A Multidisciplinary Design Methodology for Cyber-physical Systems

Frank Slomka, Steffen Kollmann, Steffen Moser and Kilian Kempf *

Institute of Embedded Systems / Real-Time Systems
Ulm University
{frank.slomka, steffen.kollmann, steffen.moser, kilian.kempf}@uni-ulm.de

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 is able to connect different application domains adequately. This paper proposes a new system based view for cyber-physical system design which can be easily adapted by MARTE or SysML, as it uses a model based design technique. Instead of defining another UML profile, we present an intuitive idea for the development of cyber-physical systems by refinement and introduce new abstraction layers that help to describe operating system and mixed-signal issues. Using new abstraction layers, it is now possible to support all views of the platform based design by using one consistent language. The approach explicitly distinguishes between the physical system and the computational system. The benefit of this new approach is presented in a case study where a cyber-physical system is designed.

1 Introduction

The design of cyber-physical systems [14] – 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. One issue is to connect different application domains of the system in a whole design process. To cope with this, many different design languages have been developed.

Model-based design with the Unified Modeling Language (UML) [20] 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 Modeling and Analysis of Real-Time and Embedded Systems (MARTE) [18] and OMG Systems Modeling Language ((OMG) SysML) [19]. A lot of different functional and extra-functional[1] diagrams and models are defined in both modeling languages. UML diagrams in MARTE are defined for embedded system design and support a lot of different views. However, a closer look at the specification of MARTE shows that the authors are software oriented. System aspects like the description of physical behavior with differential equations are

* This work was supported by the German Research Foundation.

[1] Extra-functional is similar to non-functional. But in our opinion, each requirement is needed for the functionality of a function. Therefore, requirements like time are considered as extra-functional.

marginal. However, for both, MARTE and SysML, the platform-based 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 not support hierarchical bindings and does not support operating system issues and physical design aspects needed in mixed-signal design. Therefore the system view in the Y-diagram of platform-based design [6] is poorly implemented in MARTE and UML. SysML does not support graphical representations for different types of physical and logical flows [19]. In SysML, the attribute of a flow is described by using flow properties. However, to support engineering of cyber physical systems, a clear distinction between different classes of types is needed. Such an approach is presented in this paper.

Another possibility to specify cyber-physical systems is the use of process models like the ANSI/ISA-5.1-1984 (R1992) standard [11] to describe measuring and control devices. For example, it allows the modeling of devices used by control room operators. Such a process model is able to describe different aspects of the physical environment and the connection, but the design of the computer architecture is not covered in detail.

It is inherent to the design of cyber-physical systems that the design process has to cover different application domains like automotive, avionic, industrial control, mobile communication, etc. Each domain has different views on technical and physical details. As a consequence to cyber-physical systems design, the design methodology has to consider all these different aspects. To cope with this problem, domain specific languages have been developed to cover the design challenges of specific systems. For example, the tool PREEvision of the company aquintos [3] can be used as design entry in the automotive domain. Unfortunately, other domains are not covered.

To handle the problems discussed above, we introduce a new approach for system modeling. We supply a new idea to support cyber-physical system design by introducing new symbols. These symbols are independent from UML, MARTE, or SysML but can be easily adapted to them. The goal of the methodology is to give the designer the opportunity to refine a system during design. The approach extends the object-oriented philosophy of designing software systems to multidisciplinary, multi-technology hardware/software systems. Therefore the methodology supports application design as well as platform design in a single view. This approach is only suitable if the methodology supports the refinement process and a refinement history. It should be possible to move in both directions of the refinement process, an approach that could be compared to the possibilities of version control tools.

Using such a new description language, it is necessary to include it into the design flow of cyber-physical systems. This is essential and often not sufficiently considered when new languages are designed. For example, in [23] a generic design flow for embedded systems is presented, but the granularity is not sufficient for our idea. A design flow for the automotive domain is presented in [26] and is confirmed by a real example in [13]. But in that contribution, only the automotive domain is covered. No adequate design flows exist that are suitable for cyber-physical systems and therefore we introduce an appropriate design flow in Sect. 2.

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

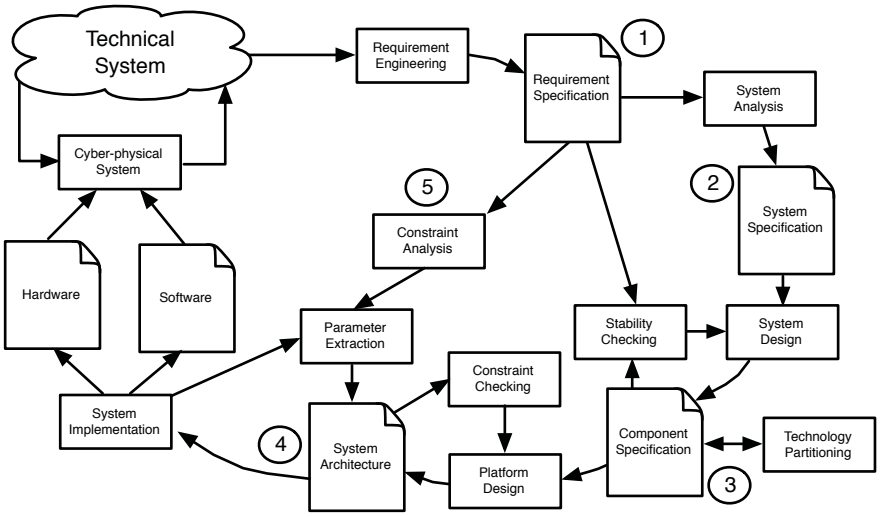

Fig. 1. Generic design flow of cyber-physical systems

2 Design Flow

Figure 1 gives an overview of the proposed design methodology which adapts object-oriented design methodologies as given in [12] to a multi-technology component approach.

In the first step, well known from software system techniques, 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. It will identify which subsystems are needed and how the information, material, or energy flows between these subsystems. Note that this step is in most cases not a software or information technology problem. It is an engineering problem, and the information technology does not play an important role at this stage. The result is a small model-based system specification which considers all aspects of the whole system.

Based on the resulting system specification ②, the system design starts. At this stage, the previously specified subsystems have to be refined and the underlying technology has to be chosen. Both parts, system design and component specification ③, are done on the application view [8]. The system and component design follow the well known steps of object-oriented methodologies, called system analysis and system design, where the design flow bases on a refinement technique. This means an engineer can first verify the system's stability independently from the computer platform which is designed later. This is an important aspect of controller dominated applications. The next step is to perform a technology partitioning. The subsystem has to be partitioned into different technology domains. This means the designer has to assign which part of the system has to be implemented in mechanical, electrical, or computational hardware and which technologies are used to realize the implementation of the hardware. Be aware that at this point in the design flow no decision concerning

software is made. At this stage we distinguish only between mechanical, analog and digital.

The step after the technology partitioning is the platform design, where components, tasks, or controllers are mapped to allocated computational resources. The platform design contains several design steps like the allocation of processing elements, communication elements, memory elements, scheduling spaces or address domains, and the binding of tasks to scheduling spaces, tasks and buffers to address spaces, scheduling spaces to processing elements, or address spaces to memory components. The result is the system architecture ④.

The next step is the constraint analysis ⑤, which is well known as hardware/software co-design or system synthesis. Further details can be found in [25] and [7]. During this design step, a constraint checker verifies the chosen computer architecture and the schedule of the chosen binding. It delivers platform parameters for the component model.

After this, the system's stability considering the influence of the platform on the application and the technical system can be verified. Based on a design space exploration step to find the cost optimal platform that allows a robust operation of the whole system, the design of the hard- and software starts. After implementing hardware and software, model parameters are extracted from the implementation specifications and the model of the cyber-physical system is checked against the specified constraints.

Considering the whole process, the system design usually starts with the platform design immediately after the system specification. This means the global view on the system design is still missing. In the following sections we will close this gap with a new system view that allows to consider multidisciplinary design criteria during the design process of cyber-physical systems. Thereby a refinement process is supported to enable stability corrections at a high level.

3 Refinement

After introducing the design flow of cyber-physical systems, we now give a closer look at the refinement process which is included. We distinguish between the hardware, software, and system refinement. It will be discussed that the classic abstraction layers of hardware and software are not sufficient for complex cyber-physical systems.

3.1 Hardware Abstractions

From hardware design, six different layers of abstractions are known. These layers are well established in the semiconductor industry and each is supported by different models of computation and verification tools. The classification is given in [8] and is divided into system level, behavioral level, register transfer level, gate level, transistor level, and layout level. As hardware design is well understood, we will not deal with these levels anymore. In the design flow given in Fig. 1 we find these parts downwards from the platform design.

3.2 Software Abstractions

Software development differs from hardware development. However, a closer look shows that software development can also be separated into different abstraction

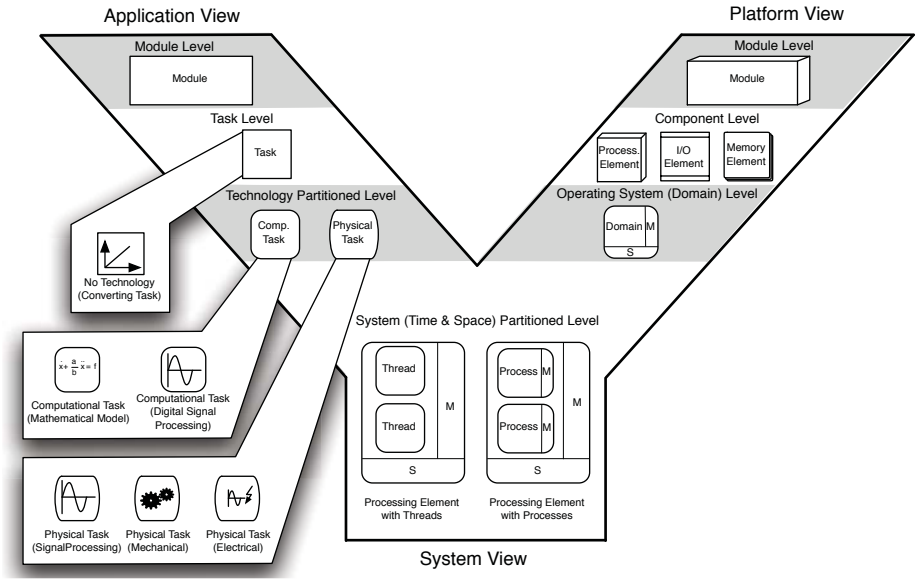

Fig. 2. Abstraction levels in the design flow

layers. This classification is adapted from [24] and is divided into architecture level, component level, algorithmic level and machine code level. The abstraction starts at the platform design as well as in hardware design, because the platform is needed to start the software design. The result is that the previous system design as given in Fig. 1 is not covered by these abstraction layers. Therefore we define a new approach to system abstraction on higher levels.

3.3 System Abstractions

Since the introduction of the system-on-a-chip paradigm, the conventional model does not apply anymore. The 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 Fig. 2 by a Y-structure. It is distinguished between the different views Application, Platform and System, as further discussed in [22]. This means the development of the application's functionality is separated from the development of the hardware platform. Hence, this approach takes into account that in many projects, hardware 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 is covered by UML and AUTOSAR [1], the system view is not supported well by these techniques.

We introduce the abstractions top-down and start with the module level in the application view. Normally, the mission level is the starting point as introduced in [4], because many cyber-physical systems are part of larger distributed systems. For example an autonomous underwater vehicle contains many different subsystems which interact: the sonar system, the navigation, the actuator of

the vehicle, and the control unit. The interaction of all these components is considered on the mission level. For this paper, the mission level is not considered any further.

Application View The application view is supported in three abstraction levels: We distinguish between the module and the task level. The task level itself is separated into two different levels: The first one is a level at which a designer only concentrates on the functionality of the system. This models the middleware abstraction of a system which is comparable to the AUTOSAR [1] standard for the automotive domain whereby our middleware is not domain dependable. It is done by the composition of the several tasks describing the behavior of specific system parts. Each behavior 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. In other words, only entities of functions are composed. In the second task level, the technology partitioned level, design decisions for the mapping of tasks to technologies are described. At this stage of the development process the designer documents the choice between digital hardware and analog electronics or mechanical elements. The computational tasks are mapped to the platform in a later stage described in detail in the step System View.

Platform View The platform view considers only the hardware for the computational tasks. The process starts with the module level to describe the approximated platform. The refinement step Component Level defines the system platform in more detail. At this level the processing elements (like CPU or buses), I/O elements, and memory elements are specified. The resulting architecture from the component level can then be refined at the operating system level. At this stage the different components are divided into domains, describing the scheduling behavior and memory architecture of the platform.

Note that this is an abstraction to support space and time partitioned platforms for the applications. For the application it is not important how the components are connected, it is important which memory can be used and when it gets resource time. Therefore this level supports an abstraction for operating systems.

System View The system view unites the application and platform view. At this level application tasks are mapped or bound to the elements of the platform. Each computational task is assigned to a platform domain describing its scheduling and memory architecture. 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.

In MARTE, allocation means the mapping of application to a platform [5]. In system synthesis, allocation means to choose a component, while binding is the term to describe mappings of tasks to allocated components. As described in [5], MARTE distinguishes between structural and behavioral bindings. In this paper we use the system synthesis terms instead, as the aim of the methodology is to synthesize embedded systems. As shown in [5], the binding mechanism in MARTE does not consider that a task needs both a computational resource as well as memory resources. If this is modeled in a naive way, for each part – computational binding and memory binding – different allocation arrows are

needed. In complex architectures this is not an elegant way because of the rising complexity. This approach is given by MARTE (see pages 132 ff. [18]).

In this paper we introduce domains. Domains are an abstract model of resources and can be modeled using the service construct in MARTE. However, services are not the concept needed by synthesis, because services hide architectural aspects. Therefore domains are a special kind of modeling element, as they allow an abstract formulation of architectures. Not in the detailed way like component diagrams, but more in an abstract way. Using the concept of hierarchical composition of domains, it is possible to describe complex hardware 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 [15]. However, the activity threads discussed in that paper are mappings of the detailed architecture to the application. This is equivalent to modeling the binding with application arrows and does not reduce the complexity. Additionally, our approach explicitly supports the design refinement of cyber-physical systems. In MARTE or SysML this is only supported by using attributes.

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 [25]. 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.

4 System Description Language

In this section we introduce our new system description language as a design entry for cyber-physical systems. As discussed in the last section, this view allows to support the refinement process in the design flow as well as the connection of different technology domains into one model. In contrast to other models, the design of the computer system is still possible, because the methodology can be easily adapted to UML/SysML and other techniques. Models given in a graphical specification language have to be intuitively understandable. As mentioned, this is one reason for the legitimacy of domain specific languages. Our intention is to consider the domain of cyber-physical systems in general. Limitations of the approach as described in this paper are a missing binding from constraints to clocks. The analog part is just drafted in the paper to separate it from the digital part. Therefore the presented methodology does not allow a compositional design flow on the analog parts of the cyber-physical system.

4.1 Application View

To support a multi-technology model, different symbol types are defined for the application view, which are depicted in Fig. 2. First of all it is distinguished between modules and tasks. Modules are blocks containing other design entities such as modules and tasks. They are used to support a hierarchical design methodology. A module may contain different tasks. However it is also possible that a module contains mathematical models, functions, or formulas. There are different types of modules specified to support intuitive and readable system models: modules keeping mathematical models, modules keeping electronic

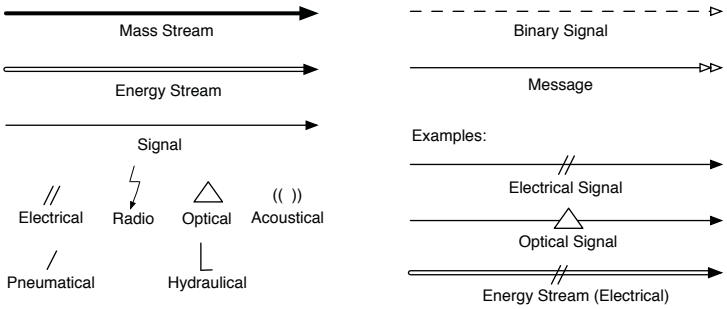

Fig. 3. Symbols of communication

systems, modules keeping electrical models, and modules keeping mechanical models. These types are supported in all abstraction levels as shown in Fig. 2. So a task can be a converting task where the technology is not specified. But it is also possible to have tasks with a technology binding and a type.

4.2 Platform View

The platform view also starts with modules to support a hierarchical design methodology. The modules can consist of several modules or components. Components are hardware elements such as memory, processors or processing elements and interface elements. Therefore components are model elements of a hardware view. The components can then be refined by domains. A domain in the platform view describes on the one hand the access to the resource through a scheduling strategy and on the other hand the memory architecture in which the domain is embedded.

4.3 Communication

To connect the different tasks in the application view, different types of communication mechanisms exist. They are depicted in Fig. 3. As well known from hardware description languages, the communication links are called signals and messages. The following classes of signals are supported: 1. Mass stream: Transport of mass. 2. Energy stream: Transport of energy. 3. Signal: Transport of information. 4. Binary signal: Computational signals as known from hardware description languages with the binary states, like true, false, undefined and high-impedance. 5. Messages are only supported in computer systems. It is also possible to define message types with complex payload data unit structures, like integer or string messages. Additional symbols are added to the communication links to characterize the link: electrical, radio, optical, acoustical, pneumatical and hydraulical types are supported. This is illustrated by three examples in Fig. 3.

Signals of different semantics can only be connected by using transformers. An example is given later in the paper: a transducer transforms an electrical signal to an acoustical/mechanical signal. Therefore design tools have to check the type of a language construct in order to separate different views and to avoid connecting different incompatible components.

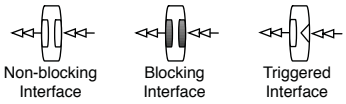

Fig. 4. Symbols of interfaces

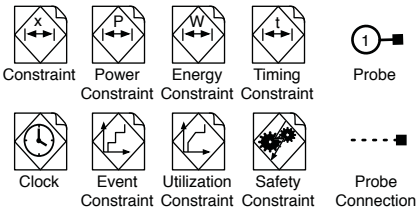

Fig. 5. Symbols of requirements

4.4 Interfaces

Figure 4 gives an overview of different graphical symbols that model interfaces to computational tasks. An interface is generally specified by a rectangle with round sides. It is distinguished between different types of interfaces that have been partly inspired by the abstract communication channels presented in [9]. There are blocking, non-blocking, and triggered interfaces. Blocking means that the sender is blocking until the data is received by the receiver. Non-blocking means that the data is stored until the receiver is ready to receive the data. Triggered means that the receiver task is triggered to receive the data. Other possibilities for interfaces between computational tasks can be found for instance in [10].

4.5 Constraints

To specify extra-functional constraints, graphical elements are defined. Such elements are a diamond in a typical comment symbol defining the type of the constraint as shown in Fig. 5. We distinguish between timing constraints, area or cost constraints, power and energy constraints, and safety constraints. Due to the compositional approach of the diagram types it is very easy to define additional constraints. Note that requirements are always specified between probes.

Assume that in this paper the time model presented in [17] is used instead of the models given in MARTE. This is done because MARTE does not give a formal semantic to its model and the MARTE model is just a syntactic extension to the time model of [17].

5 Case Study

In this section, a design example is considered. The sonar system is part of a larger project, an autonomous under water 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. Such a system is a typical example of mixed technology domains. 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. We divide the design process into three parts. The first one is

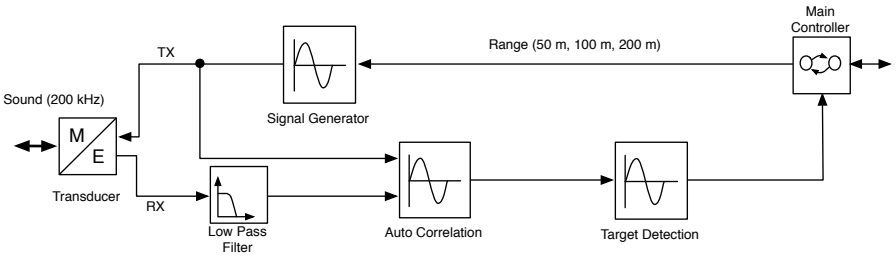

Fig. 6. Analysis components

the system design, starting with the requirements analysis. In the second step the platform design is conducted. Finally, the system is implemented with well known procedures for hardware and software design.

5.1 System Design

We start with the system design. The focus here is to perform the requirements engineering and to conduct the system analysis. Based on this, the main components are specified and the technology partitioning takes place.

Requirement Analysis The sonar has the ability to detect underwater objects (targets) by using sound signals of in this case 200 kHz. It sends an acoustic signal and then waits for echos from the targets. In order to accomplish that, the sonar has to measure the time from the emission of the signal up to the reception of the echos generated by several targets. The system should be able to detect objects in the range from 2 m up to 200 m. To support a high resolution and a fast detection of narrow targets, the range has to be chosen from three ranges: 2–50 m, 2–100 m, and 2–200 m. These ranges are available in different modes of the sonar.

System Analysis The second step in the design process is the application analysis, as shown in Fig. 6. In this step, the main functionality of the system is specified. The design of an actor oriented system starts with its design entities. In the case of the sonar system, the following analysis objects are defined: A signal generator that forms the sending signal, a transducer that transforms the electrical signal into an acoustic signal, and a receiver that filters the received signals. This step is mandatory because the signal strength of the received signal is very low compared to the strength of the sending signal. After receiving the echos of detected objects, an auto correlation function detects the echos of the sending pulse (see Fig. 6). This means that the pulse form of the received signal must be the same as the pulse form of the sending signal. After filtering, only signal echos remain. A target detection then analyzes the echos and determines the target's destination. The whole system is controlled by a main controller. In this phase, the main controller, the low pass filter, and the transducer are modeled as tasks, while the signal generator, the auto correlation, and the target detection are drawn as blocks. These blocks will be divided into subtasks during the following design steps.

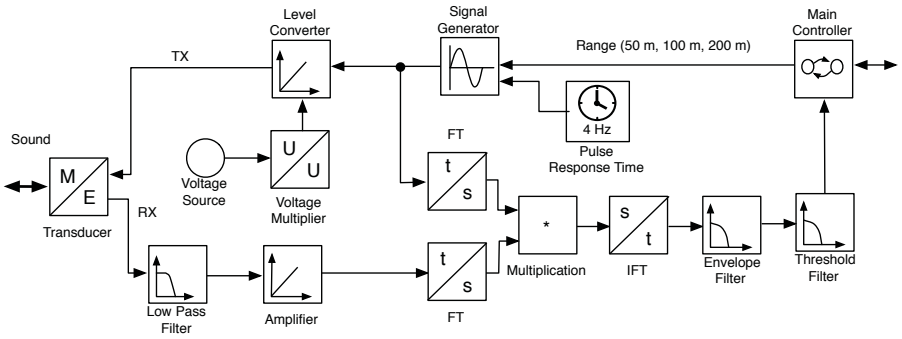

Fig. 7. Design components

System Design In the next step, the application design has to be refined. How is the auto correlation and the target detection implemented, and how does the signal generator work? The resulting application design is shown in Fig. 7.

Signal Generator: The sonar signal will be generated by a signal generator which forms digital pulses. A level converter transforms these pulses to the voltage level needed by the acoustic transducer, an X-cut crystal which transforms electrical signals into acoustic waves. The transceiver sends pulses of 200 kHz as given in the requirements. To prepare the signal detection, the sending signal generated by the signal generator is transformed by a Fourier transformation.

Auto Correlation: After amplifying the signal, the echo detection is performed by an autocorrelation function which can be implemented with a signal convolution. The autocorrelation identifies the sent pulse in the received audio spectrum. This step is necessary to be able to detect the received signal against the background noise of the sea. A convolution contains an integration of the received signal. However, it is known from signal and system theory that such an operation in the time domain can be transformed into a multiplication in the frequency domain. This transformation is performed by the Fourier transformation. The multiplication then may perform the convolution or the correlation function.

Target Detection: After transforming the signal back into the time domain, the targets are detected by an envelope filter and a threshold filter. The envelope filter reduces the information of the signal to just the interesting envelope while the threshold filter detects the received pulses and their timing. The timing of the received pulses is then sent to the main controller which calculates the range of the detected objects. The sonar data can then be sent to other system components like the navigation module of the robot.

Technology Domain Binding The first step in system refinement is the domain binding. The system architect has to decide which parts of the system are implemented in which technology. As seen in Fig. 8, the transducer and the level converter are implemented in the physical electronic domain. The system architect decides to implement the signal processing in digital electronics because the maintainability of digital systems is better and the design of digital electronic is easier than the design of analog components. However, to support

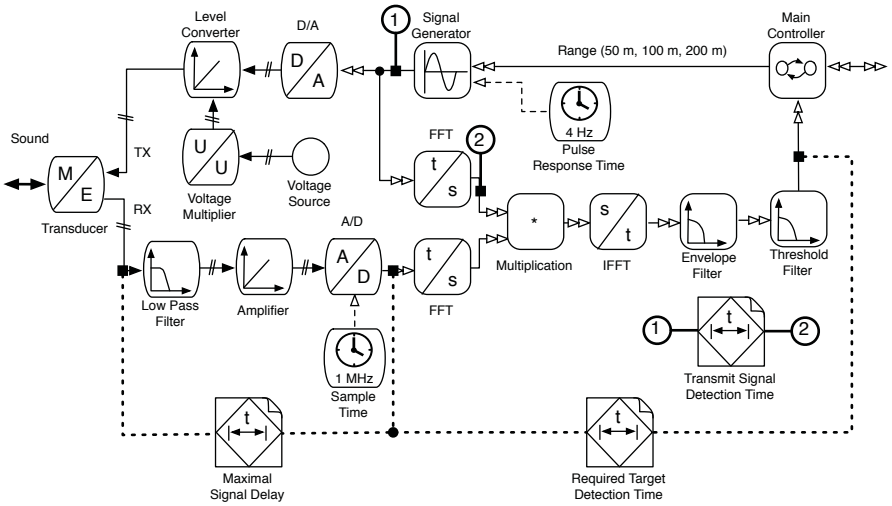

Fig. 8. Technology binding with added probes and timing constraints

this decision, a new component must be added to the signal flow. An analog digital converter is needed to discretize the received analog signal. As seen in Fig. 8, the Fourier transformation, the signal multiplication the signal generator, the inverse Fourier transformation, the envelope and threshold filter, and the main controller are now refined into digital processes. After binding tasks to a technology domain, electrical signals are refined into binary signals or messages.

Timing Requirements Fig. 8 also shows the timing requirements, which are added in this step. To sample 200 kHz signals, a sampling rate of twice that frequency is needed. However, the sampling rate of the analog-digital conversation is chosen to be 1 MHz, because the low pass filter is not perfect and the signal contains parts with frequencies above 200 kHz. So additionally, three timing specifications are added: the sampling rate of the analog-digital conversion, the required time for the target detection, and the overall computation time of the system. The time required for target detection was calculated in the following way: As given by the system specification, the sonar has three detection ranges. A short range up to 50 m, a mid range up to 100 m, and a long range up to 200 m. According to the average signal speed in water, a signal from an object in a distance of 50 m is received after 71 ms, so after sending a pulse, the system has to wait for this time before sending the next pulse. Because the 50 m range is the fastest system mode, a target detection time of 71 ms is needed. From these considerations, the maximal pulse response time may be calculated. To support the auto correlation, an additional timing requirement is needed, the transmission signal detection time. The Fourier-transformed sending signal has to be stored before the first echos are received. As stated in the AUV specification, the minimal detection range is 2 m, which leads to a transmission signal detection time of 3 ms.

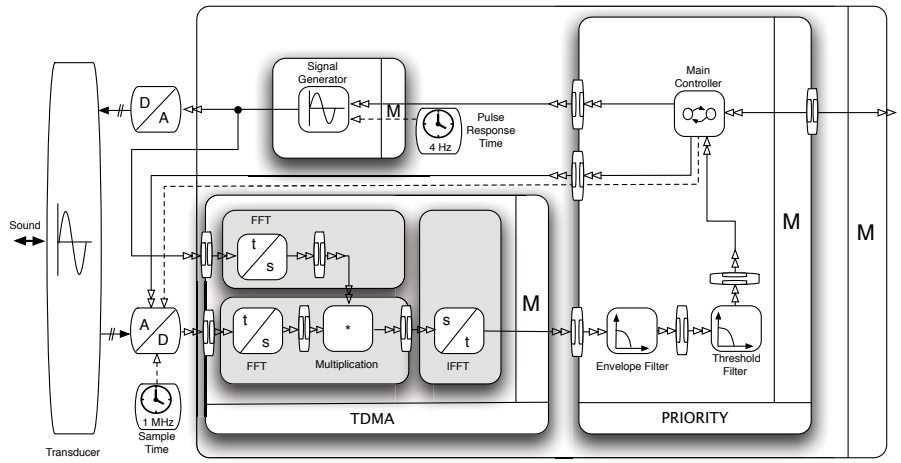

Fig. 9. Architecture of the sonar

5.2 Platform Design

After the definition of the application, the platform has to be designed. In this step, the hardware/software architecture of the system is devised. This means the required circuit techniques are selected in the analog domain, while in the digital domain, hardware technologies, hardware architectures, processing components, and software architectures are designed.

The first step is to specify buffers in the form of non-blocking interfaces. Each process has to be able to communicate with other processes through messages. To model the message communication, buffers are added to the model. Their size can not be specified at this first step because it depends on the scheduling and binding of the processes. After adding the buffers, the scheduling and memory spaces are selected. This step is an optimization process and will be performed with the help of a design space exploration.

Binding and Allocation of Computation A typical example is shown in Fig. 9. In this example, a hardware component which implements the signal generation, an instruction set processor and a coprocessor are allocated. The coprocessor implements the fast Fourier transformation (FFT) and a subsequent multiplication. The FFT coprocessor can also be used to execute the inverse Fourier transformation. This is the reason why the IFFT is bound to its own scheduling domain. All FFT operations are scheduled to the hardware coprocessor FFT that also performs the multiplication and on which a TDMA (Time Division Multiple Access) scheduling is employed. The main controller and the two digital filters are mapped onto the instruction set processor and are scheduled by a fixed-priority system. The hardware/software model is analyzed in the next step. The architecture presented in Fig. 9 can be directly used as a specification for Symta/S [21], the real-time calculus [27], or the event-spectral calculus [2] by adding the required model parameters like the worst- and best-

case execution times. The event models necessary for the analysis can be derived directly from the specification as well as the timing requirements.

Binding and Allocation of Memory In embedded systems, the platform often consists of more than one processor. In our example, two application specific hardware processors and one instruction set processor are used. The data spaces of the tasks and the message buffers for the interprocessor communication has to be bound to the system's address space. One solution could be the allocation of one memory element and the binding of all buffers and process data areas into one address space. However, there are several possible solutions. The FFT processor may have its own memory and the buffers to and from the FFT processor are bound to that memory. In order to perform a communication to the filters implemented on the instruction set processor, the buffer between this task then has to be implemented on a shared memory.

5.3 Implementation

The last step is to implement the system based on the embedded architecture. From here, well known established methodologies can be used to implement the software and hardware components as introduced in Sect. 3. For example, the system can be defined in Matlab/Simulink [16], which will provide an appropriate model for the verification. Based on code generation toolboxes it is possible to generate the hardware description language and software code for the implementation. Using automated back end processes leads to the desired implementation.

6 Conclusion

In this paper we have presented a new methodology for the description of cyber-physical systems. We have introduced a new way to model such systems in the whole. The main focus was the definition of a new entry for the design process, covering multidisciplinary design constraints. In contrast to other models, the presented approach provides the possibility to model systems with their influencing physical properties. A stepwise refinement of the system model has been introduced. One advantage is that the developed methodology is not orthogonal to the established standards MARTE or SysML and can therefore be easily adapted by these. The future work will cover a detailed discussion of the platform aspects and how domains are described at the component level. Another open issue is to go into deeper details of the physical and analog concepts of the methodology to better support the mixed-signal code generation and simulation.

References

1. AUTOSAR. http://www.autosar.org/
2. Albers, K., Slomka, F.: Event Stream Calculus for Schedulability Analysis. In: Analysis, Architectures and Modelling of Embedded Systems, IFIP Advances in Information and Communication Technology, vol. 310, pp. 102–114. Springer Boston (2009)

3. aquintos: Preevision. http://www.aquintos.com/
4. Baumann, T., Salzwedel, H.: Mission Level Design using UML 2.0. In: Proceedings of the NODe '05, Object Oriented Software Design for Real Time and Embedded Computer Systems (2005)
5. Boulet, P., Marquet, P., Piel, É., Taillard, J.: Repetitive allocation modeling with marte. In: Forum on specification and design languages (FDL07) (2007)
6. Carloni, L., De Bernardinis, F., Pinello, C., Sangiovanni-Vincentelli, A., Sgroi, M.: Platform-Based Design for Embedded Systems. In: The Embedded Systems Handbook. R. Zurawski (Ed.) (2005)
7. De Micheli, G.: Synthesis and Optimization of Digital Circuits. McGraw-Hill Science/Engineering/Math (1994)
8. Gajski, D.: High-Level Synthesis. Kluwer (1992)
9. Gerstlauer, A., Shin, D., Peng, J., Domer, R., Gajski, D.: Automatic layer-based generation of system-on-chip bus communication models. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on 26(9), 1676–1687 (2007)
10. Gladigau, J., Gerstlauer, A., Streubühr, M., Haubelt, C., Teich, J.: A System-Level Synthesis Approach from Formal Application Models to Generic Bus-Based MPSoCs. In: Proceedings of the International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS) (2010)
11. International Society of Automation: ANSI/ISA-5.1-2009. http://www.isa.org/
12. Jacobson, I., Christerson, M., Jonsson, P.: Object- Oriented Software Engineering. Addison-Wesley Longman (1992)
13. Kollmann, S., Pollex, V., Kempf, K., Slomka, F., Traub, M., Bone, T., Becker, J.: Comparative Application of Real-Time Verification Methods to an Automotive Architecture. In: Proceedings of the 18th International Conference on Real-Time and Network Systems (2010)
14. Lee, E.: Cyber physical systems: Design challenges. In: 11th IEEE Symposium on Object Oriented Real-Time Distributed Computing (ISORC) (2008)
15. Liehr, A., Rolfs, H., Buchenrieder, K., Nageldinger, U.: Generating marte allocation models from activity threads. In: Forum on Specification, Verification and Design Languages (FDL08). pp. 215–220. IEEE (2008)
16. Matlab/Simulink. http://www.mathworks.de/
17. Münzenberger, R., Dörfel, M., Hofmann, R., Slomka, F.: A general time model for the specification and design of embedded real-time systems. Microelectronics Journal 34(11), 989–1000 (2003)
18. Object Management Group (OMG): Modeling and Analysis of Real Time and Embedded systems, version 1.1 (MARTE). http://www.omg.org/spec/MARTE/1.1/
19. Object Management Group (OMG): OMG Systems Modeling Language, version 1.2 (OMG SysML). http://www.sysml.org/specs/
20. Object Management Group (OMG): Unified Modeling Langauge (UML). http://www.uml.org/
21. Richter, K.: Compositional Scheduling Analysis Using Standard Event Models - The SymTA/S Approach. Ph.D. thesis, University of Braunschweig (2005)
22. Sangiovanni-Vincentelli, A., Martin, G.: Platform-Based Design and Software Design Methodology for Embedded Systems. In: IEEE Design and Test of Computers. vol. 18, pp. 23–33 (2001)
23. Schliecker, S., Hamann, A., Racu, R., Ernst, R.: Formal Methods for System Level Performance Analysis and Optimization. In: Proceedings of the Design Verification Conference (DVCon) (2008)
24. Sommerville, I.: Software Engineering. Pearson Studium (2001)
25. Teich, J., Haubelt, C.: Digitale Hardware/Software-Systeme. Springer (2010)
26. Traub, M.: Durchgängige Timing-Bewertung von Vernetzungsarchitekturen und Gateway-Systemen im Kraftfahrzeug. Ph.D. thesis, University of Karlsruhe (2010)
27. Wandeler, E.: Modular Performance Analysis and Interface-Based Design for Embedded Real-Time Systems. Ph.D. thesis, ETH Zurich (September 2006)