Submitted to CPSNA 2013 - The 1st IEEE International Conference on Cyber-Physical Systems, Networks, and Applications.

A Tool Integration Approach for Architectural Exploration of Aircraft Electric Power Systems

Hokeun Kim, Liangpeng Guo, Edward A. Lee and Alberto Sangiovanni-Vincentelli
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley
{hokeunkim, glp, eal, alberto}@eecs.berkeley.edu

***Abstract*—For emerging safety-critical systems, it is beneficial to cope with design validation, performance estimation, and design space exploration in early design stages. In this paper, we explore the architectural choices of an aircraft electric power system (EPS) controller using Ptolemy II and Metro II. The design is modeled in separate aspects: the functional aspect models the logics and behaviors that fulfill the functionality of the controller, and the architectural aspect models the behaviors of the platform that implements the controller. The co-design benefits from the rigorous Model of Computation (MoC) in Ptolemy II, which facilitates the analysis and validation of functional aspect, as well as the flexibility and expressiveness provided by Metro II, in which complex architectural models can be built with the flexibility of changing the mapping. Co-simulation integrates the functional model and the architectural model using Metro II semantics. By clearly separating the functional aspect and the architectural aspect, the performance can be estimated at an early design stage, and the architectural exploration can be done in a more efficient manner. We show the effectiveness and extensibility of our approach using experiments and results with example candidates for the aircraft EPS controller.**

I. Introduction

Electronic systems are playing a more important role in controlling mechanical and hydraulic systems in various safety-critical applications such as aircraft and vehicles. To design a robust but economical electronic control system, the exploration of design choices in early design stages has become increasingly critical. In contrast to general electronic systems, the timing correctness is a substantial portion of the design criteria in electronic control systems. How to efficiently estimate the timing and explore the design choices in early design stages remains a challenge for designers. In the context of model-based design, complex functionalities and platforms make it more difficult to deal with it manually. We believe new design methodologies are needed to efficiently validate safety and explore implementation choices.

Traditionally, the functional and architectural aspects of the design are coupled together in one model. Designers with domain knowledge and electronic engineers work on the same model. However, due to the complex coupling relationship, changes in one aspect can easily have impact on other aspects. Since timing is part of the correctness in safety-critical systems, the electronic control system is more brittle to the changes than general electronic systems where timing is not as critical. Missing deadlines for certain time-critical tasks can have disastrous effects. Weak management of the different aspects could significantly lengthen the time to market as well as reduce reliability. We believe it is advantageous to separate the functional aspect and the architectural aspect at an early design stage and use mapping to manage the coupling. This facilitates validation and exploration with proper separation of concerns. By carrying out mapping and validation at an earlier stage, the performance can be estimated with different implementation choices, and it is easier to guarantee integrity of the safety-critical systems using the principle of correctness by construction [1]. The proposed modeling approach is useful in the context of current design practice. Since the behaviors of the function often depend on the implementation details, the separation of functional aspect and architectural aspect could help the designers better understand how different implementations affect the functions. In a context where the model of computation supports better specification of real-time constraint in functional model (e.g. PTIDES [2]), our approach is still useful. The performance under different architectures can be efficiently estimated, although the behavior of function model is more deterministic and no longer heavily affected by the implementation details.

We investigate the effectiveness of a mapping and design exploration approach by integrating Metro II [3] and Ptolemy II [4]. In order to assure safety of the system and minimize cost, architectural design space exploration is necessary. Highly complex systems typically consist of components that are heterogeneous in nature. Hence, a system design framework supporting multiple models of computation such as Ptolemy II helps. Metro II, a SystemC-based design environment for platform-based design [5], is suitable for model integration and architecture exploration because it allows the mapping to be easily changed.

In this paper, we propose a novel tool integration approach to develop a design framework that has potential to decrease design costs while enhancing reliability of safety-critical systems. This tool integration approach not only benefits from both Ptolemy II and Metro II to address the problems introduced above, but also reduces costs and complexity for development compared to previous approaches that try to develop a single grand unified framework. This newly introduced design framework provides both functional and architectural views. The main advantages of our approach are threefold: (1) co-simulation supporting multiple models of computation (2) performance prediction on given architectures (3) design

space exploration at an early stage.

II. Approach

A controller for an aircraft electric power system (EPS), a representative application of safety-critical systems, is chosen as an illustration in this paper. The functional model of an EPS controller is implemented using Ptolemy II, and a general architectural model that can be used for the EPS controller is realized as an architectural model using Metro II. Ptolemy II is chosen for the functional model of the system because it supports multiple models of computation including discrete-events (DE), synchronous dataflow (SDF) and synchronous/reactive(SR) with specific directors for each model of computation [4] [6], and thus it is appropriate for describing complex and heterogeneous components in safety-critical systems. Metro II is a SystemC-based design framework. It supports useful SystemC infrastructures for modeling concurrent behaviors [7] and provides the notion of mapping. These two models are connected to each other by mapping events of both models on a co-simulation platform using Metro II semantics, which synchronizes the events from different models.

The co-simulation platform takes two inputs. The first input is a Ptolemy II functional model with Metro directors, which are extensions of Ptolemy II directors. In addition to their roles of scheduling actors in Ptolemy II, Metro directors associate the firings of actors with events. The second input is a SystemC architectural model, which also generates concurrent events. By synchronizing the events occurring in both models, the platform can simulate them simultaneously. From the results of the co-simulation, we can evaluate the performance of the functional model when mapping to a particular architecture and explore the different architectural design choices.

Using a given specification and constraints of the aircraft EPS controller, a functional model of an EPS controller is created in Ptolemy II. The EPS controller monitors the health status of components in an aircraft's EPS and generates control signals to maintain power for all critical AC loads, while satisfying safety constraints for the power system.

An architectural model is implemented in SystemC. The architectural model includes a scheduler to arrange tasks fired in the functional model. The architectural model has the design details that reflect different implementations such as sequential or parallel executions.

III. Related Work

Architecture exploration of embedded systems has been studied previously [8] [9]. Our new approach is similar to these previous approaches in the sense that the goal is to find the most appropriate architecture for the given functional specification. However, most of the previous work targets soft real-time systems such as multimedia applications, in which the timing is a measure of the performance but not a measure of the correctness. In safety-critical applications, we believe new approaches are needed to address the fault tolerance and real-time constraints. Thus, we focus more on design exploration for meeting safety constraints than on optimization in this paper.

Some existing hardware/software co-simulation frameworks [10] have similar functions on different levels of abstraction. Synopsis Platform Architect targets at the transaction level. Other approaches, including Mentor Graphics Seamless, are designed for register transfer level abstractions, and are usually used for verification at a final stage. Researchers in embedded system design have been working on more effective and accurate hardware/software co-simulation approaches. A C/C++ based method for describing both hardware and software [11] is employed for fast co-simulation. There is also previous research on fast and accurate co-simulation using FPGA-configured soft processors [12]. Hoffmann et al. [13] develop a co-simulation framework for supporting multiple layers of abstractions. A co-simulation approach using automated software annotation [14] increases accuracy of simulation while maintaining the fast speed of behavioral simulation.

In our method, instead of separating the system by software/hardware, we decouple the modeling of functional aspects and architectural aspects. By integrating Ptolemy II and Metro II, the integrated model allows the two aspects to evolve independently. As research on safety-critical systems advances, new models of computation and novel simulation techniques for those systems will keep being developed. With our approach, applying new models and techniques for co-simulation becomes easier and quicker. In contrast, applying new models and techniques will take more effort and time if we use a single grand unified framework.

IV. Functional Model

A. Specification

An aircraft EPS [15] consists of various components such as generators, buses, contactors, and external power ports. A power failure in the aircraft can be disastrous, so it is critical to make sure the important AC and DC loads of the aircraft are always powered by at least one power source, even in the case when some generators fail. The aircraft EPS example used in this paper is shown in Fig. 1. There are four AC power sources and six contactors. Two AC loads that need power supplies are located at the opposite ends of this system. All the components are connected by power buses denoted by solid lines. The contactor is an electrically controlled switch used for switching a power circuit. With proper control signals, the contactors can dynamically change the topology of the power network. The EPS controller should produce control signals on contactors guaranteeing that both left and right AC loads are always powered. In addition, at most one of the generators should be connected on the same power bus at all times to prohibit two generators connected to each other. Meanwhile, the two AC loads are assumed to have capacitors, thus, they are allowed to be unpowered briefly within a well-defined time bound during the switching of contactors.

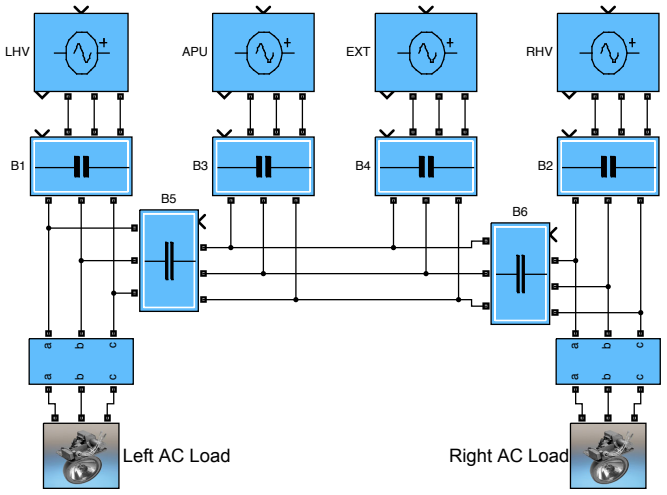

Fig. 1. A given specification of the functional model of aircraft EPS (A Simulink diagram created by P. Nuzzo)

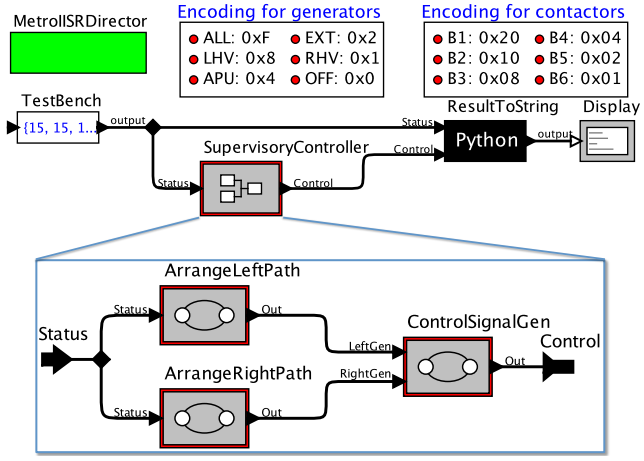

Fig. 2. Ptolemy II diagram of aircraft EPS supervisory controller functional model

B. Ptolemy II implementation

The supervisory controller is modeled using an extended version of Ptolemy II with Metro II directors and actors. The functional model here is implemented with a Metro II synchronous reactive (SR) director, which implements a model of computation where every reaction is instantaneous and simultaneous [16]. The block diagram of the functional model is shown in Fig. 2.

The supervisory controller of this functional model is a composite actor composed of three tasks. Each task is a state machine. Sensors monitor the health status of generators and contactors. The health status is used to indicate availability of generators. And the controller reads the signals from sensors as the input and produces control signals. The control signals indicate whether each contactor should connect its input and output in order to guarantee power supply for both AC loads as well as safety constraints imposed by specification.

In the model, the health status and control signals are represented in a binary format. After parsing binary encoded inputs, state machines realize the supervisory controller. Our Ptolemy II functional model emulates the binary operation using integers as inputs and outputs of the controller.

C. Tasks

The supervisory controller in the functional model is composed of three tasks as mentioned above. The three tasks are Metro II modal models that are similar to modal models in the original Ptolemy II [17] except that they associate the firing of the finite state machine with events. The modal models in the original Ptolemy II are finite state machines with states that can have refinements for representing hierarchical Ptolemy II models. The tasks are:

1) ArrangeLeftPath and ArrangeRightPath: These two tasks are designed as state machines that react for every health status input. As their names suggest, the ArrangeLeftPath (ALP) task chooses the power supply for the left AC load while the ArrangeRightPath (ARP) task undertakes the same computation for the right AC load. The output of these state machines contains which generator and path are used for each AC load.

2) ControlSignalGen: ControlSignalGen (CSG) is another task which receives the generator selection results from the two paths arranging tasks ALP and ARP, and generates control signals for contactors. The output of this task contains control signals for each contactor, which indicate how the contactors are switched. The state machine for this CSG task is also a Metro II modal model that always reacts in one cycle.

V. Architectural Model

A. Overview

The main role of the architectural model is to run synchronously with the functional model and simulate the behavior of the functional model on a specific architecture. Each architectural model includes implementation details and a set of parameters that are pertinent to the performance of the system. The architectural model is implemented using multiple SystemC processes. The architectural model consists of two types of processes, task processes and a scheduler process. The task process models the execution of the task in the functional model. And the scheduler process models the task management in the run-time system.

The functional model and the architectural model are running in different simulation processes. To communicate with the functional model, the architectural model is connected to the functional model through a named pipe, which is one of the inter-process communication methods. They communicate with each other using Metro II events by reading and writing with blocking on the pipes to synchronize their events. Task actors in the functional model send Metro II events through the pipes as requests for execution. Then the functional model tries to get from the pipes the notifications of the events as the approvals of the requests. Similarly, the task processes in the architectural model send events through the pipes as

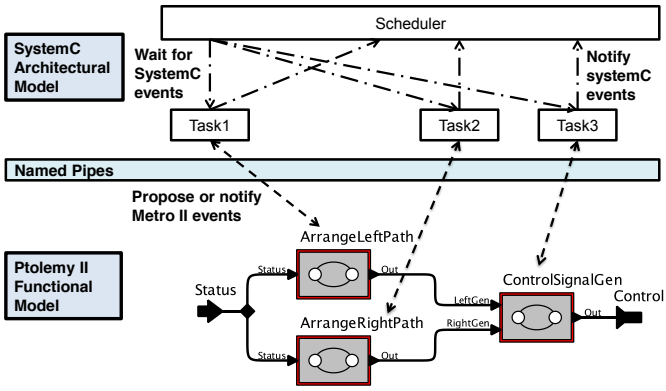

Fig. 3. Mapping and co-simulation process of the architectural model combined with the functional model

the requests for execution. Then the architectural model tries to get the notifications of the events as the approvals of the requests. The events are notified only when the mapped events appear in pairs and thus we make sure the mapped events are synchronized in the co-simulation. When the Metro II events are notified, the task actors in the functional model are fired, and the task processes are resumed. In other words, the task actors in the functional model are not allowed to fire until the corresponding task is scheduled by the architectural model. The scheduling of tasks in architectural model is determined by the priorities assigned through parameters. This mapping and communication between two models is depicted in Fig. 3.

B. Co-simulation

The process of co-simulation follows the Metro II execution semantics [18], which includes three phases: base model phase, quantity annotation phase, and constraint solving phase (Fig. 4). First, the Metro II semantics starts from the base model phase and lets the functional model and architectural model send events. And in the quantity annotation phase, the events in the architectural model are annotated with the timing information, which is computed from performance parameters of the architectures. In the constraint solving phase, the mapping of events is handled. The mapping of two events is specified as a rendezvous constraint. A rendezvous constraint is satisfied when the specified event pair appears in the constraint solving phase. An event is notified when it satisfies all the constraints. Otherwise the event has to wait. The global time is determined according to time stamps of the notified events.

C. Design choices

With Metro II semantics, it becomes possible to simulate the mapped model where the execution of tasks in the functional model reflects the implementation details in the mapped architecture.

In reality, different scheduling policies are available when implementing the same function, such as round-robin scheduling, rate monotonic scheduling, or earliest deadline first

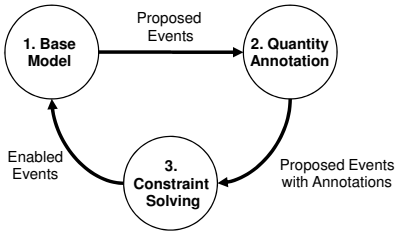

Fig. 4. Metro II: Three phase execution [18]

scheduling. Using different scheduling policies leads to systems having different performance and overhead. Sometimes the scheduling overhead can be negligible; however, it can be a critical factor determining timing behavior of time-critical systems. Priorities of tasks are also an important criterion for deciding the performance of a system. Execution order of the tasks can affect contents of cache and memory; thus, this can influence cache hit and miss rates for data access. When there is more than one task ready to run, it will choose one of them based on the given priorities. Currently, this scheduling supports only fixed-priority non-preemptive scheduling; however, it can be extended to provide dynamic scheduling by simply allowing changing the priority parameter of the scheduler at runtime. Preemptive scheduling can also be implemented by interleaving task executions when scheduling the tasks.

Execution times of tasks on a given architecture are definitely one of the most crucial elements that can affect performance of the system. In this co-simulation platform, it is assumed that the worst case execution times (WCET) of tasks can be measured on a given architecture. Whenever the scheduler chooses a certain task to execute, after being notified, the clock of the architectural model is increased by the WCET of the task. And the timing of the corresponding task in the functional model can then be computed.

Moreover, in parallel implementations, the synchronization overhead caused by parallelization should also be considered as part of critical parameters that determine overall performance of the system. In general, we can map tasks into a single processing element or multiple processing elements; the network architecture can be an on-chip network, a shared memory, the Ethernet network, or even wireless networks. In any case, there must be some communication overhead for synchronization of concurrent tasks. Therefore, the scheduler parameter also includes parallelization style and its synchronization overhead to reflect this real hardware network architecture.

VI. Experiments and Results

A. Performance prediction and design space exploration

In this section, we perform a design space exploration of a simple example by comparing design candidates characterized by parameters defined in the architectural model.

We compare three possible architecture alternatives with various parameters in task execution times, parallelization and

TABLE I
PARAMETERS OF THREE POSSIBLE EXAMPLE CANDIDATES FOR THE ARCHITECTURAL MODEL OF THE AIRCRAFT EPS CONTROLLER

Parameters		Candidate number		
		1	2	3
Scheduling overhead (ns)		10	10	10
Execution Time (ns)	ALP	40	65	50
	ARP	45	70	55
	CSG	20	40	30
Synchronization overhead (ns)		0	5	15
Parallelization of ALP and ARP		No	Yes	Yes

TABLE II
TOTAL EXECUTION TIME OF THE AIRCRAFT EPS CONTROLLER FUNCTIONAL MODEL ON GIVEN ARCHITECTURAL CANDIDATES

Candidate	Total execution time (ns)
1	1150
2	1250
3	1100

synchronization overhead. The first candidate has a single processor with high computation speed to execute three tasks in the functional model. The second and the third candidates both consist of two processors that can execute ALP task and ARP task concurrently. They differ in the sense that the second candidate has processors with slow speed and low synchronization overhead while the third candidate has medium speed processors with relatively high synchronization overhead. Other parameters such as scheduling overhead and task priorities are assumed to be same so that we can focus on the effect on performance of processing elements and the manner of parallelization with synchronization overhead. The parameters represented in numbers for these three candidates are summarized in Table I.

The results in total execution time of these tasks from the co-simulation of the functional and architectural designs using the parameters in Table I are shown in Table II. These results describe total execution times of ten iterations of the functional model for a given test bench. As stated in the section IV-C, the main tasks, ALP, ARP, and CSG are designed to react in one cycle no matter which input they accept. Therefore, the input pattern of the test bench does not necessarily affect the firing behavior of the functional model in this case, so we can replace the test bench with any test bench that can last for at least ten iterations of the scheduler.

The results in Table II reveal that there is a difference in total execution time among three candidates. This information can be used to choose one candidate as the functional model's architectural model. If the requirements of the aircraft EPS controller specify that the total execution time for ten iterations should not exceed 1200 nanoseconds, we should rule out the second candidate, which took 1250 nanoseconds for ten iterations. The first candidate can be selected over the third candidate if the cost of first candidate is less than the third candidate and other conditions are the same.

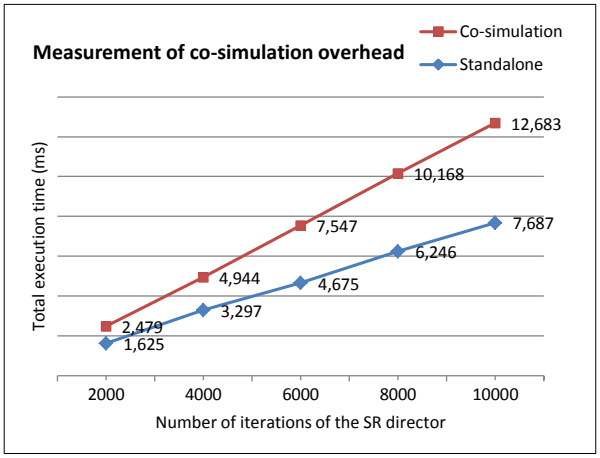

Fig. 5. Execution times of the standalone Ptolemy II functional model and co-simulation environment

B. Measuring co-simulation overhead

This co-simulation environment employs SystemC to represent the architectural model and uses named pipes for communication between the functional and architectural models as stated in the section V-A. Using SystemC and the pipes incur overhead on the Ptolemy II model, resulting in an increase of total execution time of the simulation compared to the standalone execution of Ptolemy II model. Hence, it is worth measuring the overhead imposed on Ptolemy II to assess scalability of this co-simulation environment. To measure co-simulation overhead of this environment, we compare the total execution time between the standalone version and the co-simulation version as a function of number of iterations of synchronous-reactive (SR) director.

Results of the measurement of co-simulation overhead are displayed in milliseconds in Fig. 5. This experiment is carried out in a 64-bit Ubuntu Linux 12.04 LTS with the kernel version of 3.2.0 on a machine with 2.3 GHz Intel Core i7 and 8 GB 1600 MHz DDR3 RAM. Total execution times in Fig. 5 are obtained by averaging total execution times for ten executions of each setting. 2000, 4000, 6000, 8000, and 10000 iterations of the SR director are used as experimental settings for both co-simulation version and standalone version of the Ptolemy II model in order to show a trend of execution times according to the increase of the number of iterations.

As illustrated in Fig. 5, total execution times of both the co-simulation version and the standalone version tend to increase linearly as the number of iterations grows. The results of the co-simulation version indicate it has approximately 1.58 times greater execution times compared to the standalone version. These facts suggest that this newly proposed approach has potential for extensibility and scalability.

VII. CONCLUSION

In this paper, a co-simulation environment is created using Metro II combined with Ptolemy II and SystemC for an

Aircraft EPS that supports performance prediction and comparison of architectural design candidates for design space exploration. The implementation details in the functional model and the architectural model are explained throughout this paper using block diagrams and tables.

Experiments and following results on the implementation of this co-simulation environment show effectiveness, usability and considerable potential to extend this project for more complex safety and time-critical systems with possible design candidates.

This work can be extended to cover other complex safety-critical systems with a variety of components and to support more parameters for architectural design space exploration. Possible future work would be generalizing this co-simulation environment to be applicable to other kinds of safety-critical systems and considering more architectural parameters such as network topology, memory access overheads and external I/O operation overheads.

ACKNOWLEDGMENT

This work was supported in part by the iCyPhy Research Center (Industrial Cyber-Physical Systems, supported by IBM and United Technologies), and the Center for Hybrid and Embedded Software Systems (CHESS) at UC Berkeley (supported by the National Science Foundation, NSF awards #0720882 (CSR-EHS: PRET), and #1035672 (CPS: Medium: Ptides), and #0931843 (ActionWebs), the Naval Research Laboratory (NRL #N0013-12-1-G015), and the following companies: Bosch, National Instruments, and Toyota).

REFERENCES

[1] M. Bordin and T. Vardanega, "Correctness by construction for high-integrity real-time systems: A metamodel-driven approach," *Reliable Software Technologies–Ada Europe 2007*, pp. 114–127, 2007.

[2] Y. Zhao, J. Liu, and E. A. Lee, "A programming model for time-synchronized distributed real-time systems," in *Real Time and Embedded Technology and Applications Symposium, 2007. RTAS'07. 13th IEEE*. IEEE, 2007, pp. 259–268.

[3] A. Davare, D. Densmore, L. Guo, R. Passerone, A. L. Sangiovanni-Vincentelli, A. Simalatsar, and Q. Zhu, "metro II: A design environment for cyber-physical systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 12, no. 1s, p. 49, 2013.

[4] J. Eker, J. W. Janneck, E. A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong, "Taming heterogeneity-the Ptolemy approach," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 127–144, 2003.

[5] A. Sangiovanni-Vincentelli and G. Martin, "Platform-based design and software design methodology for embedded systems," *Design & Test of Computers, IEEE*, vol. 18, no. 6, pp. 23–33, 2001.

[6] C. Brooks, E. Lee, and S. Tripakis, "Exploring models of computation with Ptolemy II," in *Hardware/Software Codesign and System Synthesis (CODES+ ISSS), 2010 IEEE/ACM/IFIP International Conference on*. IEEE, 2010, pp. 331–332.

[7] T. Grötker, S. Liao, G. Martin, and S. Swan, *System design with SystemC*. Springer, 2002.

[8] A. Kahng, B. Li, L. Peh, and K. Samadi, "ORION 2.0: A fast and accurate NoC power and area model for early-stage design space exploration," in *Proceedings of the conference on Design, Automation and Test in Europe*. European Design and Automation Association, 2009, pp. 423–428.

[9] B. Mei, A. Lambrechts, J. Mignolet, D. Verkest, and R. Lauwereins, "Architecture exploration for a reconfigurable architecture template," *Design & Test of Computers, IEEE*, vol. 22, no. 2, pp. 90–101, 2005.

[10] J. Rowson, "Hardware/software co-simulation," in *Design Automation, 1994. 31st Conference on*. IEEE, 1994, pp. 439–440.

[11] L. Séméria and A. Ghosh, "Methodology for hardware/software co-verification in C/C++ (short paper)," in *Proceedings of the 2000 Asia and South Pacific Design Automation Conference*. ACM, 2000, pp. 405–408.

[12] J. Ou and V. Prasanna, "MATLAB/Simulink based hardware/software co-simulation for designing using fpga configured soft processors," in *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*. IEEE, 2005, pp. 148b–148b.

[13] A. Hoffman, T. Kogel, and H. Meyr, "A framework for fast hardware-software co-simulation," in *Proceedings of the conference on Design, automation and test in Europe*. IEEE Press, 2001, pp. 760–765.

[14] A. Bouchhima, P. Gerin, and F. Pétrot, "Automatic instrumentation of embedded software for high level hardware/software co-simulation," in *Design Automation Conference, 2009. ASP-DAC 2009. Asia and South Pacific*. IEEE, 2009, pp. 546–551.

[15] K. Emadi and M. Ehsani, "Aircraft power systems: technology, state of the art, and future trends," *Aerospace and Electronic Systems Magazine, IEEE*, vol. 15, no. 1, pp. 28–32, 2000.

[16] S. A. Edwards and E. A. Lee, "The semantics and execution of a synchronous block-diagram language," *Science of Computer Programming*, vol. 48, no. 1, pp. 21–42, 2003.

[17] E. A. Lee and S. Tripakis, "Modal models in Ptolemy," in *3rd International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools (EOOLT)*, vol. 47. Oslo, Norway: Linköping University Electronic Press, Linköping University, 2010, pp. 11–21. [Online]. Available: http://chess.eecs.berkeley.edu/pubs/700.html

[18] D. Densmore, T. Meyerowitz, A. Davare, Q. Zhu, and G. Yang, "Metro II execution semantics for mapping," Technical Report UCB/EECS-2008-16, University of California, Berkeley, Tech. Rep., 2008.