Cyber/Physical Co-Design in Practice: Case Studies in METROII

Luca Rizzon and Roberto Passerone
Dipartimento di Ingegneria e Scienza dell'Informazione
University of Trento, I-38123 Trento, Italy
Email: luca.rizzon,roberto.passerone@unitn.it

***Abstract*—To analyze embedded systems, engineers use tools that can simulate the performance of software components executed on hardware architectures. When the embedded system functionality is strongly correlated to physical quantities, as in the case of Cyber-Physical System (CPS), we need to model physical processes to determine the overall behavior of the system. Unfortunately, embedded systems simulators are not generally suitable to evaluate physical processes, and in the same way physical model simulators hardly capture the functionality of computing systems. In this work, we present a methodology to concurrently explore these aspects using the METROII design framework. In this work, we provide guidelines for the implementation of these models in the design environment, and discuss the results gathered with the simulator for two case studies.**

I. Introduction

Embedded Systems (ESs) are computing devices specifically designed and optimized to perform a specific task. This term is used to refer to systems that exhibit features that are in contrast to general computing platforms, which are designed to execute a variety of tasks that can differ significantly from one another. PDAs, smartphones, car ECUs, avionics, industrial automation systems, home automation, among many others are all examples of instances of ESs. While these applications encompass different domains, and differ in requirements and attributes, they do share common design challenges. For example, mobile devices require a low energy consumption profile to guarantee an acceptable battery lifetime without compromising the device size or weight [1]. In the case of systems that affect individual's safety, such as for example car ABS or avionics system, the reaction time of the control algorithm is crucial; therefore, the execution time of certain tasks, and their probability of being interrupted from other events are of fundamental importance, as opposed to the case of general purpose computers.

Recently, ES applications made of software and hardware components that work together to *monitor and to control physical quantities* have received growing attention [2]. These applications, which exhibit a tight coupling between the computation domain and the physical world, are referred to as Cyber-Physical System (CPS). In a typical CPS, computation elements (processes, networking) influence the physical quantities, and conversely the effect of physical processes can influence the behavior of computing processes. This cross-relation complicates the design, especially because the computational and the physical components are handled by professionals with different profiles and are generally studied with models which differ significantly.

For engineers developing CPS it is of fundamental importance to design the cyber as well as the physical part of the system taking into account both the computational unit features, and the results of the interaction with the physical world. Both aspects, in fact, affect important system metrics, such as energy efficiency, efficient utilization of limited computational resources (memory, processors, speed), real-time constraints, and predictability of the outcome. However, due to the intrinsic difference that exists between the cyber and physical domain, design tools and simulators designed to model one world are typically not suitable to model the other, and vice versa [3]. Therefore, the current practice is to study the two worlds separately, with the risk of not properly modeling how the two interact. In particular, only very few description of studies applied to real cases have been presented in the literature [4]. In this paper, we propose a methodology to integrate computational models and physical models using a design strategy inspired by the function/architecture co-design paradigm: while cyber and physical components are conceptually separated, they are integrated over two mapping steps to result in an overall system model. We support the methodology using the METROII design exploration tool [5]. In particular, we explore the functionalities offered by the METROII design framework and how to use them in practice by studying how to analyze two very different case studies, to identify and outline best design practices that can be adopted to express and combine the cyber/physical features in METROII. The proposed practices can be adapted or extended to model other, equally diverse systems.

This paper is structured as follows. In Section II we summarize the related work, and we introduce METROII and the proposed design methodology in Section III. The following two sections are dedicated to two case studies, a binaural guidance interface (Section IV), and a thermoelectric energy recovery system (Section V), and their analysis and design space exploration. Section VI concludes the paper.

II. State of the Art

Several design methodologies have been presented in the literature and applied successfully to cope with the design challenges of CPS. In Platform-Based Design (PBD), the design moves from the application space to the architectural space through a mapping process [6]. The approach is based on the separate definition of the functionality of the system, and of the possible implementation platforms. The two are then combined by *mapping* the functionality on the different platform architectures to assesses the performance of interest in

a structured way [7]. The PBD paradigm can be enforced using contracts [8]. In this case, the design methodology is carried out in two steps. The first consists in the definition of the cyber and the physical architecture, followed by an exploration of the target platforms. In our work, we follow the PBD paradigm and extend it to a two-step mapping process to separately specify and then combine cyber and physical components.

Model-Based Design (MBD) addresses the problem of designing complex system starting from a model that represents the system behavior using numerical methods, models or measurements. Jensen et al. discuss the MBD approach in the context of CPS, by dividing the design process into steps, from physical modeling, to simulation, software synthesis and validation [9]. The authors exemplify the methodology with Ptolemy II, a modeling and simulation environment for heterogeneous embedded systems which supports MBD [10]. Our approach is more structured, and defines precise roles for the architectural and the functional elements of the models. The METROII design tool, which we use in our work, has been also used in combination with Ptolemy II, originating the design tool for timing verification called *Metronomy* [11]. Also based on Ptolemy II, PTIDES focuses on the design of event-triggered real-time distributed systems, and includes a programming model and the relative toolchain that supports MBD [12]. The PTIDES approach combines the construction of a model with a simulation of the system and network, and estimates the behavior of the system mapped onto a target platform. The implementation platform introduces delays that may affect the execution of real-time systems. The tool integrates the simulation of Discrete Events –typical of cyber components– with different physical time models, to perform timing verification, and generate executable software. An alternative approach, proposed by Noyer et al., includes the adoption of three different design tools in the workflow: 1) a tool for timing analysis, 2) a tool that supports Model Driven Development, and 3) a tool for Requirement Management (RM) [13].

The *Matlab* design environment, which follows the MBD approach, supports CPS development through *Simulink* or *SimScape*. *Matlab* is suitable to capture the functional specifications of a system as well as the architecture specifications. However, it lacks the semantic construct to be able to specify concurrency, therefore it cannot support the function/architecture co-design principle. An alternative approach to the one presented in this work is based on the adoption of *Modelica* [14]. *Modelica* is a powerful tool for modeling and simulation of mechanical, electrical, thermal, control and hydraulic systems. In its standard version, it does not provide simulation of computing systems, but since it is designed to be domain neutral, it is flexible enough to support the description of such kind of systems. Recently, a support library called *Modelica_EmbeddedSystems* was released, which represents an extension of *Modelica* from the physical domain toward the digital domain. A design approach that combines *Modelica* and METROII has been developed [15], but it keeps the two designs separated and makes them communicate via *CORBA*.

Another framework for building complex systems by composition of components is Behavior-Interaction-Priority (BIP) [16]. In BIP, the system specification is organized in three layers. The first layer describes the components in isolation, the second layer is used to specify the interaction of components activation through *connectors*, and the top layer is used to express the priorities that characterize the overall system architectures. To support CPS design, *BIP* can interact with *Simulink* physical models by using translators to convert *Simulink* blocks into *BIP* specifications [17].

To address the problem of CPS design, recent technologies are devoted to the joint modeling and simulation of domain specific software [18]. Since the intrinsic multidisciplinarity of the design process involves the study of concurrent, heterogeneous models, a design cannot be studied in a unique framework due to the different semantics in use. Some design methodologies involve the structured exploration of the cyber and physical worlds by using two different design tools, and then applying iterative algorithms to converge towards a solution that meets the design requirements [19], [20]. Other design flows use two different design tools and a new specification language to bridge programming languages that otherwise could not communicate. One example is the use of an Intermediate Format (IF) to exchange models between *Matlab/Simulink* and the C language [21]. Standards can alternatively be used for tool interoperability. The Functional Mockup Interface (FMI) is a tool-independent standard aimed to ease the collaboration in design projects where different tools and workflows are used. Using FMI, the designer distinguishes between the functionality of the component (code or binaries), and its interface data. Tools supporting FMI allow engineers to import models and co-simulate them to perform design validation, promoting the exchange of simulations model across tools [22]. Our methodology is orthogonal to this approach, which could be conveniently used to extend the integration capabilities.

III. DESIGN METHODOLOGY AND METROII

In our work, we adopt the METROII design framework [5]. A METROII system design is made of a *Functional* and an *Architectural* model, to promote reusability and separation of concerns. The Functional Model expresses what the design does in algorithmic terms, such as the control strategy. The Architectural Model describes how the system is implemented, the hardware platform where the control algorithm executes, including embedded processors, sensors, actuators, communication primitives, operating systems, firmware, and drivers. While the functional model can be used to perform control validation, the architectural model contains values and formula used to model the physical quantities of interest, to simulate physical time, power consumption, associated costs, and other features of interest.

In METROII, functional and architectural models are described as netlists of *components* that communicate through *ports*. Each port is associated to a set of methods defined by the port interface. A port can be either *required* or *provided*. Required ports specify a set of methods that the component requires from other components. Provided ports specify the methods that the component implements and that it can perform when requested by others. A *component*, in turn, contains *processes*, modeled as imperative code that implement the main functionality and the provided methods.

The basic behavior of components is initially specified in *SystemC*, then encapsulated by wrapping the code to provide

METROII interfaces. The components of a METROII system are concurrent, and synchronize through events. Each method is associated to a pair of *begin* and *end* events, which mark the start and end points of the procedure execution. The execution of a METROII system is orchestrated by an *event scheduler*, which triggers the methods of the components. Additional events are used to characterize *quantities*, which model non-functional aspects such as execution time and power consumption. *Annotators*, which are similar to event schedulers, are used to label quantities with values corresponding to the event triggering, providing the means to conduct performance analysis. In particular, end events are associated with tags expressing time the annotator uses to evaluate the execution time in the target architecture.

Event synchronization is specified through *constraints*. In particular, architectural components are synchronized with the functional components through *mapping constraints*, which schedule functions and their respective architectural implementations (tasks) simultaneously. This mechanisms implements a function/architecture co-design paradigm: functions synchronize with the architecture, which progresses according to the modeled quantities, to produce an estimate of the system performance. Likewise, abstract functional communication primitives, such as a blocking FIFO, can be synchronized to an architectural implementation, such as a shared memory.

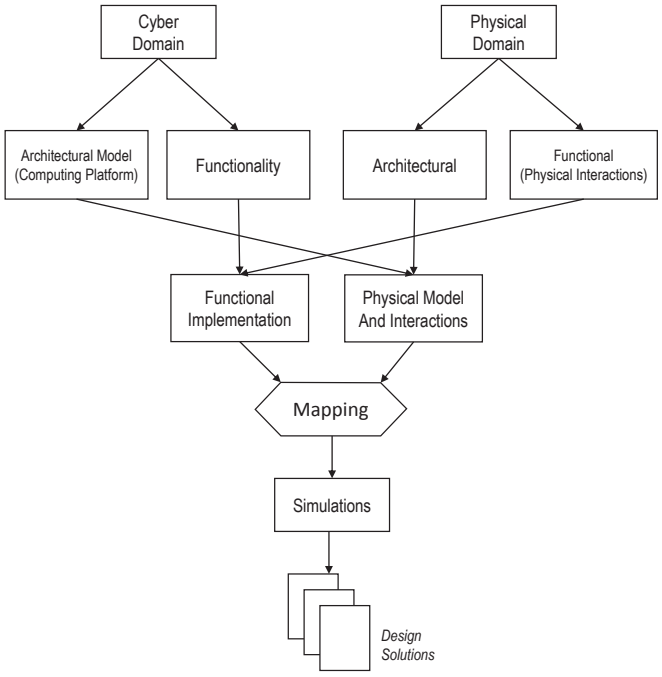

Fig. 1. Proposed workflow to model cyber-physical systems by separating functional and architectural models of the computing and physical parts of the system and conducting the simulations with the same design environment

In this work, we extend the function/architecture approach to perform cyber/physical co-design, adopting the procedure illustrated in Figure 1. We follow a paradigm similar to the function/architecture co-design appraoch. However, we first decompose the computational and the physical part of the system under consideration. Since physical quantities are handled and modeled in the architectural part of the system, to model the physical part of a CPS the designer focuses on the architectural model of the system first. Mathematical models that describe the physical behavior of the system are implemented in C/C++ inside the architectural components. While not used in our case studies, these models may involve differential equation solvers (e.g., the Euler method), when a discretized continuous time description is required. Interfaces are used to forward physical quantities to other components according to the interaction that exists between architectural elements, whether they represent a physical or a computational process. In this phase, the designer analyzes physical interaction between architectural components to identify the component organization. The interaction is implemented through ports. Then the designer analyzes the functional model of the physical part, to identify components that interact with the functionality of functional functions. A physical process that influences (or is influenced by) the behavior of the functionality of the system is modeled as a functional component. A special provision must be made if a physical process is defined only in the physical functional model. In that case, the designer must include also a virtual (or counterpart) component in the physcial architectural model, to allow the *mapping constraints* to schedule and capture its execution, even if the component is isolated from other components from the point of view of the architecture.

In the architectural model of the design, the message exchange between components is used to connect provided and required objects as in traditional ESs design. In addition, interfaces are used to forward physical quantities between components that physically interact. For example, to model how the temperature of a component *A* influences the actions of another component *B* in one of our case studies, component *A* contains a required port to a shared memory, and *B* has a provided port to the same memory. The interface is used by *B* to receive temperature values from *A*.

Once the design is completed, the overall functional model contains the functionality of the computing part of the system, with components and ports that model the interactions of physical processes with the system functionality. The architectural model contains the definition of the computing platform together with the physical models and interactions between physical and computing worlds. The METROII scheduler is used to coordinate the execution, the same way as in the design of embedded system. However, events that are triggered by physical processes (i.e., a value reaches a threshold) are ignored by the METROII scheduler, but are fired from direct, concurrent function calls performed by an architectural component.In this way it is possible to define the order of execution of code portions according to on the behavior of physical quantities that is not known before execution. In fact, the events logical ordering handled by the scheduler cannot be conditioned in runtime. Generally, the *physical time annotator* is used to evaluate the execution time of a design. To simulate the evolution in time of physical quantities, the *annotator* is used to define the time resolution of the simulation. Since the physical time annotator is used to tick the simulation time, and the logical sequence of events may change depending on the design, if the designer wants to monitor the time between two generic events the developer must insert additional code to extract time values. Designers can use the tool to evaluate the features of many implementations mapped into several

architectures to identify the solution that best matches the application constraints. At that point, having studied the design by separating the functional and architectural aspects of both computation and the physical domains, the selected solution can be implemented into the target architecture reusing the functional code.

The next two sections describe how this methodology is applied in practice, by describing in detail the design of two CPS case studies, where human behavior in the first, and heat propagation in the second, constitute part of a cyber-physical feedback loop.

IV. Binaural Guidance System

The binaural guidance interface has been developed in the context of the EU funded project *DALi* (Devices for Assisted Living). The main objective of the project is to develop a robotic platform that permits older adults to autonomously move across public spaces. The result of the project is called *c-Walker* [23]; it consists of a four-wheeled rollator equipped with sensors, computing boards, actuators, and interfaces. While a standard rollator is generally used to support gait of people with reduced strength at the lower limbs, the *c-Walker* extends the cohort of potential users including those who suffer from cognitive difficulties, and need support in order to orientate and to identify the best trajectory to follow in an unknown environment to reach the desired destination.

The user of the *c-Walker* selects the places he/she wants to visit using a touchscreen mounted in between the handlebars. The system computes the optimal route using information stored in a map database. When the user starts walking, the *c-Walker* uses different sensors to localize itself inside the environment, and compares its real time position with the planned route. The planned route may change along the way according to the dynamics of the scene monitored using a camera and a depth sensor. If the user does not deviate from the planner route, the system remains passive, as a conventional walker. If the user deviates from the optimal route, the system intervenes by actuating the user interfaces. In particular, the system uses the touchscreen to show a map of the surroundings with the current position and projected route, a pair of wearable haptic bracelets that vibrate to indicate to the user whether to move left or right, and a headphone to reproduce binaural sound stimuli which express the direction where the suggested trajectory is located.

In this work, we focus on the user audio interface [24]. The auditive interface software provides binaural sound, processed to be interpreted by the listener as originating from points in space along the planned route. The user is therefore encouraged to simply follow the sound lead. The interface software relies on data coming from the *c-Walker* regarding the spatial location of the optimal route, and on information on the user's head orientation captured using an Inertial Measurement Unit (IMU) mounted on the headphone arches, in order to adjust the sound direction in real time.

It is important to emphasize that a person can correctly interpret binaural sound signals only if they are administered with a time delay smaller than 50 milliseconds, otherwise the human localization task may infer the perceived stimuli as associated to sound events that happen later than what intended by the algorithm. This may happen frequently. To give an example, consider that the human ability to recognize the spatial location of sound improves if the listener is able to benefit of small head movements. In fact, moving the head makes it possible to collect more cues regarding an ambiguous signal, such as sounds located within the so called cone of confusion [25]. But, while the listener is moving her/his head, if the synthesized sound is emitted with a delay greater than 50 milliseconds, the sound may be associated with the final position of the head instead of being associated to the position where the head was initially. The interaction of the guidance system with the user, i.e., the emission of stimuli and the corresponding reaction of the user which results in movements, closes the loop between the localization system, the planner, and the user interface. Therefore, the time delay of the auditive interface is crucial to its correct interpretation. Several components affect this delay, from the processing time to the size of the buffers used in the physical components.

A. METROII *Implementation*

The general block diagram of the complete system is shown in Figure 2. Referring to the proposed methodology, we created a cyber model consisting of all the software components that express the functionality, as well as their corresponding architectural tasks. In addition, we have included a *Processor* and the *SoundCard* as architectural components to represent the computing platform. The physical part is needed to describe the physical time and the movements, or rather the variation of the spatial coordinates. The movements of the users are described by the *Person* component. To reflect its interaction with other components, and to map it correctly in METROII, the *Person* component is included in the architectural as well as in the functional model. The functional components are extended to include the description of physical interactions. In case a designer aims at modeling the precise human behavior [26], details regarding sound localization performance and degree of cooperation of the human can be implemented within the *Person* architectural component.

B. Physical Model

Since physical quantities of interest (i.e., time) are handled inside the METROII architectural model, physical model components belong to the architecture. The *audioProc_Task*, the *Imu_Task*, the *Planner_Task*, and the *playback_Task* have a dual role: they represent the architectural component of their functional counterparts, and they receive, modify, and forward physical data. The physical model includes the presence of the user of the *c-Walker* that is represented by the thread implemented within the *Person_Task* component. The presence of this component models the interaction of the user with the binaural guidance system. From one side, the user receives the guidance sound stimuli which convey spatial information encoded as the spatial location of the virtual sounds. On the other hand, the user movements change the position in space of the assistive device, and these variations are captured by the *Planner_Task*. The person can also move his/her head independently, and this is monitored by the *IMU_Task*. The physical model is completed including the interfaces to communicate with the *Processor* and the *SoundCard*.

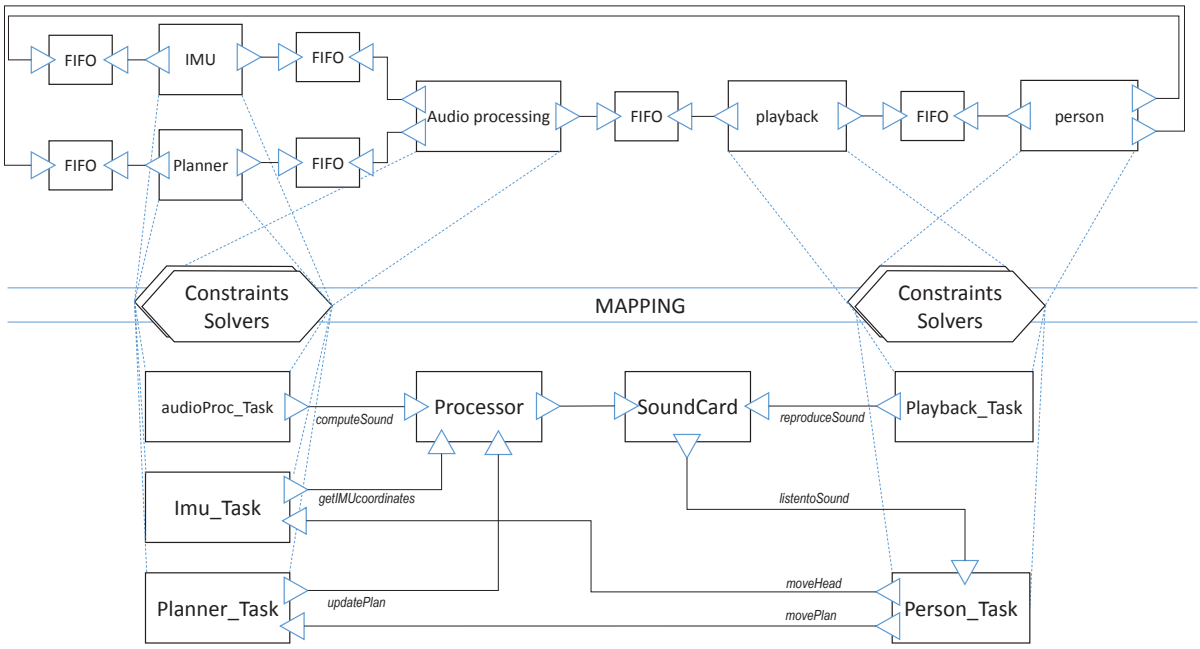

Fig. 2. The METROII model for the binaural guidance system.

From a functional viewpoint, the interface between the *playback* and the person transmits the sequence of samples representing the sound signal. But, from the physical point of view, the interface between the *SoundCard* and the *Person_Task* component models the exchange of spatial coordinates. In order to model the presence of the user, and to model the feedback between the sound reproduction and the subsequent movements of the user, a functional *Person* component is added to the cyber model of the system. Within the *Person* component it is possible to map the *Person_Task* behavior to close the loop of the system output toward the inputs.

C. Cyber Model

The cyber model consists of the software components of the binaural guidance system, and the hardware components that support their execution. In particular, the *Audio Processing* component contains the algorithm that transforms a monoaural audio signal into a binaural stimulus. The computation works on a period buffer which is then fed to the sound card to be reproduced. The size of the period buffer matters, because it influences the latency of the computation. Once a number of samples of audio output equal to the *SoundCard* period buffer size has been produced, they are forwarded to the *playback* component. The *playback* component models the reproduction of the audio signal.

The *Audio Processing* component receives inputs from the *IMU* and the *Planner*. The *IMU* provides data regarding the orientation of the head of the user. The *Planner* computes the optimal trajectory for the user, and by knowing the actual position of the *c-Walker*, provides to the *Audio processing* component the spatial coordinates of the next position to reach, i.e., the spatial location of the virtual sound source the audio processing component has to compute in the subsequent time interval. The *Audio processing* aligns the absolute spatial coordinates according to the user head orientation and starts processing sound. The cyber model is completed including a *Processor* and a *SoundCard*. The *Processor* supports the execution of all the software components, while the *SoundCard* provides the playback of the audio stimuli.

D. Mapped Systems

We have estimated the performance of the system onto two different architectures. The first is a traditional laptop with a 2.53 GHz dual core CPU and 4 GB of RAM, to have a baseline performance model on a relatively high power device. The second platform is BeagleBoard–xM, equipped with a 1 GHz ARM Cortex–A8 core and 512 MB of RAM, which we have considered as representative of a class of small size, low power devices. Because of its small dimension, the BeagleBoard would be preferable if the system were to be engineered to be wearable, rather than being carried on the *c-Walker*.

The mapping of functional onto architectural components is labeled with execution time values, which were measured on the target platforms through profiling. Besides the two platforms, we also mapped four different configurations for the Planner, which differ in terms of the minimum dimensions of the grid used to represent the environment map. This affects both the accuracy of the route determination, as well as the computation time. The communication with the IMU can be configured in two ways: 1) considering only azimuthal angles, or 2) using values of roll, yaw and pitch. The first is associated with the rendering of binaural sounds laying on the horizontal plane only, while the second is associated to the reproduction of full 3D sounds. Their implementations differ, as well as, of course, the performance. The sound processing time depends on the size of the sound buffer, which determines the period of the computation. Longer buffers allow for longer computation time, but introduce latency. Finally, the playback time depends on the semantics of the playback system call: it is considered

to be equal to the duration of the processed sound chunk for a blocking call, or is considered equal to 0 if the playback is called by a non-blocking function (we neglect the call and return time).

E. Results

We analyze the results of the model estimated processing time and simulation time, and determine if the mapped design satisfies the required execution time. Simulations run in a matter of a few seconds. Estimated execution times of a selected set of platforms for the binaural guidance system are reported in Figure 3. In the figure, the BeagleBoard is identified by the string "bb". The planner implementations are labeled from P1 to P4. Planner alternatives are listed in order of increasing accuracy. P1 corresponds to a lower degree of spatial accuracy, but it has the fastest computational time. On the contrary, P4 is slower, but performs the geometric calculation at a higher resolution. Because the planner is computationally intensive, it is always mapped onto the notebook, since the BeagleBoard would not be able to sustain it. Design names starting with 3D refer to the tridimensional sound rendering, as opposed to using the horizontal plane only. The sound buffer size is configured using three different values: 250 ms (labeled as B1), 500 ms (B2) or 1 second long (B4). The "-bp" suffix denotes a blocking playback call.

Figure 3 shows the playback time corresponding to the different buffer sizes using the continuous blue line, while the overall execution time of the different solutions is shown as colored bars, showing the planner execution time in orange, the time required by the IMU in blue, the audio processing time in grey, and the blocking playback time in yellow. Designs in which the execution time is lower than the playback time allow the system to reproduce the guidance sound continuously. If the design required execution time is greater than the playback time, the mapped system can reproduce the guidance signal only with pauses in between two consecutive signaling events longer than the delay time. This introduces glitches in the playback, which should be avoided. Among the alternatives, those with an audio processing execution time lower than 50 milliseconds should preferably be taken into considerations for implementation, since the latency affects the human recognition task the least, as discussed previously. Among these, those with a total execution time shorter than the playback time also allows the system to reproduce the guidance signal without interruption. Among the alternatives, for the system testing with participants, we chose the implementation with the shortest buffer time (B1) associated with the Planner configuration P1. Our final choice for experimentation was to adopt the notebook, because the implementations on the BeagleBoard do not respect the 50 ms deadline for the correct interpretation of the guidance sounds in any of the implementations. A more powerful portable platform should therefore be investigated for a wearable implementation.

V. Thermoelectric Energy Recovery System

In this section, we discuss the design of a microcontroller-based battery-less system for data center CPU monitoring and cooling powered by the electrical energy produced by a thermoelectric generator (TEG). The Thermoelectric Energy Recovery System (TERS) is composed of a TEG, a circuit for energy conditioning and a storage unit. The TERS interacts with two main components: a high performance data center processor, called the *host*, which we wish to monitor; and a *monitoring system*, i.e., a small wireless node that includes a low-power microcontroller and a set of sensors. The thermal energy needed to power the monitoring system comes from the heat dissipated by the high performance data center processor (the *host*). To model the thermal behavior of the *host*, we considered an ARM-based device. This thermal energy is converted into electrical energy by a Thermoelectric Generator (TEG) composed of Peltier cells. The TEG output power goes through a conditioning circuit and is then stored in a supercapacitor. When the harvested energy is sufficient to power the microcontroller and its sensors, the monitoring system wakes up and, according to its configuration, performs one of the two tasks for which it has been developed: either it serves as a wireless sensor node to monitor temperature, humidity, and other environmental parameters; or it works as an active microprocessor cooler by feeding energy to a small cooling fan. The monitoring system mounts a *MSP430* microcontroller. The *MSP* was chosen because it exhibits a very low power consumption profile. Using the harvested energy to cool the data center CPU makes it possible to over clock the *host* processor for short periods of time without overheating it, thus increasing the computing performance. A complete description of the thermoelectric energy recovery system developed to power an energy neutral wireless sensor node can be found in [27], [28], while a description of the behavior of the active cooler for over-clocking can be found in [29].

A. METROII *Implementation*

Implementing the proposed methodology, we first consider the physical models. The mathematical models have been obtained from the analysis of data collected in a dedicated experimental setup and then implemented in C/C++. We implemented the CPU heating function, the formula for the conversion of thermal gradient into electrical energy and the charge/discharge of the supercapacitor. Likewise, we characterized the energy consumption of the MSP. After that, we defined the relation between the tasks that run on the CPU with the generated temperature, and the status of the MSP with its power consumption to define the architecture models. Then, the physical–architectural components were combined with the cyber–architectural components to define the complete design.

Unlike the previous case, where events were logically ordered in time, in this system some state changes of a component are triggered by events that take place in other components, whose duration is not known in advance. This requires a more careful account of the synchronization. In fact, the data center *host* and the monitoring system functionalities are separated, but their activation is related by physical processes. The activity of the monitoring system configured to perform CPU cooling is triggered by the temperature of the ARM CPU, but the ARM CPU and the MSP are not connected from the point of view of the architecture, so there is no direct interface to trigger the event. The situation is similar for the monitoring configuration, which works as long as the recovered energy is sufficient to power the microcontroller and the sensor, independently of the state of the *host*. Events of the physical quantities are not handled by the scheduler, so the corresponding state transitions are triggered through

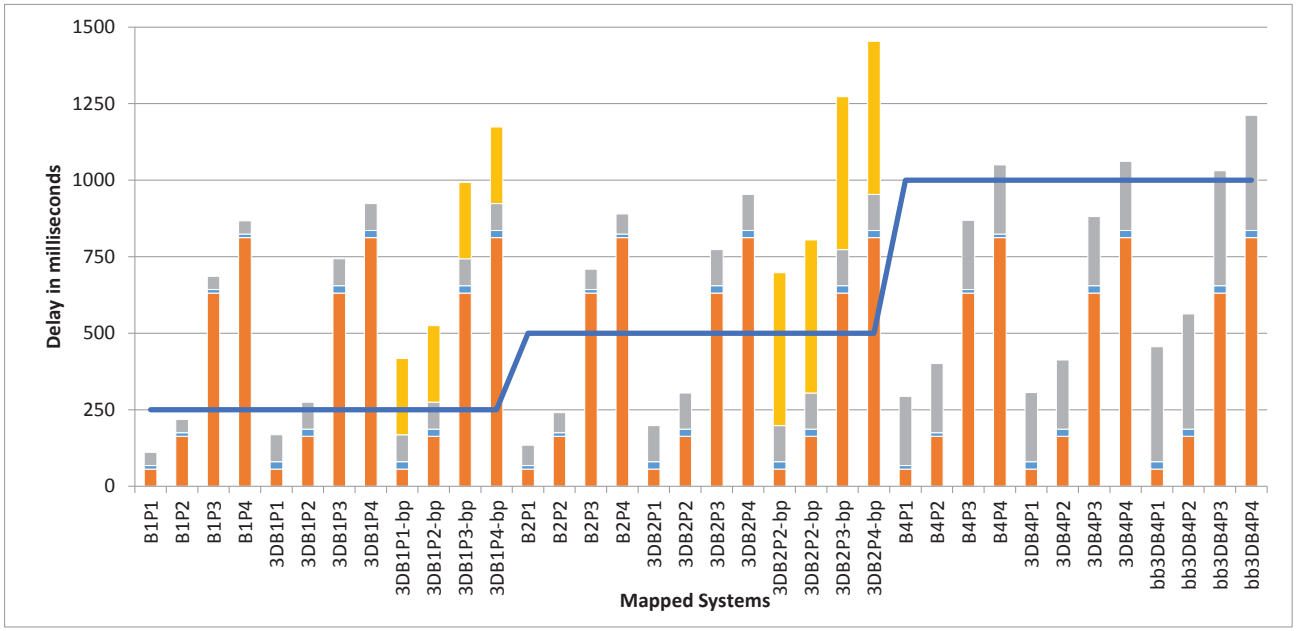

Fig. 3. The binaural guidance estimated execution time. The blue line represents the maximum execution time required to perform continuous playback. On the bars, orange is the planner execution time, blue is the time required by the IMU, grey is the audio processing time, while yellow is the blocking playback time.

direct interfaces which, in this case, relate the two CPUs. A representation of the METROII model is depicted in Figure 4. In METROII, schedulers are used to specify the logical order of begin and end events of functions, and events are used by the mapper to synchronize elements. But for the reasons listed above, not all the events are handled by the scheduler, because the start event of a functionality may be fired when a physical quantity of another component satisfies a given criteria. Moreover, the time annotator is used to specify the timing resolution of the simulation.

B. Physical Model

The physical model of the TERS is composed of the *ARM CPU*, the *TEG*, the *MSP*, and the interfaces between the aforementioned components. From the physical model point of view, the *ARM CPU* is a source of heat; and the *MSP* is a component that consumes the power stored in the *TEG* component. Unlike the cyber model, in the physical model, *ARM CPU* and *MSP* are connected together through the *TEG* component. The physical models of the two CPUs handle the physical processes and interactions (heat exchange and power consumption). The *ARM CPU* component includes the model of thermal dissipation of the processor implemented as theoretical heating and cooling curves. The theoretical heating curve defines the temperature gradient trend over time considering a constant ambient temperature as:

$$\Delta T(t) = \Delta T_{max} \cdot \left(1 - e^{-t/\tau}\right)$$

where ΔT is the thermal gradient between the CPU package and the ambient temperature, and ΔT_{max} is the target temperature difference. τ is the time constant of the system, that is the time required for the system to reach the target temperature ΔT_{max} in ideal conditions (i.e., if there were no heat exchange with the environment). Conventionally, τ corresponds to the time required for the system to reach the value $0.632 \cdot \Delta T_{max}$, and the system is considered to be in steady state after about $4 \div 5$ times τ.

When the CPU starts performing a more demanding task, the simulator computes the current value of the thermal gradient as:

$$\Delta T(t) = \Delta T_{begin} + \Delta T_{target} \cdot \left(1 - e^{-t/\tau}\right)$$

with ΔT_{begin} the value of temperature at the beginning of the new task, ΔT_{target} the steady state temperature of the new task. On the contrary, when the CPU switches from a more demanding task to a less demanding activity, or to the idle state, or when the cooling fan intervenes, the thermal gradient value is updated according to:

$$\Delta T(t) = \Delta T_{target} - (\Delta T_{target} - \Delta T_{begin}) \cdot e^{-t/\tau}$$

The values of the target temperatures were measured on our workbench during the characterization phase of the prototype device; and, observing the transitions between different tasks, we also derived values for τ. The CPU component includes the Euler forward method to resolve the thermal model equations.

The relation between the thermal gradient and the generated output power of the TEG is implemented inside the *TEG* physical/architectural component. This component contains the thermoelectric generator and the supercapacitor physical models. The thermoelectric generator model computes the actual generated power of the TEG as a function of the thermal gradient. The super-capacitor charge and discharge model is included in the *TEG* component. The generated power makes the voltage inside the super-capacitor increase as:

$$V_{sc}(t) = \sqrt{V_{sc}(t-1) + \frac{2V_{generated}(t)}{C_{sc}}}$$

When the *MSP* is powered on, it consumes power according to its state and the task it is executing. Therefore, the *TEG* component diminishes the storage voltage accordingly, with:

$$V_{sc}(t) = \sqrt{V_{sc}(t-1) - \frac{2V_{consumed}(t)}{C_{sc}}}$$

The interface between the *MSP* and the *TEG* components models the energy consumption of the *MSP*. The supplyPower interface is used to trigger state transitions of the *MSP* which works according to the energy at its disposal. For example, the interface is used to switch on the *MSP* when the voltage stored in the *TEG* supercapacitor increases over the threshold of operation of the *MSP*. This kind of state transitions are modeled directly, and not using the METROII scheduler because physical quantities are not visible to the event scheduler.

C. Cyber Model

The cyber model includes software and hardware components of the system. The model of the TERS is composed of the *host server*, the *FIFO* component, the *Host_Task*, the *ARM CPU*, the *Cooling_Task* or the *Sense_Task*, and the *MSP*. In particular, the *host server* component models the functional activity of the server (i.e., jobs/tasks the server executes). It interfaces with a *FIFO* component used to capture the job start and finish events. The *host server* is mapped into the *Host_Task* that associates the current job to a a set of parameters, such as the clock frequency, the load percentage and the job duration. The two CPUs cyber components (ARM and MSP) provide the execution to their respective tasks. The *ARM CPU* provides the execution of the jobs requested by the *Host_Task* with specific parameters. The *MSP* cyber component models the execution of the firmware implemented in the wireless sensor node. The *Cooling_Task* and the *Sense_Task* model two firmware which correspond to two possible activities that the *MSP* can provide. The first contains instructions to control the *ARM CPU* clock frequency, and to supply the cooling fan to implement the active cooling task that supports *ARM CPU* overclocking. The *Sense_Task*, instead, contains the instructions to sample the sensors that is used to implement environmental monitoring with WSN. From the cyber standpoint, the *ARM CPU* and the *MSP* are disconnected, because their tasks are unrelated.

D. Functional and Architectural Models

The resulting METROII functional model includes two components: the *host server* and its corresponding *FIFO*. Both components are part of the cyber system model. The architectural model contains components that are part of both the physical and the cyber model. The *TEG* component is purely physical, and it interacts with the *MSP* and the *ARM CPU* components. The last two contain both the physical and the cyber models. In particular, from the cyber viewpoint, the two CPUs interact with the tasks they can executed which are modeled as threads in the architectural model. Therefore, the architectural model is completed including the *Host_Task* (that executes on the *ARM CPU*), the *Cooling_Task* and the *Sense_Task* (that can execute on the *MSP*).

The *Host_Task* component contains parameters such as the load percentage devoted to the required functional activity, the clock frequency, and the CPU package temperature. The

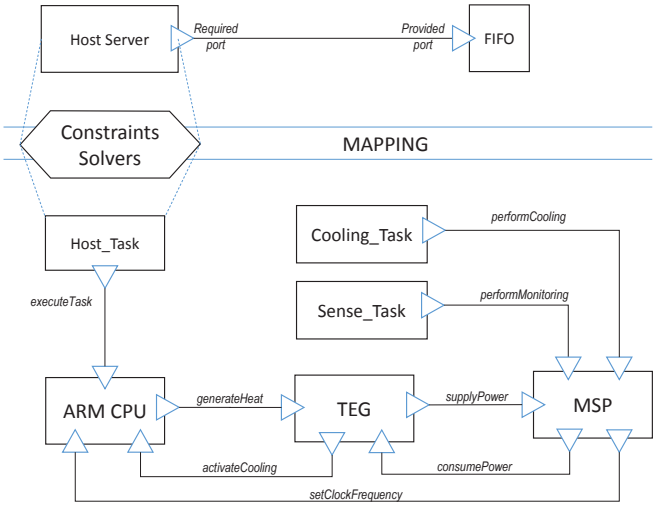

Fig. 4. The METROII model for the thermoelectric energy recovery system applied to the ARM CPU. The figure includes both the *Sense_Task* and the *Cooling_Task* that can be implemented alternatively.

temperature is computed at each iteration as a function of the CPU clock frequency and load according to the values derived from measurement made on the prototype device [27].

The *TEG* components parameters specify the efficiency of the *TEG* (η), the storage capacitance (C_{sc}), the supercapacitor maximum voltage (V_{sc}^{max}), and the present stored energy. The *TEG* component interfaces with both the host *CPU* and the *MSP* components. The *TEG* component receives from the *ARM CPU* component the CPU temperature which is used to determine the present harvested energy, according to the formula that has been derived from measurement made on the prototype device.

The *TEG* interface with the *MSP* receives request of energy, that can be served only in case the stored energy is greater than the voltage threshold that allows the *MSP* to activate. If the *MSP* is active, the *TEG* component receives request of energy, and subtracts from the present storage voltage the amount of energy required according to the task that the *MSP* component is performing. In particular, the *MSP* can be in *Idle mode* when it is active but performing neither the *Sense_Task* nor the *Cooling_Task*. If the *MSP* is performing the *Sense_Task*, it samples the sensors and transmits the data over the wireless channel, and at the same time the *TEG* component decreases the amount of stored energy accordingly.

When a *MSP* is configured to perform the *Cooling_Task*, the iteration among components becomes more articulated because of how the blocks interact with each other in order to implement the over clocking and cooling procedure discussed in [29]. First, given that the fan power consumption to perform cooling is higher than the power required to perform monitoring, when we want to simulate the cooling procedure, the value for the activation threshold of the *TEG* component is set to the corresponding, and greater, value. When the harvested energy is enough to sustain the cooling activity, the *MSP* component sends the command that makes the CPU run with the maximum available load, and at the maximum clock frequency to the corresponding *ARM CPU* component. As a consequence, the remaining time needed to perform the

running job decreases, while the CPU temperature increases. If the temperature were to reach the threshold of 82° C, the dynamic voltage and frequency scaling (DVFS) of the *host* device would intervene to scale down the clock frequency, thus reducing the generated temperature. But in this case, DVFS is not activated because the monitoring device is ready to cool down the CPU package with the following procedure. When the CPU temperature reaches 80° C the *MSP* feeds power to the cooling fan. At that time, the CPU temperature is decreased according to the fan specifications and the energy stored in the super capacitor is subtracted accordingly; at the same time, the *host* CPU continues to run at its maximum clock speed continuing to generate a lot of heat. After a while, the CPU temperature reaches again the 80° C threshold. At that time, if the stored energy is sufficient, the *MSP* can again activate the fan; otherwise, it will set the CPU to a lower clock frequency to prevent overheat. In the latter case, the host time left to complete the job is calculated again in compliance with the clock speed and load.

E. Mapped Systems

We mapped the design onto two different host devices: a 1 GHz ARM Cortex–A9 CPU Pandaboard with 1 GB DDR2 RAM, and a Samsung Arndale equipped with a 1.7 GHz dual core Cortex–A15 with 2 GB DDR3 RAM. We chose these two embedded boards because they are representatives of a class of systems with reasonable computing performance and a low heat dissipation. The simulator considers the thermal model of the corresponding *host* device. The model is written in C within the *host* component of the architectural model. We considered different values for the storage capacitance (C_{sc}), and two different cooling fans. The fan parameters in the simulator include the required voltage of operation, the power consumption, and also their cooling capacity.

F. Results

We analyze the results of the presented model estimating processing time, amount of samples transmitted, number of cooling events, amount of residual energy on the storage supercapacitor, as well as simulation time for different configurations. Figure 5 illustrates the number of cooling events and the number of packets sent during one week of operation of the system using the ArndaleBoard as a *host*, while Figure 6 shows the result obtained conducting the same analysis using the thermal model of the PandaBoard as *host*. We mapped the system using models we gathered from measurements of the devices. For each design, we evaluated the performance of the monitoring system configured to perform either cooling or sensing. We pick the design tm1C1 for deployment because it ensures the MSP executes a large number of cooling and sensing events maintaining a small size for the TEG compared to the solution with more Peltier cells (namely the tm1-4p). The very same TERS design exhibits good performance also applied to the Pandaboard (tm2C1). However, the advantage of having more Peltier cells is more evident on the Pandaboard than in the Arndaleboard (tm2-4p vs. tm1-4p). Having larger storage supercapacitor is more detrimental for the monitoring activity on the Pandaboard than on the ArndaleBoard (see blue markers for tm2C16 compared to tm1C16). In fact, the design tm1C16 makes 71.4% the number of activities of the design with a storage capacitor four time larger (tm1C4). On the contrary, the design tm2C16 sends only 37.5% the number of packets sent by the design on the Pandaboard tm2C4.

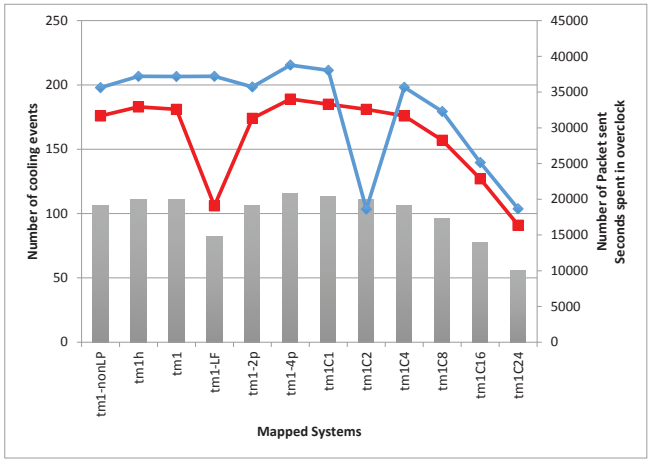

Fig. 5. Average number of cooling (in red) and sensing events (in blue) in a week for different configurations of the monitoring device applied on top of the Arndale CPU. Seconds the CPU spent in overclock (represented by bars).

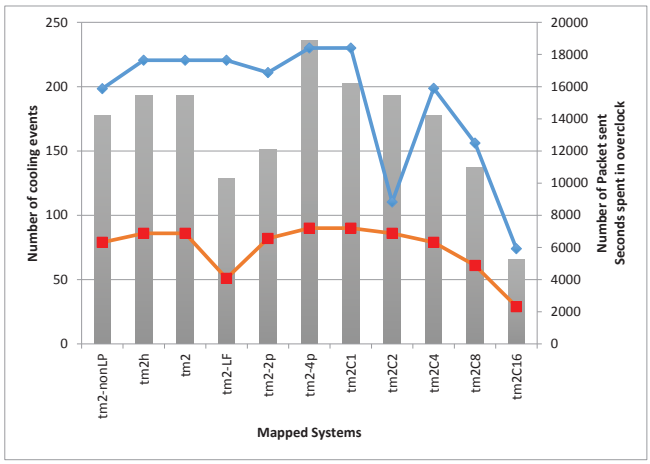

Fig. 6. Average number of cooling and sensing events (in red and blue respectively) in a week for different configurations of the monitoring device applied on top of the Pandaboard CPU. Seconds the CPU spent in overclock (represented by bars).

The simulation of one week of operation of 100 host processors, and their corresponding TEGs, with 90 monitoring devices configured to perform cooling and 10 configured for monitoring, requires about 2 hours of computation time. To simulate 2 monitoring system, one configured to perform the *Sense_Task* and the other configured to perform the *Cooling_Task*, for one week of operation, the simulator takes 105 seconds on average.

VI. Conclusion

We have proposed and discussed a methodology for the design and evaluation of Cyber-Physical System. Our approach consists in extending the traditional function/architecture co-design principle to clearly distinguish the cyber from the physical world. The two are then integrated again through a double mapping step, which brings separate functional and

architectural models together into an overall system implementation. This methodology was applied to two case studies. The case study of the audio guidance system is subject to real–time constraints. The design of the data center monitoring and cooling system is studied because its contribution to the data center energy efficiency is subject to the power availability, and because of the complex interactions that exists between components. In both cases, the METROII framework allows engineers to express the connections between the functional world as formalized by control algorithms as well as the events of the physical world to capture both aspects of the design. The results obtained from the evaluation agree with the implementations, and allowed us to choose the appropriate platform for further experimentation. The same approach can be adopted to express other types of interaction that can happen in a CPS, and to explore various design solution independently from the level of abstraction used for the system description.

ACKNOWLEDGMENT

This work was supported by the FP7 projects DALi grant n. 288917, and GreenDataNet grant n. 609000.

REFERENCES

[1] I. Minakov and R. Passerone, "PASES: An energy-aware design space exploration framework for wireless sensor networks," *Journal of Systems Architecture*, vol. 59, no. 8, pp. 626–642, September 2013.

[2] R. R. Rajkumar, I. Lee, L. Sha, and J. Stankovic, "Cyber-physical systems: the next computing revolution," in *Proceedings of the* 47^{th} *Design Automation Conference*. ACM, 2010, pp. 731–736.

[3] E. A. Lee, "Cyber-physical systems-are computing foundations adequate," in *Position Paper for NSF Workshop On Cyber-Physical Systems: Research Motivation, Techniques and Roadmap*, vol. 2, 2006.

[4] H. Kim, L. Guo, E. Lee, and A. Sangiovanni Vincentelli, "A tool integration approach for architectural exploration of aircraft electric power systems," in *IEEE 1st International Conference on Cyber-physical systems, Networks, and Applications*, ser. CPSNA13, Taipei, Taiwan, August 19–20, 2013.

[5] A. Davare, D. Densmore, L. Guo, R. Passerone, A. L. Sangiovanni-Vincentelli, A. Simalatsar, and Q. Zhu, "METROII: A design environment for cyber-physical systems," *ACM Transactions on Embedded Computing Systems*, vol. 12, no. 1s, pp. 49:1–49:31, March 2013.

[6] A. Pinto, A. Bonivento, A. L. Sangiovanni-Vincentelli, R. Passerone, and M. Sgroi, "System level design paradigms: Platform-based design and communication synthesis," *ACM Transactions on Design Automation of Electronic Systems*, vol. 11, no. 3, pp. 537–563, July 2006.

[7] D. Densmore, R. Passerone, and A. L. Sangiovanni-Vincentelli, "A platform-based taxonomy for ESL design," *IEEE Design and Test of Computers*, vol. 23, no. 5, pp. 359–374, May 2006.

[8] P. Nuzzo, A. L. Sangiovanni-Vincentelli, D. Bresolin, L. Geretti, and T. Villa, "A platform-based design methodology with contracts and related tools for the design of cyber-physical systems," *Proceedings of the IEEE*, vol. 103, no. 11, pp. 2104–2132, 2015.

[9] J. C. Jensen, D. H. Chang, and E. A. Lee, "A model-based design methodology for cyber-physical systems," in 7^{th} *International Wireless Communications and Mobile Computing Conference*, ser. IWCMC. Istanbul, Turkey: IEEE, July 4–8, 2011, pp. 1666–1671.

[10] C. Ptolemaeus, Ed., *System Design, Modeling, and Simulation using Ptolemy II*. Ptolemy.org, 2014. [Online]. Available: http://ptolemy.org/books/Systems

[11] L. Guo, Q. Zhu, P. Nuzzo, R. Passerone, A. L. Sangiovanni-Vincentelli, and E. A. Lee, "Metronomy: a function-architecture co-simulation framework for timing verification of cyber-physical systems," in *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis*, ser. CODES14, New Delhi, India, October 12–17, 2014, pp. 24:1–24:10.

[12] J. Zou, S. Matic, E. A. Lee, T. H. Feng, and P. Derler, "Execution strategies for PTIDES, a programming model for distributed embedded systems," in 15^{th} *IEEE Real-Time and Embedded Technology and Applications Symposium*, ser. RTAS09, San Francisco, CA, United States, April 13–16, 2009, pp. 77–86.

[13] A. Noyer, P. Iyenghar, E. Pulvermueller, J. Engelhardt, F. Pramme, and G. Bikker, "A model-based workflow from specification until validation of timing requirements in embedded software systems," in *Proceedings of the* 10^{th} *IEEE International Symposium on Industrial Embedded Systems*, ser. SIES15, Siegen, Germany, June 8–10, 2015, pp. 1–4.

[14] D. Henriksson and H. Elmqvist, "Cyber-physical systems modeling and simulation with modelica," in *International Modelica Conference, Modelica Association*, vol. 9, 2011.

[15] Qi Zhu, "Optimizing mapping in system level design," Ph.D. dissertation, University of California, Berkeley, 2008.

[16] B. Bonakdarpour, M. Bozga, M. Jaber, J. Quilbeuf, and J. Sifakis, "A framework for automated distributed implementation of component-based models," *Distributed Computing*, vol. 25, no. 5, pp. 383–409, 2012.

[17] V. Sfyrla, G. Tsiligiannis, I. Safaka, M. Bozga, and J. Sifakis, "Compositional translation of Simulink models into synchronous BIP," in *Proceedings of the* 5^{th} *IEEE International Symposium on Industrial Embedded Systems*, Trento, Italy, July 9–11 2010.

[18] P. Derler, E. A. Lee, and A. S. Vincentelli, "Modeling cyber–physical systems," *Proceedings of the IEEE*, vol. 100, no. 1, pp. 13–28, 2012.

[19] P. Joshi, S. K. Shukla, J. P. Talpin Inria, and H. Yu, "Mapping functional behavior onto architectural model in a model driven embedded system design," in *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, ser. SAC15. Salamanca, Spain: ACM, April 13–17, 2015, pp. 1624–1630.

[20] G. Ascia, V. Catania, and M. Palesi, "A GA-based design space exploration framework for parameterized system-on-a-chip platforms," *Evolutionary Computation, IEEE Transactions on*, vol. 8, no. 4, pp. 329–346, 2004.

[21] Y. Yang, A. Pinto, A. Sangiovanni-Vincentelli, and Q. Zhu, "A design flow for building automation and control systems," in *Proceedings of the* 31^{st} *IEEE Real-Time Systems Symposium*, ser. RTSS10, San Diego, CA, US, Nov. 30 – Dec. 3, 2010, pp. 105–116.

[22] T. Blochwitz, M. Otter, M. Arnold, C. Bausch, C. Clauß, H. Elmqvist, A. Junghanns, J. Mauss, M. Monteiro, T. Neidhold *et al.*, "The functional mockup interface for tool independent exchange of simulation models," in *Proceedings of the* 8^{th} *International Modelica Conference*. Linköping University Press, 2011, pp. 105–114.

[23] L. Palopoli et al., "Navigation assistance and guidance of older adults across complex public spaces: the dali approach," *Intelligent Service Robotics*, vol. 8, no. 2, pp. 77–92, 2015.

[24] L. Rizzon and R. Passerone, "Embedded soundscape rendering for the visually impaired," in *Proceedings of the* 8^{th} *IEEE International Symposium on Industrial Embedded Systems*, ser. SIES13, Porto, Portugal, June 19–21, 2013, pp. 101–104.

[25] D. R. Begault, E. M. Wenzel, and M. R. Anderson, "Direct comparison of the impact of head tracking, reverberation, and individualized head-related transfer functions on the spatial perception of a virtual speech source," *Journal of the Audio Engineering Society*, vol. 49, no. 10, pp. 904–916, 2001.

[26] S. Munir, J. A. Stankovic, C.-J. M. Liang, and S. Lin, "Cyber physical system challenges for human-in-the-loop control," in *8th International workshop on feedback computing. San Jose, CA, June 25*, 2013.

[27] M. Rossi, L. Rizzon, M. Fait, R. Passerone, and D. Brunelli, "Energy neutral wireless sensing for server farms monitoring," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 4, no. 3, pp. 324–334, September 2014.

[28] L. Rizzon, M. Rossi, R. Passerone, and D. Brunelli, "Wireless sensor networks for environmental monitoring powered by microprocessors heat dissipation," in *Proceedings of the* 1^{st} *International Workshop on Energy Neutral Sensing Systems*, ser. ENSSys13, Rome, Italy, November 14, 2013, pp. 8:1–8:6.

[29] ——, "Energy neutral hybrid cooling system for high performance processors," in *Proceedings of the* 5^{th} *International Green Computing Conference*, ser. IGCC14, Dallas, TX, November 3–5, 2014, pp. 1–6.