Abstract Modeling of embedded Systems Hardware

Christian Hausner and Frank Slomka
Institute of Embedded Systems / Real-Time Systems, Ulm University, 89069 Ulm, Germany

Keywords: Embedded Systems Design, Cyber-physical Systems, Electronic System Level (ESL), Platform-based Design.

Abstract: Designing cyber-physical systems is a challenge originating from the multidisciplinary and mixed-signal requirements. In order to handle this challenge, many design languages have been developed, but none covers the platform-based design and system view well. In this paper we extend our methodology by considering the aspects of the platform. A new abstraction layer, the domain layer is explained. This new layer allows the description of embedded hardware as well as system on chips in a way which can be easily understood by application or software engineers as well as hardware engineers. It closes the gap between hardware structure diagrams as given by hardware designers on system level and class diagrams as used by software engineers. Together with a new diagram type to describe hardware structures on system level the approach opens a door to describe the binding or deployment of software to operating system services and hardware in a formal way considering aspects of memory management and the structure of address spaces. Aspects not covered by common system description languages.

1 INTRODUCTION

The design of cyber-physical systems (Lee, 2008) – consisting of software as well as digital and analog hardware – is still a great challenge that is caused by the increasing complexity and the multidisciplinary requirements which are typical for mixed-signal applications. Hence, it should be taken into account that reuse of the hardware and software platform for different products is another strong requirement.

Model-based design with the UML (Object Management Group (OMG), 2013) is a common way to model software systems. During the last years it has been adapted to embedded systems. For this, UML has been extended by the profiles MARTE (Object Management Group (OMG), 2011) and SysML (Object Management Group (OMG), 2010). However, for both the system design is not considered well. Certainly it is possible to define hardware architectures as well as the binding of computational elements to processing elements, but the language does among others not support hierarchical bindings. Therefore the system view in the Y-diagram of platform-based design (Carloni et al., 2005) is poorly implemented in MARTE and UML.

Established hardware description languages like VHDL and Verilog ((Marwedel, 2011), chapter 2) or classical wiring diagrams are very detailed but not best suited for designing entire systems. They concentrate on wiring and interconnection but not on functionality. On the other hand, block diagrams as used in SysML will not graphically represent function details that are important for the platform and system design process. The software platform or operating system issues are not covered by one of these languages and methodologies.

To handle the problems discussed above, a new approach for system modeling was introduced in (Slomka et al., 2011). The approach extends the object-oriented philosophy of designing software systems to multidisciplinary, multi-technology hardware/software systems. In contrast to other modeling approaches, systems are modeled with their influencing physical properties. A system view (see Sect. 3.3) unites application design and platform design. The main focus of the previous work was to explain the design of the application and its refinement during design. Introductory information about the application view can be found in this work in Sect. 3.1 and in Figure 1, detailed information is provided in (Slomka et al., 2011). In contrast to (Slomka et al., 2011) this work shows how the architecture of the platform can be described in an intuitive way. It includes two different views of the platform and also gives the designer the opportunity to refine a system during design. These views are independent from UML, MARTE, or SysML but can be easily adapted to them.

Hausner C. and Slomka F..
Abstract Modeling of embedded Systems Hardware.
DOI: 10.5220/0004594802510258
In *Proceedings of the 3rd International Conference on Simulation and Modeling Methodologies, Technologies and Applications* (SIMULTECH-2013), pages 251-258
ISBN: 978-989-8565-69-3

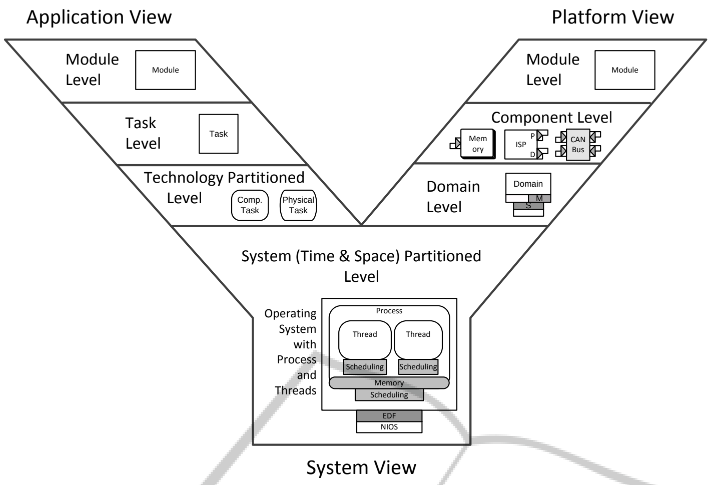

Figure 1: Abstraction levels in the design flow.

The remainder of the paper is as follows: The design flow is presented in Sect. 2 followed by the different abstraction layers in Sect. 3. In Sect. 4, the platform model is introduced. After this, an exhaustive case study is given (Sect. 5), followed by a conclusion.

2 DESIGN FLOW

The proposed design methodology is described in (Slomka et al., 2011), the following will concentrate on the platform aspects only.

In the first step a requirement specification is conducted ①. After finding the requirements, the system's functionality has to be worked out. For this, a system analysis based on the requirements is performed. The result is a small system specification which considers all aspects of the whole system. Based on the resulting system specification ②, the system design and component specification starts ③. The subsystem has to be partitioned into different

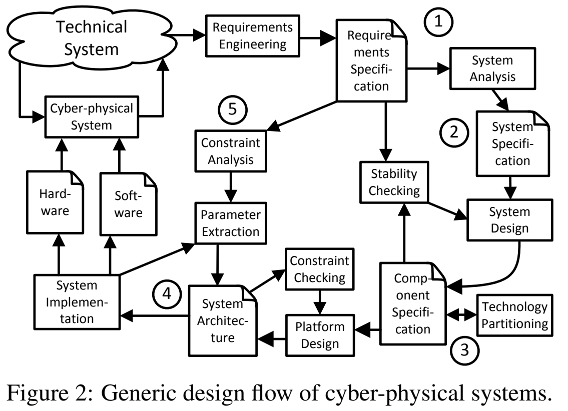

Figure 2: Generic design flow of cyber-physical systems.

technology domains.

The step after the technology partitioning is the *platform design*, where hardware elements like components, tasks, or controllers and software like operating systems and frameworks are mapped to allocated computational resources offered as services to the application. The platform design contains several design steps like the allocation of processing elements, communication elements or memory elements, the generation of platform modules and depending platform domains for resources and services that could be connected to the application by service access points. The result is the system architecture ④.

In this paper the design of the platform is described. It is discussed how a platform architecture can be described. Therefore we will not discuss the other parts of the development process in the rest of the paper.

3 SYSTEM ABSTRACTIONS

Since the introduction of the system-on-a-chip paradigm, the conventional model consisting of classic abstraction layers for hardware and software does not apply anymore. In this model six different layers of hardware abstractions are known. The classification is given in (Gajski, 1992) and is divided into system level, behavioral level, register transfer level, gate level, transistor level, and layout level. In the design flow we find these parts downwards from the platform design. Software development is here also separated into the different abstraction layers architecture level, component level, algorithmic level and machine code level. This classification is adapted from (Sommerville, 2001).

This common approach ends below the system level, but systems have to be consistently explained in the whole. Therefore a new paradigm has been developed to support system synthesis. This *platform based design* is shown in Figure 1 by a Y-structure. It is distinguished between the different views Application, Platform and System, as further discussed in (Sangiovanni-Vincentelli and Martin, 2001). This means the development of the application's functionality is separated from the development of the hardware and software platform. Hence, this approach takes into account that in many projects, platforms are used for different products. The system axis describes the mapping or binding of application functions to the hardware components. Although the application and the platform view are covered by UML and AUTOSAR (AUTOSAR development cooperation, 2013), the system view is not supported well by these techniques.

3.1 Application View

The application view as described in (Slomka et al., 2011) is supported in three abstraction levels (see also Figure 1) We distinguish between the module and the task level that is subdivided into the task level itself and the technology partitioned level. The behavior of specific system parts is encapsulated by a task and tasks may communicate with each other. Note that in this context a task does not mean an operating system task or process. The tasks are mapped to the platform in a later stage described in detail in the step "system view".

3.2 Platform View

The platform view considers hardware as well as basic software (operating systems and frameworks). It is an enhancement of the classic hardware and software abstraction layers as described in Sect. 3 but it focuses on the description of systems. In this paper the platform view is limited to the digital domain. The process starts with the module level to describe the approximated platform consisting of modules like devices, cards or chips and their interconnections. The refinement step component level defines the system platform in more detail. Components are the basic functional blocks of the platform hardware. The component level is a new abstraction level considering the entire system's hardware. It reduces hardware platforms to addressable objects and their interconnections but encapsulates from wiring and physical dimensions.

The resulting architecture from the component level can then be refined at the domain level that is especially new in our methodology. Groups of components are transformed to platform domains describing the resources and services that are offered by these components. Domains describe the scheduling behavior, memory layout and usable resources (e.g. communication buses, I/O) of the platform.

Note that this is a new abstraction level to better support the mapping of applications. For the applications it is not important how the components are connected. It is important which memory can be used, what resources are available and when it gets resource time. Therefore this level supports an abstraction for operating systems and frameworks.

3.3 System View

The system view unites the application and platform view. At this level application elements (like tasks, ports and links) are bound to the elements of the platform domains that are called service access points.

Each computational task is assigned to a platform domain describing its scheduling and memory layout. Each port of the computational task is mapped to a communication service access point of that domain. The information what ports are involved in the communication is given by the related computational link and can be distributed through the system. Based on the mapping it is possible to classify if a task is implemented as a thread sharing memory with other threads or as a process having its own memory or if an application task is implemented as a function call.

In MARTE, allocation means the mapping of application to a platform (Boulet et al., 2007). Unfortunately software engineers are talking about deployment if they want to map software functions to components. In system synthesis, allocation means to choose a component, while binding is the term to describe mappings of tasks to allocated components. In this paper we use the system synthesis terms instead, as the aim of the methodology is to synthesize embedded systems.

First we introduce domains. Domains are an abstract model of the platform and are especially useful for the binding of application elements to platforms. We introduce two types of domains that describe resources, services with access points and the memory layout. Domains group platform elements in a logical way.

However, domains and their contained elements are not the concept needed by synthesis, because they hide architectural aspects. Therefore domains allow an abstract formulation of architectures. Using the concept of hierarchical composition of domains, it

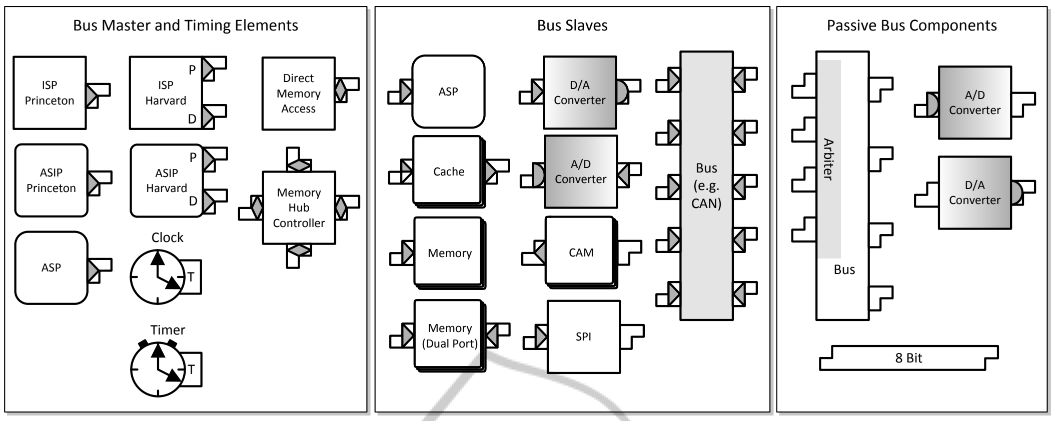

Figure 3: Symbols of components.

is possible to describe complex architectures easily. Such a model is needed because a complex binding relationship as proposed by MARTE is very hard to use if an automatic design space exploration has to be designed and implemented because there exist a lot of dependency rules between the different views. A comparable approach is presented in (Liehr et al., 2008).

In UML, MARTE, or SysML it is only possible to describe the application or platform view. Binding and mapping is only supported in a graph based approach like in (Teich et al., 1997). Memory is not considered in it. The whole system view is also missing in all cases. This gap is closed in this paper by the newly introduced system view and the concept of platform domains.

4 PLATFORM LANGUAGE

The platform view enables hierarchical design as well as refinement processes and is divided into two different views of a platform. All views describe the same platform but with different intentions.

4.1 Module and Component Level

The platform view (see Figure 1) starts with modules to support a hierarchical design methodology. Modules are for example devices, cards or chips. The modules can consist of several modules or components. Components are the basic functional hardware elements. Therefore modules and components are elements of a hardware view of addressable objects. We distinguish between different types of components (see Figure 3).

4.1.1 Bus Control Interfaces

Components contain bus control interfaces that are connection points with a technology type and a bus control interface contract. The bus control interface contract is something like the interface type in object oriented languages. At this level of detail we distinguish between the following bus control interfaces: Active, Passive, Analog and Timing. Only bus control interfaces with compatible types could be connected to each other. The interfaces and their symbols are shown in Figure 3.

The bus control interface could be an active bus control interface what implies that the component contains the controller needed for connecting to the appropriate communication system like a bus. Active bus control interfaces are symbolized by a triangle inside the bus control interface symbol. Passive bus control interfaces will not contain any controller and are therefore only seen as a wiring connection point. The interface symbol contains no triangle. Analog interfaces are specified to connect analog building blocks (see section 6). They are symbolized by a semicircle inside the interface symbol. Finally Timing interfaces connect timing elements with time users that could be almost all other components. The interface symbol differs from the other symbols and it contains a “T” indicating the timing interface.

4.1.2 Components

Model elements that can act as a bus master are called bus master components. They are equipped with active interfaces with a triangle pointing away from the component. This indicates the component can command the bus. Another possibility is a diamond symbol indicating the component can also be commanded by other components. Examples of bus masters are processing elements like instruction set processors

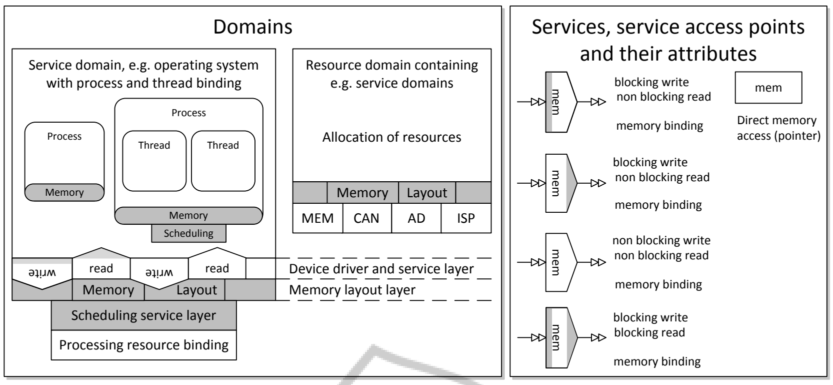

Figure 4: Symbols of domain level elements.

(ISP) or application specific instruction set processors (ASIP), direct memory access controllers and memory controller hubs also known as bus bridges. These components and their symbols can be found in Figure 3 in the area of Bus Master Components.

Bus slave components are components that will be commanded by bus masters. They are equipped with active interfaces containing a triangle pointing to the component. Bus slave components are for example memories and caches including their controllers (symbolized by a stack of memory cells) or buses with active controllers as a CAN bus for example. The memory addressing scheme – virtual or physical - could be modeled as an attribute of a component like a cache or instruction set processor. This implies that a memory management unit or a memory protection unit is not modeled as separate component but as this attribute. The interface of such a virtual addressing is indicated by a double bar triangle. These components can be found in Figure 3 in the area of Bus Slave Components.

Passive bus components will not contain an active controller. They are connected by passive bus control interfaces. For example buses without controllers are passive bus components. Another example is an AD/DA converter that should be connected to a passive bus control interface (only wires).

Timing elements are components that model timers and clocks. They can be connected to all other elements and will be symbolized by a clock with a timing interface.

4.2 Domain Level

The modules and components can then be refined by domains. In our approach we distinguish between two types of domains: Resource domains describing the resources of the platform and service domains describing the offered services with their access points that are mapped to application elements in the system view. Symbols for domains and their containing elements are shown in Figure 4). In the domain level no direct connections between model elements with arrows or lines are used, elements are connected by their names and position.

The idea is to define an abstraction layer which covers all aspects of a platform as seen by the application. In embedded or cyber-physical system design this could be a view on operating system services as device drivers or schedulers as well as the memory layout. The new layer defined in this section allows to model the platform aspects in an abstract way. It allows the designer to concentrate on the architecture and it hides the aspects of implementation details. To introduce modeling components and graphic symbols for this part is new. In further work software engineers model the platform in a software way to interpret the services as software functions like application functions and hardware engineers are not interested to consider scheduling or memory architectures as seen by software. Therefore to describe the architecture of a hardware/software system the aspects of the platform must be considered as an architecture of hardware as seen by the application (and not only by software).

4.2.1 Resource Domains and Resources

Resource domains can contain other resource domains, service domains and resources that are the substitution of hardware components. They also contain the memory layout of their resources. A resource domain contains all relevant resources of the considered system or subsystem. Resource domains inside of other resource domains are a logical group of the system resources. This enables hierarchical composition of resource domains and encapsulation of subsystems. For communication between these subsystems, a resource in the superior resource domains is needed.

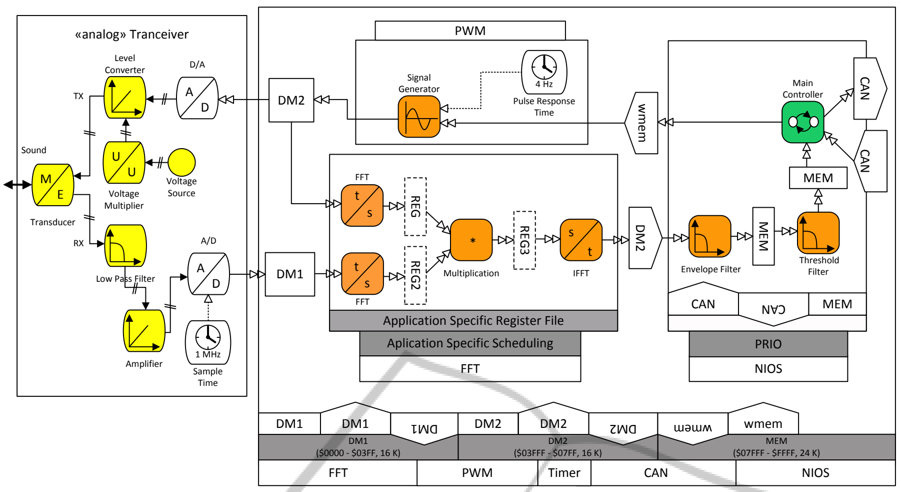

Figure 5: The system level domain architecture of the sonar system.

The root node of domains is in every case a resource domain that is a visualization of the system border.

As mentioned above, resource domains contain only relevant resources. That are resources used by the services of the containing service domains. It is not intended that all hardware components are mapped to resources e.g. an address bus is a component element that is needed for the correct work of the system but it is not used explicitly (but implicit) by services and not modeled as a resource.

Resources are the substitution of hardware components in the domain level. A domain contains all relevant resources of the considered system or subsystem modeled by the domain. As mentioned above, domains contain only relevant resources. That are resources that will be used by services.

4.2.2 Service Domains and Services

The concept of service domains establishes a service-oriented view to platforms. They are an abstraction for operating systems, frameworks, processes and threads. Service domains itself can contain service domains. At the lowest level a service domain is a process or thread. They contain services, their access points and the memory layout of these services.

Services are a concept to provide and manage computing resources. They are bound to underlying services or directly to resources. Services are divided into the following types: processing, communication, storage and timing. This categorization can be extended easily. Services are used by the application through service access points. They model the usage of services and are therefore a kind of configuration of the services depending on the current application.

4.2.3 Memory Layout

The memory layout consisting of memory partitions is available in every kind of domain. It describes the address space and size available for that domain. Resources and services will be assigned to memory partitions. A memory layout of a process describes the private memory of that process. This private memory is only virtual as it is not directly bound to a memory resource but to a memory service of the containing operating system.

5 CASE STUDY

In this section, a design example is considered that is derived from (Slomka et al., 2011). The sonar system is part of a larger project, an autonomous underwater vehicle (AUV). The major task of the sonar system is to detect objects in the water that will be used for navigation and maneuvering of the robot. The sonar system sends acoustic waves into the water and, if there is any object, receives the acoustic reverberation of that object. It consists of an electromechanical part to generate acoustic signals, an analog electronic part to drive the electromechanical sound generator and to receive the reverberation, and a digital electronic part with hard- and software for signal processing and target detection.

In this case study we will focus on the design of the platform fulfilling the requirements of the application as stated above. The application design, system design and requirements analysis are shown in detail in (Slomka et al., 2011).

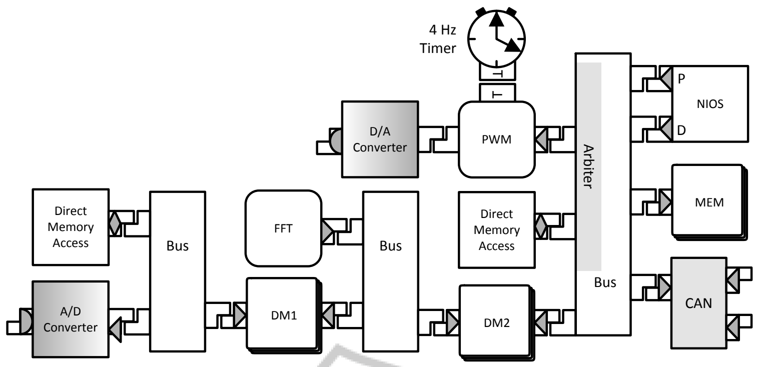

Figure 6: Platform level component architecture of the sonar system.

5.1 System Mapping and Domains

Three service domains are manually chosen for signal generation, fast fourier transformation (FFT) and main controller (see Figure 5). This is one possible kind the domain architecture chosen here for simplicity.

The next step is specifying the services to realize the communication that is modeled by messages or binary signals between computational tasks. As displayed in Figure 5 the communication services are placed between the communication links of the application. The points were the communication links are connected to the service are the service access points. This is the way to visualize the binding of communication to the platform. Tasks that transform from digital to analog domain or vice versa are known as transformers and contain a digital and an analog port. The digital port will be mapped to a communication service access point. Next, the tasks itself are all mapped to a processing service access point to enable scheduling of the tasks. Application tasks that use timing information are called timing tasks. They will be mapped to a timing access point. They provide the timing information via ports and are mapped to communication service access points.

It is already needed to create at least one resource domain containing all the hardware resources of the entire system. The result is shown in Figure 5. The domain implementing the fast fourier transformation (FFT) is not considered any more in the system level but specified in the hardware synthesis. Therefore the communication services are displayed with dashed lines and the memory layout and scheduling is defined as “application specific”.

5.2 Generation of Components

After specifying the domain architecture components and modules are generated or selected implementing that domains. This decision is a task of the system architect. Both ways are supported by the approached methodology and both ways could be mixed. In our current case study we decided to select components and modules manually from a list of available items.

Processing resources are transformed to bus masters like ISPs, ASIPs or ASPs (see Figure 3) depending on the scheduling services and attributes. Memory resources will be transformed to memory elements. If caching is used by the memory layout, cache elements are added. Timing resources are transformed to timers and clocks. Communication resources are transformed into the active or passive communication infrastructure. Resources of different domains can communicate with each other by coupling elements. Examples are memory hub controllers, buses or dual ported memories. In our case study the dual ported memory DM2 (see Figure 6) is used as a coupling element.

The automatic generation of components and modules will be done using formal transformation rules. They describe the transformation of elements of the resource domains and corresponding service domains to elements of the component level. Formal transformations are part of our future work (see Sect. 6).

6 CONCLUSIONS

The paper introduces a new abstraction level in system design. The goal is to bridge the gap between structure diagrams as used by hardware engineers and class and module based diagrams as used by software engineers. It includes graphical modeling of application and platform which are mapped together at the newly introduced system view in an intuitive graphical way. The fact is, that methods like UML are mainly used to model applications and therefore abstract from architectural details as memory layout or scheduling. But both these parts are essential for the design of cyber-physical or in general way embedded systems. The results presented in this paper are a new way to describe architectures of embedded platforms considering aspects of the design of the hardware platform. Therefore it allows the exploration of different platform implementations in early design phases of a project.

Future work will cover the meta modeling to integrate this new technique to UML as well as formal transformations to support system optimization and design space exploration. To validate our approach an implementation of the design methodology and design flow in combination with usability investigations are some more tasks to do. This will include a comparison with other modeling methodologies.

REFERENCES

AUTOSAR development cooperation (2013). AUTomotive Open System ARchitecture. http://www.autosar.org/.

Boulet, P., Marquet, P., Piel, É., and Taillard, J. (2007). Repetitive allocation modeling with marte. In *Forum on specification and design languages (FDL07)*.

Carloni, L., De Bernardinis, F., Pinello, C., Sangiovanni-Vincentelli, A., and Sgroi, M. (2005). Platform-Based Design for Embedded Systems. In *The Embedded Systems Handbook*. R. Zurawski (Ed.).

Gajski, D. (1992). *High-Level Synthesis*. Kluwer.

Lee, E. (2008). Cyber physical systems: Design challenges. In *11th IEEE Symposium on Object Oriented Real-Time Distributed Computing (ISORC)*.

Liehr, A., Rolfs, H., Buchenrieder, K., and Nageldinger, U. (2008). Generating marte allocation models from activity threads. In *Forum on Specification, Verification and Design Languages (FDL08)*, pages 215–220. IEEE.

Marwedel, P. (2011). *Embedded Systems Design - Embedded Systems Foundations of Cyber-Physical Systems*. Springer, 2nd edition.

Object Management Group (OMG) (2010). OMG Systems Modeling Language, version 1.2 (OMG SysML). http://www.sysml.org/specs/.

Object Management Group (OMG) (2011). Modeling and Analysis of Real Time and Embedded systems, version 1.1 (MARTE). http://www.omg.org/spec/MARTE/1.1/.

Object Management Group (OMG) (2013). Unified Modeling Langauge (UML). http://www.uml.org/.

Sangiovanni-Vincentelli, A. and Martin, G. (2001). Platform-Based Design and Software Design Methodology for Embedded Systems. In *IEEE Design and Test of Computers*, volume 18, pages 23–33.

Slomka, F., Kollmann, S., Moser, S., and Kempf, K. (2011). A multidisciplinary design methodology for cyber-physical systems. In *11th IEEE Symposium on Object Oriented Real-Time Distributed Computing (ISORC)*.

Sommerville, I. (2001). *Software Engineering*. Pearson Studium.

Teich, J., Blickle, T., and Thiele, L. (1997). An evolutionary approach to system-level synthesis. In *CODES*, pages 167–172.