
- The communication management policy used to select among the possible patterns the one to use in a specific communication
- The storage allocation of the application's constructs on the memories
- The execution allocation attaching application tasks to computation resources.

DIPLODOCUS main assumptions

DIPLODOCUS was created with the objective of performing early design analysis. Thus, the designer could make design decisions early in the design process, even when either the code and/or the architecture components are not yet available. To accomplish this, the analysis phase should be as fast as possible, techniques like cycle accurate simulations take long time for simulation and for building the model itself, and they need the code and the architecture complete definition before being performed. In DIPLODOCUS, and to speed up the analysis, data is abstracted. This means that tasks are exchanging abstract data samples and not valued data, only the data size matters and not the data itself.

Furthermore, DIPLODOCUS defines a clear separation between the concerns of the application and of the architecture, thus designer could test the performance of an application on many architectures and vice versa.

Models Processing and Analysis Approach

Our objective is to provide designers with tools that will help them to optimize the application, architecture and mapping models. We provide a SystemC simulation environment to simulate modeled systems and to evaluate their performance (detailed in

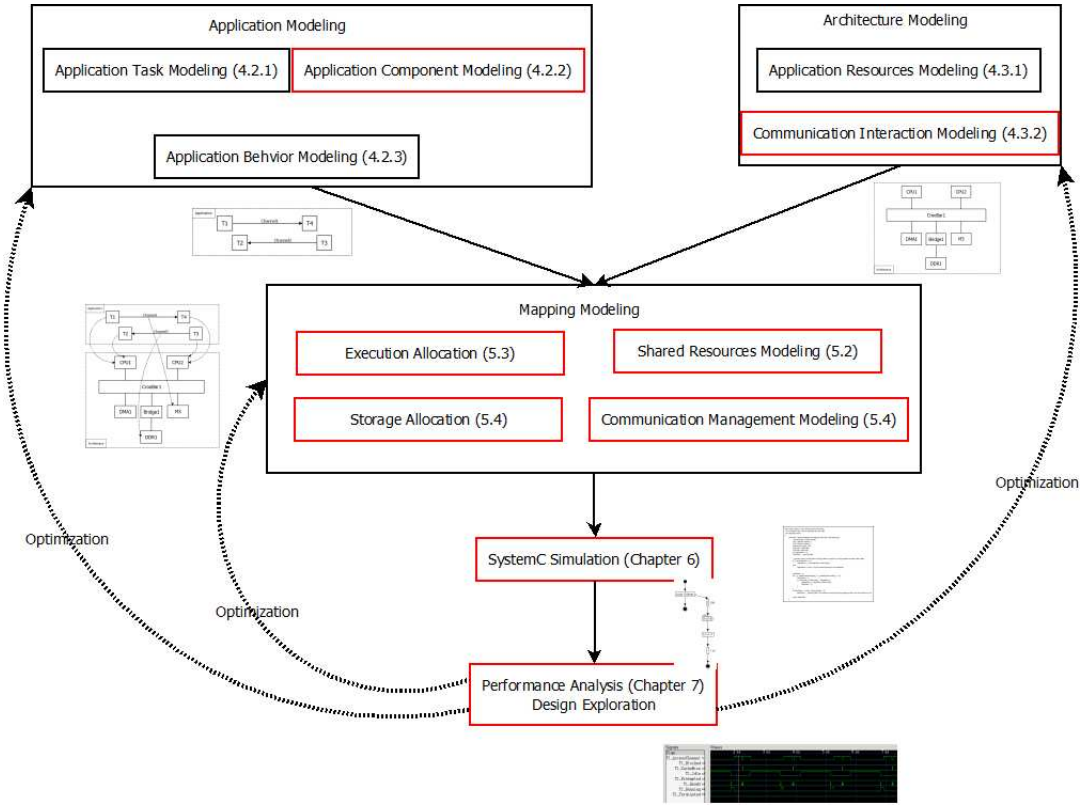

Figure 3.1: The Extended DIPLODOCUS methodology

chapter 6). The environment takes DIPLODOCUS UML models (Application, Architecture and Mapping) as inputs. It generates automatically from them corresponding SystemC code. The objective is to measure the performance metrics of the studied system, for example latency, throughput, utilization, and contentions.

The following subsections will compare the extended DIPLODOCUS approach with existing approaches and UML profiles.

3.5.2 UML for SoC Modeling

In recent years, UML increasingly has become known for its capability of unifying the specification and the design of the hardware and software parts of electronic systems. In a survey (back in 2007) organized at the margin of the 4th UML-SoC Workshop [11], participants from both academia and industry answered the question "What is the reason for your interest in UML tools and methodologies?". A vast majority of them expressed that they are considering UML to specify the system's behavior and interfaces, and to specify in a single environment systems which will involve different disciplines (HW, SW, analog, digital domains). In addition to specification of these heterogeneous systems, they were as well considering the usage of UML in the context of analysis and design space exploration of the system's architecture.

Several UML profile are proposed for SoCs modeling. Some of them are integrated in a design methodology and tools. Hereafter an overview of five well known of these profiles: TUT, UML-SoC, UML SPT, UML Marte, and the DIPLDOOCUS UML profile extended and defined in this thesis.

TUT [54] Profile extends UML 2.0 to model mutli-processors SoCs. Its objective is to enhance the support of external tools for automated analysis, profiling, and modifying the UMLmodel of an embedded system. The TUT profile defines a set of stereotypes for describing applications (including real-time requirements) and architectures as well as their mapping. Application is modeled as a network of communicating processes (conforming to the Khan Process Network). Process' behavior is modeled using the UML state chart diagram. Platform description is done as well at a high level of abstraction. However, for communication modeling some more detailed models are used (a transaction level generator) to describe the interaction between platform nodes. TUT is developed at the university Tampere, Finland, and is supported by the Koski environment and is used as input format. The mapping can be performed manually by the designer or assisted by the Koski environment tools.

UML-SoC [5] is an industry based UML profile. Proposed by Fujitsu Limited and several other companies, it intends to model SoCs. UML-SOC extends UML2.0 to propose an approach for the structural modeling, communication modeling, operation and property modeling. In fact, the proposed concepts in the context of UML-SoC are

closely related to SystemC, making possible an automatic generation of SystemC code from UML-SoC models. Unfortunately, the profile is more focused on architecture modeling and application concerns are mixed with the architecture ones.

UML-SPT profile [2] was standardized by the OMG (Object Management Group [61]) for modeling real-time systems. This profile does not only model the functionality of a system, but add the notion of quality of service (QoS) to attach non-functional information to UML models. SPT notion of QoS permits the quantification of timing and resources. Resources' performance is expressed in terms of performance parameters such as capacity, availability and timing. Clients resource (users) require a QoS from the resource they are using such as maximum deadline, and throughput. The relationship between a client and a resource is called a QoS contract. SPT was adopted by many industrial and academic approaches and several quantitative analysis of SPT models are available, and that can be classified in two categories: performance analysis and schedulability analysis. The first one uses stochastic techniques such as queuing theory or Petri nets to calculate response times, delays and resources requirements. Thus, they are aimed at determining the rate at which a system can perform a function. On the other hand, schedulability analysis uses mathematical methods such as RMA (Rate Monotonic Analysis) or holistic techniques, to predict whether a set of software tasks will meet all its timing constraints. Then, they are oriented to verify timeliness.

Marte [8] stands for *"Modeling and Analysis of Real Time and Embedded systems"* is a standard UML profile adopted by the OMG to replace the UML-SPT profile. It extends the possibilities to model the features of software and hardware parts of a real-time embedded system and their relations. It also offers added extensions, for example to carry out performance and scheduling analysis, while taking into consideration the platform services (such as the services offered by an OS). The profile is structured in two directions. First, the modeling of concepts of real-time and embedded systems and second the annotation of the models for supporting analyses of the system properties. The organization of the profile reflects this structure, by its separation into two packages: the MARTE design model and the MARTE analysis model respectively. These two major parts share common concepts, grouped in the MARTE foundations package for expressing: non-functional properties (NFPs), timing notions (Time), resource modeling (GRM), components (GCM) and allocation concepts (Alloc). An additional package contains the annexes defined in the MARTE profile along with predefined libraries.

DIPLODOCUS [82] stands for *"DesIgn sPace exLoration based on fOrmal Description teChniques, Uml and Systemc"* is a UML profile proposed by Telecom ParisTech. It targets the modeling of SoCs at a high level of abstraction with the objective

of performing early design analysis and design space exploration. It adopts the Y modeling paradigm by separating the concerns of the application and the architecture. The design space exploration in DIPLODOCUS is supported by an open source toolkit named "TTool"[63], which could additionally generate documentation from developed models. DIPLODOCUS has been created with the following assumptions:

- Application, architecture and mapping are modeled at a high level of abstraction with the same language (UML) and in the same environment (TTool).
- Application data is abstracted - by a concept of non-valued samples - leading to very fast simulations and allowing the use of formal techniques.
- Architecture modeling is done as a composition of instances of execution nodes (CPUs, DSPs, hardware accelerators ...), communication nodes (busses, routers, switches ...) and storage nodes (memories). These components are abstract generic ones and are specialized through a set of performance parameters.
- From UML DIPLODOCUS models, formal verification may be performed using automatic code generation to LOTOS or UPPAAL.

Discussion

The above described UML profiles are well-known profiles in the SoC design and modeling. All of them are adopted at a research or industrial level, and there are many tools and methodologies supporting them. Table 3.1 presents a comparison of these profiles.

The TUT, UML SPT, Marte and DIPLODOCUS profiles define a clear separation between application and architecture, while the UML-SoC profile is closely related to the SystemC language and focuses more on the architecture modeling rather than the application. With respect to the formal foundation criterion, only the DIPLODOCUS profile (among the profiles described above) present formal semantics (in LOTOS and in UPPAAL).

As specifying a system is not sufficient to study and analyze it, the UML models are used as en entry level specification in a high system level modeling approach. For instance the Marte profile is used in the Gaspard2 [18] methodology, and a complete flow is defined, starting from the Marte model down to hardware synthesis, including SystemC code generation for simulation purpose. DIPLODOCUS and its extensions presented in this thesis do not target synthesis, but rather focuses on system level analysis

Even though, the modeling contributions presented in this thesis, namely the shared resources modeling and the communication modeling and management, are generic and can be modeled using Marte or TUT; we made the choice to use the

Profile Name	Separation of Concers	Formal Semantic	Tool Support	Development Year
TUT	Yes (Y chart)	No	Koski design environment	2005
UML SOC	No (loosly coupled to SystemC)	No	Rational Rose	2005
UMLSPT	Yes (Y chart)	No	Rational Rose	2005
Marte	Yes (Y chart)	No	Papyrus, Rhapsody	2007
DIPLODOCUS + this thesis	Yes (Y chart)	Lotos and UPPAAL	TTool	2006

Table 3.1: Comparison of some UML profiles for SoC modeling

DIPLODOCUS profile. In fact, DIPLODOCUS, developed and maintained by Telecom ParisTech since 2006, is not simply a UML profile but it comes with a full tool support with the TTool [63] toolkit. DIPLODOCUS and TTool enable designers to model, generate formal specification and simulation code (the code generation for simulation is presented later in chapter 6), analyze the simulation/verification results with the same environment. Furthermore, and due to the DIPLDOOCUS metamodels developed during this PhD work, a future feasible step is to use the model transformation tools (such as kermeta [60]) to interchange models between DIPLODOCUS and other well defined profiles (such as Marte).

3.5.3 Shared Resources Contentions Modeling

Most recent Systems-on-Chip (SoC) are meant to host highly complex and interdependent applications at low cost (area, power) to the end user. The number of resources must be minimized while increasing their utilization by sharing them between multiple applications. Such shared resources have a strong impact on SoC performance because of the contention they typically induce. Indeed, end-to-end performance of a given application is the sum of the time needed to execute that application with the total contention delay of all involved shared resources.

Many design methodologies and supporting tools - including DIPLODOCUS - propose a mapping phase once application and architecture models have been performed [17]. Those methodologies extract shared resources impact on system's performances. Like our approach, some methodologies are more particularly focused

on early analysis and documentation of complex architectures, while functional modeling and synthesis of implementations is the intent of others (e.g., [68]). The following discussion compares our approach with other high-level methodologies that attempt to estimate impact of shared resources on system performances.

The back annotation techniques like MESH [22] and the one proposed in Schnerr & al. [70] extract performance latencies from a low-level simulator to annotate the higher level model. They utilize analytical and simulation techniques to estimate shared resources contention. Final code is used to estimate the performance. On the contrary, our methodology is applied early in the design flow, and so before the code is released. Also, above-mentioned techniques focus on the modeling of task scheduling and extract contention attributes related to communication and memories from low level simulations. In our approach, we extract this information from the high-level simulation of our models.

Early architecture exploration methodologies like Sesame [46] offer a clear distinction between application and architecture concerns, and facilitate flexible system-level performance evaluation. Application is modeled as a set of Khan processes while architecture is defined at a high level of abstraction in a similar way to DIPLODOCUS. So far Sesame mapping models only provide schedulers to allocate computation resources to the application Khan processes: it does not model communication architecture arbitration nor memory mapping.

Kempf et al. [49] presents a simulation framework for MP-SoC platforms. They use a virtual processing unit (VPU) to schedule the execution of tasks mapped to a processor. The important difference to our approach is that we generalize the notion of a virtual node to model accesses policies to any type of architecture resources, and that we are able to extract performance result of any shared resource.

Panama [75] enable modeling at a high level of abstraction and they capture task scheduling as well as the communication architecture and memory mapping modeling. Unfortunately they do not define a clear separation between the application and the architecture. Indeed, the language used to model tasks includes details of the underlying architecture thus reducing the reusability of models.

3.5.4 Communication Modeling

In modern multi-core and/or heterogeneous architectures, communication becomes an important issue for the the global performances as the communication infrastructure is shared between the different cores. Thus, a system level design approach should take into consideration the communication impact. Nevertheless, a communication

analysis process at higher abstraction levels is not trivial. The main challenge is the lack of timing information at these abstraction levels, which make it difficult to acquire accurate estimations to guide the space exploration.

Our objective is to estimate communication performance at a high level of abstraction, and where the required timing details are not available. The two main challenges are:

1. How to abstract the communication architecture features?
2. How to estimate the communication performances given the abstracted communication features?

In the system level modeling literature there is two main approaches for communication modeling. The **first** approach [18; 46; 47; 82] is based on the fact that the application model (usually a set of communicating tasks) drives the architecture communication behavior. For instance, a data transfer between two tasks executing on two different cores, will induce a data transfer on the communication architecture and the shared memory. As design is at a high level of abstraction, designer usually specifies the amount of data transfered, so in the performance estimation this amount information is used to calculate the communication latency while presenting some inaccuracies because the exact timing information are not captured at the high level of abstraction. The **second** approach is to use traces [75], in fact the communication architecture is be modeled at a lower of abstraction than the computation architecture and the application, than designers use traces (generated using traffic models of applications or the execution of the real code of available on one core) to enhance the accuracy.

The communication modeling approach proposed in this thesis is following the first approach and is based on a modeling construct, the *Communication pattern*. It's objective is to separate the computation from the communication. It models the interaction between the different architecture components following a communication protocol. Once, a communication (data transfer) is requested, the communication pattern is executed.

3.6 Summary

This chapter's main focus was to describe the system level design methodologies, their concepts and their trends. Few methodologies were described as well as the specification languages widely used at the system level. In addition, this chapter introduced the proposed approach (extensions to the DIPLODOCUS methodology) for modeling shared resources and orthogonalization of concerns of the computation and the communication. Our contributions to DIPLODOCUS will be detailed in the following chapters (As depicted in figure A.1).

Part II

PhD Contributions

Chapter 4

Architecture and Application Modeling

The DIPLODOCUS methodology is based on an UML profile targeting design space exploration at a high level of abstraction. It adopts the Y modeling concept, and defines a clear separation of concerns between the application and the architecture as well as between the computation and the communication. This orthogonality of concerns allows modifying one of the components, while keeping the rest at their previous (default) configuration. Thus, the mapping, for example, may be varied without touching the application or architecture models. The objective of the previous chapter was to compare the existing approaches and to introduce the contributions of this PhD thesisto the DIPLODOCUS methodology. This chapter builds on the existing DIPLODOCUS concepts introduced in [14] and [82], and defines the UML meta-models for the application (section 4.2) and for the architecture (section 4.3). It enriches the existing DIPLODOCUS with the concept of "Communication Patterns" to model the communication interactions performed by the architecture (introduced in section 4.3.2). A hierarchical application modeling approach is defined as well to permits the modeling of complex applications. This chapter provides a definition of the proposed concepts through UML meta-models to describe how the DIPLODOCUS profile extends the UML 2.0 standard. In addition, the defined profiles are illustrated by modeling examples. But first, section 4.1 presents the UML modeling basics.

4.1 UML the Unified Modeling Language: Models, Metamodels and Profiles

A model is "a representation of some phenomenon of the real world made in order to facilitate an understanding of its workings. A model is a simplified and generalized version of real events, from which the incidental detail, or 'noise' has been removed"

[71]. A meta-model is yet another abstraction highlighting properties of the model itself. A meta-model is a model whose instances are the data types of another model. This model is said to conform to its meta-model like a program conforms to the grammar of the programming language in which it is written. The meta-model defines the structure, semantics and constraints for the models.

UML 2.0 [61] is created to help designers to specify, visualize, and document models of software systems, including their structure and design. It defines thirteen types of diagrams, divided into three categories: Six diagram types represent static application structure, three represent general types of behavior; and four represent different aspects of interactions:

1. Structure Diagrams include the **Class Diagram**, Object Diagram, **Component Diagram**, Composite Structure Diagram, Package Diagram, and **Deployment Diagram**.

2. Behavior Diagrams include the Use Case Diagram (used by some methodologies during requirements gathering); **Activity Diagram**, and State Machine Diagram.

3. Interaction Diagrams, all derived from the more general **Behavior** Diagram, include the **Sequence Diagram**, Communication Diagram, Timing Diagram, and Interaction Overview Diagram.

UML is a language with a very broad scope that covers a large and diverse set of application domains. Not all of its modeling capabilities are necessarily useful in all domains or applications. A UML profile is an extension of the UML meta-model to model a specific domain. The extension mechanisms allow refining standard semantics in a strictly additive manner, so that they cannot contradict standard semantics. By definition [76], a UML profile is a specification that does one or more of the following:

- Identifies a subset of the UML metamodel.

- Specifies “standard elements” beyond those specified by the identified subset of the UML metamodel. “Standard element” is a term used in the UML metamodel specification to describe a standard instance of a UML stereotype, tagged value or constraint.

- Specifies semantics, expressed in natural language, beyond those specified by the identified subset of the UML metamodel.

- Specifies common model elements, expressed in terms of the profile.

The **DIPLODOCUS** UML profile is an extension of UML to model, at a high level of abstraction, complex System-on-Chip (SoC). The extension mechanism through

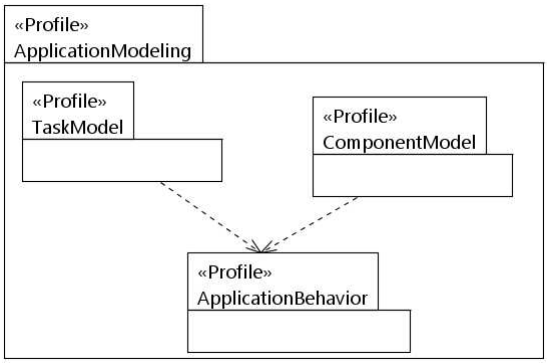

Figure 4.1: DIPLODOCUS Application Modeling profile

profiling is restricted to the use and extension of an existing modeling language metamodel without modifying the abstract syntax or semantics of the source modeling language. This chapter is organized as follows: First section 4.2 defines the Application Modeling profile that specifies the concepts for modeling the structure and the behavior of the application. Then, section 4.3 presents the Architecture Modeling concepts used to model the architecture that will execute the specified application

4.2 Application Modeling

A DIPLODOCUS Application represents the functionality that the modeled system will accomplish during its execution. The system's architecture will execute this application after the mapping phase. DIPLODOCUS defines how the structure and the behavior of the application should be modeled by extending the appropriate UML diagrams. The DIPLODOCUS "Application Modeling" sub-profile is depicted in figure 4.1. The Designer can choose to model the structure of the application using the "Application Structure Task Model" or to use the hierarchical capabilities of the DIPLODOCUS "Application Component Model". It is worth noting that both models are semantically equivalent. After defining the structure of the application, the designer defines the behavior as specified in the "Application Behavior" sub-profile. The following sections will detail these concepts.

4.2.1 Application Structure Task Model

The DIPLODOCUS Application is structured around the notion of *"task"* that holds a functionality (task's behavior). Tasks could communicate and exchange data through communication connectors. The DIPLODOCUS task modeling was first introduced by [14] and [82]. An application task model is a composition of a set of tasks and

all the communication connectors that are connecting them. In DIPLODOCUS the parallelism is inter-task, if there is no data dependency tasks can execute in parallel if they are later mapped on different computation nodes. However the behavior of a task is sequential.

Each DIPLODOCUS task, t, is characterized by the following elements:

- A behavior that describe the task's functionality (more details on that in section 4.2.3)
- A set of input communication connectors (where t is the destination task)
- A set of output communication connectors (where t is the origin task)
- A set of attributes, that are used by the task's behvior. In DIPLODOCUS an attribute can be integer or boolean
- A name that represents the identifier of the task. Task name must be unique.

On the other hand, the communication connectors in DIPLODOCUS are of two types:

1. Data Exchange Connectors, *channels*, that represents a way for modeling data exchange, without any knowledge about the implementation details of the underlying communication infrastructure or the actual contents of the data,
2. Synchronization connectors are means for tasks to exchange signals to control or request specific executions, there is two types of DIPLODOCUS synchronization connectors: *events* and *requests*

Data Exchange Connectors: Channels

A channel is a connector that carries data samples between tasks. The behavior of a task is affected when reading/writing in a channel, depending on the "type" of the channel. A channel can be: 1- blocking on read, so a task cannot read from an empty channel, 2- blocking on write, so a task cannot write to a full channel, or 3- non blocking on write or/and non blocking on read. Three types of channels are defined:

- BRBW channel: for Blocking Read/Blocking Write Channel: a task cannot read from an empty channel nor write to a full channel. It represents a finite FIFO
- BRNBW channel: for Blocking Read/Non Blocking Write Channel: a task cannot read from an empty channel but it's never blocked on writing. It represents an infinite FIFO

- NBRNBW channel: for Non Blocking Read/Non Blocking Write Channel: a task is never blocked neither on read nor on write.

Each DIPLODOCUS channel has a name (identifier) and an origin (sender) and a destination (receiver) task. It has a type (BRBW, BRNBW, or NBRNBW). The designer should as well define the ”sample size”, in bytes, is the size of a sample of data transfered on the channel and the maximum number of samples (MaxNbSamples) that can be queued in a BRBW channel.

Synchronization Connectors: Events and Requests

The “Synchronization Connectors” (events and requests) are used to communicate control information to other tasks in the system. The destination task is blocked while waiting on a synchronization connector. The origin task can be blocked while sending an occurrence (writing in the event’s FIFO). Thus, for the events we have two types of FIFOs: finite and infinite. On the other side, requests are always represented by infinite FIFOs and they are not blocking on send. Both, “Events” and “Requests” may carry some Parameters “P”

Each DIPLODOCUS synchronization connector (event or request) is defined by its name, it has an origin (sender) and a destination (receiver) task. It has a type (finite FIFO, infinite FIFO). The designer should define the maximum number of event occurrences (MaxNbOccurences) that can be queued in an finite FIFO event.

Application Task Model’s UML Representation

The DIPLODOCUS Task diagram that represents the application’s structure extends the UML **class diagram**. The UML meta-model that defines this extension is depicted in figure 4.2. Functionalities are organized under the notion of *Task*, that extends the UML metaclass ”class”. Communication Connectors (channels, event and requests) that connect DIPLODOCUS tasks extends the UML metaclass “Association Class”. A task has a dedicated behavior that will define how it will execute and define its communication scenario with the other tasks.

Task modeling example

Figure 4.3 depicts a simple application example modeled with the TTool toolkit. The application in the example is the composition of a set of tasks (Task0, Task1, Task2) and the set of Communication Connectors (ch1, evt1, done, reqTask2). The three tasks functionalities are interdependent as each task is an origin and destination for different communication connectors: Task0 exchange data with Task1 through the channel “ch1”, but it is a destination for the event “evt1” sended by Task1. With the same logic, Task1 request the execution of Task2 through the request “reqTask2” and it’s

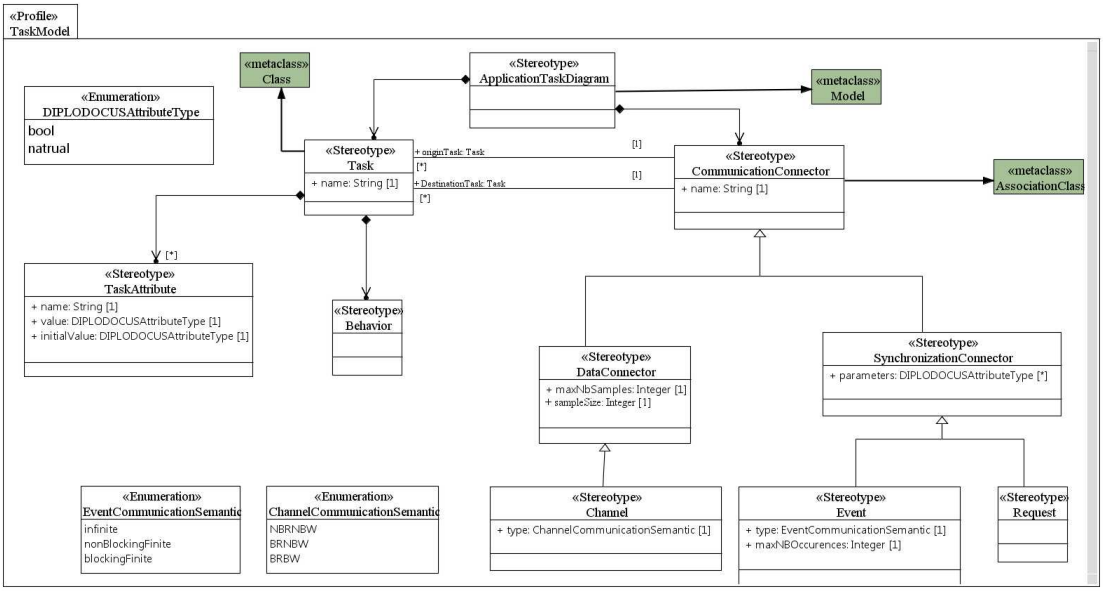

Figure 4.2: Application Structure Task metamodel

destination for the event “done” sended by Task2 to confirm the reception (execution) of the request by Task2. Moreover, Task0 has an attribute a of type Natural (integer), initialized to 0. Later in the application behavior modeling we will show how the task would manipulate the value of an attribute, send events or wait for events ...

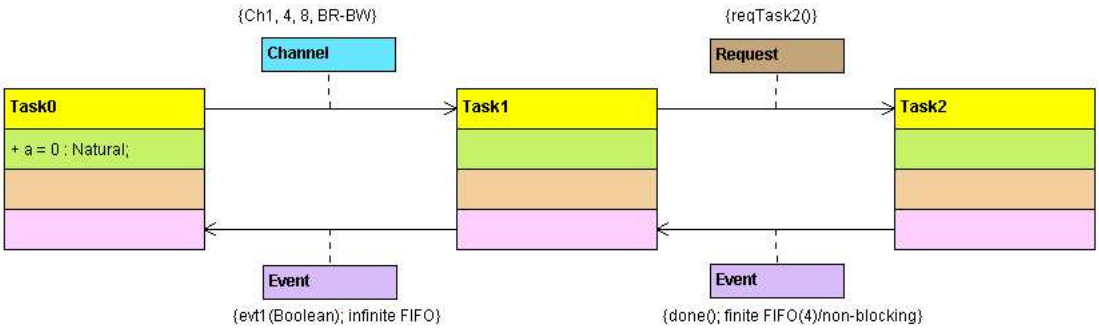

Figure 4.3: A simple application modeled by DIPLODOCUS Task Model with TTool

4.2.2 Application Structure Component Model

The application structure modeling approach presented in the previous section has the advantage of being semantically clearly defined. Formal descriptions in LOTOS or UPPAAL[42], can thus be generated using the Turtle toolkit[63]. However it suffers

some drawbacks (detailed in the next subsection), namely in terms of complexity and re-usability. To face these drawbacks and enhance the application DIPLODOCUS modeling, we introduce a new Application Structure modeling diagram based on component modeling approach. In fact, decomposing systems into individual modules for flexibility, re-usability and comprehensibility is a technique long proposed for system design. In component-based engineering, functionality is encapsulated in reusable components. This increases the re-usability and reduces the implementation of new applications by assembling existing components. It also makes programs more understandable, as design decisions that only affect a part of the application can be hidden in a module or component.

Component Model's Motivations

Models Complexity: the task modeling approach fits perfectly when the application model is small, and involves a relatively small number of tasks and connectors, but when the application is "complex" the task modeling will rapidly become cumbersome and the designer will have some difficulties navigating in the complex model. Thus, introducing the notion of hierarchical modeling aims at the models complexity and open the window for the modeling of more and more complex applications (for example the LTE application we modeled, for more details please refer to chapter 7).

Applications are inherently hierarchical: applications targeted by DIPLODOCUS are usually hierarchical. For instance a telecommunication protocol is composed of many layers and inside each layer there is a set of smaller entities that collaborate to accomplish the functionality of the layer. Thus a hierarchical DIPLODOCUS application model would reflect more appropriately the real application.

Increase models re-usability: An important aspect of component-based modeling is the reuse of previously constructed components[78]. A component can always be considered an autonomous unit within a system or subsystem. It has one or more interfaces (exposed via ports). Although it may be dependent on other elements in terms of interfaces that are required, a component is encapsulated and its dependencies are designed such that it can be treated as independently as possible. As a result, components and subsystems can be flexibly reused by connecting them together via their interfaces.

Enable collaborative development of models: As interfaces are clearly defined, components may be modeled by different designers. For instance, each component can be developed by an expert designer in the specific domain

Finally our component model has the same semantic as the task model and a translation from component to task model is provided.

Component Model's definition

DIPLODOCUS Component Based Modeling (Figure 4.4) extends the UML component diagram. It's main characteristic is the support of hierarchical modeling, thus enabling the encapsulation of functionalities in well structured groups (the composite components). A *CBD* (Component Based Diagram) is assembled out of components. There are two types of components: the primitive components (equivalent to tasks) and the composite components that are composed of composite and/or primitive components. The *CBD* is the composition of a set of primitive and composite components connected through connectors and ports. The CBD itself is a composite component.

The set of primitive and composite component encapsulated by a composite component defines a Hierarchical Level (HL). In a hierarchical level, a component's name is unique. The global component name is the composition of its name with the names of composite components encapsulating it while going up in the hierarchy up to the highest level (in other words up to the CBD).

A primitive Component is equivalent to a task in the Task Modeling approach presented in the previous section. Thus, it has behavior, a set of attributes (integer or boolean) and the communication with other components is done through the primitive ports attached to the primitive component.

A Primitive port p_{Port} is an access point to the primitive component's behavior, and it has an identifier, a direction that represents the fact that the port is orgin or destination of a communication and a type that represents the communication semantic of the port (a port can be a channel, an event or a request). The communication semantic is the same as defined in section 4.2.1

Components' ports (primitiove and composite) are connected using connectors. A connector, c, has an input port *InPort* and an output port *OutPort*. Both input and output ports must be attached to different components, and can be primitive or composite ports. Furthermore, a connector can not connect a port to itself.

When a primitive port of a primitive component is connected to a composite port of an encapsulating composite component, the later inherits its type. The composite component may later be connected to another composite port or to another primitive port. The global connection must be valid: the primitive components on both sides should be of the same type. To perform simulation and formal analysis a component model is transformed into a task model. In fact we eliminate the hierarchy, primitive components are transformed into tasks and connected ports (of the same type) are transformed into channels, events or requests.

Component Model's UML Representation

The proposed DIPLODOCUS component model extends the UML 2.0 component diagram. Figure 4.5 depicts its meta-model. It reproduces the mathematical concepts

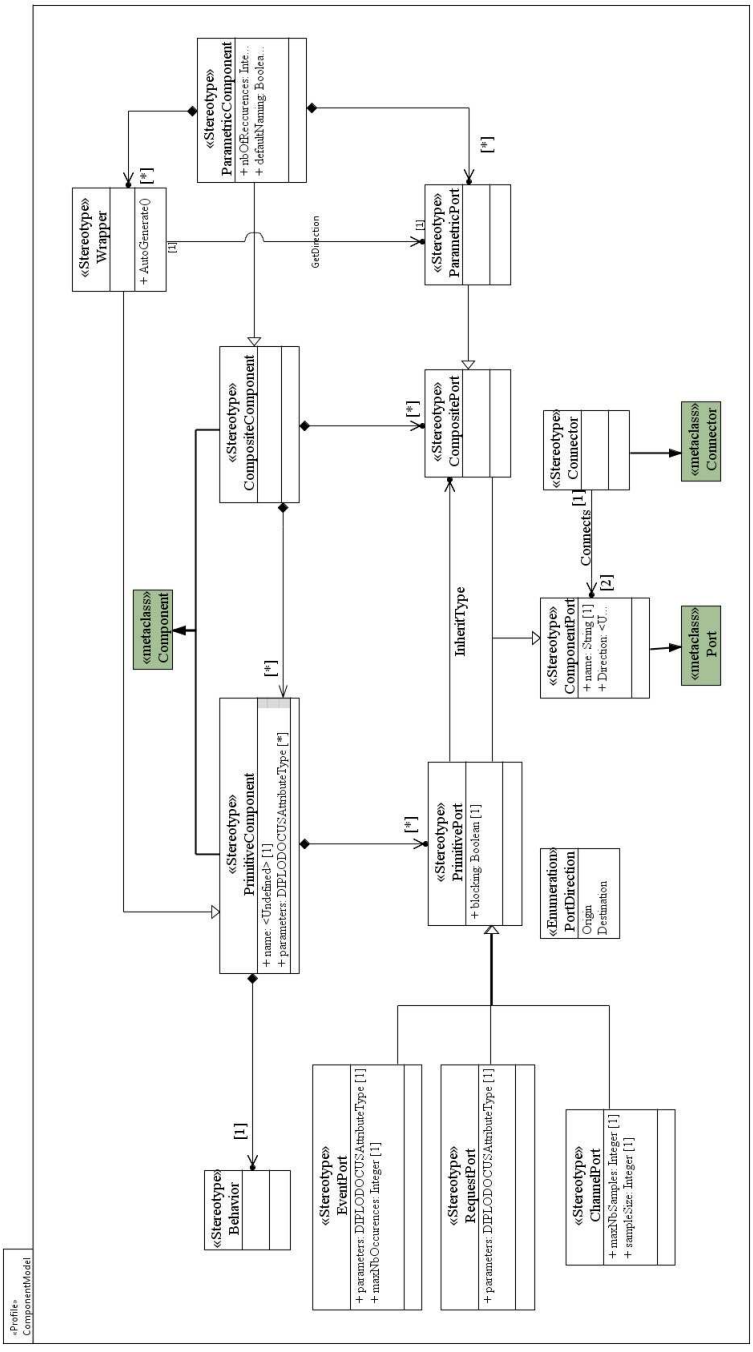

Figure 4.4: The Component Modeling metamodel

defined earlier. It defines the relation between composite and primitive components and their respective ports.

Component Modeling Example

Figure 4.6 depicts the upper hierarchical level model of the uplink physical layer of the LTE telecommunication standard. This model is used as a case study for the apporach presented in this thesis and is further detailed in chapter 7. The lime green boxes are primitive components while the yellow box is a composite component. The primitive component "SBL1_UL_Config" defines the environment of the model where it specifies the number of processed packets and many other parameters that interfere in the system execution. The "IF4" and "IF1" are representing the interfaces of the modeled system (the uplink physical layer) to the other standard layers. The composite component "SBL1_UL" represents the physical layer and is encapsulating 54 other components. It would have been extremely difficult to model all this number of tasks using the task model.

4.2.3 Application Behavior Model

To define the behavior of a task (or of a primitive component), DIPLODOCUS extends the UML activity diagram. A DIPLODOCUS behavior diagram is a set of commands that fulfills the functionality of the task. DIPLODOCUS defines 4 types of commands:

1. Control Commands (ConCommands): this command's set contains the basic control structures, namely conditional execution (IF command), repetition structure (loop command), choice command, and sequence command (define a sequence of execution between different branches). It includes as well the action command that changes the value of task's attributes.

2. Communication Commands (CommCommands): these commands involve any data or synchronization exchange between tasks. Commands are read/write to a channel, send/wait an event and send a request, check if an event's FIFO contains occurrences (NotfiyEvent command), and wait on multiple events. Tasks execution can be blocked if the involved connector is blocking and and the event's FIFO is full.

3. Abstract Execution Commands (AbsExecCommands): this subset of commands represents the computational complexity of the node while executing applications. The semantics and execution delay of these commands depend on the underlying architecture.

4. Temporal Commands (TempCommands): the Delay command represents an absolute delay.

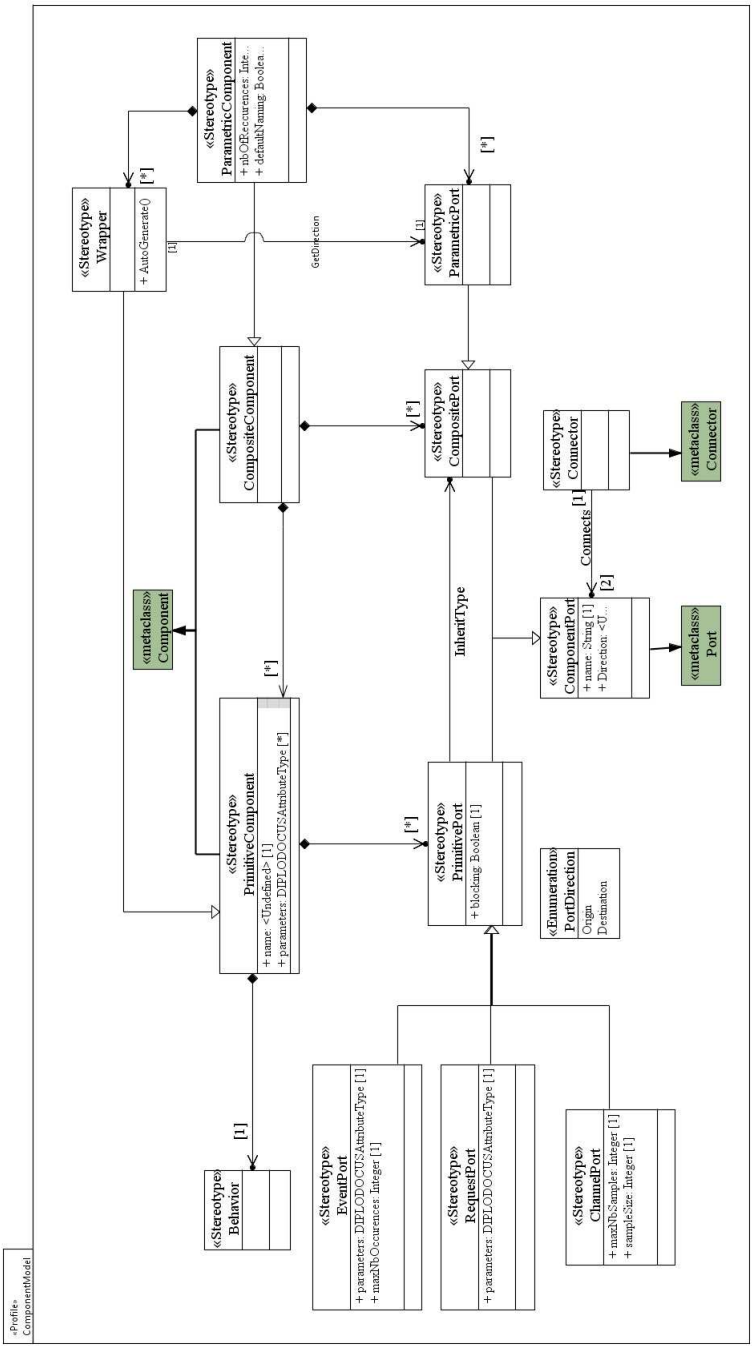

Figure 4.5: The Component Modeling metamodel

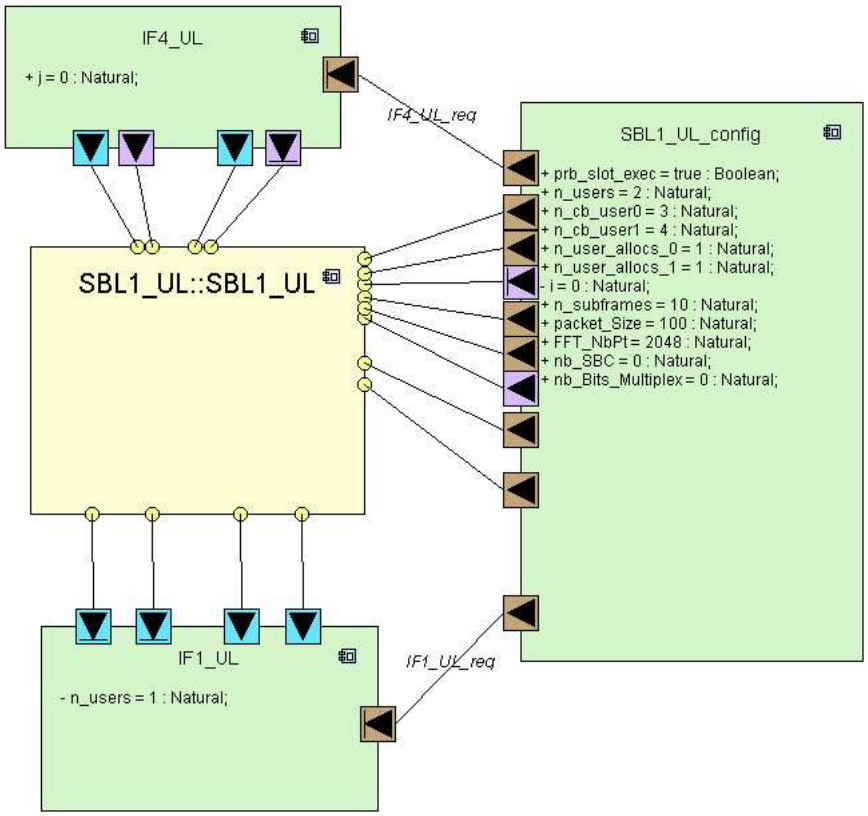

Figure 4.6: The uplink physical layer of the LTE standard modeled using the proposed DIPLODOCUS model

A DIPLODOCUS behavior command belongs to one the above defined types of commands. In addition it has an execution attribute *Commandatt*. For instance, a "send event" command has as attribute the name of the event it wants to send and the number of occurrences. However, and in order to be semantically correct, the event should belong to the set *EVENTS* that contains all the events of the task. Furthermore, when using a variable in any of the commands it should be declared in the attribute list of the task.

A DIPLODOCUS "Activity Diagram", *AD*, for task's behavior modeling is defined as a tuple:

$$AD = (C_0, C, E) \tag{4.1}$$

Where

- C is a set of all AD commands
- $C_0 \in C$ is the initial command. There is only one initial command in an *AD*

- *t* is a commands typing function: t : C $\rightarrow$ Type with Type = $\{ConCommands \cup CommCommandss \cup AbsExecCommands \cup TempCommands\}$
- *E* is the set of edges connecting the AD commands.

An edge e $\in$ E is a set of an input command *u* and an output command *v*:

$$e = \{u, v\},\ where\ u, v \in C\ and\ u \neq v \tag{4.2}$$

Some commands have only one successor, and by consequence they have only one outgoing edge. Others may have a conditional execution, for example a choice command has at least two execution successors (branches) depending on the choice condition, and thus has multiple outgoing links. In the same logic a wait on multiple event will wait for at least one event, will select the branch for this specific event and execute it, while the others branches will not execute. Thus, each command has a set of successors, this set could be empty if the command is the last command in a branch, it can be composed of one or more successors for the other commands. *succ(u)* is the set of the successors of the u command, and the following holds:

$$|succ(u)| = \begin{cases} = 0 & \text{if } u \text{ is an end of branch command} \\ = 1 & \text{if } u \text{ is Send/wait event, read/write Channel,} \\ & \text{action, abstract exec command} \\ = 2 & \text{if } u \text{ is a loop command} \\ = * & \text{if } u \text{ is a choice command it could have multiple successors} \end{cases} \tag{4.3}$$

Application Task's Behavior UML Representation

The DIPLODOCUS application behavior profile is depicted in figure 4.7. This meta-model takes the mathematical concepts defined earlier in this section and reproduces them in UML constructs. The task's behavior extends the UML activity diagram. The meta-model defines the behavior commands as UML nodes that are connected through edges extending the UML metaclass "Activity Edges". It defines all the DIPLODOCUS possible commands that can be used by the designer to define the application's behavior. The commands are grouped into the four groups identified in the earlier section.

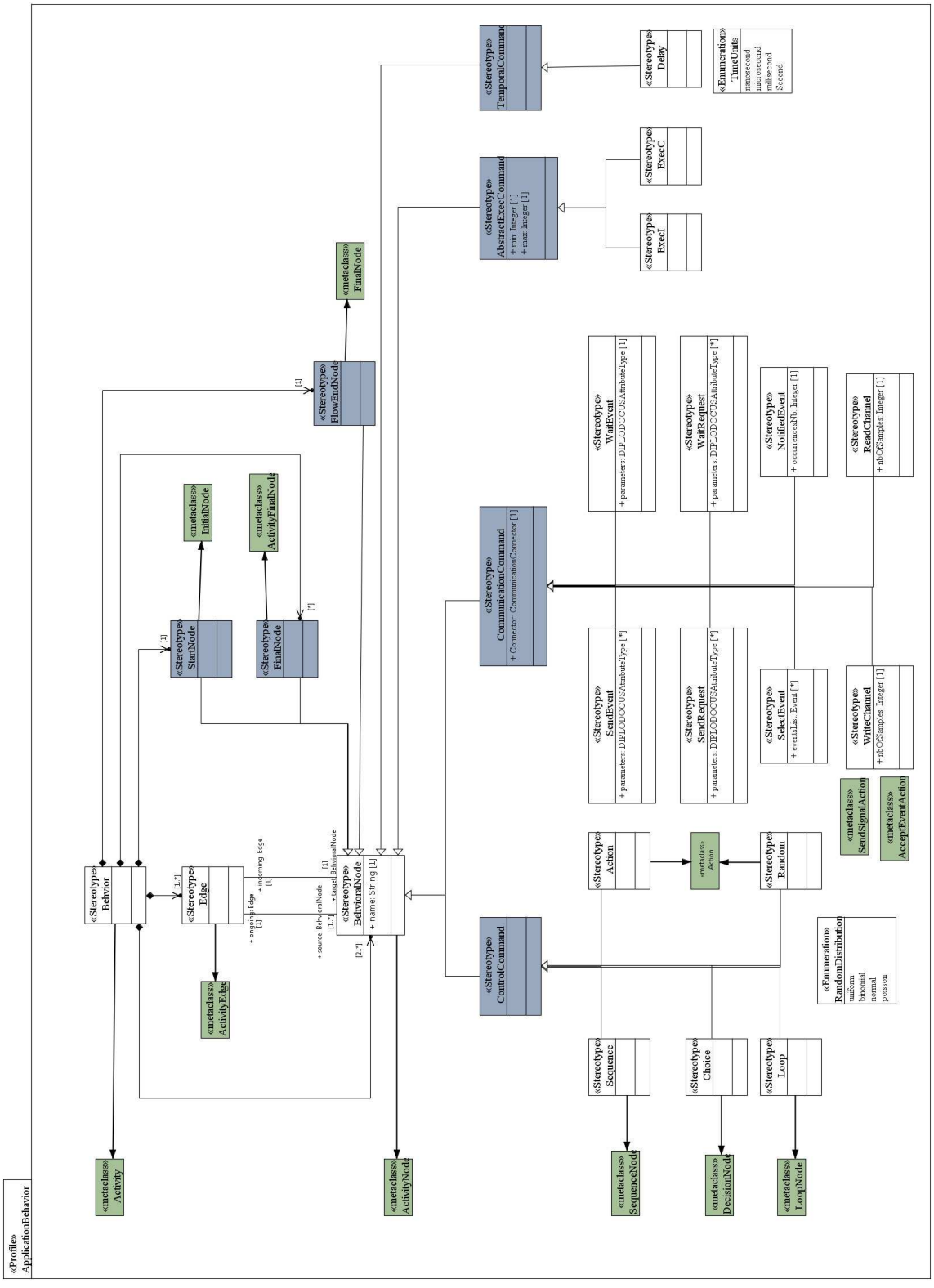

Figure 4.7: Application behavior metamodel

Application Behavior Example

Figure 4.8 depicts a simple application behavior example modeled with the TTool toolkit. In fact this is the activity diagram of the task "Task0" used in the example of section sec:AppTaskModelExample. It defines the sequence of commands that the task will perform during it's execution. The red box beside each of the DIPLODOCUS behavior command shows its type.

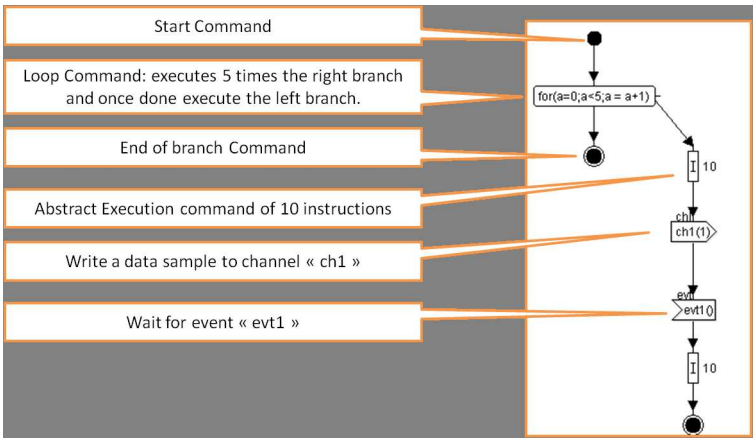

Figure 4.8: A simple application behavior modeled using the DIPLODOCUS Application Behavior profile

4.3 Architecture Modeling

A DIPLODOCUS Architecture is the set of hardware resources that will execute the application modeled as in the previous section. The "Architecture Modeling" sub-profile defines how a SoC architecture is modeled. Furthermore, section 4.3.2 introduces the concept of "Communication Patterns" to model the interactions that are handled by the architecture to perform communication operations. This modeling concept enforces the separation of concepts between execution and communication and enhances model's re-usability and correctness. The following sections describe in more details the architecture resources model and their interactions as captured by DILPODOCUS

4.3.1 Architecture Resources Model

In DIPLODOCUS an architecture is modeled as a network of **abstract** physical resources. The architecture model is not meant to execute real code, but rather the

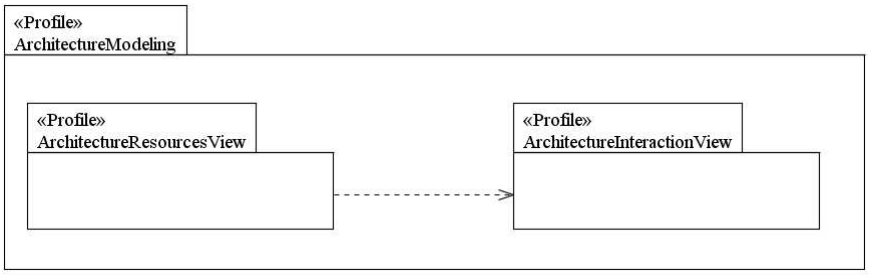

Figure 4.9: Architecture Modeling Profile

DIPLODOCUS commands as defined in 4.2.3. The architecture model need to account, only, for the timing parameters as the functional behavior is already captured by the application model which drives the simulation (the simulation process is described in chapter 6).

A DIPLODOCUS architecture *archi* is the composition of set of architecture nodes (that could be of three types: Computation, communication and storage node) and a set of deployment links that relate the nodes. In addition, the Designer can as well profit from the DIPLODOCUS architecture domain concept to structure more adequately his/her model. In fact architecture resources could be grouped into domains that are characterized by a clock speed. For example a mobile phone SoC architecture is composed of two domains: 1) the Application Processor (AP) domain that executes the designer's applications and runs the phone operating system and 2) the Baseband Processor (BP) domain that handles the used networking protocol (LTE, 3G ...). Moreover, the model could be developed by more than one designer, as each designer could model the domain of his expertise, thus specifying more accurately the architecture's parameters.

Computation Nodes

A Computation Node, *CN* represents a processing device capable of executing program code. Hence its fundamental service is to compute, it accesses the storage resources to store or load data for the executing tasks. DIPLODOCUS allows the modeling of two types of CN: CPUs and Hardware accelerators.

Each computation node has a set of parameters,*CNParam*. These parameters are strictly performance related as the objective of DIPLODOCUS is to estimate the performance on a high level of abstraction, and ultimately before the real code or the architecture are developed. This set of parameters contains the number of computation node cycles needed to execute an instrauction,*CPI*, the data and instruction cache miss rates (estimated by the designer) and the running frequency of the computation node.

These parameters are used later on, during the performance estimation phase, to

calculate the performance metrics of the system such as latency and throughput, more detailed analysis will be provided in section 6.3.2 of chapter 6

Cache Modeling

DIPLODOCUS is based on high level modeling of application, architecture and of their mapping. Thus, an accurate model of caches is not feasible because application details, such as instructions and data memory addresses, are not available (application instructions are abstracted), the designer is not aware of specific memory regions ... However, caches play a critical role in the system's overall performance. Hence, a methodology for system performance estimation must take the cache influence into consideration.

The solution adopted in this thesis is based on the experience of the designer. In fact, the designer estimates (based on her previous experiences) the cache miss rates (data and instruction miss rates) that will be encountered during the system's execution. These miss rates are used to calculate the system's performance during simulation. Thus, the calculated performance are function of the miss rates specified by the designer that could optimize them when more details, on system, are available. The miss rate specified at this level is to be considered as a requirement for the development team.

Communication Nodes

A Communication Node CommN represents communication device capable of transmitting data between the architecture nodes. Hence its fundamental service is data transfer. DIPLODOCUS allows the modeling of Busses, bridges, crossbars, and DMA . Their main objective is to carry data from/to computation nodes to the respective storage nodes. Designer should as well specify a set of parameters for each node, especially the data word size, the cost of the transfer of one word and the operating frequency

Storage Nodes

A Storage Node SN represents storage device capable of storing application tasks data. It could be on-Chip or off-Chip, shared or private, single access or dual access. They are accessible through the communication nodes. Storage nodes has the following parameters: Size (in MBytes), data word size, Latency, and frequency.

In DIPLODOCUS, architecture nodes are instantiated from a library of predefined abstract models for architecture nodes that can be customized by setting the appropriate performance parameters, thus reducing the modeling effort.

Architecture resources Model's UML Representation

Figure 4.10 presents the DIPLODOCUS "Architecture Resources" meta-model. The DIPLODOCUS architecture diagram extends the UML 2.0 deployment diagram. The centric stereotype is "Architecture node" that is the generic type of all nodes. These nodes are connected through architecture links. Even though not specified directly in the meta-model, but a bus can not be connected to another bus.

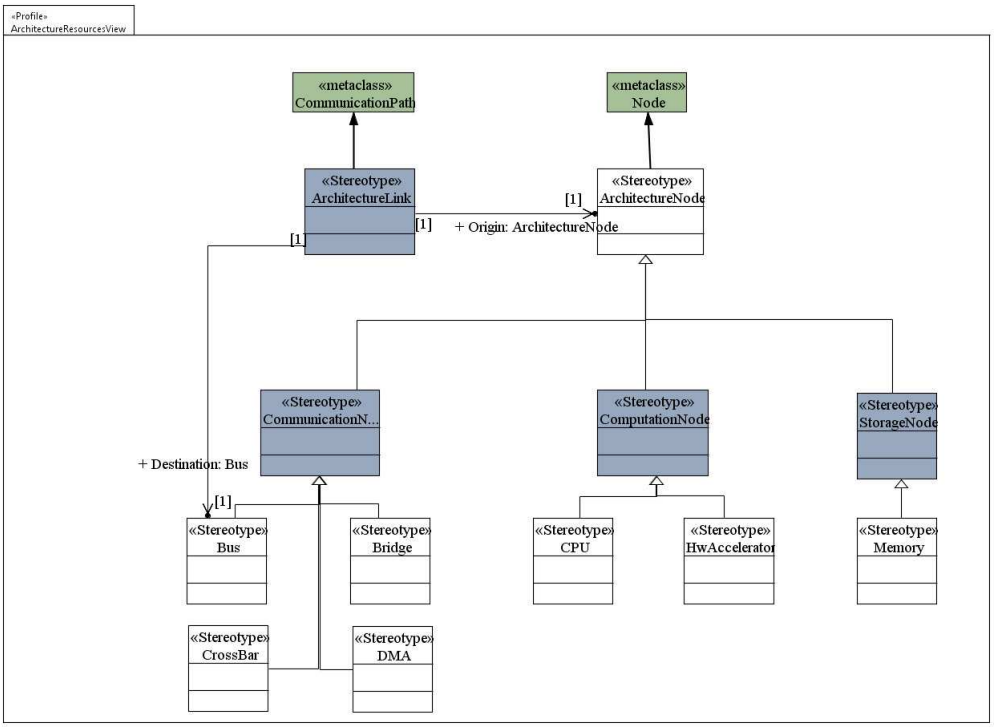

Figure 4.10: Architecture Resources Metamodel

Architecture modeling example

Figure 4.11 depicts an example of a DIPLODOCUS architecture model modeled using the TTool toolkit. In this example the architecture contains two computation nodes "Core1" and "Core2" that can access the storage resources "M3" and "DDR1" either directly or using the DMA1. The communication infrastructure includes in addition to the DMA a CrossBar and a bridge as DDR1 is an off-chip memory.

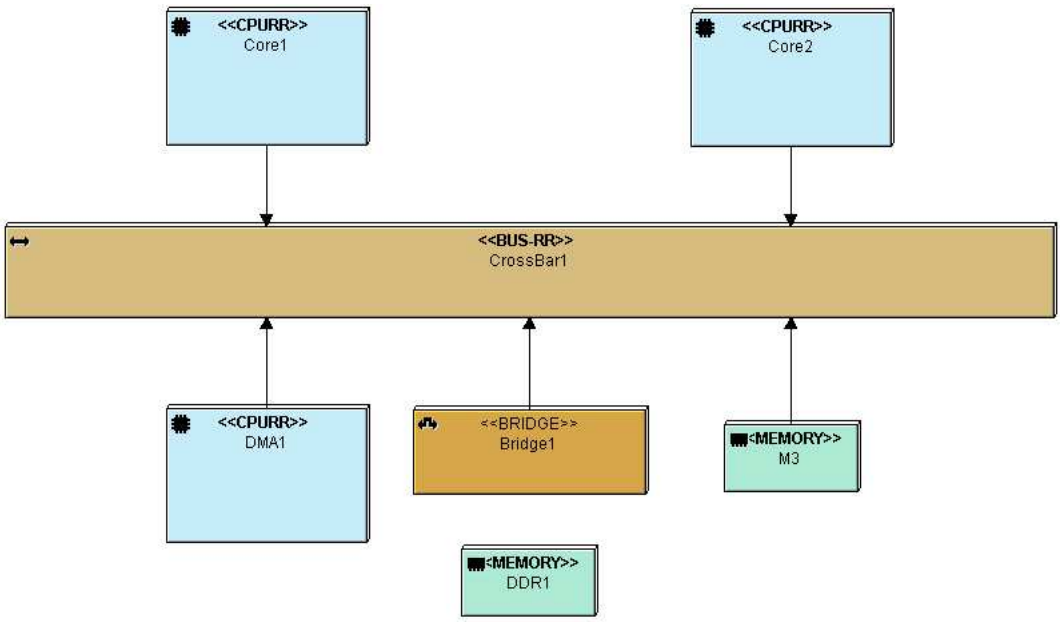

Figure 4.11: An example of a DIPLODOCUS Architecture Model with TTool

4.3.2 Architecture Communication Interaction Model: "Communication Patterns"

The execution of the application on the architecture induces data exchanges between architecture nodes. We distinguish two types of communication:

- Explicit Communications: results of the communication between tasks mapped to different computation nodes. Consequently, a message exchanged via the shared memory induces two explicit communications: one for writing into the shared memory (by the sending node) and the other for reading from the memory (by the receiving node).

- Implicit Communications: represent data and instructions that are fetched from the memory (shared or private) during the execution of a task.

When a computation node is transferring data (implicit or explicit) to a memory, the communication protocol is the same for all applications (or application's tasks). The communication cost depends on: 1) the amount of data to transfer, 2) the destination memory and 3) the set of communication nodes that will carry this data from the computation node to the memory. During this data transfer, the computation node may start the execution of another task while performing the data transfer requested by the previous one. For example in a DMA data transfer context, the computation node

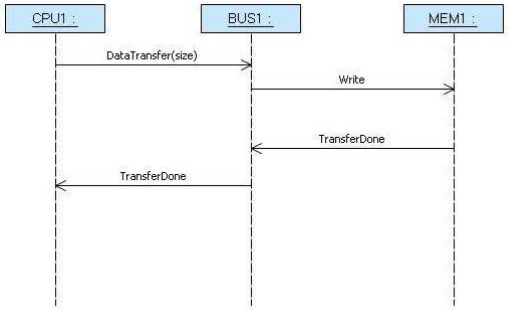

Figure 4.12: An example of simple Communication Pattern

programs the DMA for the data transfer and executes another task. We call "architecture communication behavior", the sequence of communication operations needed to complete a data transfer from a computation node to a storage node and involving one or many communication nodes (DMA, Bus, crossbar ...). One should note that different mechanisms may exist in an architecture to ensure the connection between a computation node and a storage node. In modern embedded systems, communication standards are used to define the set of interactions to perform to complete a data transfer: PCI express[65], AMBA bus[15] and Avalon[13] are examples of these communication standards. Our objective in DIPLODOCUS is not to model these standards in details and on low level of abstraction, but rather to capture the influence of communication interaction on the overall system's performance while preserving the high level of abstraction.

Keeping in mind, the objective of separating the concerns of the computation and the communication, we need to provide means to efficiently model the "architecture communication behavior" and to capture its influence on the overall system's performance. To address this issue we introduce the concept of "communication patterns", that defines a scenario of communication that ensures the data transfer between a computation node and an storage node. It should define all the exchanges (interactions) needed between the architecture nodes (CPUs, busses, Crossbars, Memories ...) involved in this scenario. An UML sequence diagram is traditionally used to represent this kind of behavior. We extended the traditional UML sequence diagram to model the communication patterns. A simple example of a communication pattern is shown in Figure 4.12. where each vertical lines (lifelines in UML notation) represents one architecture node, and each arrow represents a message exchange between two architecture nodes.

A Communication Pattern (CP) is defined as a tuple

$$CP = (N, c_0, s_0, E, Ops, \prec_E) \tag{4.4}$$

Where

- *N* is a set of the architecture nodes that compose the pattern
- c_0 is the execution node that start the communication interaction
- s_0 is the destination storage resource where c_0 wants to read/write data
- *Ops* is the set of all the operations that should be executed by all nodes of N during the communication scenario.
- *E* is the set of events that composes the operations of Ops. An operation can be a set of events.
- $\prec_E$ is a partial order relation between events of *E*. The order is partial between events belonging to operations on different nodes, but it is total between the events on the same node.

Nodes are categorized into sub-sets of "Computation Nodes" (CN), "Communication Nodes" (CommN) and "Storage Nodes"(SN), so that

$$N = \{CN \cup CommN \cup SN\} \tag{4.5}$$

The c_0 and s_0 nodes should be of types Execution Node (EN) Storage Node (SN) respectively, as defined in proporties (4.6) and (4.7)

$$c_0 \in N \; and \; type(c_0) = CN \tag{4.6}$$

$$s_0 \in N \; and \; type(s_0) = SN \tag{4.7}$$

The communication scenario starts when a computation node decides to perform a data transfer to a storage node (explicit or implicit communications). The communication pattern involves only **one** computation node (e_n) and defines one of the possible scenarios to access the specific storage node (s_n). The set E represents the events that nodes of N should perform. An event is related only to one node so that a message between two nodes involves two events: one for sending the message and for receiving it. An event evt∈E is defined as follow

$$evt = (node, type, param) \tag{4.8}$$

Where node represents the architecture node that will execute the event, type represents the functionality of the event (detailed later) and param is an integer variable that is needed for sending/receiving of the event. An operation in a communication pattern may involve at least one event (on one node) and at most all the events (on all the nodes)

In our Communication pattern, we assume (which is generally the case in real implementation) that the computation c_0 node should start the communication and though the first event in a communication pattern should be executed by the only computation node of a pattern.

We identified three categories of the "type" of an event "e": "Message Events"(ME), "Execution specification Events" (ESE) and "Conditional Events"(CE).

$$\forall evt \in E \Rightarrow type(evt) \in \{ME, ESE, CE\} \tag{4.9}$$

Message Events

A Message Event (ME) represents a communication (data, event) between two architecture nodes: an origin node, and a destination node. A Message operation (MO) involves two Message Events: the sender and the receiver, and is defined as

$$MO = \{evt1, evt2\} \tag{4.10}$$

Where evt1 is the sender and evt2 is the receiver. We define four message events: "Read", "Write", "DataTransferRequest" (when a CPU programs a DMA transfer for instance) and "TransferDone" to signal the successful completion of the communication. These operators could be **synchronous** or **asynchronous** in the sense that a computation node with synchronous messaging is blocked until the end of a Message Operation . In contrast, an asynchronous messaging permits a non blocking execution scenario, during which the computation node could execute another task while performing a communication operation (DMA transfer, caches ...).

Execution Specification Events

The second "type" of events is the Execution Specification Events. these events are meant to model situations where nodes could set some architecture specific parameters (e.g., a CPU needs some cycles to program a DMA). This consumed time is totally independent from the application. We define two execution specification operators: "Exec" which represents, in number of cycles, the cost of an execution on a computation node related to communication protocol, and "Delay" which is a time interval. This type of operators involves only one architecture node. An Execution specification Operator is composed of only one Execution specification event.

Conditional Events

Finally the third "type" of events is Conditional Events using the "loop" and "if". This event type introduces a conditional behavior to the communication pattern. It represents a set of events that are executed if and only if the condition is satisfied.

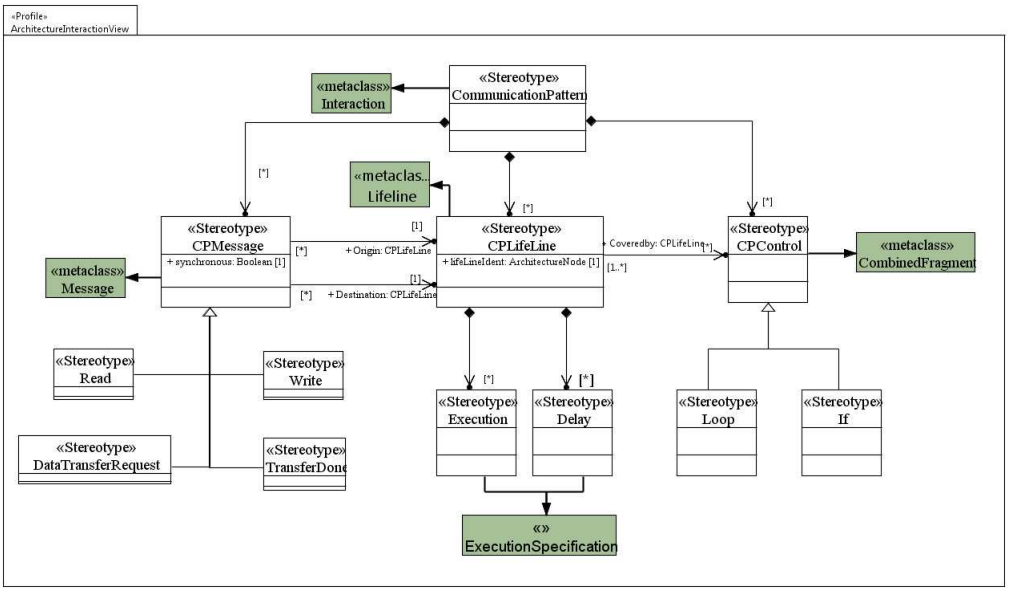

Figure 4.13: Communication Pattern metamodel

Communication Patterns' UML Representation

The DIPLODOCUS communication pattern sub profile meta-mdodel is depicted in figure 4.13. It extends the UML sequence diagram and define all the possible message that could be exchanged between architecture nodes (as described in the above section).

4.4 Summary

The objectify of this chapter is to define the application and architecture modeling in DIPLODOCUS. The starting point of our work was the DIPLODOCUS methodology introduced in [14] and [82]. We enriched the existing work by providing a mathematical and a UML definition for the application and the architecture modeling. Furthermore we added to the DIPLODOCUS methodology the component modeling diagrams to reduce the models complexity and increase their re-usability.

Another major contribution of this chapter is the communication patterns that enforce the separation of concerns of the computation and the communication in the architecture. The next chapter will illustrate the system mapping (application mapping onto the architecture) with special focus on shared resources modeling, communication management and Memory mapping.

Chapter 5

System Mapping Modeling

After defining the application and architecture modeling in the previous chapter, this chapter describes the mapping process. As one of the foundations of the DIPLODOCUS profile is the separation of concepts between the application and the architecture. A mapping phase is needed to define the binding of the concepts of the application to the concepts of the architecture. the result of this process is a system on which performance analysis will be performed.

The proposed DIPLODOCUS mapping model emphasizes on the resources sharing and possible impact on overall performance metrics such as latency, throughput and resources utilization. It introduces the *"Virtual Node"* (VN) concept to model the shared resources access control. Furthermore, it defines *"communication managers"* (CM) that will use the communication patterns defined in the previous chapter, and define how after mapping a computation resource will access a storage resource to read/write data related to an application task. The overall modeling methodology is applied to a simple example to illustrate the modeling concepts presented so far. Before describing the mapping process, this chapter starts with a simple example that shows the needed concepts to be adopted in a modeling methodology to map the application onto the architecture and to estimate the system's overall performance.

5.1 Mapping motivational example

Consider a simple application example (upper part of Figure 5.1) composed of four tasks: T1, T2, T3 and T4. T1 exchanges data with T4 through the channel "channel1" and T3 exchanges data with T2 through channel "Channel2". T1 and T4 are mutually dependent as their execution depends on the communication between them. Same for T3 and T2. The group (T1,T4) can execute in parallel (concurrently) with the group (T2,T3). This is due to the fact that there is no data exchange (Channels) or synchronization operators (Events or Requests) between the two groups.

This application will execute on an architecture (bottom part of Figure 5.1) composed of two CPUs (CPU1 and CPU2), one DMA (DMA1), one Crossbar (CrossBar1), an on-chip memory (M3) and an off-chip memory (DDR1) connected to the other architecture components through a bridge (Bridge1). This is the same architecture used in the example in section 4.3.1 (the architecture example in chapter 4). The two CPUs with the DMA share the crossbar to access the memory M3 and share the bridge and the crossbar to access the DDR.

The doted arrows in figure 5.1 depicts a **"mapping scenario"** of the application to the architecture, where:

- T1 and T2 are mapped onto CPU1, while T3 and T4 are mapped onto CPU2.
- The "Channel1", representing the data exchanged between T1 and T4. This data exchange is done using the shared M3 memory. "channel2", representing the data exchanged between T2 and T3 using the DDR1.

Thus the first step in the mapping is the **"allocation"** of the architecture resources to the applications tasks and channels. In the mapping scenario presented above, when T1 and T4 exchange data through "Channel1", the data will be transfered to/from the memory M3 through the CrossBar1. In a similar way when T2 and T3 exchange data, the data will be transmitted using the CrossBar1 and the Bridge1 to reach the memory DDR1.

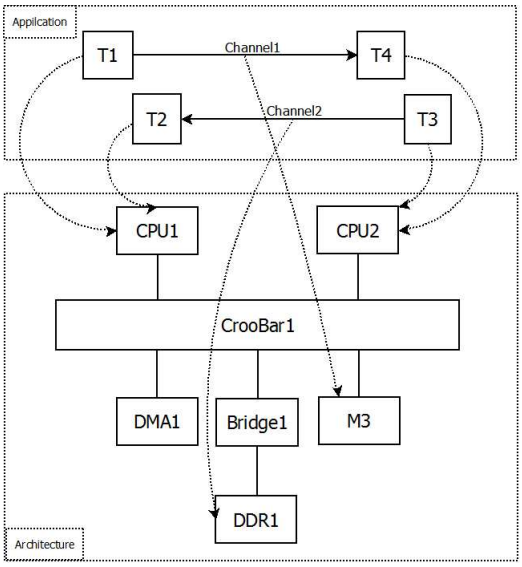

Figure 5.1: Mapping Demonstrative example

As application's tasks can run **concurrently**, an arbitration policy is needed for each CPU to select a task to execute among the ready ones. For instance, if the scheduling policy of CPU1 is priority based, and T1 is of higher priority than T2, then T1 will

be chosen to execute first. In this case, T2 execution is delayed until T1 finishes its execution, including an additional cost due to communication with T4 (access cost to the crossbar and to the memory). This concurrent execution of tasks is translated into a concurrent access to the communication and storage nodes. Thus, a data exchange operation between two tasks, for example T1 writes data samples in Channel1 with destination T4, is translated after mapping as a communication between the two CPUs, CPU1 and CPU2. This communication involves two shared architecture nodes, the crossbar and the memory M3 (as "Channel1" is mapped to M3) whose access is determined, as well, with access policies. Thus, it is primordial to define the resource **sharing** techniques to capture correctly the system's execution.

It is possible as well to imagine a communication scenario where the data is transfered using the DMA. This means that the architecture provides two different paths to transfer data between the CPUs and the memory M3: first path is direct through the crossbar to M3, the second path involves a DMA data transfer. Hence, in a mapping methodology, there should be a modeling component that defines how to **manage** the communication in that case.

The overall latency of application's execution on the architecture is equal to the sum of:

1. Execution cost of the application functionalities (Tasks) on the computation resources

2. Communication cost due to storage nodes access through the communication nodes

3. Latency due to the contention on shared nodes. The contention's cost depends on the shared nodes involved in the execution and on the communication path used for the data transfer.

Chapter Outline

The objective of the DIPLODOCUS Mapping modeling Profile (depicted in figure 5.2) proposed in this chapter is to define how the architecture resources are allocated to the application constructs. The mapping modeling profile is composed of four sub-profiles:

- Shared Resources Modeling (section 5.2): it defines the access control to shared nodes. This control provides the designer ideas about bottle necks in the architecture, suitable scheduling policies on Computation nodes, suitable arbitration policies for communication nodes ...

- Execution Allocation (section 5.3): it is a allocation function that specifies for each task the computation node on which it will execute

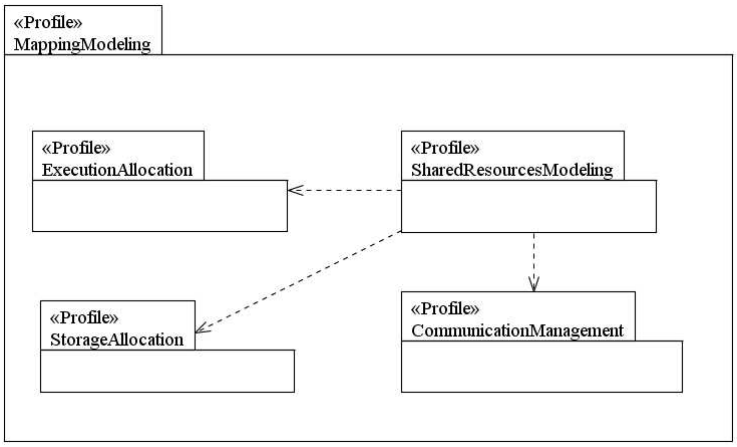

Figure 5.2: DIPLODOCUS Mapping Modeling profile

- Storage Allocation (section 5.4): it is an allocation function that specifies on which storage node the task's data and code are stored. It defines as well where the application channels (representing data exchange) data is mapped.

- Communication Management Modeling (section 5.5): it controls how a computation node after mapping will transfer data to storage nodes, using the communication patterns introduced in the previous chapter (section 4.3.2 of chapter 4). It defines as well a policy to choose, when possible, among different available paths to transfer the data (like the direct or DMA communication scenarios presented in the above example).

The following sections will define each one of these sub-profiles with more details. In addition, the example introduced in this section will be furthermore enriched with the mapping modeling concepts.

5.2 Shared Resources Modeling

Application execution on top of the architecture will result in a concurrent access to shared resources, thus a control access must be applied. In fact, in a modern SoC, heterogeneous applications are running concurrently; they share the same computation resources. It is common as well to have more than one computation resource (CPUs, DSPs, Hardware accelerators, ASIP, etc.). These computation resources may share the same communication and storage resources. Operating systems are controlling the computation resources shared between the different applications; bus arbiters are controlling the bus shared between different computation resources, and memory con-

trollers manage concurrent memory accesses. Thus providing a definition of shared resources and of their access policies is primordial to accurately model the system.

5.2.1 Resource definition

A resource by definition is any physical or virtual entity that provides services (computation for example) to accomplish an activity and achieve desired outcome (a task execution for example). In the DIPLODOCUS context, architecture nodes are **physical** resources that will enable the correct execution of the application and they are of three types: Computation, Communication and storage resources.

These **physical** resources are shared to optimize their utilization, and are accessed through resources requests. In fact, during the execution of the application on the architecture, three types of resource's requests (not to be confused with the *Request* connector used in the application) can be identified:

- Computation requests, *ComputationReq*, that are generated by application tasks to execute on computation nodes (e.g. CPUs).
- Communication requests, *CommReq*, that are generated by computation nodes to the communication nodes (e.g. Bus) in order to transfer data generated/requested by tasks.
- Storage requests, *StorageReq*, that are generated by computation and communication nodes to storage nodes (e.g. memories)

A Resource Request should specify as well the:

- *priority* that is used, when the resource is shared using a fixed priority based access policy. When dynamic priority scheduling is used, the priority is dynamically calculated and this parameter is not specified.
- *amount* that specifies the resource's amount that the requester needs. For instance a computation node would request to write a data *amount* of size x to a storage node.
- *type* of the request as identified above.

As each resource can be shared between different requesters, resources should have an access policy that selects a request among pending ones. The next subsection introduces the virtual node concept to control shared resources.

5.2.2 Shared Resources' Control: The *"Virtual Node"*

The *''Virtual Node"* (VN) is defined as a generic modeling component that controls the access to a resource by implementing an "access policy". It allocates the controlled **resource** to a *requester*, for example the VN of a CPU allocates the CPU to a task that is ready to execute, or the VN of a bus allocates the bus bandwidth to a CPU that is trying to reach the memory or other architecture nodes that are connected to the bus. The Virtual Node has a three-steps execution semantic:

1. It waits for incoming Resources requests. The VN has a queue to store the requests.

2. It selects a request among the possible ones, according to its **allocation policy**. After the expiry of a switching delay (Context switch), the resource is allocated to the requester.

3. It waits either for the selected request to finish its execution or for a new incoming request. In that latter case, the allocation is re-evaluated (step 1).

In addition the virtual nodes has a type. It inherits the type of the resource it controls. Hence, a Virtual Node could be a computation, communication or storage VN. By controlling a shared resource, the virtual node divides the resource between the requesters as if it creates a **Virtual** resource for each one of them. This virtualization function permits the definition of classes of policy accesses, for instance, a computation node can be virtually allocated to two separate applications by using a time sharing policy (as detailed in subsection 5.2.2), or many cores can be grouped into one virtual resource in a dynamic scheduling scenario (as detailed in subsection 5.2.2). To control both types of resources (physical and virtual), there is two categories of virtual nodes: Local Virtual Node (*LocalVN*) that controls the access to a Physical Resource and a Generic Virtual Node (*GenericVN*) that controls a Virtual Resource.

GenericVN can be stacked hierarchically, in such a way that a GenericVN could be connected to another GenericVN that is connected to another GenericVN or a Local VN. However, a GenericVN could be connected to only one LocalVN. Furthermore, each physical resource is controled by one and only one Local virtual node. To further illustrate this stacking mechanism, the following two sub-sections provide two examples on how to use the concepts introduced above on two well known resources sharing that exists in the domain: the hierarchical and dynamic scheduling scenarios.

Hierarchical scheduling

Embedded systems can concurrently execute different real-time heterogeneous applications. For instance in a modern mobile device multimedia application such as video

or audio could execute concurrently with control applications (telecommunication protocols). These applications may have specific scheduling requirements (soft real time, intensive data transfer or execution loop, etc). Furthermore, applying one access policy to all applications is not the optimal solution [74]. Using the Shared Resources Modeling (SRM) presented in the previous section, designer can use a hierarchical stacking of virtual nodes to optimize resources sharing when heterogeneous groups of requesters request the resource. A main virtual node controls a hardware resource and a secondary virtual node controls each group of requesters. This approach allows us to optimize the access policies to satisfy requirements of all groups. This hierarchical composition of virtual nodes can be used for computation, communication and storage resource sharing.

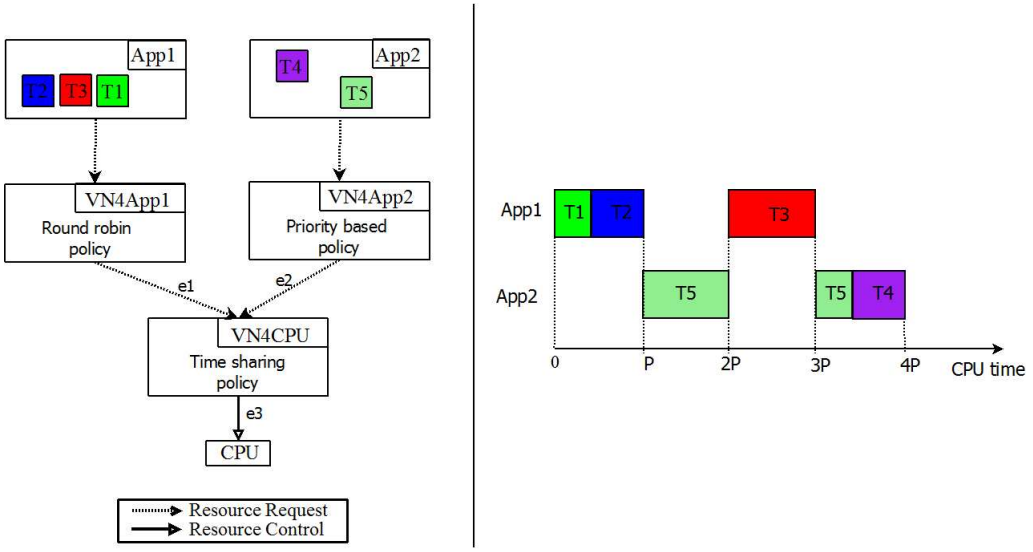

Figure 5.3: A simple hierarchical scheduling example

The left part of figure 5.3 shows an example of a hierarchical scheduling of two classes of applications; "App1" is controlled using a round robin policy while "App2" is controlled by a priority based policy. The CPU is shared between the two applications by a time sharing policy implemented by the main VN (LocalVN), VN4CPU. It allocates a time slot of the CPU time to each one of the two applications. The generic virtual Nodes VN4App1 or VN4App2, controlling respectively the applications App1 and App2, allocate the available execution time (a time slot) to one or more tasks depending on its access policy.

The right part of figure 5.3 shows an execution scenario of this hierarchical scheduling model. The CPU's time is divided into periods "P", using the "Time Sharing policy" of the local virtual node. In this example each application gets half of the CPU's time. The generic virtual nodes choose which tasks of the applications will execute

during the available execution time.

Dynamic scheduling

In modern SoCs, it is common to find multi-core processors and/or multiple computation resources. To optimize the utilization of these resources dynamic scheduling techniques are generally used. All tasks compete for execution on all processors and a global scheduler controls the set of available cores (a virtual resource) and dispatch the ready for execution tasks on the available cores using it is defined access policy. Each one of the local cores is controlled by a Local Virtual node that will define which one of the dispatched tasks will execute at an instant t.

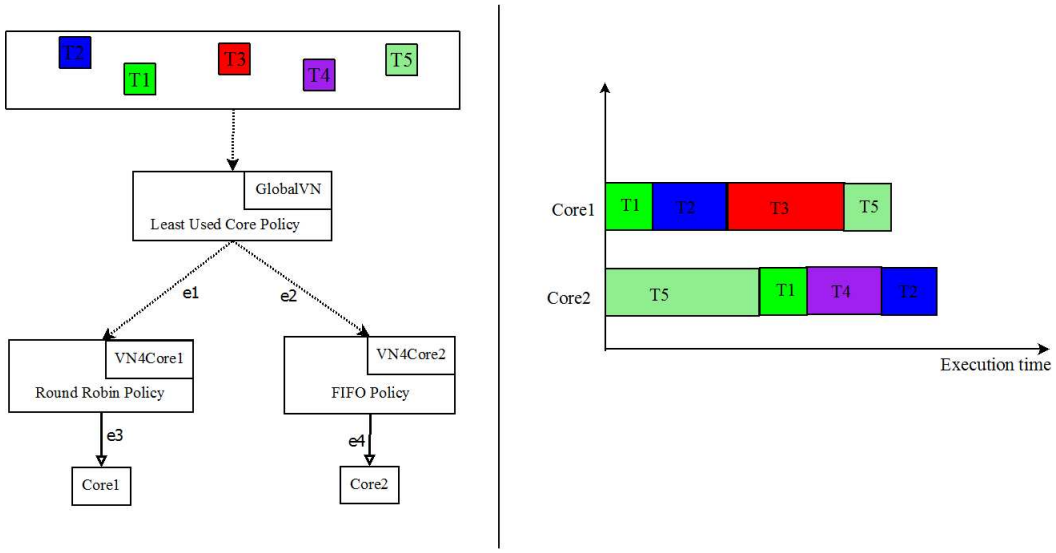

Figure 5.4: A simple dynamic scheduling example

Figure 5.4 shows an example of a global Virtual Node (GlobalVN) that is dipatching the application's tasks (T1 ... T5) on two local Virtual Nodes (VN4Core1 and VN4Core2), each one of them is controlling a core (Core1 and Core2 respectively). The global virtual node uses a "Least used Core" policy, where it dynamically forward the incoming requests from application tasks to the Core that is available or is less used. The time chart on the right part of the figure shows the dynamic execution of the tasks on the cores. T1 is first executed on Core1 but on its second execution it was executed on Core2, as the later was available while Core1 was executing T3.

5.2.3 Virtual Node vs Real Implementation

The virtual node concept ensures the modularity of scheduling. By simply changing the access policy of a virtual node, the designer can evaluate the impact of the new ac-

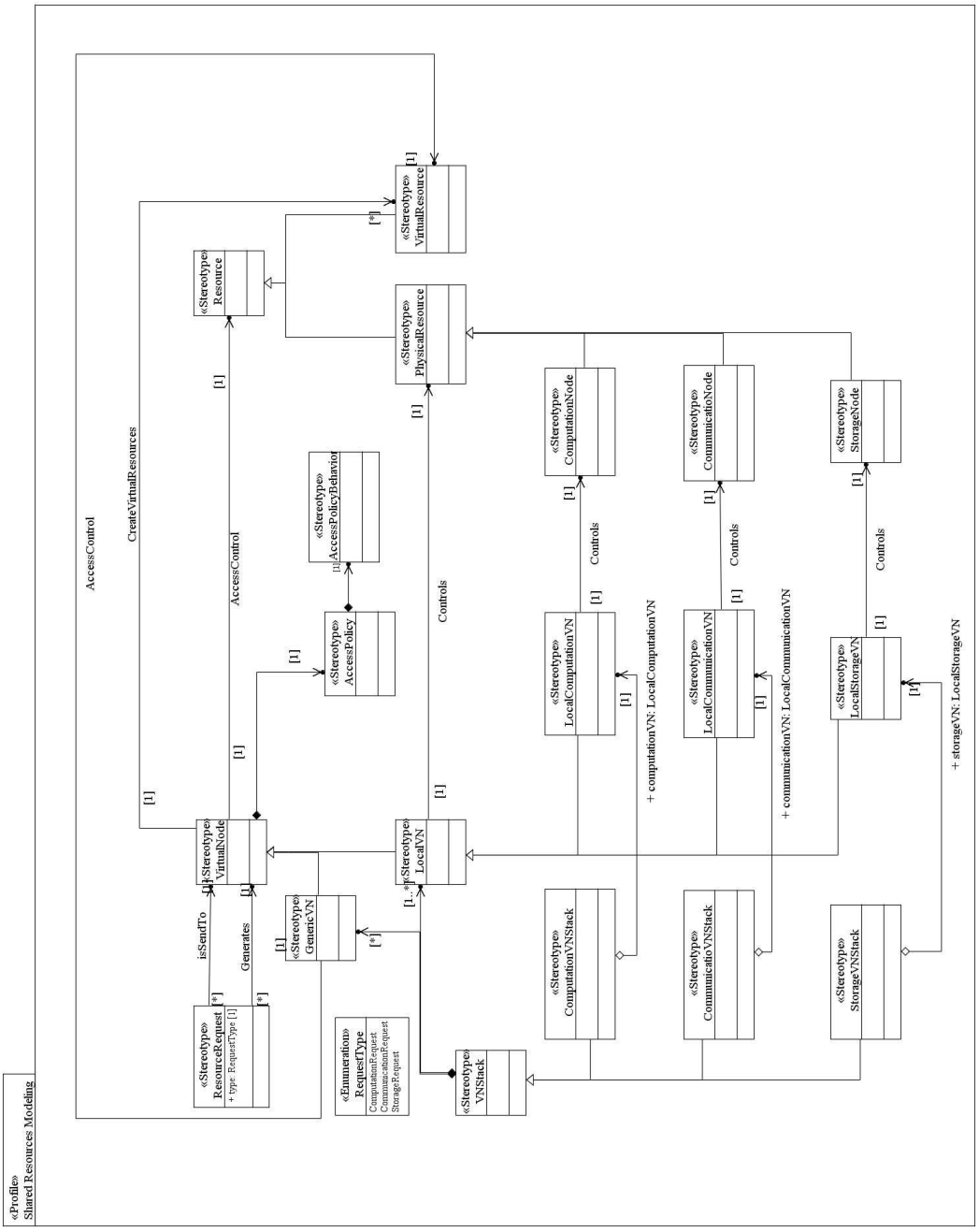

Figure 5.5: DIPLODOCUS Shared Resources Modeling profile

cess policy on the overall system. However, one may think: What is the equivalent of a virtual node and of the shared resources model in a real implementation? and the answer is that a virtual node is the abstract model of scheduler for computation resources, and of arbiter for communication resources, and of memory controller for storage resources. Which means, that it abstracts both software and hardware access policies. Finally, one can think of the virtual node as the scheduling part of the operating system that will run the architecture.

UML Representation

The DIPLODOCUS Shared Resource metamodel is depicted in figure 5.5. This metamodel reproduces the concepts defined earlier with UML constructs. The Virtual Node is materialized by the stereotype *"VirtualNode"* which extends the the UML metaclass *"artifact"* that will be attached to an architecture node in the architecture deployment diagram. The *"VirtualNode"* owns an *"AccessPolicy"* that in it turns has an *"AccessPolicyBehavior"*. This latter will define how the virtual node will choose, among the requesters, the one who will access the resource.

The virtual node can control Physical and virtual resources resources. Thus, two sterotypes: *"LocalVN"* and *"GenericVN"* extend the virtual node to control respectively the physical and virtual resources.

To model the hierarchical and dynamic scheduling scenarios, virtual nodes (Local and generic) are stacked; and generic virtual nodes controls the virtual resources created by the local virtual nodes. The stereotype *"VNStack"* models these scenarios. It models a set of Generic and Local virtual nodes.

5.3 Execution Allocation

Each computation node is controlled by a local computation virtual node, that in its turn may create computation virtual resources (for example the case of hierarchical and dynamic scheduling scenarios presented in the previous section). These virtual resources will be controlled by generic computation virtual nodes. The *Execution Allocation view* of the DIPLODOCUS mapping profile bind a task to a computation virtual node (generic or local). Each task is in concurrence with the other tasks mapped on the same virtual node to access the computation resource.

The Task Execution Allocation is set of mappings of all application's tasks to computation virtual nodes. For each task t, a *taskExecAlloc* function is defined. This function specifies the virtual node that will control the execution of the task t by allocating the computation resource using its access policy and the priority that is specified when the resource is shared using a fixed priority based access policy. When dynamic priority scheduling is used, the priority is dynamically calculated and this parameter is

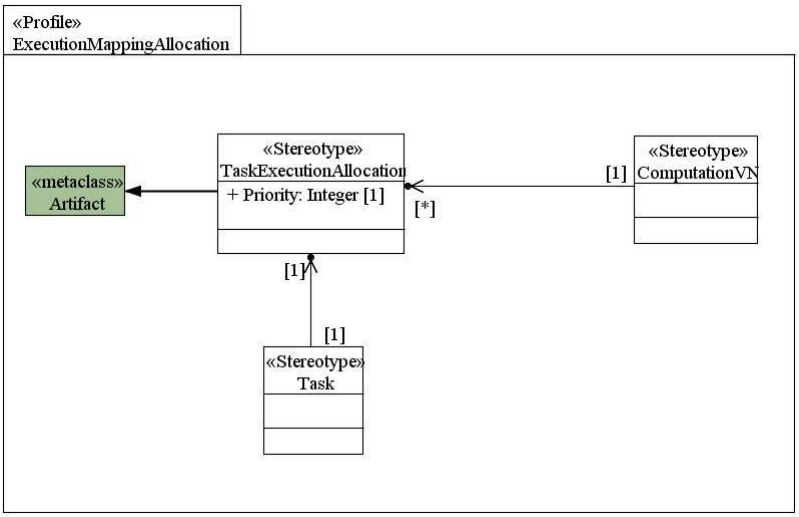

Figure 5.6: DIPLODOCUS Execution Mapping profile

not used.

Example Consider the mapping scenario of the simple example of the section 5.1. Tasks T1 and T3 will execute on CPU1 and CPU3 respectively. The task T1 is mapped to the virtual node controlling the CPU1 with a priority 2, and task T3 is mapped to the virtual node controlling the CPU2 with a priority 1. The Execution Allocation of this application is: *ExecutionAllocation* = $\{\{T1, VN4CPU1, 2\}, \{T3, VN4CPU2, 1\}\}$

UML Representation

The DIPLODOCUS Execution Allocation profile is depicted in figure 5.7. The main stereotypes defined is: *TaskExecutionAllocation*, it extends the UML metaclass *artifact* of the UML deployment diagram. The Execution allocation in DIPLODOCUS will be performed on the computation virtual nodes (Local or generic).

5.4 Storage Allocation

In the application modeling presented in section 4.2 of Chapter 4, *"tasks"* exchange data through *"channels"*. After mapping, this data is stored on shared memories. In addition, tasks have their code and data stored on storage nodes. The *Storage Allocation View* of the DIPLODOCUS mapping profile represents the memory distribution of application's tasks and channels on Storage nodes.

The Task Storage Allocation, *TaskStorageAllocation*, is a set of all the mappings of all the application tasks' code and data to storage nodes. For each task the *taskStor-*

ageAlloc defines on which memory the code and data are mapped. Furthermore it defines*CodePer* and *DataPer* are integers, specifying respectively the percentage of Code and of data of the task *t* mapped on *sn*. A task's code and data could be dispersed on different memories.

Example Consider the simple application introduced at the beginning of this chapter. The tasks T1 and T4 are exchanging data through the channel Channel1. T1 executes on CUP1 while T4 executes on T4. The DDR1 and M3 are two accessible shared memories for both CPUs. Let's consider in this example that the code of the task T1 is stored in the memory M3 while the data is distributed on both memories M3 and DDR1 equally. The same distribution is applicable for task T4. The Task Storage Allocation of this application is: *TaskStorageAllocation* = $\{\{T1, M3, 100, 50\}, \{T1, DDR1, 0, 50\}, \{T4, M3, 100, 50\}, \{T4, DDR1, 0, 50\}\}$

The Channel Storage Allocation, *ChannelStorageAllocation*, is the set of the mappings of all application channels to storage nodes. Channels carry data samples between tasks, and this samples are stored on shared memories. For each channel the *channelstorageallocation* specifies the memory where the channel data is stored

Example In the previous example, consider that the data exchanged between tasks T1 and T4 is stored on the memory M3. The Channel Storage Allocation of this application is: *ChannelStorageAllocation* = $\{Ch1, M3\}$

UML Representation

The DIPLODOCUS Storage Allocation profile is depicted in figure 5.7. This metamodel takes the mathematical concepts defined earlier in this section and reproduce them in UML constructs. Two main stereotypes are defined, the *TaskStorageAllocation* and the *ChannelStorageAllocation*, to model the constructs introduced in definition 8 and definition 9. Both stereotypes extends the UML metaclass *artifact* of the UML deployment diagram. The storage allocation in DIPLODOCUS will be performed on the architecture diagram that extends the UML deployment diagram. The multiplicity associations *taskStorageMapping* and *channelStorageMapping* specify that a task or a channel's data can be stored on at least one memory.

5.5 Communication Management Modeling

The Communication Manager (*CM*) is a mapping component that controls the communication between a computation node and a storage node. A Computation node's *CM* has a list of all the **communication patterns** (as defined in section 4.3.2 of Chapter 4) that the computation node can use. This list permits to the CM to know all the stor-

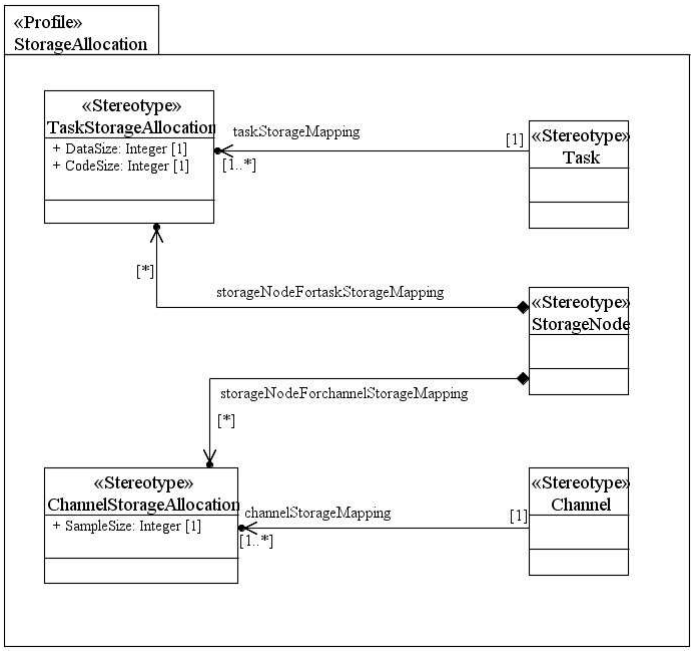

Figure 5.7: DIPLODOCUS Storage Mapping profile

age nodes that the computation node can access. In addition, the CM has a selection function that selects the communication pattern to be used if more than one pattern are available. The selection function could be for example: select the pattern with the least contention, or select the pattern with the highest data rate. It could be as well fixed by the designer, where only one specific pattern is always used.

UML Representation

The Communication management concepts are represented as well in the "DIPLDOOCUS Communication Management" profile (depicted in figure 5.8). The stereotype *CommunicationManger* is attached to Computation node. It owns a list of all the communication patterns available for the node, represented by the UML attribute *patternsList*. The operation *PatternSelect* defines how the computation node will transfer the data to destination using the available communication patterns.

5.6 Mapping Validation

The DIPLODOOCUS mapping in four views as defined in the previous section, takes in consideration the shared resources and communication management aspects and binds the application constructs to the architecture resources. In order that the map-

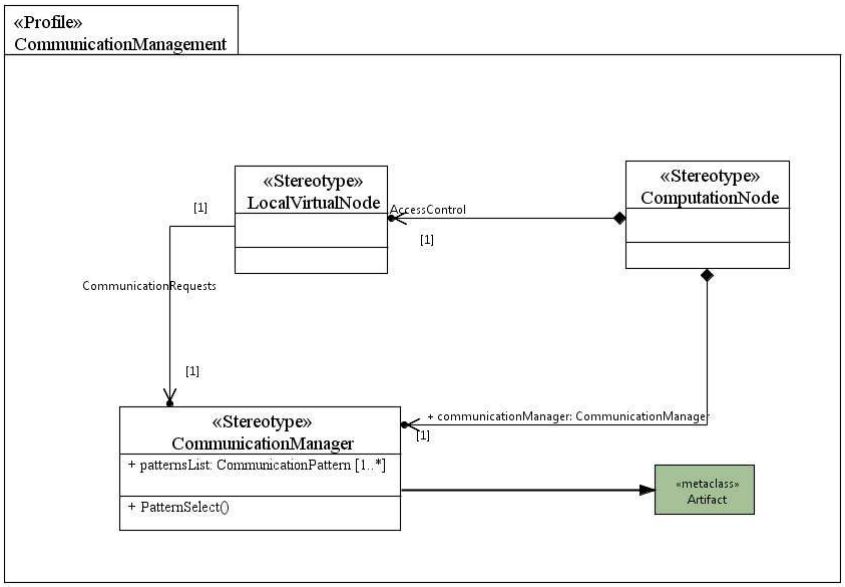

Figure 5.8: DIPLODOCUS Communication Management Modeling profile

ping performed is valid, the following properties should be satisfied:

Property1: if two communicating tasks T1 and T2 are mapped on two different computation nodes, then these computation nodes should at least have one shared memory on which the channels are mapped. In other words, in the communication patterns lists of the *Communication managers* of both CPUs there should be a pattern where the destination is the memory where the channel is stored.

Property2: If a task t is mapped on a computation node, and its data are on a memory M, then the *Communication Manager* of this computation node should have in his communication pattern list a *Communication Pattern* that has as destination the memory M.

5.7 Mapping overall scenario

Once the designer(s) completed his system model and specified the four views of the mapping, as defined in the earlier sections. The system is now ready for analysis through simulation (as defined in chapter 6) and/or formal verification (using the UPPAAL and LOTOS specifications generated automatically from the DIPLODOCUS UML models using teh TTool toolkit [63]).

Figure 5.9 depicts one possible execution scenario of the system after mapping. This scenario is as follow:

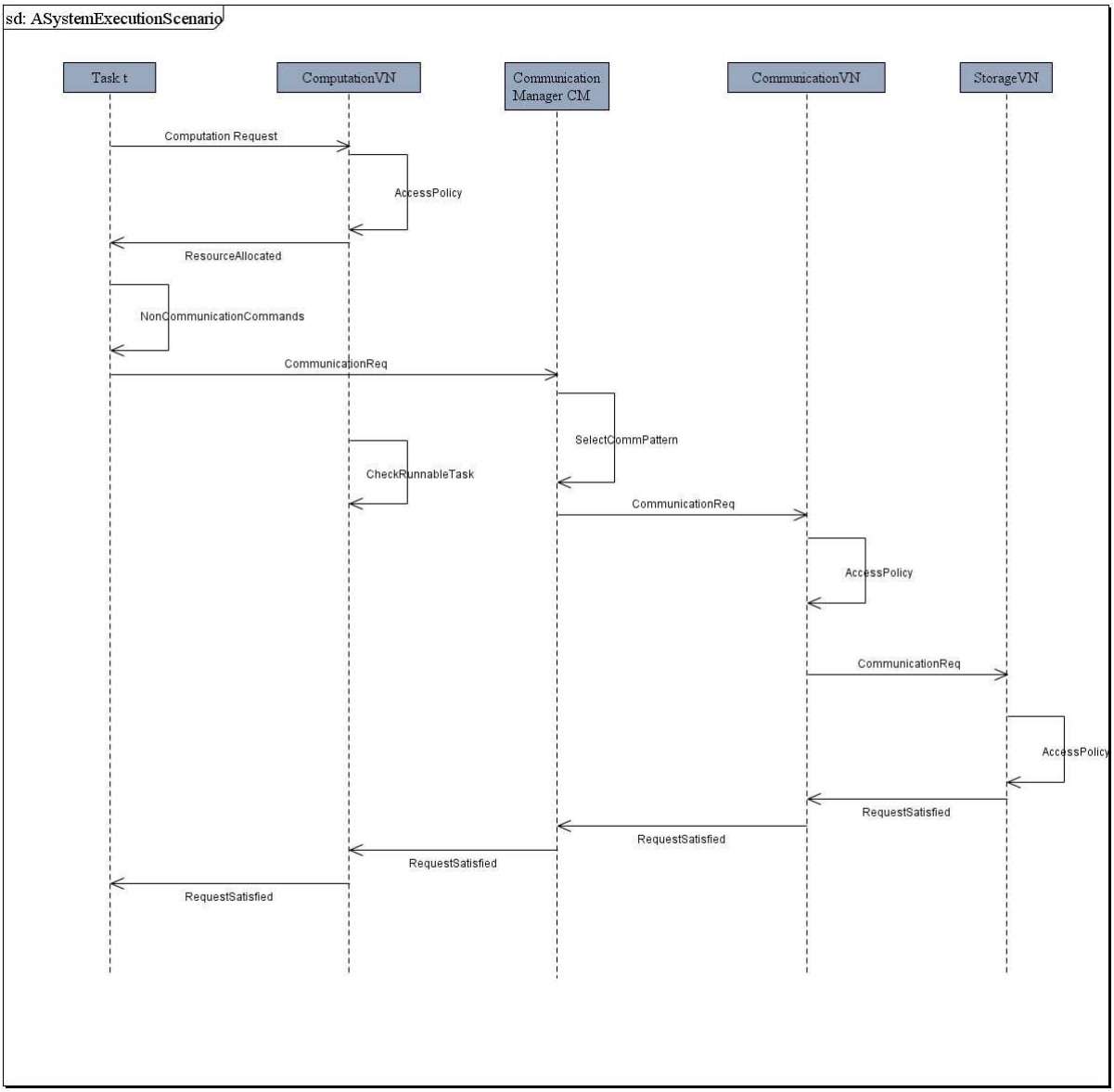

Figure 5.9: System Execution scenario after mapping

1. A ready for execution task, *t1*, will issue an execution request to the computation virtual node *ComputationVN* where it is mapped (As specified in the Task Allocation Mapping in section 5.3)

2. The virtual node, through its defined access policy, will decide if it will allocate the resource (the computation resource) to the requesting task *t*. It may interrupt the execution of another task.

3. The task *t* will start the execution. The Task's behavior may include a data exchange (through channels) with other tasks, it may as well, need to fetch data or code from the memory. Thus, invoking an access to the storage nodes.

4. The task *t1* generates a communication request that has as **destination**, the storage node where the data should be stored/fetched. The storage allocation view defined in section 5.4 specifies on which storage node (memory), the data/code of the task exists and the channel's data sample are stored. It calls the computation node's Communication Manager, *CM*, (section 5.5) to control the data transfer.

5. The "Communcation Manager" searches in its Communication pattern's list, the communication pattern that can routes the data to the specified Storage node (as specified in the communication request). There should be at least **one** communication pattern that satisfy the request. If there is more then one communication pattern, than the communication manager runs a selection function to choose one of them.

6. After specifying the communication pattern to be used, the Communication Manager, *CM*, transfers the communication request to the following node as defined in the communication pattern. The communication request waits for the communication virtual node to get the access to the communication resource. The request should access all the communication nodes that are relating the computation node to the storage node, as defined by the communication pattern.

7. Meanwhile, the computation virtual node can start the execution of another runnable task, while the data belonging to task *t1* are in transfer phase.

In the above scenario, contention on shared resources is not represented. However, when multiple tasks are executing concurrently on multiple computation nodes and data transfer (the communication interaction of the platform modeled by the different communication patterns); the execution scenario becomes quickly more complex. The execution time of a task is the sum of its execution on the computation node, the memory access cost (due to communication with other tasks or due to data load/store from the memory), and the contention cost on the shared resources.

5.8 System Mapping Example

This section reproduces the example presented in section **??**, and enrich it with the mapping components and specification defined in this chapter. Figure 5.10 depicts this example. It shows the system model on three layers: Application (blue), Architecture (yellow) and Mapping (green).

The application task model is composed of tasks: T1, T2, T3 and T4, and of two channels: Channel1 (carrying data from T1 to T4) and Channel2 (carrying data from T3 to T2). This application will execute on top of an architecture composed of two CPUs (CPU1 and CPU2), one DMA, one bridge, one bus and two memories (M3 and DDR1). The communication patterns model how the architecture nodes interacts to accomplish data transfers from computation nodes to storage node. For instance,"*CPU1DMAM3*" is the communication pattern that defines how the CPU1 will transfer data to the memory M3 using the DMA. The other Communication patterns are: CPU2DMAM3, CPU1M3, and CPU2M3.

The mapping layer ensures the binding of the architecture resources to the application constructs. As defined earlier in this chapter, the DIPLODOCUS mapping consists in four components:

- Shared Resources Modeling SRM = {VN4CPU1, VN4CPU2, VN4DMA, VN4Bus1, VN4Bridge1, VN4M3, VN4DDR1} is the set of all the virtual nodes that will control the architecture resources. The computation, communication and storage queues (c1, c2, com1, com2, com3, s1, s2) stock the respective requests. The virtual nodes will access these queues to choose the next executable resource request.

- Communication Management is ensured by the communication managers. Each computation node has one and only one communication manager. In this example, CM4CPU1 and CM4CPU2 are the communication managers of CPU1 and CPU2 respectively. Each one of these CM owns a list of all possible communication paths

- Execution Allocation = {{T1, VN4CPU1, 1}, {T2, VN4CPU1,2}, {T3, VN4CPU2,2}, {T4, VN4CPU2,2}}.

- Storage Allocation

  1. TaskStorageAllocation = {{T1, M3, 10, 20}, {T1, DDR, 90, 80}, {T2, M3, 100, 100}, {T3, M3, 100, 100}, {T4, DDR1, 100, 100}}
  2. ChannelStorageAllocation = {{Channel1, M3}, {Channel2, DDR1}}

5.9 Summary

The objective of this chapter was to define the Mapping modeling in DIPLODOCUS. The main contributions were the definition of the Virtual Node concept to model the resource sharing and of the communication manager to control data transfer in the system. In addition, an allocation process is defined to allocate computation resources and storage resources to the the application artifacts. Once the system mapping is performed (like the example in the previous section), The system designer could generate a SystemC code for simulation,in order to analyze the system's performance. The proposed mapping is applied to an industrial size model (the LTE physical layer uplink model) in chapter 7.

The following chapter will describe the SystemC simulation environment developed for DIPLDOOCUS models simulation. A definition of the simulation model of computation is provided, especially how concurrency and timing are taken in consideration to simulate accurately the system modeled using the DIPLODOCUS UML profile as defined in this chapter and in the previous one.

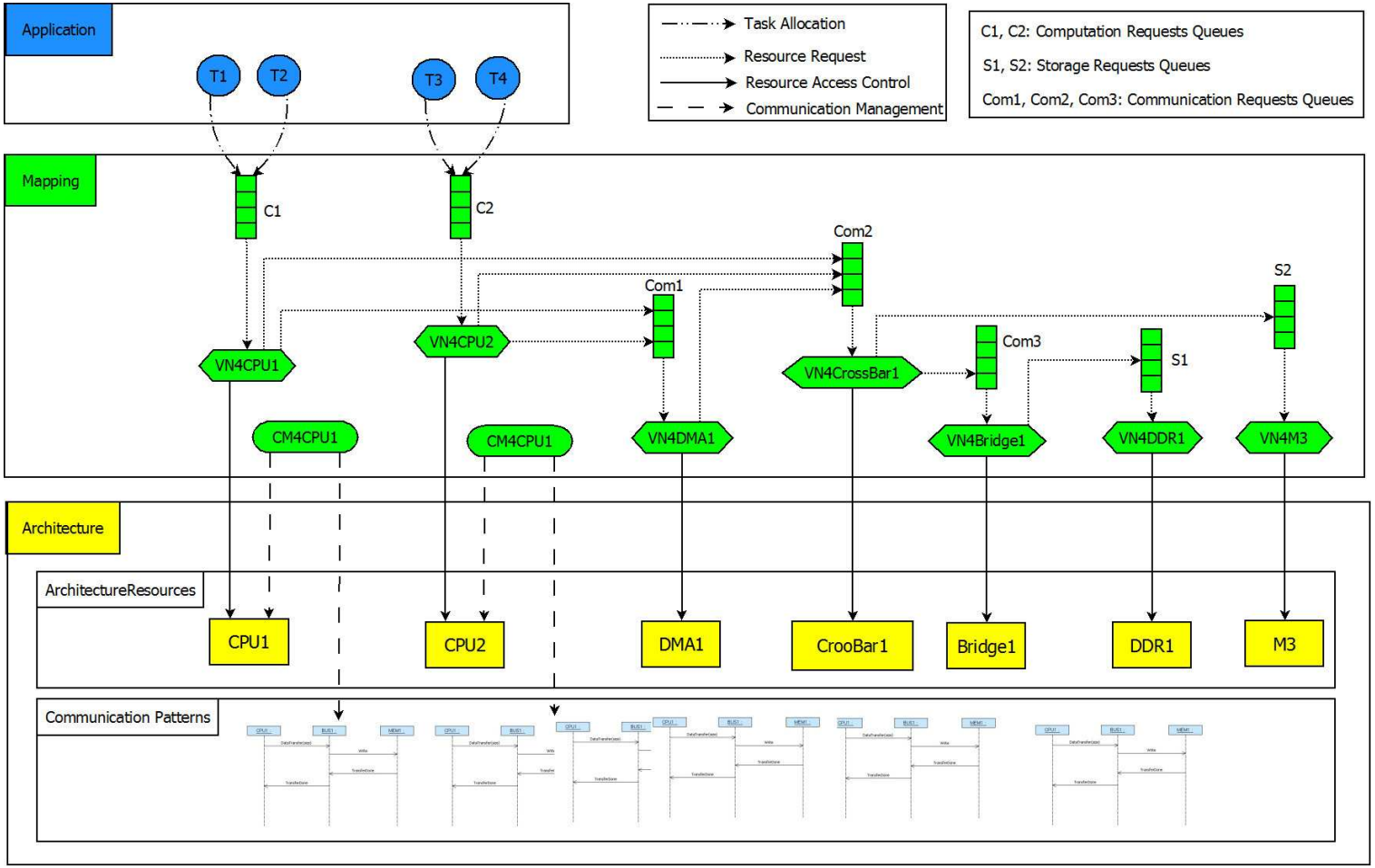

Figure 5.10: System Mapping Example

Chapter 6

Models Simulation for Performance Analysis

The DIPLODOCUS UML profile, and its extensions presented in the previous chapters, provide the designer with constructs to model separately the application and the architecture. It ensures, as well, the orthogonalization of the communication and computation concerns. The mapping process binds the application to the architecture, and permits the modeling of shared resources and of communication management. The following logical step is to analyze the modeled system and extract performance metrics. A performance analysis process based on simulation is presented in this chapter. Notably, a SystemC simulation environment is developed for models performance estimation. Performance metrics such as latency, throughput, utilization, scheduling policies impact and many others are extracted using observers to guide the designer in the *Design Space Exploration* process.

6.1 Introduction

The objective of this PhD work is to develop a methodology for performance estimation of complex SoCs at a high level of abstraction and in early design stages. After modeling the system using the extended DIPLDOOCUS UML profile (as presented in chapter 4 and 5), a performance analysis step is needed to check whether the system satisfies the design's objectives. As the application and architecture models are totally independent from each other, so a designer can easily evaluate candidate architectures using the same application model. This separation of concerns permits the exploration of the mapping of two different applications on a given architecture during first stages of projects. Thus, the designer can identify and make design decisions.

The performance analysis process presented in this chapter consists is based on a SystemC simulation environment for DIPLODOCUS UML models. The simulation's

objective is to rapidly estimate the system's performance, with a special focus on the capture of the impact of shared resources and communication process on the overall performances. The described simulation environment takes DIPLODOCUS UML models (Application, Architecture and mapping) as inputs. It generates automatically from them corresponding SystemC code.

The simulation semantics defined in this chapter target a SystemC code that is at the same level of abstraction as the UML model. The final objective is to perform early performance estimations and not to generate implementation code. The ideal would be to analyze systems when neither the software or hardware implementations exist. The Design decisions taken at this stage will guide the implementation.

Chapter Outline

Section 6.2 is dedicated to the SystemC language. It briefly describes its objectives, than details the model of time on which it relies and how it is used for system design. It describes as well the SystemC simulation kernel responsible of management of concurrency and of timing. Than, section 6.3 describes the proposed simulation environment, based on SystemC, for the simulation of DIPLDOCOCUS UML models. It defines how concurrency and timing are taken in consideration in order to simulate the Application, Architecture and mapping models. Section 6.4 presents the performance metrics extraction from the simulation. In a first step some predefined metrics are automatically calculated during the simulation, than in a second more advanced step, *Observers* are introduced to capture additional performance metrics. Observers are defined by the designer. The simulation environment presented in this chapter is used for the simulation of an industry size application example (LTE uplink physical layer) in chapter 7.

6.2 State of the Art on SystemC

SystemC is a modeling language created by the Language Open Group of the Open SystemC Initiative (OSCI) [43]. It supports different models of computation and allows the design of heterogeneous systems [79], [27]. Furthermore, it can be used to describe the hierarchical structure and the behavior of complex embedded systems. SystemC can be used to describe the system at different levels of abstraction. Using SystemC a system can be described at functional level, architectural level, and implementation level.

The SystemC language is described in IEEE standard 1666-2005 [4]. It defines a C++ library that consists of classes, macros, and templates which can be used to model a concurrent system using hardware-oriented data types and communication mechanisms. OSCI provides an open-source implementation of the SystemC framework.

Using this implementation, a SystemC model can be compiled by standard compliant C++ compilers and can be executed. This execution simulates the model and provides information and traces that can be used to analyze and validate the model.

6.2.1 System's Design in SystemC

Complex systems consist of many independently functioning components. These components may represent hardware, software, or any physical entity. In SystemC, These components are represented by *Modules*. In fact, a system model is structured by using modules. A SystemC module *SC_MODULE* encapsulates a part of the system which is being modeled and may has communication ports to communicate with other modules within the model. The designer can use C++ function calls to specify the modules inter-communication and is not forced to use the SystemC constructs. As SystemC supports modules hierarchy, a module can contain other modules as well.

SystemC processes are used to define the behavior of a SystemC modules. Because the behavior of the model is defined in C++ member functions, all valid C++ language constructs can be used. For example, the behavior of a module can be described by using advanced data structures and algorithms from the standard C++ library or any other C++ library. The process is the basic unit of execution in SystemC.

Each process has a sensitivity list which is a list of SystemC events. An event is something that happens at a specific point in time, for example a change of value on an input port. A SystemC process is defined in the form of a C++ member function that is registered with the SystemC simulation kernel by using the SC_THREAD, SC_CTHREAD, or SC_METHOD macros. Each type of these SystemC processes has a defined semantic:

- **SC_Method**: An SC_METHOD process is started by the SystemC simulation kernel whenever one of the events on its sensitivity list occurs. It always runs to completion before it returns control to the simulation kernel.

- **SC_THREAD**: An *SC_THREAD* relies on the *"Wait()"* method to suspend its execution. When wait() executes, the state of the current thread is saved (context switch). The SC_THREAD process is resumed by the simulation kernel at the specified time, or when one of the events on its sensitivity list occurs.

- **SC_CTHREAD**: An *SC_CTHREAD* process is a special kind of SC_THREAD which is only sensitive to a certain edge of a clock input port. This explains the extra C in the macro name SC_CTHREAD.

6.2.2 Concurrency

SystemC is developed with the aim to model and simulate complex systems. These systems are the composition of many hardware and software blocks and many things that may run in parallel and access concurrently to shared resources. For example, in a embedded system, two CPUs execute in parallel and may request access to a shared memory through a shared bus. In this simple example the two CPUs the memory and the bus are executing in parallel.

To model concurrency SystemC uses processes to model concurrency. As with most event-driven simulators, concurrency is not true concurrent execution. In fact, simulated concurrency works like cooperative multi-tasking. In other words, the concurrency is not preemptive. Each process in the simulator executes a small chunk of code, and then voluntarily releases control to let other processes execute in the same simulated time space. The SystemC simulation kernel is responsible for starting processes and managing which process executes next. Due to the cooperative nature of the simulator model, processes are responsible for suspending themselves to allow execution of other concurrent processes.

Events are used as well to handle the concurrency in SystemC is handled by using processes and events (more details on subsection 6.2.3). An event is something that happens at a specific point in time. It has no value and no duration. SystemC uses the *sc_event* class to model events. This class allows explicit launching or triggering of events by means of a notification method. Once it occurs, the processes waiting for it (using the sensitivity list) become ready to execute. Programmer can perform only three actions with an *sc_event*: wait for it (using the SystemC *Wait()* method, or as an event in the sensitivity list), initiate it (using the SystemC *notify()* method), or cancel it (using the SystemC *cancel()* method).

6.2.3 SystemC Simulation Kernel

The SystemC simulation kernel provided by the OSCI open-source implementation, defines how the SystemC code is simulated. It manages namely the concurrency and the timing aspects. It adopts an event-driven process scheduler that mimics the passage of simulated time and allows parallel processes to synchronize and communicate in order to model the hardware and "software" components of a system. The SystemC scheduler is non-preemptive, and is deterministic with reference to events occurring and different simulation times. It is not deterministic with reference to events that occur at the same simulation time.

Once the SystemC code is available the designer can simulate the model using the OSCI SystemC simulation. The simulation process is divided into two phases: the elaboration phase and the simulation phase:

1. During the **Elaboration phase** the modules are instantiated and initialized by

executing their constructors. During this initialization the connections between the modules are set up. Because the modules are instantiated and connected by executing C++ code **any valid C++ language construct** can be used. During this phase, all processes are placed initially into a ready pool. Once this phase terminated, the simulation kernel is invoked, and the system simulation can now start.

2. During the **Simulation phase** the model is simulated according to the SystemC simulation kernel semantic. The kernel controls the timing and the order of execution of the SystemC processes. One by one processes are randomly taken from the ready pool by designating them as running and invoked. Each process executes until it either completes (e.g., via a return) or suspends (e.g., calls wait()). During execution, a process may invoke immediate event notification (i.e., event.notify()) and possibly cause one or more waiting processes to be placed in the ready state. It is also possible to generate delayed or timed event notifications. Completed processes are discarded. Suspended processes are placed into a waiting pool. Simulation proceeds until there are no more processes ready to run. Than the simulation kernel enters in the waiting state. At this point, three possible options are available: waiting processes or events that are zero time delayed, non-zero time delayed, or neither.

    - There may be processes or events waiting for a delta cycle delay (.notify(0)). In this case, waiting pool processes with zero time delays are placed back into the ready pool. Zero time events originating from delayed notifications may cause processes waiting on those events to also be placed into the ready pool. In this case the kernel will check for ready processes and execute them. Time is not incremented.
    - There may be processes or events, scheduled for later, waiting for a non-zero time delay to occur. In this case, time is advanced to the nearest time indicated. Processes waiting on that specific delay will be placed into the ready pool. If an event occurs at this new time, processes waiting on that event are placed into the ready pool. Another round of evaluation occurs if any processes have been moved into the ready pool.
    - It is possible that there were no delayed processes or events. Since there are no processes in the ready pool, then the simulation simply ends.

Discussion

The simulation environment presented in this chapter focuses on the simulation and performance estimation of the DIPLODOCUS models. Its main objective is to calculate performance metrics at a high simulation speed to enable the designer to rapidly

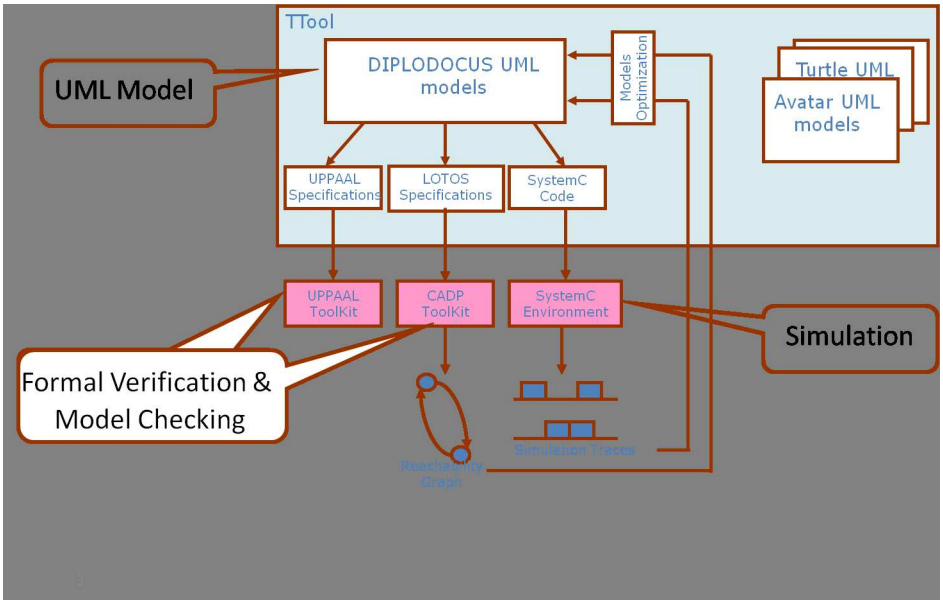

Figure 6.1: Simulation and formal verification with TTool

evaluate different design decisions. This simulation environment is based on SystemC is used as simulation language. However, when it's possible C++ constructs are used as they they simulate faster than SystemC constructs. In difference with the existing SystemC simulation kernel, the described simulator supports interruptions.

6.3 A SystemC Simulation Environment for DIPLODOCUS models

The DIPLODOCUS UML profile is supported by an open source toolkit, TTool[63]. In addition to DIPLODOCUS, TTool supports other UML profiles such as Turtle and AVATAR. After modeling the system using one of the supported profiles, TTool can generate formal specifications (UPPAAL [42] and LOTOS [3]) of the models to enable formal verification using external tools (the CADP toolkit [35] for the LOTOS specifications and the UPPAAL toolkit for the UPPAAL specifications). Simulation code can be generated as well for simulation purposes. The new SystemC simulator presented in this chapter comes with a generator of SystemC code from DIPLODOCUS models. the other profiles can have as well their own generators for simulation code, for instance Turtle models can be translated to a JAVA code for execution.

Figure 6.1 describes the TTool modeling, and simulation code, and formal specification generation. The formal specification and the simulation code are used to evaluate the system. Formal verification will primely verify the functionality of the system

as they cover all the design space. Hence, when the system is complex, formal verification process could reach its limits (state explosion scenarios). Simulation's main objective is to estimate the system's performance. The simulation environment presented in this chapter does not intend to cover all the design space, but to estimate the performance of one mapped system. After system's evaluation phase, designer could optimize the system model, by changing the mapping, the application and or the architecture. The following subsections present the SystemC simulation environment for DIPLODOCUS and how it supports concurrency, timing and interruptions, and how it extracts performance metrics of system's models using performance observers. This performance estimation process is not targeting, for the time being, the system's power estimation. This PhD work can be extended to integrate the power estimation in the estimation process.

6.3.1 DIPLODOCUS SystemC simulator concurrency

The concurrency in the system is due to the concurrent execution of different tasks on the computation resources. In fact, when a task is in the "ready to execute" state it will request the computation virtual node on which it is mapped. As tasks can run concurrently, multiple computation requests are treated simultaneously in the system (more details on this were provided in section 5.1 of chapter 5). These concurrent computation requests will generate concurrent communication requests to access to storage resources to read/write data (data exchanged between tasks or tasks' data and instructions). Hence, in a system there are three concurrency types:

1. Tasks concurrent access to a computation resource: when multiple ready to execute tasks request the same computation resource.

2. Communication resources concurrent access: when computation resources executing different tasks are trying to access the storage resources using the shared communication resources.

3. Storage resources concurrent access: when a storage resource is shared between multiple computation resources that are trying to access it through multiple communication resources.

The access to shared resources is controlled by the virtual nodes. Each architecture resource has one virtual node (or more than one in the dynamic and hierarchical scheduling scenarios), that runs in concurrency with all the other virtual nodes. In addition, all the application's tasks can run concurrently. As Concurrency in SystemC is modeled using the *"SC_MODULE"* construct and its processes, DIPLODOCUS *"Tasks"* and *"Virtual Nodes"* are modeled as SystemC modules.

Furthermore, *"Communication managers"* control the data transfer from computation nodes to storage nodes and they run in parallel to tasks and virtual nodes. In

fact, the extended DIPLODOCUS presented in this thesis advocates a clear separation between the execution and communication using the communication patterns and the communication managers. The communication managers are as well modeled using the SystemC modules.

	DIPLODOCUS UML Construct	SystemC/C++ construct
Application	DIPLODOCUS Task	SC_Module
	Application Task Behavior Command	C++ class
	Application Channel	C++ class
	Application Events and Requests	C++ class
Architecture	HW_Resources	C++ class
	Architecture communication pattern	C++ class
Mapping	Virtual Node	SC_Module/Sc_Thread
	Access Policy	C++ class
	Communication Manager	SC_Module/Sc_Thread

Table 6.1: DIPLODOCUS UML constructs and their corresponding SystemC constructs

Table 6.1 shows the translation of DIPLODOCUS UML constructs into SystemC constructs. The concurrent constructs (Tasks, Virtual Nodes, Communication managers) are translated into SystemC modules, while the other DIPLODOCUS constructs are translated into normal C++ classes.

6.3.2 System's Timing: From DIPLODOCUS commands to physical time

In the system performance estimation process the central issue is that computation has no timing behavior as long as the target platform is unknown. Thus, it becomes primordial to define the timing behavior of the system and especially how this behavior is determined.

DIPLODOCUS application's tasks represent the system desired functionality, their behavior is modeled through a set of commands (as described in section 4.2.3 of chapter 4). These commands could be of four types:

1. Control Commands: that represent basic control structures (if, loop ...), and the action command that could change the value of task's attributes. Their execution on a DIPLODOCUS computation node is done in *zero-delay* and does not count in the computation complexity.

2. Communication Commands: These commands involve any data or synchronization exchange between tasks. Commands are here for read/write to a channel

and send/receive an event or a request (sending an event is done in zero delay). the execution of these commands involves an access to the communication resources in order to read/write data from storage nodes. In addition to a cost due to cache misses.

3. Abstract Execution Commands (Exec): this subset of commands represents the computational complexity of the node while executing applications. The semantics and execution delay of these commands depend on the underlying architecture. Furthermore, they contribute to the communication cost as their execution may induce cache misses, and consequently access to communication and storage resource. These commands represent the abstraction of the real code of tasks.

4. Temporal Commands: the Delay Command represents an absolute delay. During the simulation, the execution of this command is equivalent to putting the task on hold during the specified delay, once the delay expires the tasks returns to ready state.

In other words, the execution of a DIPLODOCUS command could be performed in a zero delay (the control commands), or may induce a computation complexity and/or communication complexity. A methodology for performance's estimation should be capable of calculating these complexity metrics based on the DIPLODOCUS UML models. The following discussion will detail how to calculate the physical time needed for the execution of DIPLODOCUS commands based on the parameters of the execution platform.

The transformation process from DIPLODOCUS command to physical time is done in two steps: calculation of computation timing cost and calculation of communication timing cost. Both types of timing are functions of the modeling parameters specified in the DIPLODOCUS application model, of the architecture performance parameters specified in the DIPLODOCUS architecture model and of contention on shared resources (monitored using the mapping constructs of virtual node and communication managers).

Computation Timing Cost

This corresponds, in a first step, to the calculation of the number of cycles needed to execute an Abstract Execution Command (*Exec*) on a computation node without the communication cost that it may induce. The first parameter used in the calculation of this cost is the number of "Abstracted Instructions" (*nbAbsIns*) specified by the designer. For instance, *"Execi 100"* corresponds to the abstraction of the execution of 100 abstract instructions. The second parameter is the number of cycles per instruction (*CPI*) needed by the computation node to execute an instruction. This parameter is

specified when modeling the architecture.

The Number of cycles (*NbCyclesPerExec*) required to execute a number *nbAbsIns* of abstract instructions of a task mapped on a computation node that can execute an instruction in a *CPI* cycles is calculated as follows:

$$NbCyclesPerExec = nbAbsIns * CPI \tag{6.1}$$

Once the required number of cycles is calculated it is easy to calculate the time needed to perform the execution. In fact, the computation frequency is specified in the architecture model. However, the execution may take longer than this calculated number if contention on the computation resource occur during the execution (the SystemC simulator presented in this chapter supports interruptions). Furthermore another computation cost must be added to the total time, which is the context switch time. In fact, each time a task starts or resumes its execution after interruption, the scheduler needs to execute (context switch). This time is fixed and depends only on the used scheduler (computation virtual node Access Policy)

Communication Timing Cost

The execution of the application on the architecture introduces two types of communication: **Explicit** communication resulting from the inter-task communication (using the DIPLODOCUS channels) and **Implicit** communication resulting from the need of tasks to fetch instructions and data from storage resources to accomplish their execution. In both types the time required to perform the data transfer is a function of: the data amount to transfer, the sum of data transfer cost on each communication resource on the selected communication pattern and finaly the access cost to the storage node.

In the case of channels reading/writing the data amount (*DataAmount*) is specified by the designer. In fact, the designer defines a write or read with a specific number of samples and s/he specifies the size of samples in byte (more details in section 4.2.3 of chapter 4). However in the case of implicit communication (data and instruction cache miss) the data amount depends on the cache miss percentage and cache line size. Once the *DataAmount* is specified, the data transfer cost on a communication node is defined as follows:

The transfer cost (*TransferCost*) of an amount of data, *DataAmount*, on a communication resource with a word size *WordSize* that could be transfered without interruption (nor cache mechanism interference) in a delay of *TransferCostOneWord* is calculated as follows:

$$TransferCost = (DataAmount/WordSize) * TransferCostOneWord \tag{6.2}$$

This transfer cost should be done for all the communication nodes on the selected communication pattern. Finally the access cost to the storage node is calculated in a similar way To that time one should add the contention cost due to contention on the communication and storage resources using to perform the data transfer. One should add to this cost the **ContextSwitchTime** due to the change of the requester on communication and storage ressources.

Figure 6.2 shows the calculation of CPU's cycles required for the execution of 600 abstract instructions on a CPU where data is stored on an external memory. The final time encapsulate both computation timing and communication timing. Data and instructions are stored on an external memory "M3" accessible through a bus. The Read/write of a word to that memory costs 40 CPU cycles. The cache miss percentages are specified by the designer in the architecture model. The execution of these 600 abstract instructions consumes 2664 CPU cycles. This calculation is done without the possible contentions (these contentions and possible interrupts are detailed in section 6.3.3) . Note that this calculation is done in the simulator directly and for each execution.

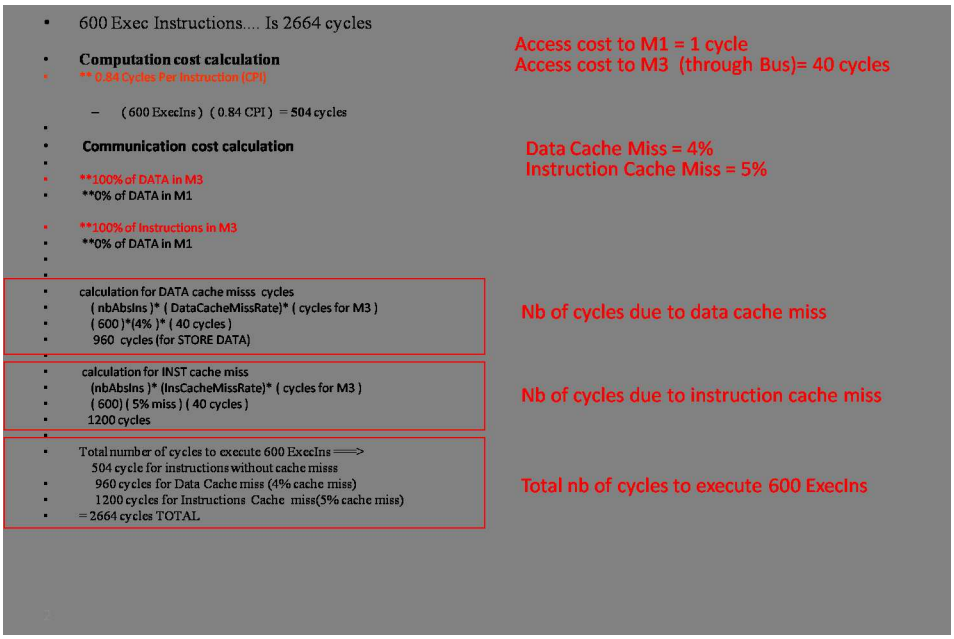

Figure 6.2: Calculation of the number of CPU's cycles needed to execute a DIPLODOCUS "Exec" command

6.3.3 Simulation's Timing semantics and Interruptions Support

The previous section showed how to calculate the required time for computation and communication without contention. The next step is to define the timing semantic of the system during simulation. Mainly how simulation time is advanced on computation and on communication, how interruptions are taken in consideration, and how timing is recalculated after an interruption.

The simulator presented here is based on SystemC, so it uses the SystemC kernel to manage time. SystemC provides two ways to advance simulation time: the *"Wait"* construct that suspends the execution of a thread during a specific time, and the more implicit *"sc_event"* construct, where an event can be notified for a future time. The event can see its future notification canceled (event.cancel()) or its notification's time changed. The SystemC kernel (its simulation behavior is described in section 6.2.3) executes all modules ready at a specific time and advances the simulation time to the next event.

Timing semantic in this context refers to *who* and *how* time will be managed. For **who** question, there is two possible solutions: first solution is that a centric simulation unit plays the interface between the simulation constructs and the SystemC simulation kernel, the second solution is that each concurrent construct (namely: tasks, virtual nodes and communication managers) manages its time. Besides, interruptions are always possible where computation virtual nodes can interrupt tasks' execution and communication managers can be interrupted by communication and storage virtual nodes. The second solution is adopted as it fits with the DIPLODOCUS concurrency as described in section 6.3.1. The following two subsections will describe **how** the computation and communication timings are managed.

Computation timing

During its execution, the task manages its time. In fact each SystemC/DIPLODOCUS task has a *"Timer()"* method that is sensitive to an sc_event, *"TimerEvent"*. When a command, that needs time to execute (Abstract execution commands or reading/writing a channel or a delay command) starts its execution, it fires the task's timer with the time needed, *"Delay"*, calculated as described in the previous section. The timer notifies the "TimerEvent" with the specified time (TimerEvent.notify(Delay)).

Once the time elapsed, the SystemC kernel fires the event, *"TimerEvent"*, and the task's *Timer* method is activated as it is sensitive to the firing of the event. The timer in its turn notifies the command when the requested time is reached. Then, the command execution is finished and the next command can start its execution. If the task has finished its execution or it is blocked waiting for an event or for data on a channel, it signals its new state to the computation virtual node that in its turn can select another task to start its execution. The scenario presented above, is depicted in the upper part

of the sequence diagram in figure 6.3.

The lower part of the sequence diagram depicts the scenario when the task's execution is interrupted by the virtual node. In fact, when an interruption occurs, the computation virtual node sends the preemption signal to the task. Then the task calls its interrupt method, that cancels the timer event and calculates the elapsed time and subtracts it from the total needed time. The subtraction's result is the remaining needed time to complete the execution of the command. Once the scheduler reselect the task to resume its execution the already calculated remaining time is re-injected to the timer to complete the execution of the command.

There is another timing effect to take in consideration, which is the **context switch** cost. In fact, when a computation node is executing a task and wants to start the execution of another task there is a cost to pay representing the storing of the state of the task and the execution of the system scheduler to select the next running task. This time is modeled using the *wait* constructs and it is non interruptible.

Communication Timing

Communication timing behavior is managed by the communication manager. It has its own "Timer" that controls time in a similar way to tasks' timing presented above. In fact, when a communication request is created by the task to access a storage node the communication manager selects the communication pattern that will transfer the data. Then, it transfers the communication request to each communication node in the pattern to acquire access to them. The time needed to perform the transfer without preemption is calculated as defined in the previous section. Then the communication manager's timer notifies a "timer event" with the calculated time. When an interruption occurs (a higher priority computation node wants to use one of the communication nodes or one of the communication nodes uses a time sharing policy and the time accorded to the request is finished), the data amount transfered before the interruption is calculated, as well as the remaining amount and updates the request with it. Once the interruption finishes the communication manager could resume the transfer if it succeeds to acquire all the resources in the pattern.

6.3.4 The simulator in a nutshell

After modeling the application and the architecture and defining the mapping the designer uses the provided simulation environment to analyze and optimize the system. This flow of modeling, simulation, analysis and optimization adopted by DIPLODOCUS and its extensions is presented if figure 6.4. The simulator is an event driven simulator and works at a high level of abstraction and is faster than the real time execution (chapter 7 presents an industrial case study where more information on simulation speed are provided).

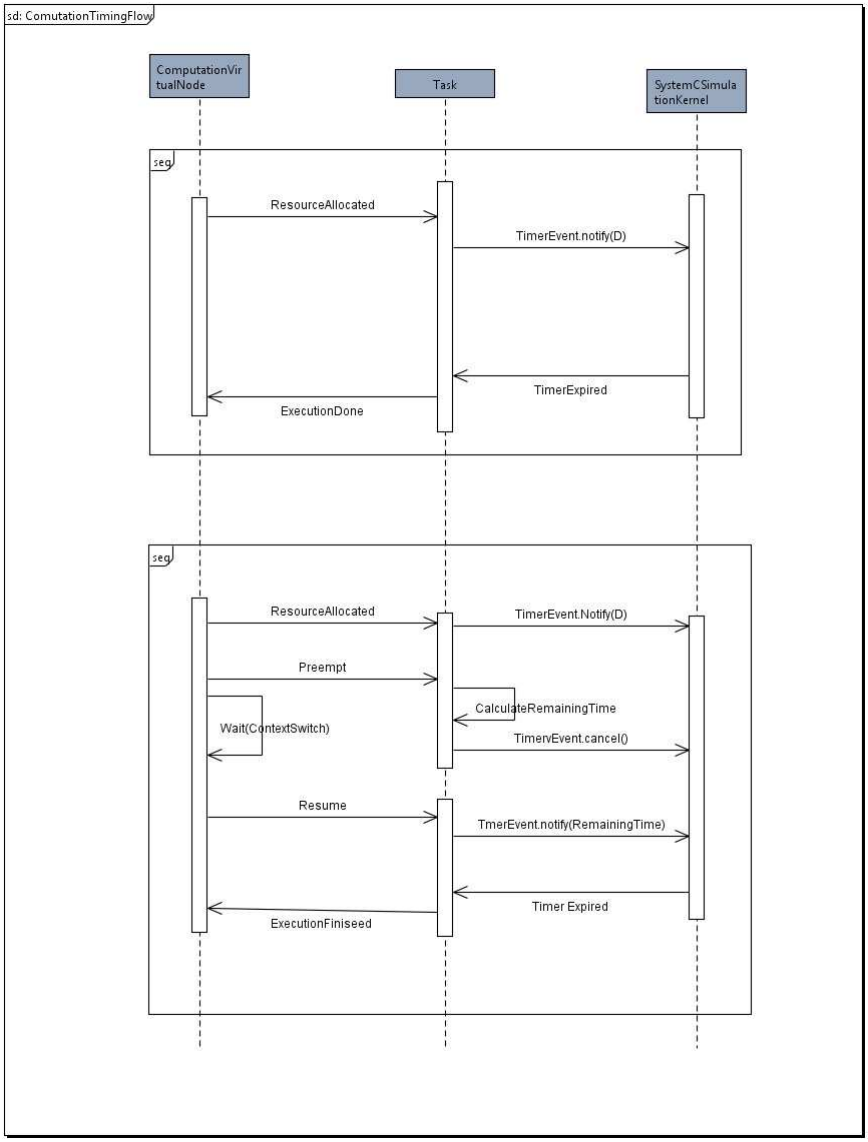

Figure 6.3: Task Timing behavior

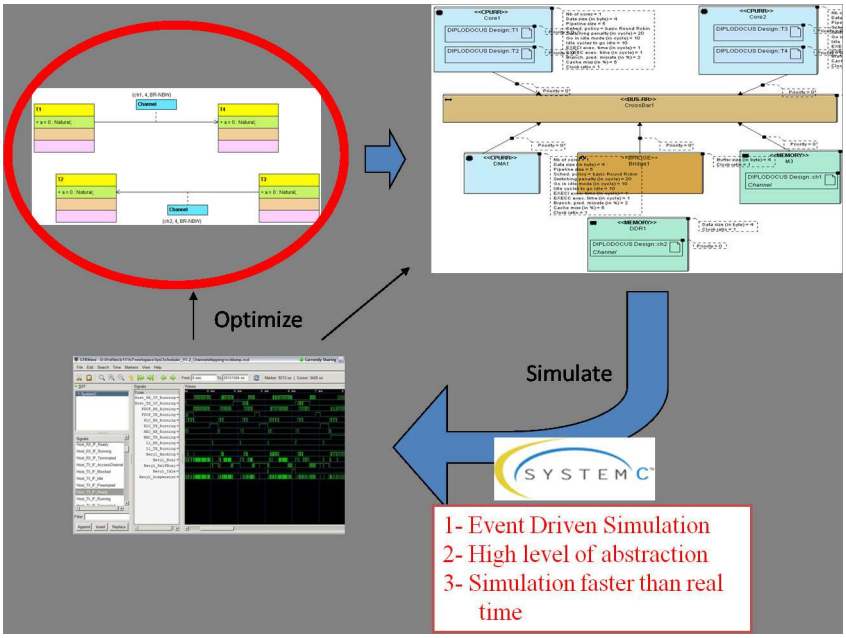

Figure 6.4: The DIPLODOCUS Flow - Performance estimation and optimization of system models

The first requirement of a simulation environment that is meant to simulate DIPLODOCUS models is that the simulation code should be equivalent to the DIPLODOCUS UML models. Thus, it should first defines the SystemC/C++ constructs for all the DIPLODOCUS UML constructs. The SystemC representation of DIPLODOCUS application architecture and mapping is provided in the following subsections.

Application

A DIPLODOCUS application represents the system desired functionality. It is a set of communicating *"tasks"*, each has a behavior composed of a set of commands that could be communication, control, execution abstraction and delay commands (as described in section 4.2.3). Tasks exchange data samples through communication *"channels"* and exchange synchronization information through *"events"* and *"requests"*.

Figure 6.5 depicts the C++/SystemC code representing the DIPLODOCUS application. The left part of the figure is an excerpt of the application file where tasks, channels, events, requests and task observers are instantiated. This file specifies as well how the communication connectors are bound and attached to tasks. The upper right part of the figure is an excerpt of the task T3 code. It instantiates the commands (task's behavior) and then it defines how their execution order (the Tasks Command

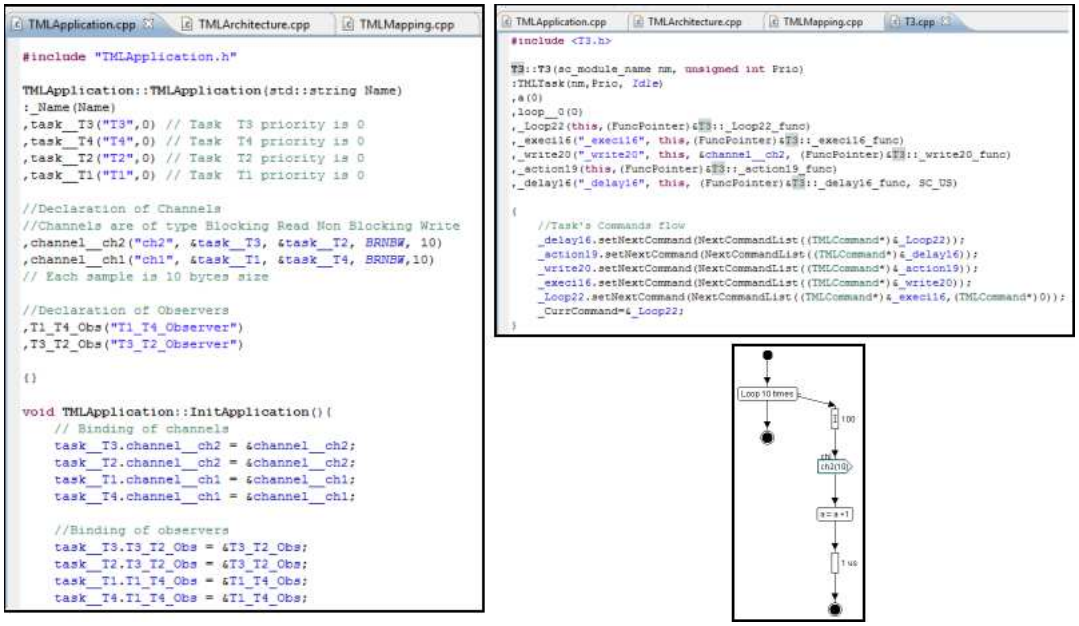

Figure 6.5: An excerpt of the SystemC generated for the Application model

Flow section). Finally the lower right part of the figure is the DIPLODOCUS UML diagram that represents the same task T3.

Architecture

DIPLODOCUS architecture model (as described in section 4.3) is meant to execute the application. It is a set of two main profiles

- Architecture resources which are Computation, communication and storage resources. They are characterized by performance parameters. These parameters are used to calculate the system's execution time as described in this chapter.

- Communication patterns, defining how data will be transfered from computation nodes to storage nodes through communication nodes. Hence, modeling the communication protocols implemented in the architecture

Figure 6.6 shows an excerpt of the SystemC/C++ code for the modeled architecture, where each resource is specified with a set of performance parameters. The bottom part of the figure presents the code for a communication manager transferring data from a computation node to a storage node using a crossbar. In the architecture specification file, this could pattern in instantiated for each computation node that wants to access a memory using the crossbar.

```
DIPLODOCUSArchitecture::DIPLODOCUSArchitecture(std::string Name)
:_Name(Name)
//Architecture nodes instantiation with their performance parameters
,CPU1("CPU1", 4, 4, 1, 1000)// Data cache miss = 4, Ins cache miss = 4, CPI = 1
,CPU2("CPU2", 4, 4, 1, 1000)//
,DMA1("DMA1", 0, 1, 1, 450)
,CrossBar1("CrossBar1", 50, 500, 128)
,CPU1_M2("Core1_M2", 1, 512, 500, 8)// M2 access cost is 1 cycle, Running frequency= 500 MHz
,CPU2_M2("Core2_M2", 1, 512, 500, 8)// M2 size is 512 KB, data word size = 8 bits
,M3("M3", 20, 500, 1056, 8)
,DDR1("DDR1", 40, 400, 1056, 64, 20, 75)// DDR data size = 64 bit

//Communication pattern instantiation
,CommPattern_CPU1_M3_Pattern("CommPattern_CPU1_M3_Pattern", &CPU1, &SwitchingMatrix, &M3 )
,CommPattern_CPU2_M3_Pattern("CommPattern_CPU2_M3_Pattern", &CPU2, &SwitchingMatrix, &M3 )
,CommPattern_CPU1_DMA1_M3_Pattern("CommPattern_CPU1_DMA1_M3_Pattern", &Core1, &DMA1, &CPU1_M2,  &SwitchingMatrix, &M3 )
,CommPattern_CPU2_DMA1_M3_Pattern("CommPattern_CPU2_DMA1_M3_Pattern", &Core2, &DMA1, &CPU2_M2, &SwitchingMatrix, &M3 )
,CommPattern_CPU1_DDR1_Pattern("CommPattern_CPU1_DDR1_Pattern", &CPU1, &SwitchingMatrix, &DDR1 )
,CommPattern_CPU2_DDR1_Pattern("CommPattern_CPU2_DDR1_Pattern", &CPU2, &SwitchingMatrix, &DDR1 )
```

a) An excerpt of the Architecture specification file

```
Core_M3_Pattern::Core_M3_Pattern(std::string Name, TMLArchitectureResource* Core, TMLArchitectureResource* CrossBar, TMLArchitectureResource* Mem )
: TMLCommPattern(Name)
,_ConsumerMem(Mem)
,_ProducerCore(Core)
,_CommMedium(CrossBar)
,_WriteAction25(std::string("_WriteAction25"), &_CommMedium, &_ConsumerMem, BlockingAction,(CommFuncPointer)&Core_M3_Pattern::_WriteAction25_func)
,_DTAction25(std::string("_DTAction25"), &_ProducerCore, &_CommMedium, BlockingAction,(CommFuncPointer)&Core_M3_Pattern::_DTAction25_func)
{
	//Action chaining
	_WriteAction25.SetNextAction(array(1,(TMLCommAction*)0));
	_DTAction25.SetNextAction(array(1,(TMLCommAction*)&_WriteAction25));
	_CurrAction = &_DTAction25;

}
```

b) An excerpt of the C++ file representing a communication pattern

Figure 6.6: An excerpt of the SystemC code generated for the Architecture model

Mapping

As defined in Chapter 5, the mapping process binds the application constructs to the architecture nodes and defines the access policies used to access shared resources. The mapping should define teh followwing components:

- Virtual nodes, which control the access to shared resources and where each resource has at least one virtual node.
- Access Policy where each node has an access policy that specifies which requester will acquire the resource
- Communication managers, to control the communication and data transfer from computation nodes to storage nodes.
- Task execution allocation, where tasks are mapped to computation virtual node where they will execute
- Storage allocation, where data and instructions of tasks are stored

Figure 6.7 shows an excerpt of the mapping file in the SystemC simulator, where all the mapping steps are showed. **add comments on this mapping file**

6.4 Performance Monitoring

High level Performance estimation and monitoring can assist in detecting potential performance problems during early development stages and identifying alternative solutions. Performance metrics are extracted during the system simulation. They help designer to understand the performance interrelationships within their design prior to deployment and to answer the following questions(among many others):

1. How does the architecture affect response times and throughputs for the modeled application?
2. How does the allocation of architecture resources to application influences performance?
3. How does the access policies adopted to access shared resources influence performance?
4. How does the system perform if the input throughput changes?

Our simulator targets the extraction of three types of performance metrics : Latency, Throughput and Mapping Efficiency.

```
DIPLODOCUSMapping::DIPLODOCUSMapping(DIPLODOCUSApplication* Appli_P, DIPLODOCUSArchitecture* Archi_P):
//Mapping is the binding of architecture resources to application constrcuts
Application_P(Appli_P)
,Architecture_P(Archi_P)

//Access to each architecture resource is controlled by a VN using an access policy
,VN_CPU1("VN_CPU1", &AccessPolicyVN_CPU1, 0)
,VN_CPU2("VN_CPU2", &AccessPolicyVN_CPU2, 1)
/* 0 and 1 are the priorities of the VNs, in this example CPU1 is of higher priority
*than CPU2 when accessing communication and storage resources*/
,VN_DMA1("VN_DMA1", &AccessPolicy_DMA1, 2)
,VN_Bridge1("VN_Bridge1", &AccessPolicy_Bridge1, 1)
,VN_CrossBar1("VN_CrossBar1", &AccessPolicy_CrossBar1, 2)
,VN_M3("VN_M3", &AccessPolicy_M3, 0)
,VN_DDR1("VN_DDR1", &AccessPolicy_DDR1, 1)

/*Each computation resource has a communication manager that defines how it will access storage resource.
 * The communication manager has a list of all communication patterns available for the resource */
,CPU1CommManager("CPU1CommManager")
,CPU2CommManager("CPU2CommManager")


{
```

```
void DIPLODOCUSMapping::TaskStorageAllocation(){

   // Set instruction memory map for task__T4
   MappMem._Mem = &(Architecture_P->M3);
   MappMem.Per = 70;
   Application_P->task__T4.SetInsMappingMem(MappMem);
   MappMem._Mem = &(Architecture_P->DDR1);
   MappMem.Per = 30;
   Application_P->task__T4.SetInsMappingMem(MappMem);

   // set data memory allocation for task__T4
   MappMem._Mem = &(Architecture_P->DDR1);
   MappMem.Per = 100;
   Application_P->task__T4.SetDataMappingMem(MappMem);
```

```
void DIPLODOCUSMapping::ChannelStorageAllocation(){
   // Set memory map for channel__ch2
   MappMem._Mem = &(Architecture_P->M3);
   MappMem.Per = 100;
   Application_P->channel__ch2.SetMappingMem(MappMem);



   // Set memory map for channel__ch1
   MappMem._Mem = &(Architecture_P->DDR1);
   MappMem.Per = 100;
   Application_P->channel__ch1.SetMappingMem(MappMem);
}
```

```
void DIPLODOCUSMapping::TaskStorageAllocation(){
   // Attach tasks to the computation virtual Nodes
      VN_CPU1.AddTask(&(Application_P->task__T1));
      VN_CPU1.AddTask(&(Application_P->task__T2));
      VN_CPU2.AddTask(&(Application_P->task__T3));
      VN_CPU2.AddTask(&(Application_P->task__T4));
}
```

Figure 6.7: An excerpt of the SystemC generated for the Mapping model

Latency metrics

Latency is the time interval during which the response to an event must be executed. In DIPLODOCUS, latency is the time required to execute a task or a group of tasks (the application itself is a set of task), or to read/write a data sample on a channel. Some application have strict latency constraints, for example the LTE mobile communication protocol requires that the execution done on a data packet do not exceed 1 ms time frame [33].

Throughput metrics

Throughput, by definition, is the average rate of message delivery. In DIPLODOCUS, throughput is the average rate of data transfered on a communication pattern during the system's execution (in byte/second). Another category of throughput is the application throughput, which represent the average rate of data transfered between tasks during their execution.

Mapping Efficiency metrics

The efficiency metrics contains the resources utilization and the resources' access contention. Resource utilization refers to the capability of the modeled system to use efficiently the architecture resources (for instance a CPU is over-charged used while another one is under charged) . However, utilization is affected by access contention to shared resources. If the mapping is not appropriate or if execution needs are under estimated the resources' utilization will not be "optimal".

6.4.1 Simulator Default monitoring

By default the simulator extracts a set of performance metrics and saves them into a simulation log. The first performance monitoring means provided by the simulator is the VCD (Value change dump) waveforms. These waveforms provides an overview of the system's execution, task's interdependency, utilization of computation resources and can be used by the designer to detect some execution anomalies like tasks exceeding their execution deadlines or tasks blocked waiting for another task execution while they are mapped on different computation nodes. Figure 6.8 provides an example of a waveform depicting a part of the execution of a task "T1". Designer can identify when the task is ready to execute but waiting to get get access to the resource, or when the task execution is causing a cache miss. In fact, all the execution states of all tasks during the simulation are logged into the VCD file, and be studied by the designer using any VCD standard complaint tool. In this study, the free tool "GTKWave" [24] is used.

After over-viewing the system's execution the designer, can study more detailed extracted metrics. For instance, after simulation, the simulator generates for each task the

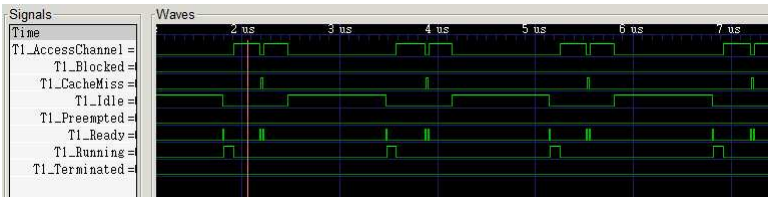

Figure 6.8: A VCD diagram of the task T1 execution and the evolution of its state in time

individual execution time of each of its execution, and the total number of instructions and cycles. On the architecture side, the utilization of architecture resources (CPUs, busses, memories ...) is generated as well as the contention on each of this resources. Finally the throughput (average rate of data transfer) of each communication pattern as well as the contention are provided to enable the designer to identify the communication bottlenecks. For an example on the use of these metrics please refer to chapter 7 where they are extracted for an industrial use case.

6.4.2 Personalized Performance Metrics: Observers

The previous subsection showed the default metrics extracted during the system simulation. However, sometimes the designer is interested in investigating in more details some modeling aspects. For example, s/he can be interested by the latency of execution of two dependent tasks, or for example in a telecommunication protocol case the total latency of execution on a specific packet of data by a set of tasks (the protocol stack for instance). s/he can be interested as well by monitoring the execution of a part of a task. Thus, a more personalized mean for performance monitoring is required. The solution proposed here is to use *observers*.

Observers are attached to a system to estimate its internal state. In the DIPLODOCUS context an observer is an object that monitors the execution of the system to extract its performance parameters. It is a C++ class that executes in zero delay (does advances the simulation time). The execution of an observer does not change the state of the simulated system, however, it increases the simulation time.

An observer has two plugs (C++ class methods): the *"Start Observation point"* and the *"End Observation point"*. These plugs could be attached to tasks, for instance an observer "start observation point" attached to a task task1 first command and its "end observation point" to another task task2 last command, measures the time needed to execute both tasks . In fact using the observers designer can monitor the timing execution of a task or of a group of tasks. The observer code is designer defined, thus she has the freedom to observe the execution parameters she wants.

6.5 Summary

This chapter presented a performance analysis simulator for the DIPLODOCUS UML models. The SystemC simulation code is generated from UML models using the TTool toolkit and a model translator (written in JAVA) for this purpose. Concurrency and timing issues are modeled using the SystemC constructs (modules, events) in order to use the SystemC simulator kernel to manage the simulation timing. Furthermore, performance Observers are defined to monitor the system's execution and gets the advanced performance metrics.

The next chapter will present the modeling of an industrial system, LTE protocol's physical layer implemented on a Freescale multi-core platform. The complete system is modeled using the DIPLODOCUS profile and its extensions presented in the previous chapters. The performance estimation is done using the simulator described in this chapter and promising results shows that the global approach is efficiently usable in a design methodology.

Part III

Approach's Validation

Chapter 7

Use Case Study: SoC Modeling for LTE Base Station

This chapter is dedicated to the validation of the proposed system-level modeling approach described in the previous chapters. The contribution of this thesis work is demonstrated by applying the proposed modeling concepts to a LTE communications system. First, a brief description of the LTE standard is presented. Then a discussion on the design challenges imposed by the standard in terms of throughput and processing power requirements. This chapter describes the use case study modeling. It is considered the physical layer of LTE to exemplify the proposed methodology. The execution platform is a (or architecture) is a six-core DSP communicating via a matrix interconnect and a three-level memory hierarchy: cache, on-chip memory, and off-chip memory. The mapping defines the scheduling and arbitration policies to access the resources (using the virtual node concept) and the allocation of architecture resources for the application. The modeled system is simulated using the SystemC simulator described in chapter 6. The simulation results, including performance metrics and parameters extracted using the observers (such as the time required to process a packet received by the LTE physical layer) are compared with an existing implementation of the LTE platform.

7.1 LTE: The Long Term Evolution Standard

The first mobile communication protocol (GSM) was circuit switched and voice oriented, however, it provided some data services such as text messaging. Later on, the first 3G networks were based on circuit switched data, with packet-switched services as an add-on. It was until the 3G evolution into HSPA and later LTE/LTE-Advanced that packet-switched services and IP became the main objective of protocol's design. The voice services provided by the earlier protocols remain in LTE but they will be

provided over IP. The main drivers of LTE are:

- Data rate: Voice services still have high priority in LTE. However they require lower data rates. Data applications such as web-browsing, streaming and file transfer drive the data rates for mobile systems, from orders of kbit/s in 2G networks to orders of Mbit/s in 3G, getting even close to Gbit/s in 4G networks. The higher data rate services drive the design of the radio interface.
- Delay: Interactive services such as real-time gaming, but also web browsing and interactive file transfer, have requirements for very low delay. There are, however, many applications such as e-mail and television where the delay requirements are not as strict. One of the major quality of services provided by LTE, is the guarantee of a small delay of data transfer.
- Capacity: From the mobile system operator's point of view, it is not only the peak data rates provided to the end-user which are important, but also the total data rate that can be provided on average from each deployed base station site per hertz of licensed spectrum. This measure of capacity is called spectral efficiency. In the case of capacity shortage in a mobile system, the Quality-of-Service (QoS) for the individual end-users may be degraded.

7.1.1 Overall LTE Network Architecture

Figure 7.1 describes the LTE architecture and network elements. This figure shows as well the division of the architecture into two main high level domains: Evolved UTRAN (Evolved Universal Terrestrial Radio Access Network E-UTRAN) and Evolved Packet Core Network (EPC). The high level architectural domains are functionally equivalent to those in the existing 3GPP systems. The new architectural development is limited to Radio Access and Core Networks, the E-UTRAN and the EPC respectively.

E-UTRAN and EPC together represent the Internet Protocol (IP) Connectivity Layer. This part of the system is also called the Evolved Packet System (EPS). The main function of this layer is to provide IP based connectivity. All services are offered on top of IP, and circuit switched nodes and interfaces seen in earlier 3GPP architectures are not present in E-UTRAN and EPC at all. IP technologies are also dominant in the transport, where everything is designed to be operated on top of IP transport.

Evolved Packet Core Network (EPC)

The Evolved Packet Core (EPC) also known as core network, enables packet communication with the Internet. The Serving Gateways (S-GW) and Packet Data Network Gateways (P-GW) ensure data transfers and Quality of Service (QoS) to the mobile

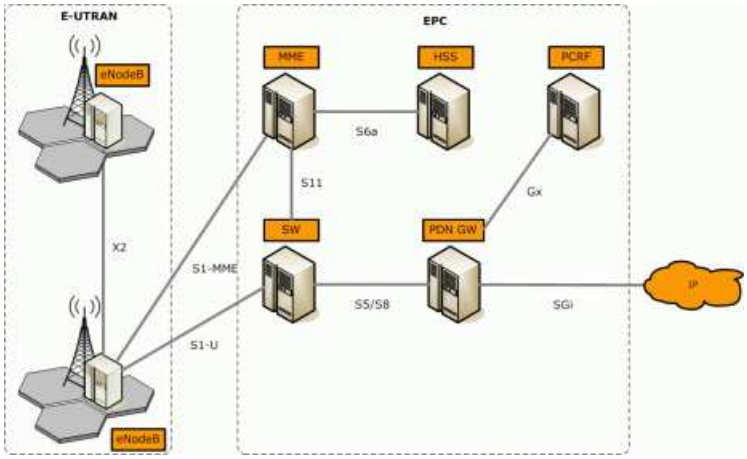

Figure 7.1: LTE Network

user equipment (UE). The Mobility Management Entities (MME) are scarce in the network. They handle the signaling between UE and EPC, including paging information, UE identity and location, communication security, and load balancing. The radio-specific control information is called Access Stratum (AS). The radio-independent link between core network and UE is called Non-Access Stratum (NAS). MMEs delegate the verification of UE identities and operator subscriptions to Home Subscriber Servers (HSS). Policy Control and charging Rules Function (PCRF) servers check that the QoS delivered to a UE is compatible with its subscription profile. For example, it can request limitations of the UE data rates because of specific subscription options.

One of the big architectural changes in the core network area is that the EPC does not contain a circuit switched domain, and no direct connectivity to traditional circuit switched networks is needed in this layer. Functionally, the EPC is equivalent to the packet switched domain of the existing 3GPP networks. There are, however, significant changes in the arrangement of functions and most nodes and the architecture in this part should be considered to be completely new.

The Evolved Universal Terrestrial Radio Access Network (E-UTRAN)

The E-UTRAN is responsible for all the radio-related functionality of the overall network, including for example, scheduling, radio-resource handling, re-transmission protocols, coding, and various multi-antenna schemes. The EPC is responsible of functions that are not related to the radio interface but which are needed for providing a complete mobile-broadband network. This includes for example: Authentication, charging functionality, and setup of end-to-end connections. Handling these functions separately, instead of integrating them into the RAN (Radio Access Network). This is beneficial as it allows to serve several radio-access technologies with the same core

network.

The development in E-UTRAN is concentrated on one node, the evolved Node B (eNodeB). All radio functionality is collapsed there, i.e. the eNodeB is the termination point for all radio related protocols. As a network, E-UTRAN is simply a mesh of eNodeBs connected to neighboring eNodeBs. An U-UTRAN manages the radio resources and ensures the security of the transmitted data. It is composed entirely of eNodeBs. One eNodeB can manage several cells. A cell is usually three-sectored with three antennas (or antenna sets), each covering 120 degrees. The user mobile terminals (commonly mobile phones) are called User Equipment (UE). At any given time, a UE is located in one or more overlapping cells and communicates with a preferred cell; the one with the best air transmission properties. LTE is a duplex system, as communication flows in both directions between UEs and eNodeBs. The radio link between from eNodeB to the UE is called the **downlink** and the opposite link between UE and its eNodeB is called **uplink**. These links are asymmetric in data rates as most Internet services require higher data-rates for the downlink than for the uplink. Fortunately, it is easier to generate a higher data rate signal in an eNodeB than in an UE, which is powered by a battery.

7.1.2 Key Technologies of the 3GPP LTE Air Interface

OFDMA for Downlink Multiple Access

OFDMA, *Orthogonal Frequency-Division Multiple Access*, is a digital multi-carrier modulation method OFDM used for 3GPP LTE and several other radio-access technologies, such as WiMAX [9] and DVB broadcast technologies [7]. OFDMA is a technology that has been shown to be well suited to the mobile radio environment for high rate and multimedia services [80].

OFDM achieves high data rate and efficiency by using multiple overlapping carrier signals instead of just one carrier. The key advantage of OFDM over single carrier modulation schemes is the ability to subdivide the bandwidth into multiple frequency sub-carriers which carry the information streams. These sub-carriers are orthogonal to each other delivering higher bandwidth efficiency. Therefore, a guard time is added in each OFDM symbol to combat the channel delay spread, this guard time is called cycle prefix.

OFDM signal generation consists of multiplexing the original data stream into *Nc* parallel data streams; each one of these data streams is modulated with a different sub-carrier frequency using linear modulation. Then, the resulting signals are transmitted together in the same band. Correspondingly, the receiver consists of Nc parallel receiver paths because of the Nc equally spaced orthogonal sub-carriers of OFDM symbol behaves as Nc independent narrow-band flat fading channels. In short, OFDM converts the wide-band frequency selective fading channel into Nc narrow-band flat

fading channels thus the equalization can be performed in the frequency domain by a scalar division carrier-wise with the sub carrier related channel coefficients. This fact reduces dramatically the equalization complexity.

While OFDMA is adopted for the LTE downlink multiple access, SC-FDMA (a technique built over OFDM modulation) is a new multiple access technique that utilizes single carrier modulation, DFT spread orthogonal frequency multiplexing, and frequency domain equalization. It has a similar structure and performance as OFDM. SC-FDMA is currently adopted as the uplink multiple access scheme for 3GPP LTE.

MIMO

Multiple-Input Multiple-Output (MIMO) is a technique based on the use of multiple antennas at both the transmitter and receiver to improve radio link communication performance. MIMO technology is considered in the new wireless communications standards such as 3GPP LTE or WIMAX since it offers significant increases in data throughput and link range without additional bandwidth or transmit power. It achieves this by higher spectral efficiency (more bits per second per hertz of bandwidth) and link reliability or diversity (reduced fading).

MIMO can be split into transmit diversity and spatial multiplexing techniques and it depends on the channel condition which MIMO technique to select. Transmit diversity increases coverage and quality of service (QoS) because it relies on transmitting multiple redundant copies of a data stream to the receiver; while spatial multiplexing increases the spectral efficiency because it transmits independent and separately data streams from each of the multiple antennas.

MIMO systems present two modes of operation: Open-loop and closed-loop. While open loop MIMO systems only knows the channel state information (CSI) at the receiver side, closed-loop MIMO systems also knows the CSI at the transmitter side and it can improve the throughput and reliability of a MIMO system.

The MIMO technique combined with OFDM (MIMO-OFDM) [84] have demonstrated to deliver high spectral efficiency. This is appropriate for wide-band systems because OFDM simplifies the receiver structure. This is achieved by decoupling selective flat fading MIMO channels into sub-carriers. Then the fading process experienced by each sub-carrier is close to frequency flat, and therefore, it can be modeled as a constant complex gain. This consideration allows to obtain the MIMO channel matrix of transmission coefficients per sub-carrier and simplify the implementation of a MIMO scheme provided this is applied on a each sub-carrier.

7.1.3 LTE Radio Link Protocol Layers

The information sent over an LTE radio link is divided in two categories: The user-plane, which provides data and control information, irrespective of LTE technology.

The second category is the control-plane information, which gives control and signaling information for the LTE radio link. The protocol layers of LTE are displayed in Figure 7.2 and are subdivided in:

1. *Packet Data Convergence Protocol* (PDCP) performs IP header compression to reduce the number of bits to transmit over the radio interface. PDCP is responsible for ciphering and, for the control plane, integrity protection of the transmitted data, as well as in-sequence delivery and duplicate removal for handover. The service provided by PDCP to transfer IP packets is called a radio bearer. A radio bearer is defined as an IP stream corresponding to one service for one UE.

2. *Radio-Link Control* (RLC) performs the data concatenation and then generates the segmentation of packets from IP-Packets of random sizes, which comprise a Transport Block (TB) of size adapted to the radio transfer. The RLC layer also ensures ordered delivery of IP-Packets; Transport Block order can be modified by the radio link. Finally, the RLC layer handles a re-transmission scheme of lost data through a first level of Automatic Repeat reQuests (ARQ). The RLC provides services to the PDCP in the form of radio bearers. There is one RLC entity per radio bearer configured for a terminal.

3. *Medium-Access Control* (MAC) handles multiplexing of logical channels, hybrid-ARQ re transmissions, and uplink and downlink scheduling. The MAC provides services to the RLC in the form of logical channels. Finally, the MAC layer contains the scheduler that is the decision maker for both downlink and uplink radio parameters.

4. *Physical layer* (PHY) handles coding/decoding, modulation/demodulation, multi-antenna mapping, and other typical physical-layer functions. This layer creates physical channels to carry information between eNodeBs and UEs and maps the MAC transport channels to these physical channels. The use case modeled in the following sections focuses on the physical layer with no distinction drawn between user and control planes. The physical layer manipulates bit sequences called Transport Blocks.

An example of the LTE Data Flow through the protocol layers

During the data transmission flow, the data block are segmented and concatenated layer after layer, all the way from the original data in IP packets to the data that is sent over the air. Figure 7.2 illustrates this flow for the downlink, it shows an example of three IP packets (two on one radio bearer and one on another radio bearer). It summarizes these block operations.

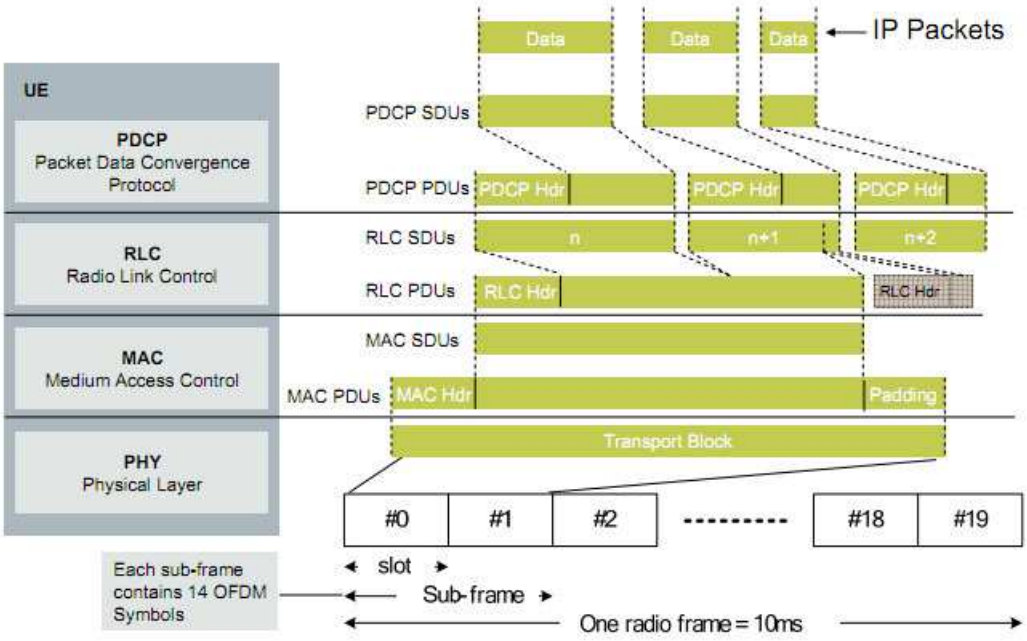

Figure 7.2: LTE Data Flow through the protocol layers

The data flow in the case of uplink transmission is similar. The PDCP performs (optionally) IP-header compression, followed by ciphering. A PDCP header is added, carrying information required for deciphering in the terminal. The output from the PDCP is forwarded to the RLC. The RLC protocol performs concatenation and/or segmentation of the PDCP SDUs and adds an RLC header. The header is used for in-sequence delivery (per logical channel) in the terminal and for identification of RLC PDUs in the case of re transmissions. The RLC PDUs are forwarded to the MAC layer, which multiplexes a number of RLC PDUs and attaches a MAC header to form a transport block. The transport-block size depends on the instantaneous data rate selected by the link adaptation mechanism. Thus, the link adaptation affects both the MAC and RLC processing. Finally, the physical layer attaches a CRC (Cyclic Redundancy Check) to the transport block for error-detection purposes, performs coding and modulation, and transmits the resulting signal, possibly using multiple transmit antennas.

7.2 Use Case Modeling

The following sub-section will describe briefly the LTE uplink physical layer signal processing part and its modeling using the extended DIPLODOCUS methodology. Our objective is to study its execution on a Freescale multi-core platform and analyze

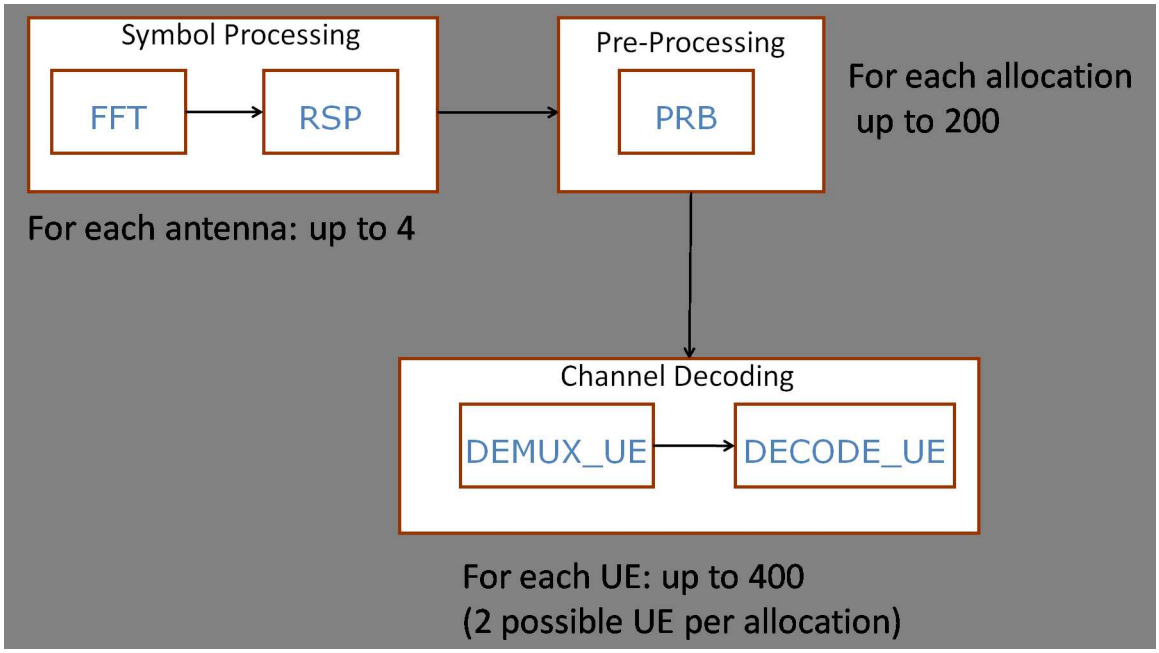

Figure 7.3: LTE Physical Layer uplink flow

the global performances as shown in section 7.3). Our objective is to compare the modeling results to the existing implementation.

7.2.1 Scope of the use case

The scope of this modeling effort, made to validate the modeling approach presented in the earlier chapters, is illustrated in figure 7.3. It concentrates on the uplink LTE physical layer in the eNodeB (in other words the signal processing part of the LTE standard). While, the downlink baseband process is itself divided into channel coding that prepares the bit stream for transmission and symbol processing that adapts the signal to the transmission technology, the uplink baseband process performs the corresponding decoding (figure 7.3). The physical layer uplink and downlink baseband processing must share the eNodeB digital signal processing resources which requests a high computational power. The objective of this study is to model the most computationally demanding use cases of LTE, i.e, the high uplink bandwidth (20 MHz).

An eNodeB can have up to 4 transmit and 4 receive antenna ports. For each of these antenna a symbol processing based on FFT is executed (The RSP: Reference Symbol Processing). The eNodeB controls both uplink and downlink time and frequency allocations. The allocation base unit is a block of 1 millisecond (corresponding to the Transmission Time Interval: TTI). Figure 7.4 depicts the LTE frame structure. LTE frames (Transport Blocks) are 10 msec in duration and are segmented into byte aligned

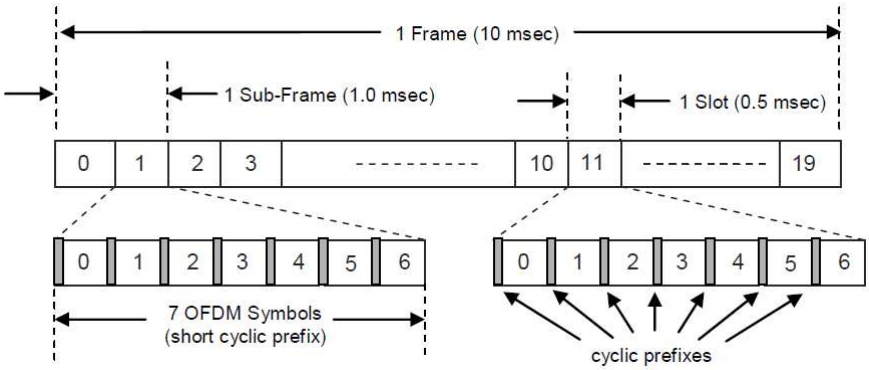

Figure 7.4: LTE Generic Frame Structure

segments with a maximum information block size of 6144 bits. They are divided into 10 subframes, each sub-frame being 1.0 msec long. Each sub frame is further divided into two slots, each of 0.5 msec duration. Slots consist of either 6 or 7 OFDM symbols, depending on whether the normal or extended cyclic prefix is employed. The OFDMA symbol structure consists of three types of sub-carriers, data sub-carriers for data transmission, pilot sub-carriers for estimation and synchronization purposes and Null sub-carriers for no transmission. OFDMA may support frequency reuse of one, i.e., all cells/sectors operate on the same frequency channel to maximize spectral efficiency.

The smallest amount of resources that can be allocated in the uplink or downlink to a user is called a resource block (RB). An Physical RB (PRB) is 180 kHz wide and lasts for one 0.5 ms time slot. For standard LTE, an RB comprises 12 sub-carriers at a 15 kHz spacing (cyclic prefix). The maximum number of RBs supported by each transmission bandwidth is 200 resource blocks for a bandwidth of 20 MHz. PRB main functions are SNR (Signal to Noise Ratio) estimation and MMSE equalization. One resource block can contain data related to one or two users.

The PRBs are mapped to a Virtual Resource Blocks (VRB) in a localized or distributed manner. The frequency and time allocations to map information for a certain user to PRBs is determined by the eNodeB scheduler depending on the actual radio channel and transmission traffic.

Figure 7.3 represents the data flow processing of the LTE uplink physical layer. For each antenna, there is a reference symbol processing task (RSP), then for each resource block allocation (number of allocation can be up to 200) a Physical Resource block is executed, and then, depending if the resource block is allocated to one or two users, a VRB or two respectively are executed.

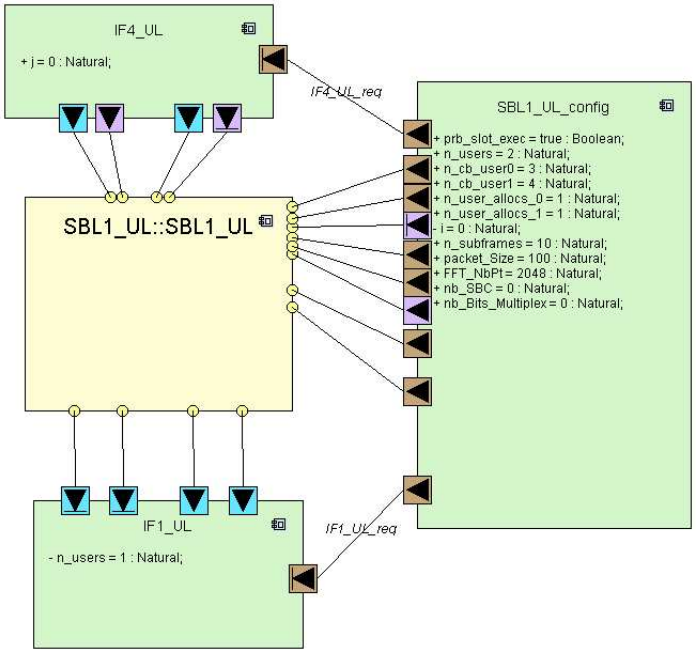

Figure 7.5: The higher level of hierarchy in the LTE uplink physical layer model

7.2.2 Application Model: LTE Physical Layer

The modeling of an application in DIPLODOCUS starts with the modeling of its structure. We used the DIPLDOCUS Application component model to model the LTE uplink physical layer. The component diagram, due to the hierarchy it defines, permits to model complex applications. In fact, it would have been very difficult to model this use case using the simple task model. Components' ports provide interfaces to describe the connections between the application parts.

The Top level of the structure hierarchy is depicted in figure 7.5. The different components composing the application have the following functionalities:

- IF4-UL is the input interface with a buffering large enough to allow layer 1 uplink processing of the sub frame N, while SC-FDMA symbols of the sub frame N+1 are being received and OFDMA symbols of the sub frame N-1 are being sent. Typically, a double buffering mechanism is selected which allows coping with up to 1 ms processing uplink and downlink latency inside the layer.

- IF1_UL is the output interface of Layer 2 towards higher protocol level (the MAC layer) .

- SBL_Uplink_config is the component that define the models parameters such as the number of LTE sub-frames for the input, number of users per allocation,

etc. By modifying the behavior of this component the designer can test different execution scenarios.

- The composite component SBL1_UL represents the uplink model itself. It encapsulates the set of components that are needed to process a sub frame.

The data flow of the protocol goes through components ports through DIPLODOCUS channels. On the other side synchronization information are transferred via events and requests. Figure 7.6 depicts the DIPLODOCUS model of an LTE allocation. As defined in the previous section, the data processing starts with the physical resource block (PRB) for each allocation (up to 200 possible allocation) then each resource block could have data belonging to two different user equipment. The events (purple component ports) carry the actual processed sub-frame helping to trace the data processing. This information is used later by observers to calculate the overall time needed to execute an LTE sub-frame.

Data belonging to different users can be processed in parallel by different cores, thus, the tasks and the channels and the events are duplicated for each user (as well for each allocation) which increase the complexity of the model. We made the choice to model only two allocations and there are two users for each allocation.

The behavior of tasks and primitive components in DIPLODOCUS is described using the DIPLODOCUS activity diagrams as defined in section 4.2.3 in Chapter 4. Figure 7.7 depicts an excerpt of the activity diagram that models the behavior of one of the PRB tasks. The task starts executing after receiving a request from the SBL1_UL_config, then for each user (one or two possible users, this number is parameter defined as well by SBL1_UL_config), the PRB task calculates the number of cycles to execute (PRBExec) and the data samples size it will exchange (read or write) with the other tasks (through the channels).

Number of cycles of the abstract execution commands

The number of cycles to execute by the tasks in the LTE use case is based on the number of bits it receives multiplied by the number of cycles needed by the task to perform the processing of one bit. This number is either estimated, called as well intelligent guess based on the expertise of the designer, or it is known as in our case where Freescale designers provided us with this number. In fact, the code representing the LTE uplink physical layer exists and after its execution on the platform (without cache misses and contentions) we calculated the number of cycles.

7.2.3 Architecture Model: Freescale MSC8156 Multi-Core DSP

The main objective of this use case study is not to perform an architecture exploration but rather to test if the DIPLODOCUS model and the analysis performed on it match

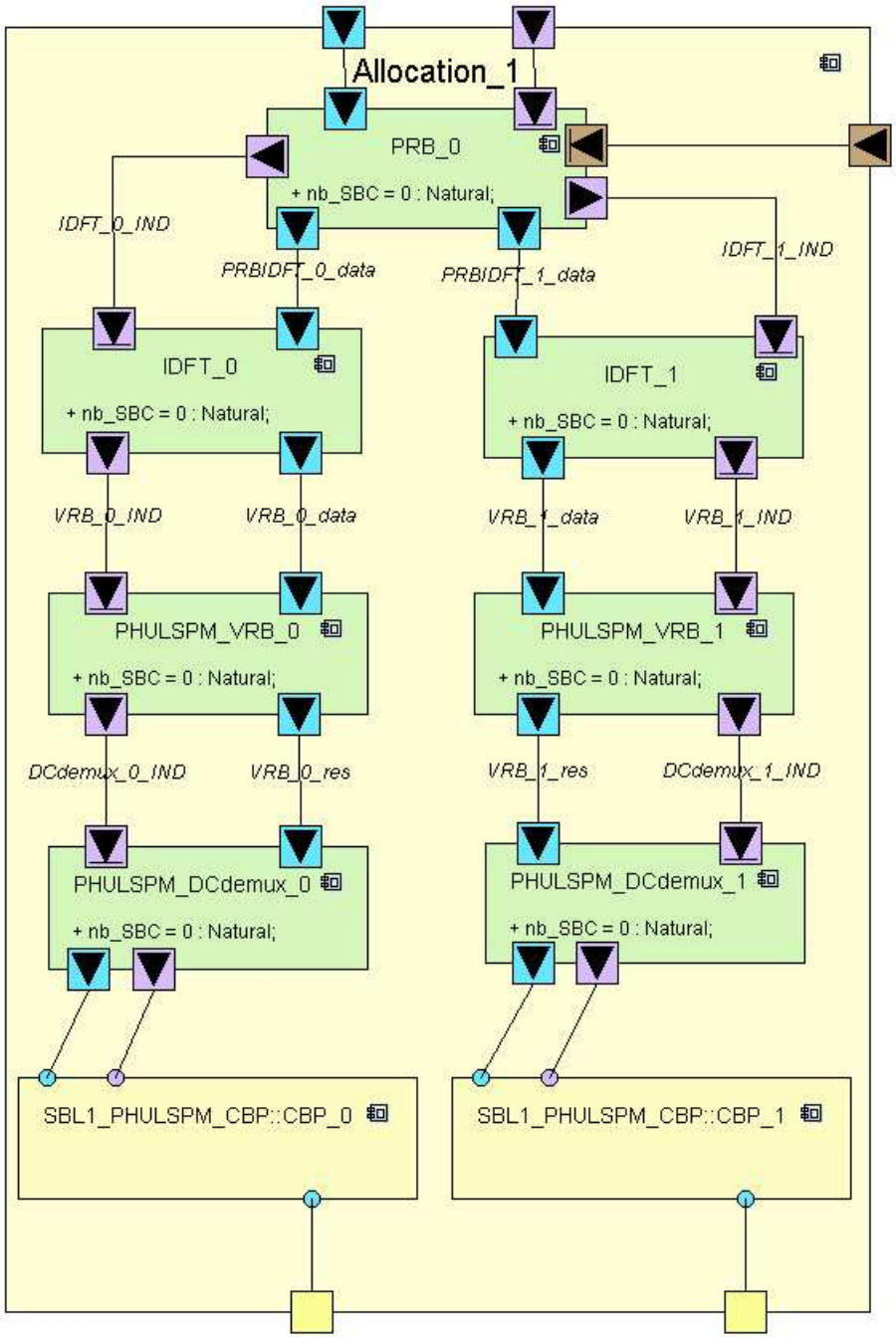

Figure 7.6: Application Structure Task metamodel

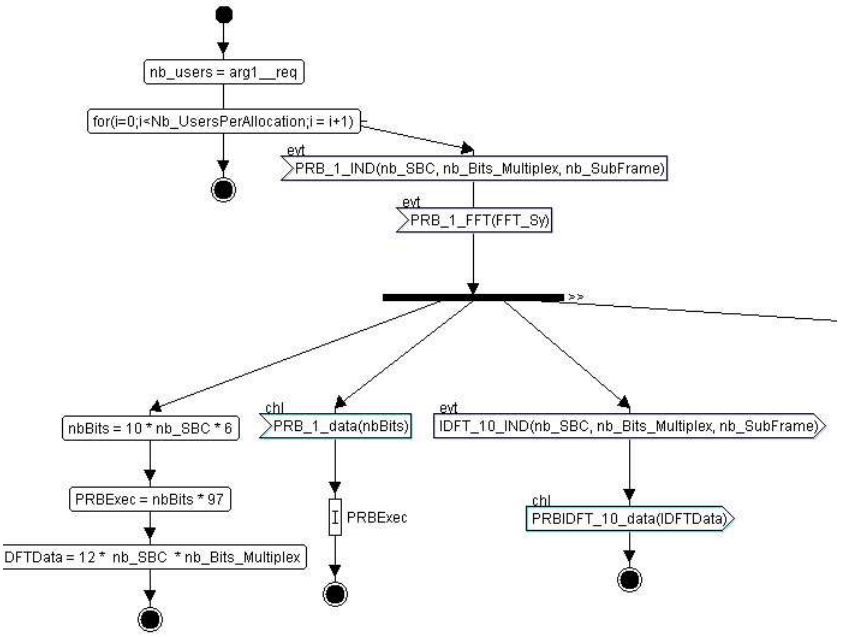

Figure 7.7: An excerpt of the Behavior of a Physical Resource Block (PRB) Task

the real implementation. Thus, the architecture was chosen to be an existing Freescale LTE implementation. Figure 7.8, represents the DIPLODOCUS model of the Freescale MSC8156 DSP that is integrated in a vast spectrum of LTE base stations.

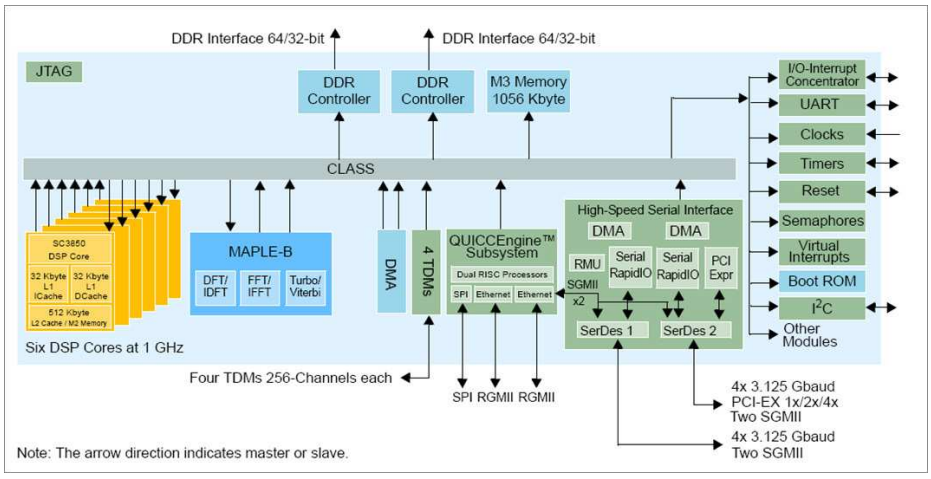

Figure 7.8: An example of a DIPLODOCUS Architecture Model with TTool

This architecture contains six DSP cores connected through a ”non-blocking” communication matrix (called CLASS). A hardware accelerator (the Maple) that performs

the most expensive (in computing power) tasks (such as the FFT and the CRC). The memory system is a three level hierarchy, where each DSP has its own memory, and there is an on-chip shared memory (M3) and two off-chip shared memories (DDR1 and DDR2). DSP cores and Maple can access the shared memories directly using the CLASS or via the system DMA. The DSP cores communicate with the Maple via its own memory accessible via the Maple DMA.

After specifying the architecture resources and their parameters (frequency, data size etc.), we defined the communication patterns for of the computation resources (the six DSP cores and the Maple accelerator).

Cache effect estimation

The DSP cores are doted with L1 caches for both data and instruction. The presence of these caches is an additional factor that affects the estimation accuracy. The main objective of cache memory is to reduce the total amount of application cycles, since there are fewer stalls caused by communication with memory. However, they make system analysis more complex due the induced non-determinism.

In this use case, the cache miss ratio was provided as input to the DIPLODOCUS architecture model by system engineers based on their knowledge of the LTE protocol and the DSP implementation. The objective is to quantify and model the variation of an application execution time in a system with cache.

7.2.4 Mapping Model

After defining the application and architecture models of the use case, this section describes the mapping model that enables the execution of the application on the architecture.

Shared resources modeling and access policies

The modeling of shared resources is done using the concept of virtual node. Each virtual node has an access policy that determines at each time for which requester the shared resource will be allocated. Figure 7.9 presents the SystemC code of a priority based access policy for a computation virtual node. It returns a pointer to the highest priority task that is ready to execute. By redefining the C++ method "<SchedPolicy"> the designer can redefine her/his own (new) access policy.

A task can be in different scheduling states (depicted in figure 7.10) depending on its internal execution sequence or on the access policy decisions, these states are:

1. Idle: This execution is the first or a new one of the task
2. Ready: Task is ready to execute and wait to be selected by the access policy

```
/*this access policy is for a priority based scheduling
 * it will choose from a list of ready task the one with
 * the highest priority
*/
      TMLTask* TMLPriorityBased::SchedPolicy(TMLTask* RunningTask){
            _RunningTask = RunningTask;
            TML_TaskList::iterator i;
            TML_TaskList::iterator j;
            bool found_Higher_Prio;
            TMLTask* TempTask;
            TMLTask* NextTask;
            int HighestPrio = 0;
            NextTask = _RunningTask;

            // get the priority of the actual running task to compare it to the priorities of other ready tasks
            if (!_RunningTask == 0)
                  HighestPrio = _RunningTask->GetPriority();
            else
                  HighestPrio = 100;// 100 the lowest priority for the scheduler


            NextTask = 0;
            for ( i=_ReadyTaskList.begin(); i!=_ReadyTaskList.end(); ++i){
                  TempTask = *i;
                  if (TempTask->GetPriority() < HighestPrio ){
                        HighestPrio = TempTask->GetPriority();
                        NextTask = *i;
                  }
            }
            if (NextTask == 0 && _RunningTask != 0)
                  NextTask = _RunningTask;// the actual running task is the highest priority and will continue to run

            return NextTask;
      }
```

Figure 7.9: An example of a priority based access policy for a computation virtual node

3. Running: task is executing its sequential behavior (DIPLODOCUS commands)

4. Preempted: Task execution is preempted by the access policy because other(s) task(s) with higher priorities are ready to execute. During this state the task is ready to run. We choose to not merge the preempted and the ready state to facilitate further analysis especially this help to know the time during which the task was preempted.

5. Blocking: Task is blocked while trying to read/write on a communication channel (DIPLODOCUS events or channels)

6. Terminated: Task has finished its execution. If task is periodic or requested by other tasks the next state will be the idle state

Table 7.11 depicts the set of virtual nodes used to control the architecture nodes, as well as their access policies. Each architecture node is controlled by a virtual node. The computation virtual nodes had a priority based access policy, as the one described above, while communication and storage virtual nodes use a first come first served policy.

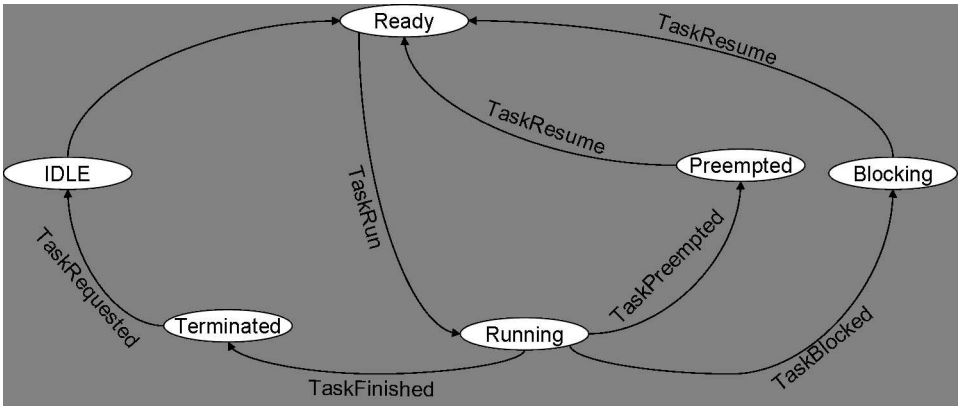

Figure 7.10: Possible States of a task during its execution

Virtual node Type	Architecture Nodes	Virtual node Name	Virtual Node Access Policy Type
Computation VN	Six DSP Cores	VN_Core1	Priority based
		VN_Core2	Priority based
		VN_Core3	Priority based
		VN_Core4	Priority based
		VN_Core5	Priority based
		VN_Core6	Priority based
	Four Hardware PE: Processing Elements	VN_MapleFFT	Priority based
		VN_MapleDFT	Priority based
		VN_MapleTVPE	Priority based
		VN_MapleCRC	Priority based
Communication VN	Two DMA	VN_SystemDMA	First Come First Served
		VN_MapleDMA	First Come First Served
	One Non blocking communication matrix	VN_SwitchingMatrix	First Come First Served
Storage VN	One on-chip memory	VN_M3	First Come First Served
	Two off-chip memories	VN_DDR1	First Come First Served
		VN_DDR2	First Come First Served

Figure 7.11: Virtual nodes and their access policies

Use Case Execution and Storage Allocation

The execution allocation corresponds to define for each task which virtual node will control its access to the computation resource. In this case study a one level scheduling

hierarchy (no scheduling or dynamic scheduling is adopted). Figure 7.12 depicts an excerpt of the SystemC code representing this step. For intance the task IF1_UL is mapped to the virtual node of the core 1, while the task CBP_0 is mapped to the virtual node of the core 2.

```
VN_Core1.AddTask(&(Application_P->task__IF1_UL));
VN_Core1.AddTask(&(Application_P->task__IF4_UL));
VN_Core1.AddTask(&(Application_P->task__SBL1_UL_config));
VN_Core1.AddTask(&(Application_P->task__CBP_1));
VN_Core2.AddTask(&(Application_P->task__CBP_0));
VN_Core1.AddTask(&(Application_P->task__CBP_1_0));
VN_Core2.AddTask(&(Application_P->task__CBP_1_1));
VN_Core2.AddTask(&(Application_P->task__PHULSPM_DCdemux_1));
VN_Core4.AddTask(&(Application_P->task__PHULSPM_DCdemux_0));
VN_Core4.AddTask(&(Application_P->task__DCdemux_1_0));
VN_Core4.AddTask(&(Application_P->task__DCdemux_1_1));
VN_Core2.AddTask(&(Application_P->task__PHULSPM_VRB_1));
```

Figure 7.12: Excerpt of the Execution allocation of LTE uplink to the computation Virtual nodes

After defining the execution allocation, the next step is to define the storage allocation. In other words to define on which memories is located the data and the code of tasks and the data corresponding to application channels. We reproduced the same storage allocation as done in the Freescale implementation. Tasks' code was distributed between the on-chip memory M3 (70% of the data) and on the off-chip memory DDR (the remaining 30 %) while tasks' data was on the DDR. Channels samples were memory mapped on the M3 and on the DDR memories.

7.3 Use case analysis

This section details the results of the LTE uplink physical layer case study. After describing the application and architectures models in the previous section, this section presents the results of the mapping and the performance estimation calculated using the simulation environment described in chapter 6. All results are gathered on a 2.53 GHz Intel i5 core laptop running Windows 7 with 4GB of RAM.

Table 7.1 resumes the use case parameters. It has been assumed an uplink bandwidth of 20 MHz, and a MIMO configuration of 2 antennas at the receiver. The use case intends to model the processing of ten LTE sub-frames, each sub-frames contains

Parameter	Value
Carrier Frequency	2.14 GHz
Uplink Bandwidth	20 MHz
MIMO receivers	2
Transmission Time Interval: TTI	1 ms
Sub-frame duration	0.5 ms
OFDM symbols per TTI	14
Number of PRB allocation per sub-frame	2
Number of sub-carriers per PRB	12
Number of uplink users per allocation	2
Use case total number of sub-frames	10
Use case total number of users	20

Table 7.1: LTE uplink physical layer use case parameters

two PRB allocations. The number of uplink users per allocation is two, thus this use case model the data transmission and processing of 20 users.

7.3.1 Application Execution Performance Metrics

The UML models of the use case are transformed to SystemC code and simulated using the simulation environment presented in chapter 6. Figure 7.13 shows a sample of the execution flow of the LTE uplink physical layer tasks on the modeled architecture. The IF4_UL interface sends the data relative to each sub-frame, then the reference symbol processing is done (the FFT and RSP tasks). The figure depicts the execution of data related to only one user (the use case covers 4 users). The PRB tasks forward the data related to the user to the other tasks (IDFT, VRB, DCDemux, and CBP). IF1_UL waits until all the processing for all of the users (4 in this case) is done representing the end of the processing on the relative sub-frame. For instance, the red line (between 1 ms and 2 ms) represents the end of the processing of the first sub-frame.

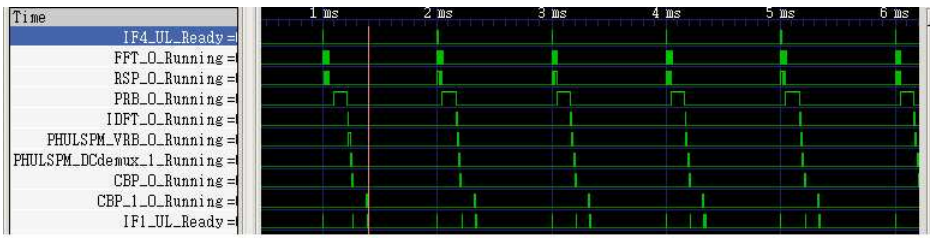

Figure 7.13: LTE uplink physical layer flow for one user

Figure 7.14 traces the execution of the task PRB_0 during the first sub-frame.

At the beginning, it was blocked, PRB_0_Blocked, waiting for data from the reference symbol processing part (FFT and RSP). Then, once the data is available the task changes of state to access the channel and read the corresponding data (PRB_0_AccessChannel). Then the task is now allowed to execute its computation part (PRB_0_Running), an execution that supposes that instructions and data are in the cache. However, as defined earlier, the cache miss ratio is set for this use case at 4%, thus, there is a cost to pay for fetching data from external memories (PRB_0_CacheMiss). Then, the task will write data relative to the following tasks in the flow (as showed in figure 7.13) and its state passes again to PRB_0_AccessChannel, once this data exchange accomplished the task is again blocked waiting for new data to process.

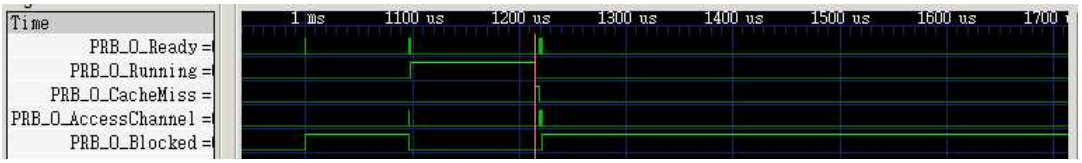

Figure 7.14: An excerpt of the execution of the PRB0 task

After examining the LTE uplink execution flow using the VCD diagrams, figure 7.15 shows the percentage of computation taken by each group of tasks on the overall execution time. The ”PRB” part, represents the sum of time consumed by the two allocations, is the most demanding on computation resources. The previous figures enable the designer to quickly evaluate the execution of the system, identify the computation demanding tasks, and validate that the processing flow is executing as desired.

7.3.2 Comparison of simulation results to real implementation results

Given that the execution of the real LTE uplink physical layer code on the Freescale DSP platform is very accurate, it is used as a reference for validation. The results are reported in table 7.16. This table validates the accuracy of our models. It shows for each group of tasks the simulated and the measured MCPS (Millions of Cycles per second) consumed to accomplish its execution. The approach presented in this PhD work proved its accuracy as the error percentage is less than 10%. Thus the designer can be confident of the modeling results.

In addition to the total number of MCPS, the above table provides as well a comparison of the number of MCPS consumed for memory access (Memory MCPS). This metric represents the total number of cycles consumed when the application tasks were exchanging data or suffering from cache misses and the architecture computation nodes were performing this exchange of data (through access to communication resources).

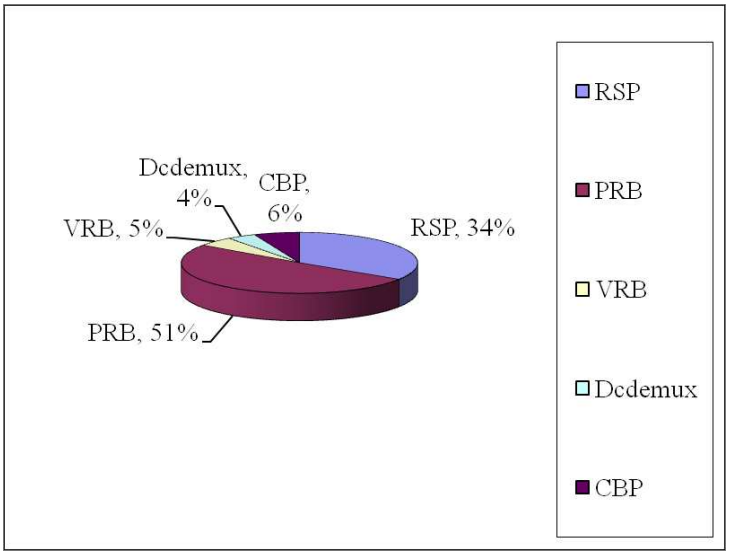

Figure 7.15: LTE uplink Tasks computation complexity

	MCPS			Core MCPS		Memory MCPS	
Task Group	Simulated	Measured	Error %	Simulated	Measured	Simulated	Measured
RSP	353,4	361,3	2,2	316,0	326,4	37,4	34,9
PRB	524,8	540,2	2,8	497,6	500,9	27,2	39,3
VRB	51,7	52,6	1,8	44,4	45,0	7,3	7,6
Dcdemux	40,9	41,8	2,2	35,9	25,9	5,0	15,9
CBP	64,6	65,0	0,6	57,5	54,9	7,1	10,1

Figure 7.16: LTE Tasks execution performance metrics

The last metric shown in this table is the ”Core MCPS” that represents the total number of cycles consumed by computation resources to execute application tasks but without the communication cost.

Figure 7.17 depicts the comparison between the number of MCPS measured on the real implementation and the number of MCPS extracted using the DIPLODOCUS UML models and SystemC simulator. One should note that the simulation time needed to perform the model’s simulation is faster than the real time execution. In fact, this use case represents the processing of ten LTE sub-frames that need to be executed in less than 1 ms for each, thus 10 ms for the sub-frames. the model’s simulation took 4

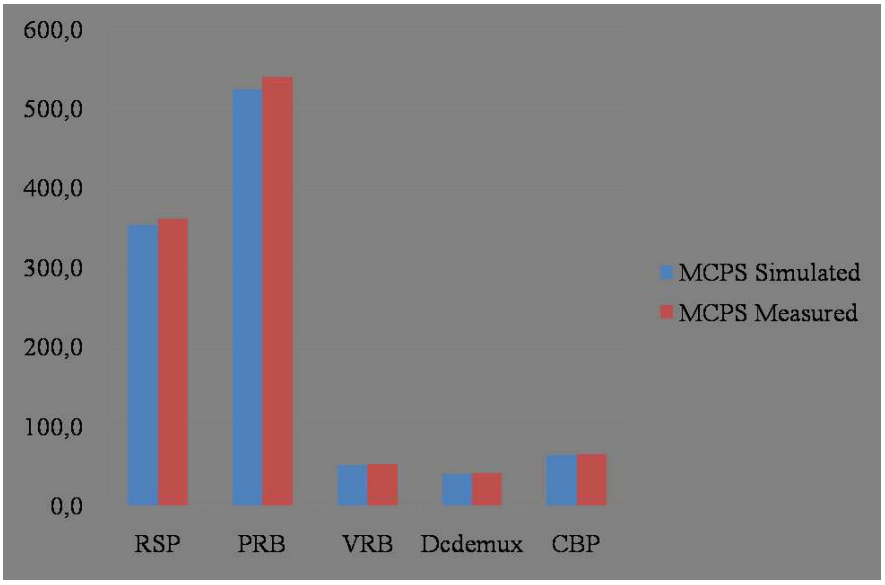

Figure 7.17: Simulated MCPS compared to measured MCPS

ms.

7.4 Use case study conclusion

The modeling approach proposed in this PhD thesis was applied to a real implementation example. The comparison of the modeling/simulation results with the real implementation confirmed the accuracy and efficiency of our approach. The case study in the previous section proved that using the input and help of design engineers and feeding the DIPLODOCUS models with good estimates about the tasks execution complexity enabled us to obtain an accurate estimation the system's global performances.

However, one should ask how good are the data. What happens if the complexity estimates proved to be wrong or far from the real results. Furthermore, DIPLODOCUS models are ideally created earlier in the design flow, so what if the designers integrated new functionalities or architecture elements that were not modeled. The answer to these questions would be that the designer should keep her/his models updated with the latest implementation results. A perspective research step is to connect DIPLODOCUS to the design flow and automatically update the models with new results.

Chapter 8

Conclusions and Perspectives

This thesis began with the statement that system level design and high level analysis is the solution to the increasing complexity of modern SoC. Abstraction, by hiding some aspects of the design, helps the designer to only consider those which help her/him to take early design decisions. Abstraction coupled with modeling reduce the complexity and enable faster system analysis. However, the challenge remains to preserve the analysis results accuracy.

This thesis presented our contributions for a system level design methodology for SoC performances estimation. Those contributions were on both methodology and tools levels. These include the definition of modeling constructs with UML meta-models and thus participating to the definition of the DIPLODOCUS methodology developed by Telecom ParisTech, a SystemC simulation environment to analyze the UML models and the modeling of an industrial use case to validate the proposed approach.

Firstly, new UML modeling components, the communication patterns, were defined to ensure the separation of concerns of the computation and the communication. They enable the modeling of the communication interactions between the architecture resources to deliver (read or write) application data to storage resources. Secondly, virtual nodes were defined to capture the shared resources impact on the overall system performances, they allow the modeling of access policies that control the shared resources.

While the DIPLODOCUS UML high level modeling is used to model the system, a SystemC simulation environment is used to analyze its performance. UML models are transformed into SystemC code that capture the concurrency and timing behavior of the models and use them to extract performance metrics such as latency, throughput and end-to-end delays. The proposed simulation is proven faster than the real-time execution.

Even though the proposed modeling approach is targeting system design at high level of abstraction, the presented contributions demonstrate promising results and the

comparison of the modeling/simulation results with the real implementation confirmed the efficiency and accuracy of our approach.

Limitations and possible enhancements

Chapter 7 proved that the methodologies proposed and developed in this thesis dissertation is capable of correctly estimating system's performance when the computation complexity of system's tasks is approximatively known, or proposed by the implementation engineers due to their experience. However, what if this information is not available in advance? The system designer using DIPLODOCUS and its extensions will use some estimates, called in the industry "intelligent guess" to build the model. Thus, the estimates may vary from the real implementation results. In addition, with advances of the design project, the designer will have newer estimates and needs to re-inject these estimates in the model. If not, the model will be outdated. A possible enhancement to the DIPLODOCUS methodology is to integrate a **feedback loop** (back annotation) to update the model estimates with new experimental results when available.

Perspectives

Power consumption estimation. The simulation environment presented in this thesis is mainly dedicated to extract temporal performance metrics. Thus, the utilization (number of executed cycles and the number of idle cycles) of architecture nodes. Taking into consideration this metric, the designer can estimate the power consumption estimation of the system. In fact, designers has some good estimates on the energy consumption of a resource (a processor for example) to execute one cycle, as well as the energy consumed when the node is idle. Hence, it is possible to get some energy estimates for the DIPLODOCUS model. In addition, as the virtual nodes can easily implement new access policies, one can imagine that some energy aware access policies will be modeled and used in DIPLDOOCUS models.

Models transformation and exchange with other UML standard profiles. DIPLODOCUS is a UML profile that extends UML 2.0 to model SoCs on a high level of abstraction. It comes with a toolkit (TTooL [63]) that permits the generation from the UML models of simulation code (C++ and SystemC code) and of formal specifications (LOTOS and UPPAAL). However, DIPLODOCUS is not the only UML profile to target the SoC modeling, as some other profiles and particularly the Marte OMG profile had a similar objective. Some initiatives, such as open embedd [62], are trying to define a complete design flow starting from Marte models down to the implementation. Thus, one possible enhancement of the DIPLODOCUS UML

environment is to use model transformations tools (such as Kermeta [60]) to transform DIPLODOCUS UML models to Marte (or other UML profiles) and thus permits the use of their advantage. The transformations can be done in both ways so the other UML profiles can re-use the DIPLODOCUS environment.

Integration in a Design Flow. The model transformation enables the transformation of models build on a specific hypotheses (abstraction of some aspects of the system for instance) into models that focus on other aspects of the design. The transformation process can be as well from system level models to a lower level of abstraction. In the DIPLODOCUS context, this transformation consists first of all at enriching the DIPLODOCUS models with the lower level details, it can be on multiple steps and not completely automatic as the designer can use predefined libraries (RTL or TLM libraires). Thus, enabling the integration of DIPLODOCUS in a SoC design flow.

Bibliography

[1] IEEE standard verilog hardware description language. *IEEE Std 1364-2001*, pages 0_1 – 856, 2002. 32

[2] Object management group, uml profile for schedulability, performance, and time. 2005. Version 1.1. 2005. OMG document: formal/05-01-02. 42

[3] Information processing systems – open systems interconnection – lotos – a formal description technique based on the temporal ordering of observational behaviour. *ISO 8807:1989*, 2006. 98

[4] IEEE standard systemc language reference manual. *IEEE Std 1666-2005*, pages 0_1 –423, 2006. 33, 94

[5] Object management group, uml profile for system on a chip (soc). 2006. Version 1.0.1. OMG Document, 06-08-01. 41

[6] IEEE standard vhdl language reference manual. *IEEE Std 1076-2002*, pages c1 – 626, 2008. 32

[7] Digital video broadcasting (dvb): Framing structure, channel coding and modulation for digital terrestrial television, 2009. ETSI standard ETS EN 300 744 v. 1.1.2. 120

[8] Object management group, uml profile for modeling and analysis of real time and embedded systems. 2009. MARTE specification version 1.0. OMG document: formal/2009-11-02. 42

[9] Mobile wimax part i: A technical overview and performance evaluation, August 2006. WiMAX Forum. 120

[10] 3rd Generation Partnership Project: 3GPP. The mobile broadband standard, 2011. http://www.3gpp.org. 20

[11] 4th UML-SoC Workshop. Uml for soc design. http://websrv2.c-lab.de/uml-soc/uml-soc07/survey2007.pdf. 41

[12] Alcatel-Lucent. Lte products. http://lte.alcatel-lucent.com/. 21

[13] ALTERA. Avalon interface specification, 2011. http://www.altera.com/literature/manual/mnl_avalon_spec.pdf. 68, 167

[14] L. Apvrille, W. Muhammad, R. Ameur-Boulifa, S. Coudert, and R. Pacalet. A uml-based environment for system design space exploration. In *Electronics, Circuits and Systems, 2006. ICECS '06. 13th IEEE International Conference on*, pages 1272 –1275, dec. 2006. 25, 49, 51, 71, 159

[15] ARM. Amba bus specification, 1999. http://www.arm.com/products/system-ip/amba/amba-open-specifications.php. 68, 167

[16] F. Balarin, Y. Watanabe, H. Hsieh, L. Lavagno, C. Passerone, and A. Sangiovanni-Vincentelli. Metropolis: an integrated electronic system design environment. *Computer*, 36(4):45 – 52, april 2003. 26, 160

[17] F. Balarin, Y. Watanabe, H. Hsieh, L. Lavagno, C. Passerone, and A. Sangiovanni-Vincentelli. Metropolis: an integrated electronic system design environment. *Computer*, 36(4):45 – 52, april 2003. 35, 44

[18] Rabie Ben Atitallah, Philippe Marquet, Éric Piel, Samy Meftali, Smail Niar, Anne Etien, Jean-Luc Dekeyser, and Pierre Boulet. Gaspard2: from MARTE to SystemC Simulation. In *Proceeedings of the DATE'08 workshop on Modeling and Analyzis of Real-Time and Embedded Systems with the MARTE UML profile*, Washington, États-Unis, 2008. 25, 35, 36, 43, 46, 159

[19] A. Benveniste, P. Bournai, T. Gautier, M. Le Borgne, P. Le Guernic, and H. Marchand. The signal declarative synchronous language: controller synthesis and systems/architecture design. In *Decision and Control, 2001. Proceedings of the 40th IEEE Conference on*, volume 4, pages 3284 –3289 vol.4, 2001. 25, 159

[20] P.; Edwards S.A.; Halbwachs N.; Le Guernic P.; de Simone R. Benveniste, A.; Caspi. The synchronous languages 12 years later. Proceedings of the IEEE, 2003. 33

[21] G. Berry. The constructive semantics of pure esterel, 1999. 25, 33, 159

[22] A. Bobrek, J.J. Pieper, J.E. Nelson, J.M. Paul, and D.E. Thomas. Modeling shared resource contention using a hybrid simulation/analytical approach. In *Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings*, volume 2, pages 1144 – 1149 Vol.2, feb. 2004. 35, 36, 45

[23] C. Brooks, E.A. Lee, and S. Tripakis. Exploring models of computation with ptolemy ii. In *Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2010 IEEE/ACM/IFIP International Conference on*, pages 331 –332, oct. 2010. 26, 35, 160

[24] Tony Bybell. Gtkwave: a vcd waveforms wiewer, 2011. http://gtkwave.sourceforge.net/. 112

[25] Carbon. Soc designer plus: Rapid development of virtual platforms. http://www.carbondesignsystems.com/soc-designer-plus/. 26, 162

[26] Celoxica. Handelc language reference manual, 2005. http://babbage.cs.qc.edu/courses/cs345/Manuals/HandelC.pdf. 25, 159

[27] Bill Bunton Anna Keist David C. Black, Jack Donovan. *SystemC: From the Ground Up*. Kluwer Academic Publisher, 2 edition, 2009. 94

[28] D. Densmore and R. Passerone. A platform-based taxonomy for esl design. *Design Test of Computers, IEEE*, 23(5):359 –374, may 2006. 13, 37

[29] CoFluent Design. Cofluent, cofluentstudio. http://www.cofluentdesign.com/index.php/cofluent-studio. 26, 37, 160

[30] CoFluent Design. Cofluent studio extension for sysml and marte. http://www.cofluentdesign.com/index.php/ja_JP/solutions/uml-sysml-marte.html. 37

[31] Ericsson. Lte products. http://www.ericsson.com/ourportfolio/products/lte-radio-access-network-products?nav=fgb_101_220. 21

[32] 3GPP Releases for HSPA. The hspa broadband standard, 2011. http://www.3gpp.org/HSPA. 20

[33] 3GPP Releases for LTE. The lte broadband standard, 2011. http://www.3gpp.org/LTE. 20, 112

[34] Jianwen Zhu Domer R. Gerstlauer A. Shuqing Zhao Gajski, D.D. *SpecC: Specification Language and Methodology*. Kluwer Academic Publishers, 1 edition, 2002. 33

[35] Hubert Garavel, Radu Mateescu, Frédéric Lang, and Wendelin Serwe. Cadp 2006: A toolbox for the construction and analysis of distributed processes. In

Werner Damm and Holger Hermanns, editors, *Computer Aided Verification*, volume 4590 of *Lecture Notes in Computer Science*, pages 158–163. Springer Berlin / Heidelberg, 2007. 10.1007/978-3-540-73368-3_18. 98

[36] A. Gerstlauer, C. Haubelt, A.D. Pimentel, T.P. Stefanov, D.D. Gajski, and J. Teich. Electronic system-level synthesis methodologies. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 28(10):1517 –1530, oct. 2009. 29

[37] Frank Ghenassia. *Transaction-Level Modeling with Systemc: Tlm Concepts and Applications for Embedded Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. 25, 28, 159, 162

[38] Mentor Graphics. Catapultc an eda tool for full-chip high-level synthesis. `http://www.mentor.com/esl/catapult/overview`. 25, 159

[39] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous dataflow programming language lustre. *Proceedings of the IEEE*, 79(9):1305–1320, September 1991. 25, 33, 159

[40] T.A. Henzinger and J. Sifakis. The discipline of embedded systems design. *Computer*, 40(10):32 –40, oct. 2007. 13, 23, 156

[41] Thomas Henzinger and Joseph Sifakis. The embedded systems design challenge. In Jayadev Misra, Tobias Nipkow, and Emil Sekerinski, editors, *FM 2006: Formal Methods*, volume 4085 of *Lecture Notes in Computer Science*, pages 1–15. Springer Berlin / Heidelberg, 2006. 10.1007/11813040_1. 24, 158

[42] Anders Hessel, Kim Larsen, Marius Mikucionis, Brian Nielsen, Paul Pettersson, and Arne Skou. Testing real-time systems using uppaal. In Robert Hierons, Jonathan Bowen, and Mark Harman, editors, *Formal Methods and Testing*, volume 4949 of *Lecture Notes in Computer Science*, pages 77–117. Springer Berlin / Heidelberg, 2008. 10.1007/978-3-540-78917-8_3. 54, 98

[43] Open SystemC Initiative homepage. System level design with systemc. `http://www.systemc.org`. 33, 94

[44] huawei. Lte products. `http://www.huawei.com/radio_access_network/lte.do`. 21

[45] Chafic Jaber, Andreas Kanstein, Ludovic Apvrille, Amer Baghdadi, Patricia Le Moenner, and Renaud Pacalet. High-level system modeling for rapid hw/sw architecture exploration. In *Proceedings of the 2009 IEEE/IFIP International Symposium on Rapid System Prototyping*, RSP '09, pages 88–94, Washington, DC, USA, 2009. IEEE Computer Society. 25, 159

[46] S Jaddoe, M Thompson, and A. D Pimentel. Signature-based calibration of analytical performance models for system-level design space exploration. *Transactions on High-Performance Embedded Architectures and Compilers (Trans. on HiPEAC)*, 4(4), 2009. 45, 46

[47] Tero Kangas. *Methods and Implementations for Automated System on Chip Architecture Exploration*. PhD thesis, Tampere University of Technology, 29 September 2006. 25, 26, 35, 36, 46, 160

[48] T. Kempf, G. Ascheid, and R. Leupers. *Multiprocessor Systems on Chip: Design Space Exploration*. Springer, 2011. 31

[49] T. Kempf, M. Doerper, R. Leupers, G. Ascheid, H. Meyr, T. Kogel, and B. Vanthournout. A modular simulation framework for spatial and temporal task mapping onto multi-processor soc platforms. In *Design, Automation and Test in Europe, 2005. Proceedings*, pages 876 – 881 Vol. 2, march 2005. 45

[50] K. Keutzer, A.R. Newton, J.M. Rabaey, and A. Sangiovanni-Vincentelli. System-level design: orthogonalization of concerns and platform-based design. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 19(12):1523 –1543, dec 2000. 29, 31

[51] B. Kienhuis, E. Deprettere, K. Vissers, and P. Van Der Wolf. An approach for quantitative analysis of application-specific dataflow architectures. In *Application-Specific Systems, Architectures and Processors, 1997. Proceedings., IEEE International Conference on*, pages 338 –349, jul 1997. 31

[52] R. Kocik and Y. Sorel. A methodology to reduce the design lifecycle of real-time embedded control systems. In *Proceedings of European Simulation and Modelling Conference, ESM'04*, Paris, France, October 2004. 36

[53] Hermann Kopetz. The complexity challenge in embedded systems design. In *Proceedings of the 11th IEEE International Symposium on Object/Component/Service-Oriented Real-time Distributed Computing (ISORC 2008)*, Orlando, Florida, USA, May 2008. IEEE Computer Society. 13, 23, 156

[54] P. Kukkala, J. Riihimaki, M. Hannikainen, T.D. Hamalainen, and K. Kronlof. Uml 2.0 profile for embedded system design. In *Design, Automation and Test in Europe, 2005. Proceedings*, pages 710 – 715 Vol. 2, march 2005. 41

[55] C. Laot, A. Glavieux, and J. Labat. Turbo equalization: adaptive equalization and channel decoding jointly optimized. *Selected Areas in Communications, IEEE Journal on*, 19(9):1744 –1752, September 2001. 22

[56] Sebastien Le Beux, Laurent Moss, Philippe Marquet, and Jean-Luc Dekeyser. A high level synthesis flow using model driven engineering. In Guy Gogniat, Dragomir Milojevic, Adam Morawiec, and Ahmet Erdogan, editors, *Algorithm-Architecture Matching for Signal and Image Processing*, volume 73 of *Lecture Notes in Electrical Engineering*, pages 253–274. Springer Netherlands, 2011. 10.1007/978-90-481-9965-5_12. 25, 159

[57] Lip6. Dsx: Desidn space explorer. https://www-asim.lip6.fr/trac/dsx. 26, 160

[58] G. Martin, L. Lavagno, and J. Louis-Guerin. Embedded uml: a merger of real-time uml and co-design. In *Hardware/Software Codesign, 2001. CODES 2001. Proceedings of the Ninth International Symposium on*, pages 23–28, 2001. 25, 159

[59] W. Mueller, R. Domer, and A. Gerstlauer. The formal execution semantics of specc. In *System Synthesis, 2002. 15th International Symposium on*, pages 150 –155, oct. 2002. 33

[60] Pierre-Alain Muller, Franck Fleurey, Didier Vojtisek, Zoé Drey, Damien Pollet, Frédéric Fondement, Philippe Studer, and Jean-Marc Jézéquel. On Executable Meta-Languages applied to Model Transformations. In *Model Transformations In Practice Workshop*, Montego Bay, Jamaïque, October 2005. 44, 141, 170

[61] OMG. Object management group, 2011. http://www.omg.org/. 42, 50

[62] OpenEmbeDD. Open source platform for model driven engineering. http://openembedd.org/home_html. 140, 170

[63] TELECOM ParisTech. TTool toolkit, 2011. http://labsoc.comelec.enst.fr/turtle. 43, 44, 54, 86, 98, 140, 170

[64] O. Pasquier, F. Muller, J. P. Calvez, D. Heller, and E. Chenard. *The MCSE approach for system-level design*, pages 213–224. Kluwer Academic Publishers, Norwell, MA, USA, 2001. 37

[65] PCI-SIG. Pci express base specification, 2011. http://www.pcisig.com/specifications/pciexpress/. 68, 167

[66] A. D Pimentel. The artemis workbench for system-level performance evaluation of embedded systems. *Journal of Embedded Systems*, 3(3):181–196, 2008. 26, 160

[67] A.D. Pimentel, C. Erbas, and S. Polstra. A systematic approach to exploring embedded system architectures at multiple abstraction levels. *Computers, IEEE Transactions on*, 55(2):99 – 112, feb. 2006. 29

[68] Katalin Popovici, Xavier Guerin, Frederic Rousseau, Pier Stanislao Paolucci, and Ahmed Jerraya. Efficient software development platforms for multimedia applications at different abstraction levels. *Rapid System Prototyping, IEEE International Workshop on*, 0:113–122, 2007. 45

[69] A. Sangiovanni-Vincentelli. Quo vadis, sld? reasoning about the trends and challenges of system level design. *Proceedings of the IEEE*, 95(3):467 –506, march 2007. 24, 158

[70] J. Schnerr, O. Bringmann, A. Viehl, and W. Rosenstiel. High-performance timing simulation of embedded software. In *Design Automation Conference, 2008. DAC 2008. 45th ACM/IEEE*, pages 290 –295, june 2008. 45

[71] Ulf Schünemann. Modeling and abstraction, 2004. http://web.cs.mun.ca/~ulf/mod/rel.html. 30, 50

[72] Freescale Semiconductor. Freescale embedded processing solutions. http://www.freescale.com/. 22

[73] Freescale Semiconductors. Freescale msc8156: Six core high performance dsp, 2011. DSP Data Sheet. 22

[74] I. Shin, A. Easwaran, and I. Lee. Hierarchical scheduling framework for virtual clustering of multiprocessors. In *Real-Time Systems, 2008. ECRTS '08. Euromicro Conference on*, pages 181 –190, july 2008. 79, 164

[75] M. Silbermintz, A. Sahar, L. Peled, M. Anschel, E. Watralov, H. Miller, and E. Weisberger. Soc modeling methodology for architectural exploration and software development. In *Electronics, Circuits and Systems, 2004. ICECS 2004. Proceedings of the 2004 11th IEEE International Conference on*, pages 383 – 386, dec. 2004. 37, 45, 46

[76] OMG specification for UML. Unified modeling language resource page, 2011. http://www.uml.org/. 14, 34, 50

[77] Synopsys. Platform architect for soc architecture performance analysis and optimization. http://www.synopsys.com/Systems/ArchitectureDesign/pages/PlatformArchitect.aspx. 26, 160

[78] Clemens Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley, 2 edition, novembre 2002. 55

[79] Grant Martin Stuart Swan Thorsten Grötker, Stan Liao. *System Design with SystemC*. Kluwer Academic Publishers, 2002. 94

[80] 3GPP TR 25.892 V6.0.0. *Feasibility Study for Orthogonal Frequency Division Multiplexing (OFDM) for UTRAN enhancement (Release 6)*, 2004. 120

[81] A. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *Information Theory, IEEE Transactions on*, 13(2):260 – 269, April 1967. 22

[82] M. Waseem, L. Apvrille, R. Ameur-Boulifa, S. Coudert, and R. Pacalet. Abstract application modeling for system design space exploration. In *Digital System Design: Architectures, Methods and Tools, 2006. DSD 2006. 9th EUROMICRO Conference on*, pages 331 –337, 0-0 2006. 29, 35, 42, 46, 49, 51, 71

[83] RYSAVY Research white paper. Hspa to lte-advanced: 3gpp broadband evolution to imt-advanced (4g), September 2009. http://www.3gamericas.org/documents/3G_Americas_RysavyResearch_HSPA-LTE_Advanced_Sept2009.pdf. 20

[84] Hongwei Yang. A road to future broadband wireless access: Mimo-ofdm-based air interface. *Communications Magazine, IEEE*, 43(1):53 –60, jan. 2005. 121

Appendix A

Résumé en français

A.1 Introduction

Le développement et l'évolution des systèmes sur puces modernes (System-on-Chip "SoC") peuventêtre caractérisés par leur principale finalité qui est la miniaturisation. Des caractéristiques trés avancées sont associées à cette finalité en termes de performance, de consommation d'énergie, sécurité, viabilité et bien d'autres. Toutes ces caractéristiques sont réalisables grâce aux impressionnantes avancées techniques et technologiques dans les deux domaines des semi-conducteurs et de l'ingénieurie logicielle.

Par ailleurs, le cycle de développement devient de plus en plus court pour pouvoir confronter la concurrence. De plus et pour différencier leurs produits, les industriels intègrent plus de fonctionnalités ce qui augmente la complexité de conception. Par exemple, un téléphone mobile intègre aujourd'hui, en plus du protocole de télécommunication, un appareil photo, un lecteur de musique, un GPS, des outils de navigation sur Internet et de nombreuses autres applications.

Cette complexité de concpetion accrue oblige les concepteurs à re-évaluer leur méthodologies de conception. Initialement, la conception se faisait au niveau transistor, puis elle aévolué au niveau des portes logiques et enfin au niveau de transfert de registres (RTL: Register Transfert Level). Les outils développés au niveau RTL permettent la vérification de comportement du système. Cependant, la complexité et l'hétérogénéité croissantes des systèmes sur puces poussent les concepteurs àélever le niveau d'abstraction au niveau système afin de cibler la conception de l'ensemble du système, plutôt que de se focaliser sur les composants individuels.

Dans ce contexte, de nombreuses méthodes et approches de conception au niveau systèmes ontété proposées. Ces méthodologies fournissent aux concepteurs les moyens pour modéliser les systèmes, puis pour les analyser et les optimiser. L'analyse du système peut cibler différents domaines tels que la vérification fonctionnelle,

l'estimation des performances et l'estimation de la consommation d'énergie. En outre, les niveauxélevés d'abstraction permettent uneévaluation plus rapide des performance du système. Toutefois, le premier défi reste à définir le niveau d'absatrction approprié qui préserve la précision des résultats.

Comme les systèmes sur puce (SoC) les plus récents sont destinés à accueillir des applications complexes et interdépendantes à faible coût (surface, puissance) pour l'utilisateur final, le nombre de ressources doitêtre minimisé tout en augmentant leur utilisation en les partageant entre plusieurs applications. Ce partage de resources a un fort impact sur les performances à cause des contentions qu'il induit généralement. En effet, les performances globales d'une application donnée dépendent de la somme du temps nécessaire pour l'exécuter plus le temps de contention pour l'accès aux ressources partagées. La contention des ressources, les délais de communication et de nombreux autres facteurs ont un effet considérable sur les performances globales du système. De ce fait, l'identification des principaux facteurs critiques pour la performance du système et comment les modéliser constituent d'autres défis majeurs dans l'analyse des performances.

Une fois les métriques de performance (tels que la latence, le débit, l'utilisation des ressources, etc) identifiées et modélisées, le concepteur peutévaluer le système en cours de conception (par le biais des mécanismes de simulation et/ou d'analyse formelle) afin d'explorer diverses combinaisons d'architecture (nombre d'unités de calcul, topologie de communication à utiliser, hiérarchie de la mémoire, etc) et de définir les applications à considérer (nombre de tâches, parallélisme entre les tâches, etc.). L'objectif du concepteur est de vérifier si les performances du système satisfassent les exigences de conception. Ainsi, l'extraction rapide des métriques de performance nécessaires pour effectuer l'analyse du système est uneétape primordiale dans la conception. Le concepteur pourrait alors choisir, parmi différentes solutions possibles, celle qui est la plus adaptée. Ainsi le troisième défi à considérer concerne la façon d'analyser les métriques de performance du système à concevoir afin de vérifier le respect des exigences de conception.

Objectif de cette thèse. *L'objectif de ce travail de thèse est de fournir aux concepteurs systèmes les moyens nécessaires (au niveau méthodologique et au niveau outils) pour estimer les performances du système etévaluer rapidement les décisions de conception, idéalement trés tôt dans le flot de conception. La définition du niveau d'abstraction et des méthodes pour l'analyse des models abstraits constituent la clé pour une méthodologie résussie de conception au niveau système. La modélisation à haut niveau d'abstraction facilite les prises de décisions de conception au niveau système puisque le concepteur n'a pas besoin d'avoir une expertise approfondie des logiciels à bas niveau d'abstraction, du matériel et des outils de conception.*

Ce travail de thèse se situe dans le cadre de la méthodologie DIPLODOCUS

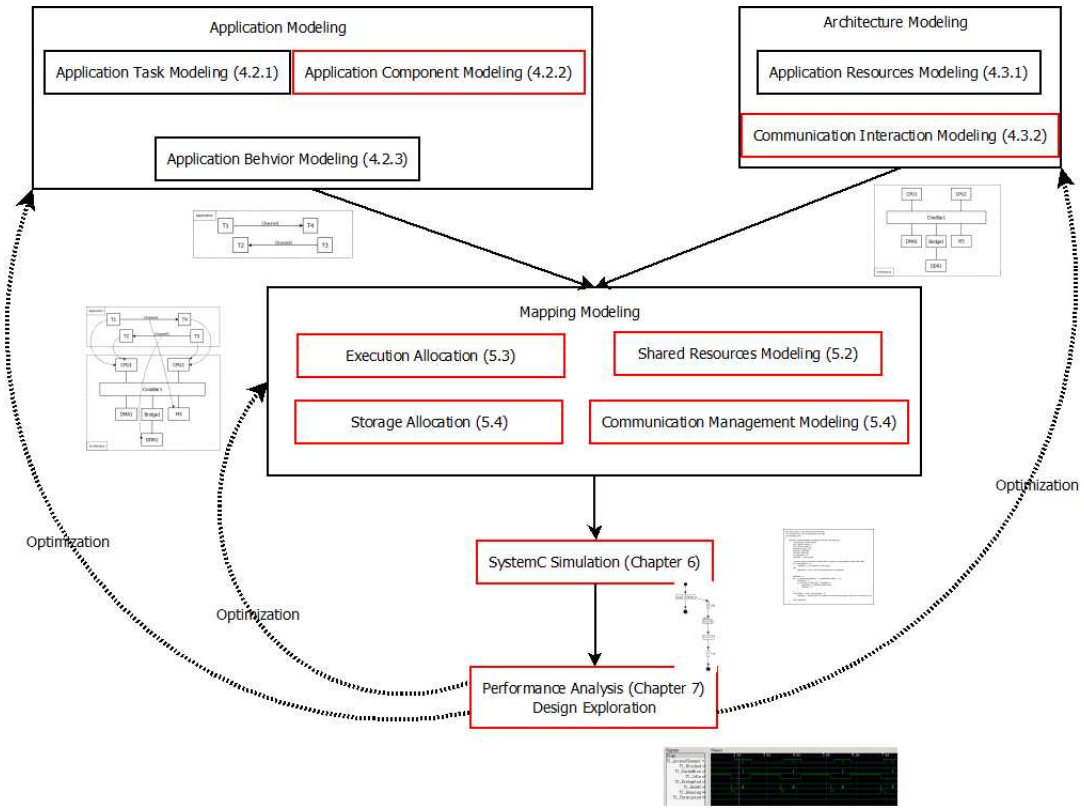

Figure A.1: Notre thèse dans le contexte de la méthodologie DIPLODOCUS

développée à Télécom ParisTech, qui définit un profil UML ciblant l'exploration de l'espace de conception à haut niveau d'abstraction. La figure A.1 présente nos contributions (encadrées en rouge) à la méthodologie DIPLODOCUS. Les numéros entre parenthèses spécifient le chapitre ou la section où chaque contribution est définie et expliquée. La section suivante décrit brièvement les différentes contributions de notre thèse.

A.1.1 Contributions de la thèse

Afin d'atteindre les objectifs identfiés ci-dessus, la thèse présente des contributions qui peuventétre classées en trois catégories:

- **Contributions au niveau conceptuel**: Définition du profile UML (les méta-modèles) DIPLODOCUS et surtout en enrichissant l'éxistant avec:
  - **La modélisation des resources partagées**: Permettant d'étudier l'effet de

ce partage sur les performances globales du système.

- L'**Orthogonalisation** des concepts de l'exécution et de la communication pour mieuxévaluer le coût des communications.

- **Contributions au niveau de la simulation**: Un simulateur en SystemC aété développé pour simuler les modèles UML proposés. La simulation est faite à haut niveau d'abstraction et elle est plus rapide que l'exécution en temps réel.

- **Contributions au niveau expérimental**: L'approche proposée aété appliquée sur uneétude de cas industriel. L'implémentation de la couche physique du protocle de télécommunication mobile de 4ème génération (LTE; Long Term Evolution) sur un DSP muli-core produit par Freescale aété modélisé en utilisant notre apporche et les résultats ontété validés en les comparant avec l'implémentation réelle, afin estimer la précision.

A.1.2 Plan de la thèse

Le rapport de thèse est divisé en six chapitres:

Le **chapitre 2** donne un aperçu du domaine de la conception de systèmes sur puce. Il décrit la complexité accrue de la conception qui se manifeste au niveau de l'architecture matérielle, logicielle ainsi que dans leur intégartion. Il décrit aussi le flot de conception typique qui, en partant des spécifications client et moyennant divers raffinements (modèle niveau système, prototype virtuelle, prototype physique), aboutit à un SoC fonctionnel. Ce flot couvre à la fois l'application et l'architecture ainsi que leur intégration et validation.

Le **chapitre 3** a pour objectif de situer la methodologie DIPLODOCUS ainsi que ses extensions présentées dans cette thèse par rapport aux autres méthodologies de conception niveau système. Après une brève description des objectifs et concepts niveau système (notamment la modélisation et l'abstraction, la séparation des concepts de l'application et de l'architecture, et l'exploration de l'espace d'architecture), il décrit les langages de spécifications les plus utilisés ainsi que quelques méthodologies existantes. Puis, dans une deuxième partie, ce chapitre compare les méthodologies existantes à celle que nous proposée. Ainsi une comapraison avec les autres profils UML est fournie.

Le **chapitre 4** fournit une définition sous forme de meta-modèle UML du profil DIPLODOCUS. Il définit ainsi le sous profil "ApplicationModeling" pour modéliser la structure (en modèle de taches ou en composants) et le comportement

(en diagramme d'activité UML) de l'application. Ensuite, il définit le sous profil "ArchitectureModeling" qui dans une première partie permet la modélisation des resources de l'architecture (resources d'exécution, de communication et de stockage) puis dans un deuxième temps introduit le concept de "motif de communication" (Communication pattern en anglais) pour modéliser l'interaction des resources d'exécution avec les resources de stockage à travers les resources de communication. En fait, un noeud d'exécution lorsqu'il essaie d'accéder (en lecture ou enécriture) à une resource de stockage, il suit un protocol bien déterminé qui est indépendant de l'application qui s'exécute. Ainsi pour séparer l'exécution de la communication, les motifs de communications définissent le protocol (ou les protocols) qu'une resource d'exécution pourra utiliser pour communiquer. De ce fait, et gréce au simulateur présenté au chapitre 6, le concepteur peut identifier le débit sur un motif d'exécution, l'utilisation des resources de communication et les possibles goulots d'étrangelement dus aux contentions. Les motifs de communication sont une extension du diagramme UML de séquence.

Le **chapter 5** propose un processus de "mapping" de l'application sur l'architecture (modélisé comme décrit dans le chapitre 4). Le mapping définit comment l'application va s'exécuter sur les resources de l'architecture. Etant donné, que dans un système sur puce moderne, des applications hétérogènes concourent pour accéder aux mêmes ressources d'exécution. Ces derniers partagent la même infrastructure de communication et des ressources de stockage. Les systèmes d'exploitation contrôlent l'exécution des applications sur les ressources d'exécution; les arbitres des bus assurent le contrôle des bus entre les différentes ressources d'exécution, et les contrôleurs de mémoire gérent les accès rivaux aux mémoires. Ainsi, la performance globale du système dépend des politiques d'accès aux ressources partagés. Le choix d'une politique d'accès peut affecter les performances du système complet. Ce chapitre propose le concept du "Virtual Node" (VN) pour analyser l'effet du partage des ressources sur les performances du système. Le VN est un composant de modélisation générique qui contrôle l'accès à une ressource partagée selon une politique d'accès (qui peutétre défini par le concepteur sous forme d'un diagramme d'activité UML). Il alloue la ressource partagée à un demandeur d'accès (par exemple le VN d'un CPU alloue le CPU pour une tache pour s'exécuter en la choisissant parmi toutes les taches prétes à s'exécuter). D'autre part, et afin de controler la communication des resources d'exécution, nous proposons le concept "gestionnaire de communication" qui gère la communication d'une entité d'exécution avec les autres entités de l'architecture. Un gestionnaire de communication possède une liste de toutes les resources de stockage à lesquelles une resource d'exécution peut accéder ainsi qu'une liste des motifs de communication (définis dans le chapitre 4) qui décrivent les protocoles de communications. Ce chapitre fini avec un simple exemple de modélisation qui illustre les différents concepts de modélisation introduits précedemment.

Le **chapter 6** se focalise sur l'analyse et l'estimation des performances des systèmes modélisés en utilisant les concepts proposés dans les chapitres 4 et 5. Après une description du langage SystemC et de la manière avec laquelle il gère la concurrence, le chapitre présente un simulateur développé en SystemC qui permet de simuler les modèles UML DIPLODOCUS et d'en extraire des métriques de performances (latence, débit, utilisation, contention sur les resources patagées, ...) ainsi que des graphiques au format VCD qui illustrent l'évolution de l'exécution de l'application sur l'architecture. Le simulateur permet aussi l'utilisation des observateurs, développés (en C++), par l'utilisateur pourétudier l'exécution d'une partie de l'application. Par exemple, un observateur peut calculer le tempsécoulé entre la réception des données et le traitement de ces derniers par une tâche ou encore le temps d'ecxcution de deux tâches, etc.

Le dernier chapitre, **chapter 7** est dédié à la validation de l'approche de modélisation proposée. Il représente la contribution au niveau expérimental de la thèse par application des concepts développés à un système de communication LTE. Après une bréve description du protocole LTE et des défits qu'il impose au niveau de la conception mais surtout aux niveaux du débit et de la puissance de traitement, ce chapitre décrit la modélisation du système. L'application considérée est la couche physique de LTE. L'architecture est UN DSP à six coeur communiquant via une matrice et une hiérarchie mémoire à trois niveaux: cache, mémoire sur la puce, et mémoire à l'extérieur accessible via un bus. Le mapping contient des politiques d'ordonnencements et d'arbitrage d'accès aux ressources. Ensuite, le chapitre présente les résultats de simulation, notamment les métriques de performances comme définis dans le chapitre 6, et des paramètres extraits grâce aux observateurs (comme le délai nécessaire au traitement d'un paquet LTE reçu par la couche physique). Ces paramètres sont comparés avec l'implémentation réelle de la plateforme LTE.

A.2 Complexité de la conception des systèmes sur puces

Un système sur puce (SoC) intègre des composantes logiciels et matériels et il est conçu pour fournir des fonctionnalités spécifiques à l'environnement [40], [53]. Le système doit être capable de réagir à des stimuli en continu et avec le comportement souhaité par le concepteur, c'est à dire, en satisfaisant des contraintes de temps d'exécution, de la consommation d'énergie, et du coût. Tels systèmes sont de plus en plus complexes et hétérogènes. La mise en oeuvre des SoC pour les applications modernes dans les différents domaines partage les tendances suivantes:

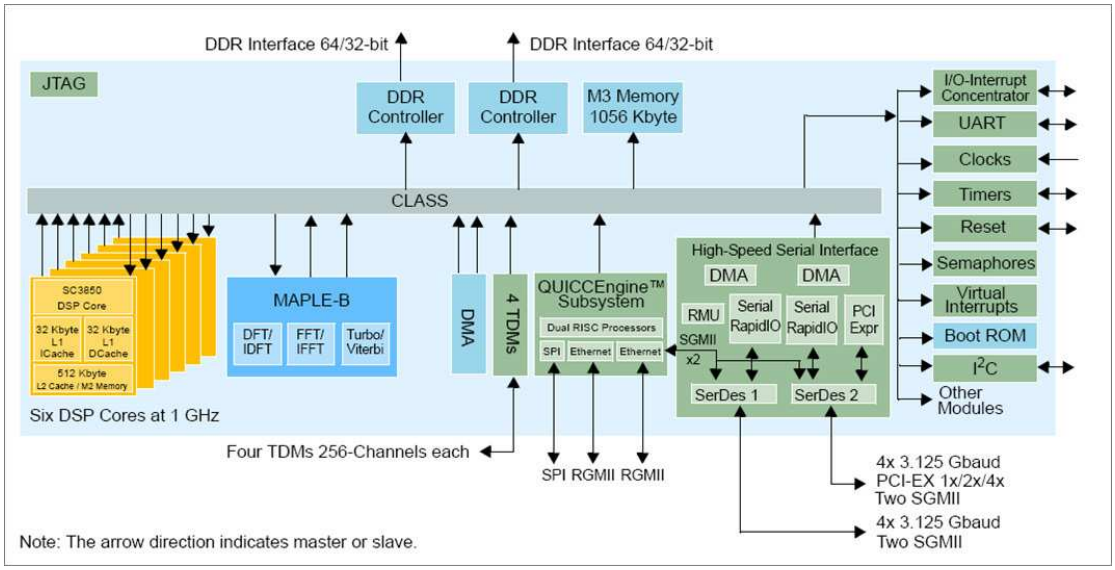

Figure A.2: Le DSP multi-coeur Freescale pour la couche physique du protocole LTE

1. Les nouvelles fonctionnalités et services à ajoutée conduisent à l'augmentation de traitement et les exigences de communication.

2. Les normes sont introduites plus rapidement et deviennent plus sophistiquées. Ceci exige une flexibilité accrue de la mise en oeuvre des SoC pour introduire els systèmes plus rapidement aux marchés.

3. La ré-utilisation des composants développés est augmenté, afin de raccourcir les délais de commercialisation et de réduire les coûts de production.

4. Pour les périphériques mobiles et les capteurs, une taille réduite et un meilleur rendement énergétique deviennent des facteurs de coût qui aident à la différenciation des produits.

En fait, la complexité de la conception du système a considérablement augmenté dans les trois dimensions suivantes:

1. L'hétérogénéité et la complexité de l'architecture matérielle: les nouvelles technologies de conception et la grande densité d'intégration ont augmenté la puissance de calcul des SoC modernes, ce qui permet à des fonctions encore plus sophistiquées à être enfouis dans des puces de plus en plus petites. Un SoC moderne peut intégrer des processeurs à usage général (GPP), des processeurs dédiés (ASIP), des topologies de communication différentes, une hiérarchie mémoire plus complexe, d'entrée différente et dispositifs de sortie ... Le Freescale DSPMSC8156 (figure A.2) est un bon exemple d'une telle architecture.

2. l'hétérogénéité et la complexité du logiciel embarqué: Comme la puissance de calcul des systèmes sur puce grandisse, des fonctionnalités plus avancées sont introduites. Le développement logiciel consomme actuellement une grande partie du budget de mise en oeuvre des SoC. Par exemple, plus d'un million de lignes de code sont intégrées aujourd'hui dans un téléphone mobile [69]. En contraste avec les systèmes logiciels traditionnels où le processus d'abstraction laisse de côté tous les aspects matériels du système et que seuls les aspects fonctionnels du code comptent, le logiciel embarqué est plus couplée à l'architecture matérielle ce qui limite la réutilisation du code, lorsque les spécifications du systèmeévoluent. En outre, des applications hétérogènes partagent la même architecture matérielle: par exemple, l'architecture matérielle d'un téléphone mobile doit être capable de gérer simultanément les appels vocaux et l'échange de données, tout en traitement des tâches complexe de traitement d'images comme la capture vidéo. Cette hétérogénéité augmente la complexité du partage des ressources et de l'optimisation de la conception.

3. La complexité de l'intégration: Au cours de la phase d'intégration, des composants logiciels et matériels sont intégrés afin de créer le SoC. Comme ces composants sont généralement développés par des équipes différentes et parfois dans des pays différents (et / ou par des sociétés différentes), cette phase est d'une extrême complexité [41]. Le travail requis pour la vérification et la modification au cours de cette phase est couteux et nécessite beaucoup de temps.

A.3 Flot de conception d'un système sur puce

La conception d'un système sur puce est un processus itératif, qui vise la mise en oeuvre d'un produit basé sur les spécifications du client. Un flot de conception du système sur puce est une succession d'étapes d'améliorations et d'optimisations pour aboutir à la conception du système. Chaque étape du flot se décompose en la modélisation et / ou la mise en oeuvre, la vérification et l'intégration des matériels et logiciels. Un flot idéal commence par une phase de description de haut niveau de la fonctionnalité et de l'architecture. Le concepteur s'attend à ce que le flux de conception de SoC lui permette de:

1. Ré-utiliser le code existant et les modèles développés pour les produits précédents pour accélérer le cycle de conception. Il faut noter que, généralement, les nouveaux produits sont uneévolution des produits existants et rarement une révolution. Par conséquent, la réutilisation des conceptions peuvent entraîner des réductions considérables du facteur coût.

2. Identifier les décisions de conception le plus tôt possible dans le flot de conception. Les modifications dans les phases ultérieures de conception sont coûteuses. Pour cela, le fait de trouver les goulets d'étranglement et d'estimer les performances du système tôt permettent de réduire le coût de la conception et à augmenter la productivité.

3. Fournir un produit fonctionnel, qui satisfait les exigences des clients. Ainsi, le processus adopté pour la vérification et la validation doitêtre correct et précis afin d'aboutir au produit exigé.

La spécification de la conception à chaque étape doit couvrir les spécifications de la fonctionnalité et de l'architecture du système. Actuellement, il n'ya pas un langage ou un format standard pour la spécification de la conception. En plus de la spécification, chaque niveau du flot de conception des SoC devrait définir des méthodes de vérification et de validation pour vérifier la satisfaction de la conception aux exigences.

Langages de spécifications

Plusieurs langages de programmation et de paradigmes de modélisation sont utilisés pour spécifier les fonctionnalités d'un SoC. Bien que le logiciel embarqué est généralement écrit en C, la fonctionnalité pourrait être décrite avec un paradigme différent: elle pourrait être directementécrites en C ou spécifiée en utilisant les langages synchrones comme Esterel [21], Lustre/SCADE [39], Signal [19], à partir de laquelle certains outils peuvent générer automatiquement du code en C et de vérifier formellement le système. Plus récemment, UML est proposé pour permettre la spécification des applications très complexes en fournissant un large éventail de constructions de langage. En outre UML a l'avantage d'être indépendant de l'implémentation [58][56].

Pour la conception de l'architecture matérielle, la conception au niveau système revient à n'importe quel niveau d'abstraction qui est au-dessus du niveau de transfert de registre (RTL). La modélisation au niveau transaction (TLM) [37], la modélisation comportementale, algorithmique, et fonctionnelle sont des termes souvent utilisés pour indiquer des niveaux plus élevés d'abstraction dans la conception du matériel. En raison de sa popularité et son efficacité, le langage de programmation C et ses langues dérivées gagnent du marché. De nombreuses approches et outils proposent la synthèse de code C en code RTL; Catapult C (Mentor) [38], Handel-C [26] sont des exemples bien connus de ces approches et d'outils. En outre, de nouvelles approches basées sur UML visent la modélisation du matériel et du logiciel des SoC. Où le code de simulation, la spécification de vérification formelle ou un code de synthèse peuvent être générée des modèles UML de haut niveau; Gaspard2 [18] [56], Koski cite Kangas, ainsi que la méthodologie Diplodocus décrite et étendue dans cette thèse [14][45].

Idéalement, un langage de spécification devrait couvrir plusieurs nombreux niveaux d'abstraction de sorte qu'il peut être utilisé tout au long du processus de conception.

Vérification et Validation

La phase de vérification et validation est une phase du développement du système, qui est effectué aux différents niveaux d'abstraction et où le logiciel et le matériel sont analysés pour vérifier qu'ils remplissent les propriétés et exigences souhaitées. Les techniques les plus courantes pour la validation et la vérification de conception sont la simulation et la vérification formelle. Bien que la simulation permet l'évaluation des systèmes complexes, la vérification formelle est un processus pour vérifier si un système satisfait une propriété donnée en vertu de toutes les entrées possibles, et elle est appliquée aux systèmes avec des contraintes critiques de surêté pour garantir l'exactitude.

A.3.1 Les étapes d'un flot de conception

La figure A.3 montre un flot typique de conception de SoC. Il commence à partir de la spécification du client et aboutit après diverses améliorations à un produit, idéalement prêt pour le marché. A partir des spécifications du client, les concepteurs expérimentés déduisent les exigences de conception et définissent un premier projet de la spécification du système. Puis, à travers trois étapes principales, la conception va évoluer de la spécification du client à un modèle au niveau système qui par le prototypage virtuel et une analyse plus approfondie aboutira au produit final avec la phase de prototypage. Dans chaque étape, le logiciel et le matériel sont pas idéalement développé/modélisé en parallèle et une étape d'intégration permet d'évaluer les progrès de la conception et la satisfaction aux exigences.

A.3.2 Conception au niveau système

Les approches pour élever le niveau de l'abstraction de la conception des SoC sont appelés "méthodologies de conception au niveau système". Leur objectif est d'aider les concepteurs à prendre et à valider les décisions de conception à un stade précoce de la conception du système. Ils permettent aux concepteurs de modéliser, de simuler, d'explorer, de vérifier et d'affiner une conception du système .Certains frameworks, en outre, fournissent un flot de conception en intégrant un ensemble de raffinements pour transformer un modèle au niveau système à une implémentation. Ptolemy [23], Artemis [66], CoFluent Studio [29], Metropolis [16], Koski [47], Design Space Explorer (DSX) [57], Platform Architect (CoWare) [77], SoC Designer (ARM - Carbon

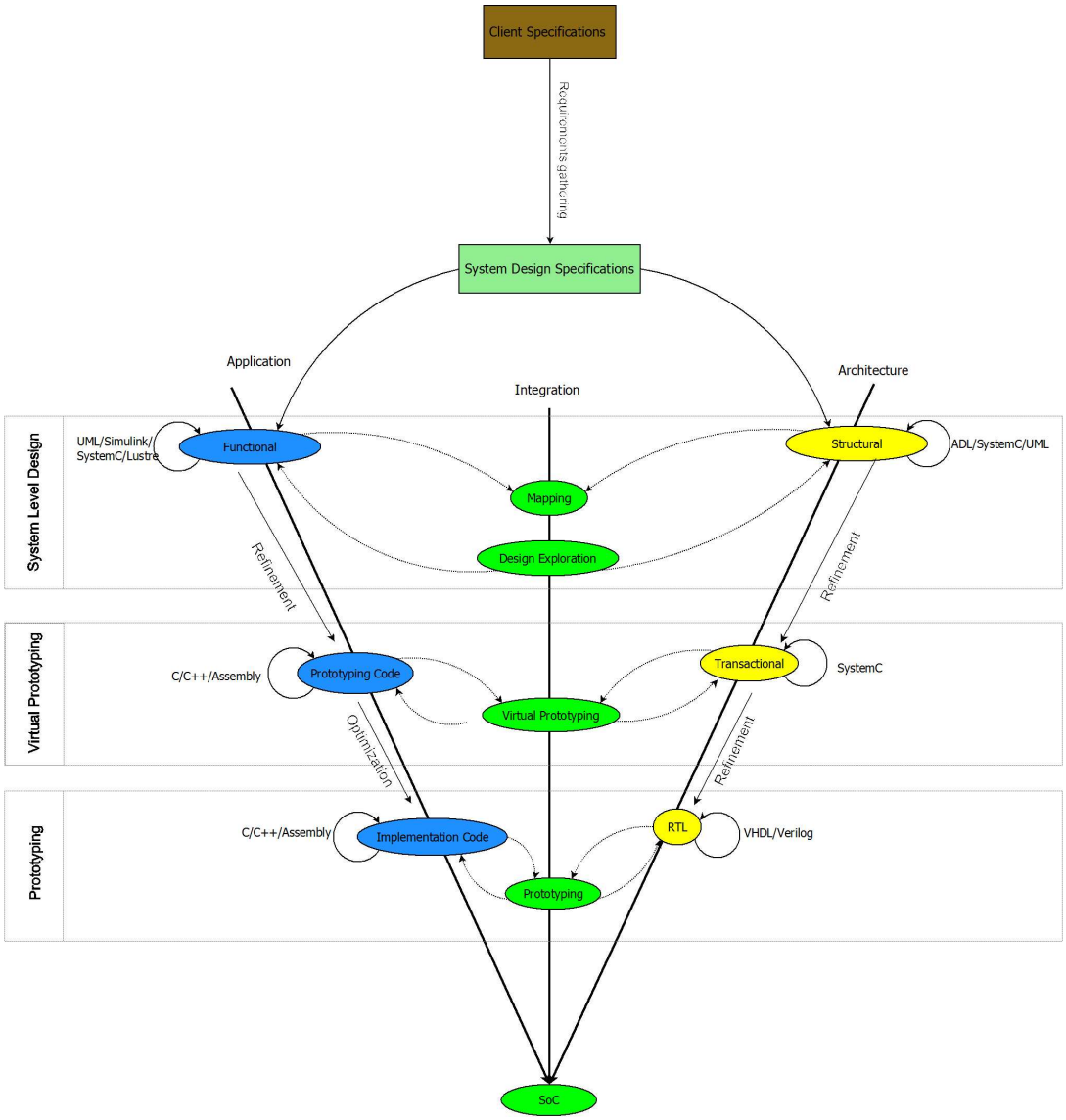

Figure A.3: Flot de conception pour les systèmes sur puces

Design Systems) [25] et beaucoup d'autres sont des exemples bien connus de framework et d'outils de conception au niveau système.

La plupart de ces approches adoptent une séparation claire entre la modélisation de l'application et celle de l'architecture. Ainsi, une phase de mapping est nécessaire pour intégrer les deux modèles et pour définir l'exécution de l'application sur l'architecture. Après la phase de mapping, le système estévalué afin de vérifier s'il répond aux exigences de conception. Cette phase est appelée: l'exploration de l'espace de conception. Son objectif est de trouver un modèle optimal qui correspond aux exigences. Les choix de conception comme le nombre de processeur nécessaires ou combien de mémoires on-chip et off-chip sont nécessaires, sont idéalement prises à ce stade. Dans la figure A.3, la conception au niveau système est la premièreétape après la spécification du système basé sur les spécifications du client. L'application et l'architecture sont modélisées d'abord, puis leur mapping et la phase d'exploration de l'espace de conception. En se basant sur la validation et le test du système au cours de cette exploration, le concepteur peut modifier ses modèles. Notre thèse se situe Ã ce niveau d'abstraction.

A.3.3 Prototypage virtuel

Le prototypage virtuel est la deuxième étape principale dans le processus de développement d'un SoC. Son objectif est de valider une conception donnée avant de s'engager à faire un prototype physique. En fait, le processus de conception souffre de problèmes de productivité dus à l'effort nécessaire pour vérifier et valider le système. La vérification du système est généralement effectuée à un niveau intermédiaire jusqu'à un faible niveau d'abstraction: les implémentations des prototypes logicielles au niveau des transactions et des modèles de l'architecture matérielle au niveau transfert de registres. Le prototypage rapide virtuel participe à la validation du système et à l'identification des goulots d'étranglement potentiels dans l'implémentation qui ne pouvaient pas être facilement identifiés dans l'étape précédente. Un prototype virtuel peut démontrer aux clients la faisabilité et de montrer l'évolution des technologies possibles, et donc de manière significative réduire le temps de mise sur le marché.

La modélisation au niveau transactionnel (TLM [37]) est très utilisée pour créer des prototypes virtuels. Cette étape intervient juste après l'étape de conception au niveau du système. Le code est plus mature et le concepteur peut faire une analyse plus précise. Même si, le temps de simulation est nettement plusélevé que lors de la conception au niveau système.

A.3.4 Prototypage

Le prototypage correspond à l'intégration du logiciel optimisé lors de l'étape de conception précédente sur le matériel du produit final ou sur une plateforme FPGA qui

correspond au matériel réel. Les derniers tests de vérification et de validation préparent pour l'étape finale de la conception de SoC correspondant à la création des masques matériels qui seront la base de la production en silicium du SoC. à cette étape du flot de conception, il est très coûteux de faire des changements de conception. Par conséquent, la grande majorité des outils et des approches tentent de valider le système lors des étapes précédentes.

A.4 Contributions de la thèse

A.4.1 Contrôle des ressources partagées: Le "virtual node"

Le *"noeud virtuel"* (VN pour Virtual node en anglais) est défini comme un composant générique de modélisation qui contrôle l'accès à une ressource en mettant en oeuvre une politique d'accès. il alloue la **ressource** contrôlée à un *demandeur*, par exemple, le VN d'un CPU alloue le CPU à une tâche qui est prête à s'exécuter, ou le VN d'un bus alloue la bande passante du bus à un CPU qui tente d'accéder à la mémoire ou à d'autres noeuds de l'architecture qui sont connectés au bus. Le noeud virtuel dispose d'une sémantique d'exécution en troisétapes:

1. Il attend les requêtes des ressources. Le VN les stocke dans une file d'attente.

2. Il sélectionne une requête en fonction de sa **politique d'accès**. Après l'écoulement d'un délai (le context switch), la ressource est allouée au demandeur.

3. Il attend que soit la demande sélectionnée termine son exécution ou qu'une nouvelle requête soit arrivée pour réévaluer l'allocation (étape 1).

Le noeud virtuel a un type qu'il hérite du type de la ressource qu'il contrôle. Par conséquent, un noeud virtuel pourraitêtre un VN d'exécution, de communication ou de stockage. En contrôlant une ressource partagée, le noeud virtuel divise la ressource entre les demandeurs, comme s'il crée une **ressource virtuelle** pour chacun d'eux. Cette fonction de vitalisation permette la définition des classes de politique d'accès, par exemple, un noeud d'exécution peutêtre virtuellement allouée à deux requêtes distinctes en utilisant une politique d'accès de partage du temps,ou plusieurs noeuds d'exécution peuventêtre regroupées en une seule ressource virtuelle dans un scénario d'ordonnancement dynamique. Pour contrôler les deux types de ressources (physiques et virtuels), il ya deux catégories de noeuds virtuels: le noeud virtuel local (*LocalVN*) qui contrôle l'accès à une ressource physique et le noeud virtuel générique (*GenericVN*) qui contrôle une ressource virtuelle.

Les GenericVN peuvent être empilés de façon hiérarchique, de telle sorte qu'un GenericVN peutêtre connecté à un autre GenericVN qui est connecté à un autre GenericVN ou à un LocalVN. Toutefois, un GenericVN pourrait être relié à un seul LocalVN. En outre, chaque ressource matérielle est contrôlée par un et un seul noeud local virtuel. Pour mieux illustrer ce mécanisme d'empilement, les deux sous-sections suivantes fournissent deux exemples sur la façon d'utiliser les concepts présentés ci-dessus sur deux politiques de partage des ressources bien connues: les scénarios d'ordonnancement hiérarchique et dynamique.

Ordonnancement hiérarchique

Les systèmes embarqués peuvent exécuter en temps réel et simultanément des applications hétérogènes. Par exemple, dans un téléphone mobile modern, des applications multimédias comme la vidéo peuvent être exécutées en parallèle avec des applications de contrôle (protocoles de télécommunication). Ces applications peuvent avoir des exigences spécifiques d'ordonnancement. En outre, l'application d'une seule politique d'accès à toutes les applications n'est pas la solution optimale [74].

Avec le concept des noeuds virtuels présenté ci-dessus, le concepteur peut utiliser un empilement hiérarchique de noeuds virtuels pour optimiser le partage des ressources lorsque des groupes hétérogènes demandent les ressources. Un noeud virtuel principal contrôle une ressource matérielle et un noeud secondaire contrôle une ressource virtuelle pour chaque groupe de demandeurs. Cette approche nous permet d'optimiser les politiques d'accès pour mieux répondre aux exigences de tous les groupes. Cette composition hiérarchique de noeuds virtuels peut être utilisée pour le partage des ressources de l'exécution, de la communication et du stockage.

La partie gauche de la figure A.4 montre un exemple d'un ordonnancement hiérarchique de deux classes d'applications : "App1" est contrôlée à l'aide d'une politique "round robin" alors que "App2" est contrôlée par une politique basée sur la priorité. Le CPU est partagée entre les deux applications par une politique de partage du temps ("time sharing") mise en oeuvre par le VN principale (LocalVN),VN4CPU. Il alloue un intervalle du temps CPU à chacune des deux applications. Les noeuds virtuels génériques VN4App1 etVN4App2, en contrôlant respectivement les requêtes proviennent des App1 et App2, allouent le temps d'exécution disponibles (un intervalle de temps) à une ou plusieurs tâches selon la politique d'accès.

La partie droite de la figure A.4 montre un scénario d'exécution de ce modèle d'ordonnancement hiérarchique. Le temps CPU est divisé en périodes "P", en utilisant la politique "Time Sharing" du noeud local virtuel. Dans cet exemple, chaque application obtient la moitié du temps du CPU. Les noeuds virtuels génériques choisissent les tâches des applications qui vont s'exécuter pendant l'intervalle d'exécution disponible.

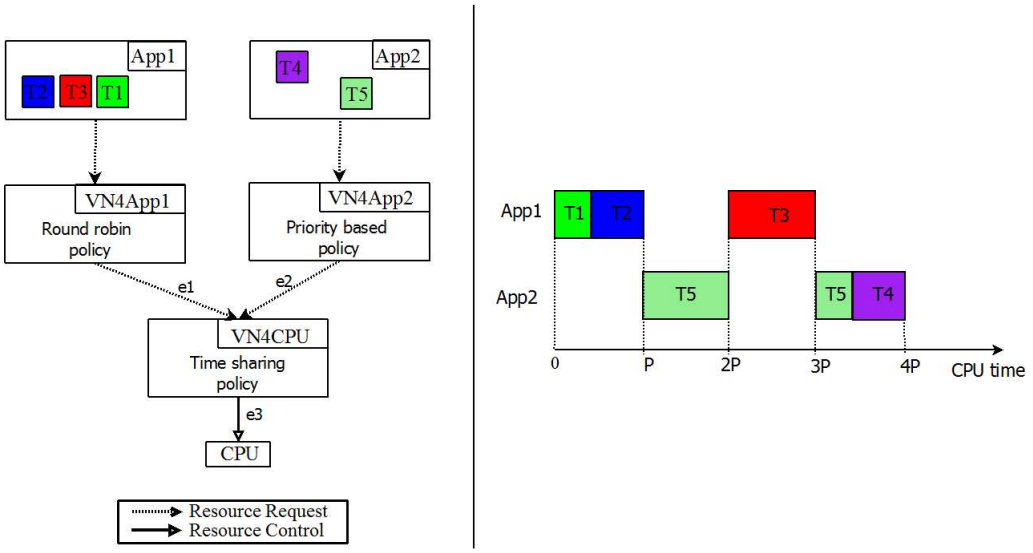

Figure A.4: Un simple exemple d'ordonnacement hiérarchique

Ordonnancement dynamique

Dans les SoC modernes, il est fréquent de trouver des processeurs multi-noyau et/ou des ressources de calcul multiples. Pour optimiser l'utilisation de ces ressources, des techniques d'ordonnancement dynamiques sont généralement utilisées. Toutes les tâches sont en compétition pour s'exécution sur tous les processeurs et un ordonnanceur global contrôle l'ensemble des coeurs disponibles (une ressource virtuelle) et expédie les tâches prêtes sur les noyaux disponibles en appliquant sa politique d'accè. Chacun des noyaux locaux est contrôlé par un noeud local virtuel.

La Figure A.5 montre un exemple d'un noeud virtuel global (GlobalVN) qui est expédie les tâches de l'application (T1 ... T5) sur deux noeuds virtuels local (VN4Core1 et VN4Core2), chacun d'entre eux contrôle un noyau d'exécution (Core1 et Core2 respectivement). Le noeud virtuel global utilise une politique d'accès " le moins utilisé", où il expédie dynamiquement les requêtes entrantes provenant des tâches de l'application vers le noyau le moins utilisé. Le diagramme de temps sur la partie droite de la figure montre l'exécution dynamique des tâches sur les noyaux. T1 est d'abord exécuté sur Core1 mais pendant sa deuxième exécution elle aété exécutée sur Core2, quiétait disponible alors que Core1 a été en train d'exécution T3.

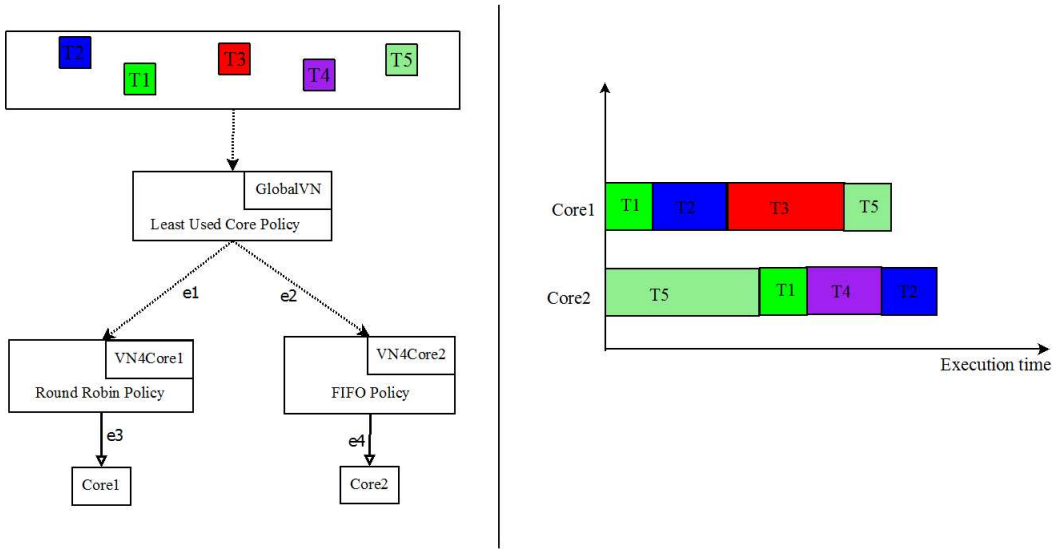

Figure A.5: Un simple exemple d'ordonnacement dynamique

A.4.2 Modèle d'interaction des noeuds d'architecture Interaction communication: "les motifs de communication"

L'exécution de l'application sur l'architecture induit deséchanges de données entre les noeuds de l'architecture. Nous distinguons deux types de communication:

- Les communications explicites: dues à la communication entre les tâches mappés à différents noeuds de calcul. Par conséquent, par l'intermédiaire d'un messageéchangé la mémoire partagée induit deux communications explicites: un pour l'écriture dans la mémoire partagée (par le noeud d'envoi) et l'autre pour la lecture de la mémoire (par le noeud de réception).

- Les communications implicites: représentent des données et des instructions qui sont lues/écrites de la mémoire (partagée ou privée) au cours de l'exécution d'une tâche.

Lorsqu'un noeud d'exécution effectue unn transfert de données (implicite ou explicite) à une mémoire, le protocole de communication est le même pour toutes les applications (ou les tâches de l'application). Le coût de la communication dépend de: 1) la quantité de données à transférer, 2) la mémoire de destination et 3) l'ensemble des noeuds de communication qui porteront ces données à partir du noeud d'exécution à la mémoire. Au cours de ce transfert de données, le noeud d'exécution peut commencer l'exécution d'une autre tâche tout en effectuant le transfert de données demandé par la précédente. Par exemple, dans un contexte de transfert de données DMA, le noeud d'exécution programme le DMA pour le transfert de données et exécute une

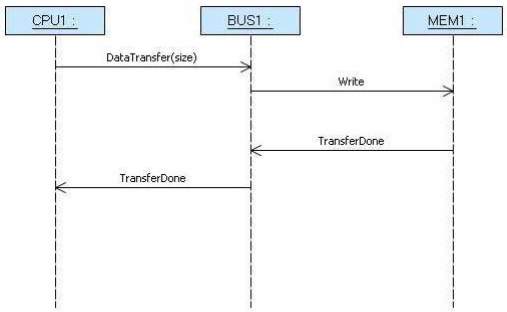

Figure A.6: Un simple exemple de motif de communication

autre tâche. Nous appelons ”le comportement de communication de l'architecture”, la séquence d'opérations de communication nécessaires pour compléter un transfert de données à partir d'un noeud de calcul à un noeud de stockage et impliquant un ou plusieurs noeuds de communication (DMA, Bus ...). Il convient de noter que différents mécanismes peuvent co-exister dans une architecture pour assurer la connexion entre un noeud de calcul et un noeud de stockage. Dans les systèmes embarqués modernes, les normes de communication sont utilisés pour définir l'ensemble des interactions à effectuer pour compléter un transfert de données: PCI Express [65], AMBA bus [15] et Avalon [13] sont des exemples de ces normes de communications. Notre objectif dans Diplodocus n'est pas de modéliser ces normes dans les détails et à faible niveau d'abstraction, mais plutôt de saisir l'influence de l'interaction de communication sur la performance du système dans son ensemble tout en préservant le haut niveau d'abstraction.

Gardant à l'esprit, l'objectif de séparer les concepts de l'exécution et de la communication, nous devons donner les moyens de modéliser efficacement ”le comportement de communication de l'architecture” pour capturer son influence sur les performances du système global. Pour résoudre ce problème, nous introduisons le concept de ”motifs communication”, qui définit un scénario de communication qui assure le transfert de données entre un noeud d'exécution et un noeud de stockage. Il convient de définir tous leséchanges (interactions) entre les noeuds nécessaires (processeurs, bus, des mémoires ...) impliquées dans ce scénario. Un diagramme de séquence UML est traditionnellement utilisé pour représenter ce genre de comportement. Nous avons étendu le diagramme de séquence UML traditionnel pour modéliser les scénarios de communication. Un simple exemple d'un motif de communication est illustré à la figure A.6. Où chacun des lignes verticales (lignes de vie dans la notation UML) représente un noeud d'architecture, et chaque flèche représente unéchange de messages entre deux noeuds d'architecture.

A.4.3 Comparaison des résultats de simulation aux résultats réels d'implémentation

L'exécution du code réel LTE de la couche physique sur la plate-forme Freescale DSP est utilisé comme une référence pour la validation de notre approche. Les résultats sont rapportés dans le tableau A.7. Ce tableau confirme l'exactitude de nos modèles. Il indique pour chaque groupe de tâches les MCPS mesurées (millions de cycles par seconde) consommés pour accomplir son exécution. L'approche présentée dans ce travail de thèse a prouvé sa précision en affichant un taux d'erreur inférieur à 10%. Ainsi, le concepteur peut être confiant des résultats de la modélisation.

	MCPS			Core MCPS		Memory MCPS	
Task Group	Simulated	Measured	Error %	Simulated	Measured	Simulated	Measured
RSP	353,4	361,3	2,2	316,0	326,4	37,4	34,9
PRB	524,8	540,2	2,8	497,6	500,9	27,2	39,3
VRB	51,7	52,6	1,8	44,4	45,0	7,3	7,6
Dcdemux	40,9	41,8	2,2	35,9	25,9	5,0	15,9
CBP	64,6	65,0	0,6	57,5	54,9	7,1	10,1

Figure A.7: Les paramètres de performance des tâches LTE

En plus du nombre total de MCPS, le tableau ci-dessus fournit aussi une comparaison du nombre de MCPS consommées pour accéder à la mémoire (Memory MCPS). Cette métrique représente le nombre total de cycles consommés lorsque les tâches de l'application ont échangé de données ou lorsqu'elles étaient bloquées pendant un acc ès à la cache. La derni ère métrique dans ce tableau est le "Core MCPS" qui représente le nombre total de cycles consommés par les ressources d'exécution pour exécuter les tâches de l'application, mais sans le co ût de communication.

A.5 Conclusions et perspectives

Cette thèse a commencé avec l'affirmation que la conception au niveau système et l'analyse à haut niveau d'abstraction est la solution face à la complexité croissante des SoC modernes. L'abstraction, en masquant certains aspects de la conception, aide le concepteur à ne considérer que ceux qui l'aident à prendre des décisions tôt dans le flot de la conception. L'abstraction couplée avec la modélisation permet de réduire la

complexité et d'analyser rapidement le système. Cependant, le défi reste de garder la précision des résultats de l'analyse.

Cette thèse a présenté nos contributions pour développer une méthodologie de conception au niveau du système pour l'estimation des performances des SoC. Ces contributions sont à la fois au niveau méthodologie et au niveau outils. Il s'agit notamment de la définition des concepts de modélisation avec des méta-modèles UML et leur intégration dans Diplodocus, qui est une méthodologie développée par Télécom ParisTech. Nous avons aussi développé un environnement de simulation SystemC pour analyser les modèles UML. La validation de l'approche proposée est faite grâce à la modélisation d'un cas d'utilisation industriel.

Tout d'abord, de nouveaux composants de modélisation UML ont été définis, les motifs de communication, afin d'assurer la séparation des concepts de l'exécution et de la communication. Ils permettent la modélisation des interactions de communication entre les ressources de l'architecture pour délivrer (lire ou écrire) des données de l'application aux des ressources de stockage. Deuxièmement, les noeuds virtuels ontété définis pourévaluer l'impact des ressources partagées sur les performances globales du système, ils permettent la modélisation des politiques d'accès qui contrôlent les ressources partagées.

Bien que la modélisation UML Diplodocus à haut niveau est utilisé pour modéliser le système, un environnement de simulation SystemC est utilisé pour analyser les performances. Les modèles UML sont transformés en code SystemC qui capturent le comportement concurrentiel et temporel des modèles et de les utiliser pour extraire des mesures de performance telles que les latences et le débit de bout en bout. La simulation proposée est plus rapide que l'exécution en temps réel.

Même si l'approche de modélisation proposée vise la conception du système à un niveauélevé d'abstraction, les contributions présentées démontrent des résultats prometteurs et la comparaison des résultats de la modélisation/simulation avec l'implémentation réelle a confirmé l'efficacité et la précision de notre approche.

Perspectives

Estimation de la consummation de l'énergie. L'environnement de simulation présenté dans cette thèse est principalement dédié à l'extraction des mesures de performance temporelles comme l'utilisation des ressources de l'architecture qui correspond au ratio entre nombre de cycles exécutés et le nombre de cycles d'inactivité. En prenant en considération cette mesure, le concepteur peut estimer la consommation d'énergie du système. En fait, les concepteurs ont quelques bonnes estimations sur la consommation d'énergie d'une ressource (un processeur par exemple) pour exécuter un cycle, ainsi que de l'énergie consommée lorsque le noeud est inactif. Par conséquent, il est possible d'obtenir quelques estimations d'énergie pour les modèle Diplodocus. En outre, comme les noeuds virtuels peuvent facilement mettre en oeuvre

de nouvelles politiques d'accès, on peut imaginer que certaines politiques d'accès qui visent la réduction de la consommation seront modélisées et utilisés dans les modèles DIPLDOOCUS.

Transformation de modèles et d'échange avec d'autres profils standard UML. Diplodocus est un profil UML qui étend UML pour la modélisation des SoC à un niveauélevé d'abstraction. Il est livré avec une trousse à outils (TTooL[63]) qui permet la génération des modèles UML de code de simulation (C + + et SystemC code) et de spécifications formelles (LOTOS et UPPAAL). Toutefois, Diplodocus n'est pas le seul profil UML qui cible la modélisation des SoC, et d'autres profils, et en particulier le profil OMG Marte, ont un objectif similaire. Certaines initiatives, telles que openembedd[62], tentent de définir un flot de conception complet à partir de modèles haut niveau Marte pour arriver à l'impléméntation. Ainsi, un éventuel renforcement de l'environnement Diplodocus UML sera d'utiliser des outils de transformations de modèles (comme Kermeta[60]) afin de transformer les modèles diplodocus en modèles UML Marte (ou d'autres profils UML) permettant ainsi l'utilisation de leur outils dédiés. Les transformations peuventêtre fait dans les deux sens, ainsi les autres profils UML peuvent ré-utiliser l'environnement de Diplodocus.

L'intégration dans un flot de conception. La transformation du modèle permet la transformation, en s'appuyant sur une hypothèse spécifique (abstraction de certains aspects du système par exemple) des modèles vers d'autres modèles qui mettent l'accent sur d'autres aspects de la conception. Le processus de transformation peutêtre bien aussi à partir de modèles au niveau système vers un niveau inférieur de l'abstraction. Dans le contexte Diplodocus, cette transformation consiste tout d'abord à enrichir les modèles diplodocus avec les détails de niveau inférieur, elle pourraitêtre faite sur plusieursétapes et sans forcementêtre complètement automatique. Le concepteur peut utiliser des bibliothèques prédéfinies (RTL ou libraires TLM). Ainsi, permettant l'intégration de Diplodocus dans un flot de conception SoC.