# Modeling SPIRIT IP-XACT with UML MARTE

Charles André[†], Frédéric Mallet[†*], Aamir Mehmood Khan[*], Robert de Simone[*]
AOSTE Project, I3S/INRIA
INRIA Sophia-Antipolis Méditerranée.[*]
Université de Nice-Sophia Antipolis, France.[†]
{Charles.Andre, Frederic.Mallet, Aamir.Mehmood, Robert.De_Simone}@sophia.inria.fr

## Abstract

*Large System-on-Chips (*SOC*) are now built by assembly of existing components, modeled at different representation levels(*TLM, RTL ... *). The* IP-XACT *standard was recently developed to help normalize interfaces of IP components, and ease their composition. Currently it does not fully face timing representation issues. The equally recent* MARTE UML *profile focuses explicitly on the rich expression of time (physical or logical), clocks, and timing features.*

*We attempt here to embed* SPIRIT IP-XACT *representations as* MARTE *models. This leaves open the possibility to extend the formalism with timing aspects, while it provides graphical editing functionalities and a way to experiment on extensions.*

## 1. Introduction

Reuse and integration of heterogeneous Intellectual Property (IP) from multiple vendors is a major issue of System-on-Chip (SoC) design. Existing tools attempt to validate assembled designs by global co-simulation at the implementation level. This fails more and more due to the increasing complexity and size of actual SoCs. Thus, there is a clear demand for a multi-level description of SoC, with verification, analysis and optimization possibly conducted at the various modeling levels. In particular, analysis of general platform partitioning based on coarse (extra functional) abstraction of IP components is highly looked after. This requires interoperability of IP components described at the corresponding stages, and the use of traceability to switch between different abstraction layers. Although this is partially promoted by emerging standards, it is still insufficiently supported by current methodologies. Such standards include OCSI SystemC, SPIRIT Consortium IP-XACT, Silicon Integration Initiative OpenAccess API, and also recent Unified Modeling Language [1] (UML)-based standards like the UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE [2]) that specifically targets real-time and embedded systems.

System Modeling requires representation of both structural/architectural/platform aspects at different levels of abstraction and behavioral/functional aspects possibly considering timing viewpoints such as untimed/asynchronous/causal, logical synchronous/cycle-accurate or physical/timing models. Semantics provides behavioral meaning to full systems from the combination of the behavior of its components. For *system structure* representation, UML uses component and composite structure diagrams, while SYSML [3] uses block diagrams. Tools like Esterel Studio, and virtual platforms like CoWare, Synopsys CoreAssembler and ARM RealView, introduce their own architecture diagrams. IP-XACT provides some ADL (Architecture Description Language) features for externally visible common interfaces and recognized state encodings, together with administrative information (ownership, tool chains, versioning ...). It uses its own XML syntax, for specification of IP meta-data and tool interfaces.

For *component behavior* representation, SystemC provides programming libraries to represent IP component behavior at different abstraction levels, from Transaction Level Modeling (TLM) to RTL but it requires additional support for architecture modeling. IP-XACT relies on SystemC, VHDL, and Verilog HDL to specify the actual behavior of components. UML behavioral diagrams provide a support for describing "untimed" algorithms, while MARTE also provides support for logically or physically timed behaviors. Generally speaking, the behavioral models in UML are more abstract than commonly used SystemC levels, such as TLM/PV, TLM/CC, or RTL. In fact, state, sequence and activity diagrams can be thought as closest to TLM/CP (Communicating Processes) models, which are still seldom used in SystemC methods. Then MARTE could be seen as introducing the relevant *timed* version at this level (like PVT does for PV), through logical time and abstract user-defined clock threads.

One good feature of UML is its extendability, with the

profiling mechanism. On the other hand the set of considered features should be as much as possible standardized and agreed upon (if to allow interoperability), which is the main goal of the SPIRIT consortium. Therefore a UML profiling approach for IP-XACT based on relevant metamodels would allow easy creation and extension of model editors for IP-XACT as the standard evolves, as well as experimentations with prototypal versions. Models can then be translated into IP-XACT syntax for tool integration. We chose to do this by specializing the MARTE profile, which already provides a number of modeling features for extra-functional aspects (such as logical timing elements).

So our work goes in two steps: first we create an ad-hoc UML profile for IP-XACT, then we study how a proper subset of MARTE can be used to add modeling expressivity for timed behaviors. The resulting profile should be uploadable into existing graphical tools (*e.g.*, Eclipse UML, MagicDraw by NoMagic, Rational Software Architect by IBM, Artisan).

This paper presents the details of our metamodeling (Section 2) and profiling (Section 4) of IP-XACT in UML reusing MARTE wherever possible. In Section 3, we give an overview of the aspects reused from MARTE: the subprofiles for Generic Resource Modeling (GRM) and Hardware Resource Modeling (HRM) for the structural aspects, the TIME subprofile to provide abstract behavioral aspects of IPs and the ALLOCATION subprofile for their mapping onto the virtual platform itself. As a running example, we use the IP-XACT specification of the Leon2 processor, releases as part of the IP-XACT 1.4 RC1 distribution package (*http://www.spiritconsortium.org*).

**Related work**: UML profile dedicated to SOC modeling have been proposed, such as UML4SOC [4] and UML4SYSTEMC [5], but they do not consider timing aspects. UML/SYSML have also been proposed as representation formalisms to support analysis and produce SystemC outputs [6]. The yearly DAC satellite workshop UML for SOC design deals with these issues (*http://www.c-lab.de/uml-soc*).

## 2. SPIRIT IP-XACT metamodel

IP-XACT defines seven kinds of models: Component, Bus definition, Abstraction definition, Design, Abstractor, Generator chain, Configuration. We consider only the first four.

### 2.1. Component

Component is the basic model element in SPIRIT. Every IP is described as a component without distinction of type, whether they represent computation cores (processors, co-processors, DSPs), peripheral devices (DMA controllers, timers, UART), storages (memories and caches), or interconnects (simple buses, multi-layer buses, cross bars, networks on chip).

Figure 1 shows the main features of components as considered in IP-XACT, their interface and their memory hierarchy. A component identifier is unique, it specifies its name, the containing library, the vendor and the version number. A textual description may comment its intended role. A component also contains a precise description of the memory hierarchy: addressSpaces and memory mappings.

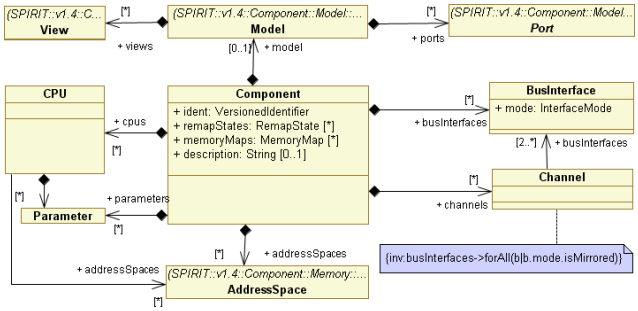


**Figure 1. IP-XACT Component metamodel.**

The hardware model of a component describes its physical ports, the different views available (RTL, TLM, documentation) and a set of parameters.

The *view* mechanism is a provision for having several models of the same component at different levels of abstraction. The *ports* can be wire ports (for RTL) or transactional (for TLM) ports. The formers only allows purely binary values or vectors of binary values.

BusInterface allows for grouping together ports that collaborate to a single protocol. Components communicate with each other through their bus interfaces tailored for a specific bus. The bus interfaces map the physical ports of the component to the logical ports of the abstraction definition. They also identify the interface mode (master, mirrored master, slave, mirrored slave). The mirroring mechanism guarantees that an output port of a given type is connected to an input port of a compatible type.

Channels describe multi-point connections between components when the interfaces are not directly compatible and require some adaptation.

### 2.2. Abstraction and Bus Definition

A BusDefinition describes the high-level attributes of the interfaces connected to a bus. For instance, it defines a maximum number of masters and slaves, and whether a master interface can be directly connected to a slave interface or should rather go through mirrored master/slave interfaces. A bus definition can extend another bus definition, thus allowing the definition of compatibility rules with legacy

buses. For instance the AHB (Advanced High-performance Bus) definition extends the AHBlite definition. When extending another bus definition, some constraints must be enforced. For instance, an extending bus definition must not declare more masters and slaves than the extended one.

An AbstractionDefinition gives lower-level attributes for a given BusDefinition. There can be several abstraction definitions for the same bus definition, like AHB_rtl and AHB_tlm. In the same way, an abstraction definition can extend another one with also some compatibility constraints to enforce. The abstraction definition defines the ports, which have to be defined by the bus interfaces, and constrains them (type, direction . . . ).

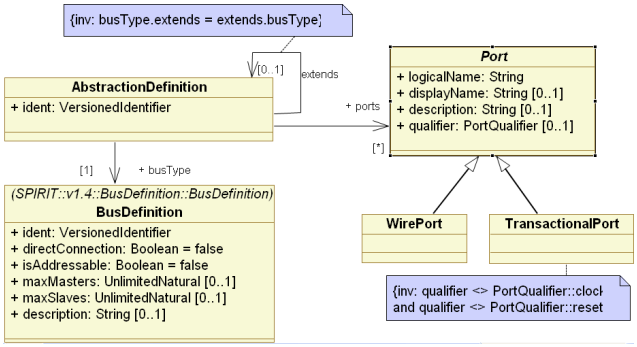


**Figure 2. IP-XACT BusDefinition metamodel.**

### 2.3. Design

A Design represents a system or a sub-system, it defines a set of component instances and their interconnections. Ad-hoc connections connect two ports directly, wire ports but also transactional ports, without using a bus interface. Interconnections are point-to-point connections of bus interfaces from two siblings components, whereas hierarchical connections connect components at different hierarchical levels (*e.g.*, a parent to one of its children).

## 3. The UML MARTE Profile

### 3.1. General overview

The new OMG UML profile for MARTE supersedes and extends the former UML profile for Schedulability, Performance and Time (SPT[7]). MARTE adresses new requirements: specification of both software and hardware model aspects; separated abstract models of applications and execution platforms; modeling of allocation of the former onto the latter; modeling of large domains of Time and Non-Functional properties.

MARTE consists of three packages. The first package defines the *foundational concepts* used in the real-time and

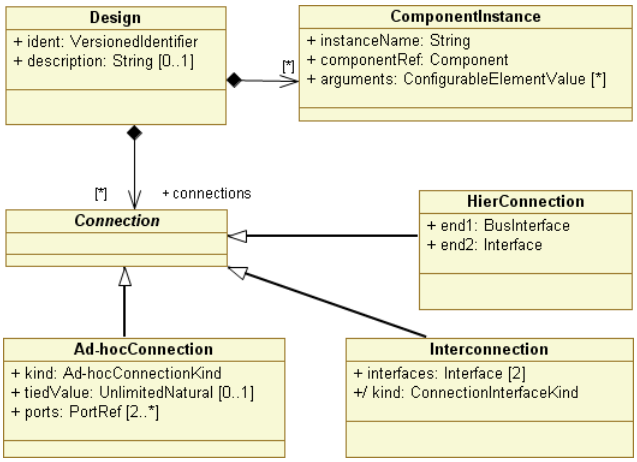


**Figure 3. IP-XACT Design metamodel.**

embedded domain. These foundational concepts are refined in two other packages to respectively support modeling and analysis concerns of real-time embedded systems.

The second package addresses *model-based design*. It provides high-level model constructs to depict real-time embedded features of applications, but also detailed software and hardware execution platforms.

The third package addresses *model-based analysis*. It provides a generic basis for quantitative analysis sub-domains.

### 3.2. Highlights of MARTE parts relevant to IP-XACT profiling

Our profile for SPIRIT reuses several model elements from the first and second packages. This subsection briefly describes these borrowings.

The central concept of *resource* is introduced in the Generic Resource Modeling (GRM) package of MARTE. A *resource* represents a physically or logically persistent entity that offers one or more *services*. A Resource is a classifier endowed with behavior (a BehavioredClassifier in UML terminology), while a ResourceService is a behavior. Resource and ResourceService are types of their respective *instance* models.

Several kinds of resources are proposed in MARTE like ComputingResource, StorageResource, CommunicationResource, TimingResource. Two special kinds of communication resource are defined: CommunicationMedia and CommunicationEndPoint. The communication endpoint acts as a terminal for connecting to a communication media; typical associated services are data sending and receiving.

For structural modeling, MARTE enriches the concepts defined in the UML composite structures. StructuredComponent defines a self-contained entity of a system, which

may encapsulate structured data and behavior. An *interaction port* is an explicit interaction point through which components may be connected. There exist two kinds of ports: MessagePort and FlowPort. MessagePort supports a request/reply communication paradigm, whereas FlowPort enables flow-oriented communication paradigm between components. Thus, MARTE supports both message and flow oriented communication schemas.

The MARTE *Allocation* associates functional application elements with the available resources (the execution platform). This comprises both spatial distribution and temporal scheduling aspects, in order to map various algorithmic operations onto available computing and communication resources and services.

Both Resource and Allocation refer to *Time*. In MARTE, Time can be chronometric (refering to physical time) or logical. The time model is defined in the Time package of MARTE and it supports concepts as timed events, timed behaviors, timed values. Details about the MARTE Time and Allocation models are presented in a previous paper[8].

The Detailed Resource Modeling (DRM) package of MARTE specializes these concepts. It consists of two subpackages: Software Resource Modeling (SRM) and Hardware Resource Modeling (HRM). Only the latter is used and described in the following.

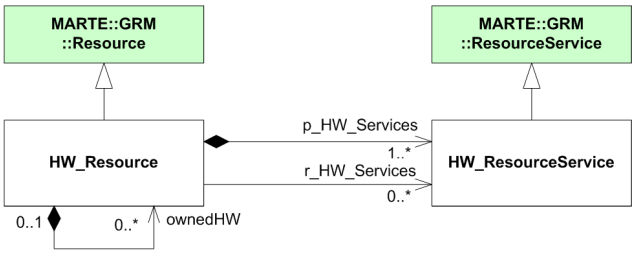


**Figure 4. MARTE hardware resource metamodel.**

As shown in Figure 4, HW_Resource (HW_ResourceService, resp.) specializes Resource (ResourceService, resp.) defined in the GRM package. A hardware resource *provides* at least one resource service (hence the prefix 'p_' in the role name) and may *require* ('r_' prefix) some services from other resources. Note that a HW_Resource can be hierarchical. The HRM package is further decomposed in two sub-packages: HW_Logical and HW_Physical. The former provides a functional classification of hardware entities, the latter defines a set of active processing resources used in execution platform modeling and close to several SPIRIT concepts.

HW_Resource are specialized in the same way as the generic resources of the GRM package. Figure 5 is an excerpt of the GRM class hierarchy.

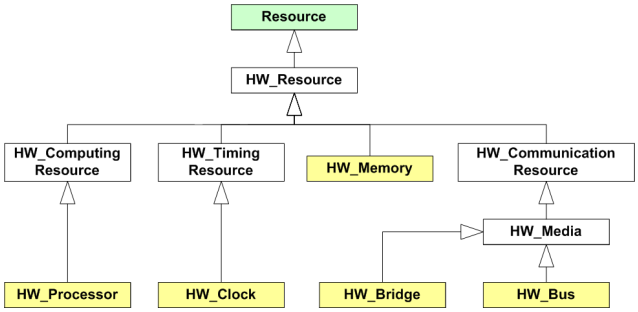


**Figure 5. MARTE HRM metamodel.**

## 4. Our UML profile for SPIRIT IP-XACT

Our UML profile for IP-XACT is defined by mapping the IP-XACT concepts identified in the metamodel (see Section 2) to UML metaclasses. Following B. Selic [9], we have tried to define *stereotypes* with parsimony and to create new ones when no equivalent concepts were available in UML or in MARTE. In addition to the stereotypes we have also defined a *model library* to provide a set of data types equivalent to IP-XACT primitive types.

In Section 4.1 we describe our mapping of the IP-XACT concepts to the UML world. In Section 4.2 we study how a proper subset of MARTE can be used to add modeling capability for timed behavior.

### 4.1 Structural description

As we have seen in Section 2, there are two phases in the definition of an IP-XACT model. During the first phase, the components and the buses are defined. The second phase integrates component instances into a design and specifies the interconnections. Contrary to other approaches that use component diagrams for both phases we recommend the use of class/component diagrams for the first phase and the use of composite structure diagrams for the integration phase. Our approach eases the reuse of already defined components and allows for having several parts (component instances) of the same classifier without corrupting the classifier itself. This also results in very simple composite structure diagrams while maintaining all the detailed information on the classes themselves. Figure 6 shows a partial composite structure diagram of the Leon2 design. The Leon2 processor is connected to a memory through a AHB-bus component. The connector hides the bus definition and interfaces.

Figure 7 represents the first phase with a detailed view of these components and buses along with an expanded view of the bus interfaces connected to the ports.

In the following we describe our mapping of IP-XACT concepts to UML keywords. These mapping rules, summarized in Table 1, allowed the automatic generation of IP-

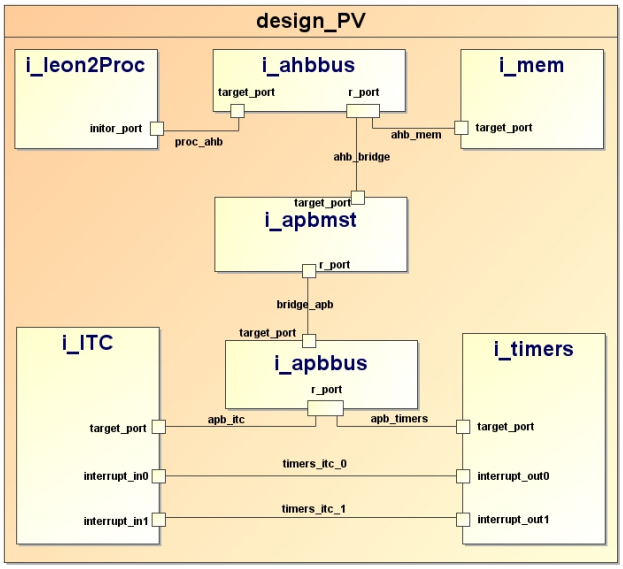


**Figure 6. The Leon2 design in UML.**

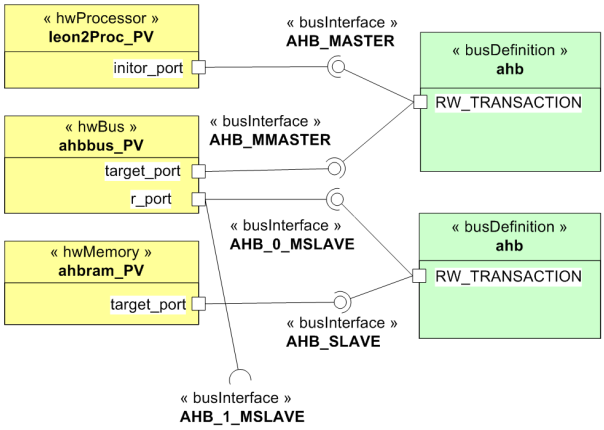


**Figure 7. Component and bus definitions in UML.**

XACT from the UML model of the Leon2 processor. The model transformation was performed using an XSLT (eXtensible Stylesheet Language Transformation) description.

| SPIRIT IP-XACT | UML |
|---|---|
| Component | «hwResource» |
| VLNV | VersionedIdentifier |
| CPU | «hwProcessor» |
| Port | «wirePort», «transactionalPort» |
| Bridge | «hwBridge» |
| Clock Driver | «hwClock» |
| Timing Constraints | Model Library |
| Bus Definition | «busDefinition» |
| Abstraction Definition | «abstractionDefinition» |
| Bus Interface | «busInterface» |
| Bus Interface Mode | InterfaceModeKind |
| Design | Composite Structure |
| Component Instance | Part |
| Connection | Connector |

**Table 1. Mapping IP-XACT concepts to UML**

### 4.1.1 Component

To distinguish hardware components (which must be transformed into IP-XACT components) from software components, we use the MARTE HwResource stererotype and its specializations. IP-XACT considers all IPs equally without distinction of type, however, existing MARTE models using specialized stereotypes (like HwProcessor) can also be transformed into IP-XACT components.

IP-XACT objects are uniquely identified by their identifier VLNV (Version, Library, Name, Vendor). Our model library introduces a custom data type called VersionedIdentifier to represent these identifiers.

Components use ports to interact with the communication media. UML ports are extended by the MARTE HwEndPoint stereotype. We further specialize this stereotype with two new stereotypes WirePort and TransactionalPort to add IP-XACT specific features, like the direction and width.

In components, bus interfaces map physical ports of the component to logical ports of an abstraction definition. Such a mechanism does not exist in MARTE, so we extend the UML Interface metaclass with a stereotype called BusInterface to represent this concept. Master ports usually initiate the transactions, thus they *require* services from other components. In our profile, interfaces associated with the master ports are represented as *sockets*, a usage dependency in UML. Figure 7 shows an example with the initor_port of the leon2Proc_PV component. Interfaces associated with slave ports are represented as *balls* (a realization dependency in UML), since slave ports *provide* services to other components. In Figure 7, the target_port of the ahbram_PV component is such an example. The stereotype BusInterface refers to a BusDefinition and an AbstractionDefinition with its properties busType and abstractionType. It also has an enumeration literal to set its *mode* (master, slave or system) and a Boolean property to identify whether it is a *direct* or a *mirrored* interface. The enumeration InterfaceModeKind is defined in our model library.

We reuse the MARTE stereotype HwBridge to represent IP-XACT bridges. Finally, the port clock drivers are modeled with the MARTE stereotype HwClock.

#### 4.1.2 Bus and Abstraction Definition

There is no equivalent neither in UML nor in MARTE for the IP-XACT important concepts of BusDefinition and AbstractionDefinition. Our profile introduces two stereotypes for this purpose. Figure 7 illustrates that with the example of the BusDefinition ahb.

When a BusDefinition (or an AbstractionDefinition) extends another one, this extension is modeled by the MARTE Refinement mechanism. MARTE refinement is very similar to UML refinement but makes explicit the constraints implied by the refinement. We can use these constraints to specify the IP-XACT compatibility constraints, for instance.

#### 4.1.3 Design

The design component represents a system or a subsystem. It contains component instances and interconnects them. No additional stereotypes are required for this level. We recommend the use of structured classifiers (UML hierarchical classifiers) with a composite structure diagram. The component instances become the *parts* of the structured classifier, both the interconnections and the ad-hoc connections are modeled as UML *connectors*. The hierarchical interconnections are represented with a delegate dependency between the port of the parent and one port of one the children.

### 4.2 Behavioral description

Neither IP-XACT nor SYSTEMC provide any support to model at an abstraction level higher than SYSTEMC PV (Programmer View). However, combining UML state machines and activities with the MARTE time model gives the opportunity to make models at a *timed* CP (Communicating Processes) level. Figure 8 shows a timed activity that describes a timer at a much higher level than the equivalent PV description, not to mention the RTL description. Using such activities or SYNCCHARTS [10] (the synchronous version of UML state machines) enables formal validation of composite models at the system level and gives a golden timed model from which lower level descriptions can be derived or at least compared.

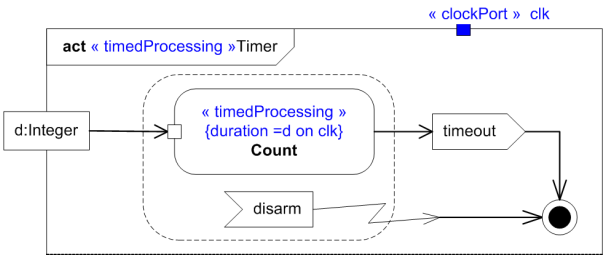


**Figure 8. Timed Activity.**

## 5 Conclusions

We have demonstrated how the UML MARTE could be used to represent the various IP-XACT features. We focused on four top-level elements, but plan to extend it to the full IP-XACT standard. The main interest of this work in our view is that it allows the of MARTE for possible extensions of IP-XACT with timing representations.

## References


[1] Object Management Group. *Unified Modeling Language Superstructure, V2.1.2*. OMG Available Specification, November 2007. OMG document number: ptc/2007-11-02.

[2] OMG. *A UML profile for MARTE*. OMG Adopted Specification, August 2007. OMG document number: ptc/07-08-04.

[3] OMG. *Systems Modeling Language (SysML) v1.0*. OMG Available Specification, September 2007. OMG document number: formal/2007-09-01.

[4] OMG. UML *profile for System on a Chip v1.0.1*. OMG Available Specification, August 2006. OMG document number: formal/06-08-01.

[5] E. Riccobene, P. Scandurra, A. Rosti, and S. Bocchio. *A SoC Design Methodology Involving a UML 2.0 Profile for SystemC*. In Proc. of the Conf. on Design, Automation and Test in Europe (DATE), March 2005, Munich, Germany. IEEE, pages 705–709.

[6] A. Viehl, T. Schönwald, O. Bringmann, and W. Rosenstiel. *Formal performance analysis and simulation of UML/SysML models for ESL design*. In Proc. of the Conf. on Design, Automation and Test in Europe (DATE), March 2006, Germany. IEEE, pages 242–247.

[7] OMG. *UML Profile for Schedulability, Performance, and Time Specification v1.1*. OMG, January 2005. OMG document number: formal/05-01-02.

[8] C. André, F. Mallet, and R. de Simone. *Modeling time(s)*. In MoDELS 2007, LNCS 4735, pages 559–573, Springer Verlag, October 2007.

[9] B. Selic. *A systematic approach to domain-specific language design using UML*. In 10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'07), pages 2–9. IEEE, 2007.

[10] C. André. *Computing SyncCharts Reactions*. Electronic Notes in Theoretical Computer Sciences, 88(9):3–19, 2004.