Tools

Giacomo Barbieri*, Patricia Derler, David M. Auslander, Roberto Borsari, and Cesare Fantuzzi

Design of mechatronic systems through aspect and object-oriented modeling

Entwurf mechatronischer Systeme mittels Aspekt- und Objektorientierter Modellierung

DOI 10.1515/auto-2015-0071
Received October 15, 2015; accepted January 22, 2016

Abstract: Design of mechatronic systems involves the use of multiple disciplines, from mechanics to electronics and computer science. Different granularities of hybrid co-simulations with increasing details can be used during the design process. However, there is the need of modeling tools for effectively managing the necessary abstraction layers. This work proposes a combination of Aspect-Oriented and Object-Oriented modeling for reaching the goal. Moreover, it shows how the utilization of these tools can facilitate design-space exploration, segregation of domains of expertise and enhances co-design.

Keywords: Mechatronic systems, design life-cycle, hybrid co-simulation, aspect-orientated modeling, object-orientated modeling.

Zusammenfassung: Der Entwurf mechatronischer Systeme involviert mehrere Disziplinen, von der Mechanik über die Elektronik bis zur Informatik. Während des Entwurfsprozesses können verschiedene Granularitätsebenen hybrider Co-Simulationsmodelle mit zunehmendem Detailierungsgrad genutzt werden. In jedem Fall werden Werkzeuge benötigt, welche die verschiedenen Abstraktionsebenen effizient managen. Der vorliegende Beitrag schlägt die Kombination von Aspekt- und Objektorientierter Modellierung vor, um dieses Ziel zu erreichen. Der Beitrag zeigt auf, wie die Nutzung dieser Werkzeuge die Untersuchung möglicher Designvarianten, die Trennung der Expertendomänen und damit ein verbessertes Design ermöglichen.

Schlüsselwörter: Mechatronisches System, Entwicklungslebenszyklus, hybride Co-Simulation, Aspekt-orientierte Modellierung, Objekt-Orientierte Modellierung.

1 Introduction

Mechatronic systems (MTSs) are physical applications controlled through software-based (e. g. microcontrollers etc.) and/or digital logic controllers (e. g. FPGA etc.) [11]. The design of MTSs can be considered as the integration of the following domain specific designs [3]:

1. *mechanical*: design of the physical application;
2. *control*: design of the control system and selection of controllers in which control system will be deployed;
3. *interfaces*: design of the interface between physical application and controllers. The interface consists of sensors, actuators, communication means and protocols, converters etc.

In current practice, mechanical designs, control designs and their interfaces are rarely simulated together. Instead, physical prototypes are preferred. A significant reason for that is the complexity of modeling software deployments. In fact, control system is usually deployed in software-based controllers; these controllers can be seen as a hierarchy of physical and logical layers. In general, they consist of hardware architecture, Real-Time Operating System and application environment. Each logical layer has a scheduling strategy for computing processes and a memory management policy. Moreover, hardware architectures use computation means (e. g. pipelines etc.) and memories (e. g. cache, registers etc.) designed to optimize average-case rather than worst-case performance [9]. This makes

***Corresponding author: Giacomo Barbieri,** University of Modena and Reggio Emilia, Via Amendola 2, 42122 Reggio Emilia, Italy, e-mail: giacomo.barbieri@unimore.it
Cesare Fantuzzi: University of Modena and Reggio Emilia, Via Amendola 2, 42122 Reggio Emilia, Italy
Patricia Derler: National Instruments, 2168 Shattuck Ave., 94704 Berkeley, USA
David M. Auslander: UC Berkeley, 5120 Etcheverry Hall, 94720 Berkeley, USA
Roberto Borsari: Tetra Pak Packaging Solutions SpA, Via Delfini 1, 41123 Modena, Italy

controllers highly not deterministic and difficult to simulate.

While it may be possible to implement a system that satisfies the requirements just relying on physical prototypes, there is no guarantee that the system is optimal with respect to performance, since interactions among disciplines are not considered. Therefore, our proposal is to use simulation for deriving necessary performances of non deterministic elements (i. e. controllers and communication networks) which assure an "optimal" solution, and then verify their fulfillment through measurements on prototypes [21]. Modeling tools and methodologies are necessary for reaching this task. This paper identifies modeling tools, while the definition of methodologies will be matter of future work.

Inter-disciplinary influences and overall system behavior can be modeled through co-simulation [5]. Here, Domain Specific Models of each disciplines are merged and integrated. In particular, design of MTSs requires *hybrid co-simulation*: continuous dynamics combined with discrete mode changes and discrete events [15]. Then, functional behaviors can be implemented in this platform and integrated with their non-functional behaviors. It is common to use the term *functional behavior* for a model that describes the core intended behavior of a system. For example, a functional model of a DC motor may describe the feedback control law and the physical dynamics of the motor. So called *non-functional behaviors* might include properties of an implementation such as control law deployment in a software-based controller.

Different abstractions of hybrid co-simulations can be performed during the design of MTSs for modeling non-functional behaviors and ideally deriving specifications of the elements necessary for deploying system functionalities. When non-functional behaviors are implemented, models grow in two directions:

- *Vertical*: mathematical models refine including more details. For example, an incremental encoder can be modeled as a velocity integrator in a functional model, while as two square wave generators in a non-functional model.
- *Horizontal*: orthogonal information must be annotated to actors for modeling non-functional behaviors and for describing alternatives during design-space explorations. For example, an actor which implements a process may be characterized with its execution time (non-functional behavior). Two hardware architectures may be evaluated for deploying the process. Each alternative will result in a certain execution time for the process on the basis of the hardware performances, scheduling strategy and CPU load.

Therefore, models which implement non-functional behaviors can become awkward increasing time necessary for the modeling activity; as shown in [2]. Modeling tools are necessary for keeping models simple also when non-functional behaviors are implemented. This work proposes a method based on Aspect-Orientated (AO) and Object-Orientated (OO) modeling.

The paper is organized as follows: related works are described in Section 2. Section 3 illustrates AO and OO modeling, while Section 4 models typical elements of MTSs through the defined tools. Section 5 applies the approach to a case study. Conclusion and future work are discussed in Section 6.

2 Related work

In order to integrate the disciplines involved in the design of MTSs, hybrid co-simulations must be performed. Different simulation environments can be used for hybrid co-simulation.

In the Functional Mock-up Interface (FMI) standard [17], a model is exported as a Functional Mock-up Unit (FMU) and instantiated in a host simulation environment. FMI provides two mechanisms for interaction of FMUs:

- *Model exchange*: every FMU contains the mathematical model of the referenced model, while host simulator is responsible for computation and communication of FMUs.
- *Co-simulation*: FMUs independently compute their mathematical model. Host simulator implements coordinator algorithms (master algorithms) for managing communication among FMUs.

Currently, both approaches have limitations. In model exchange, FMUs and host simulator authors have to agree on the semantics. Such an agreement has been shown to rarely exist, for example, due to failure of the Hybrid Systems Interchange Format [18]. Since an FMU for co-simulation includes its own simulation engine, there is no requirement for matching semantics of the host simulator. However, host simulator master algorithms are even today based on constant communication step size strongly restricting the efficiency of FMU solvers. Various research is dedicated to finding master algorithms for deterministic and efficient hybrid FMI co-simulations [4, 20].

Moreover, in the actual form of the standard, it is impossible to model cross-cutting concerns. In fact, every FMU is an "atomic" element accessible through methods defined from the standard. Cross-cutting concerns would

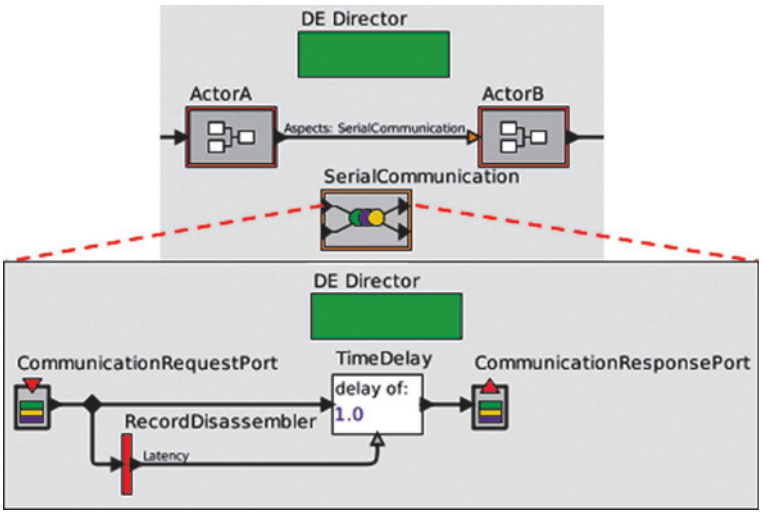

Figure 1: Communication aspect for modeling latency due to serial communication.

require additional methods for constraining the FMU execution (as shown in Section 3).

In order to avoid issues related to interpretation semantics, the solution may be to use simulation tools which natively support hybrid co-simulation. MATLAB, Labview, Modelica and other tools which constitute the "de-facto" standard for MTSs may be used. Moreover, frameworks for modeling in Simulink the effects of software deployment as schedulers, processes, and communication means and protocols are proposed in [7] and [12]. However, in their current version, these tools do not allow AO modeling and just Modelica supports OO modeling. We consider AO and OO fundamental tools for the design of MTSs as will be illustrated in next sections. Therefore, Ptolemy II [19] is utilized in this paper which allows hybrid co-simulation, and both AO and OO modeling.

3 AO and OO modeling in Ptolemy II

Ptolemy II modeling approach is based on actors: concurrent components that communicate by exchanging messages. An actor has a set of input ports, output ports and state. In contrast with other actor-oriented approaches [1, 13], a director/Model of Computation (MoC), rather than the individual actors themselves, defines details of scheduling and communication. This is possible because every actor implements a standard executable interface: an interface constituted of methods which are invoked by the director. The computation of an actor is performed through the calling of its `fire` method. Typically this method involves reading inputs, processing data and producing outputs.

A model can be annotated with additional information that is orthogonal to the model. An example would be the evaluation of the cost for implementing a certain system. By annotating each model element with a price, the cost of the entire system can be statically computed. AO modeling enables the annotation of actors with information that is evaluated dynamically. Ptolemy II provides two types of aspects [2]:

- *Communication aspect*: wrap the communication between actors by intercepting token transmission. The communication aspect operates on the token from the sending actor (e. g. delays, modifies, drops the token) before the receiving actor gets the token.
- *Execution aspect*: wrap the execution of an actor; i. e. the aspect behavior is executed before the `fire` method is called.

An example of a communication aspect is shown in Figure 1. This aspect models the latency introduced by a serial communication. The communication between `ActorA` and `ActorB` is annotated with the aspect and configured with parameters describing the latency on that connection. The link will be annotated with a textual parameter that binds the communication link to the `SerialCommunication` aspect.

This aspect may also be associated to other connections in the model. It assumes that a physical wire is utilized for each connection. If networks must be modeled, *composite aspects* which implement arbitrary models of protocols and network elements should be utilized.

OO implementation in Ptolemy II is described in [14]. An example is shown in Figure 2. The class definition icon is outlined in light blue to distinguish it visually from an instance. `ClassCounter` adds `CounterParam` value to a `Count` variable every time a trigger signal is received. Then, a subclass is derived from `ClassCounter`. The subclass inherits actors, ports, and parameters from the base

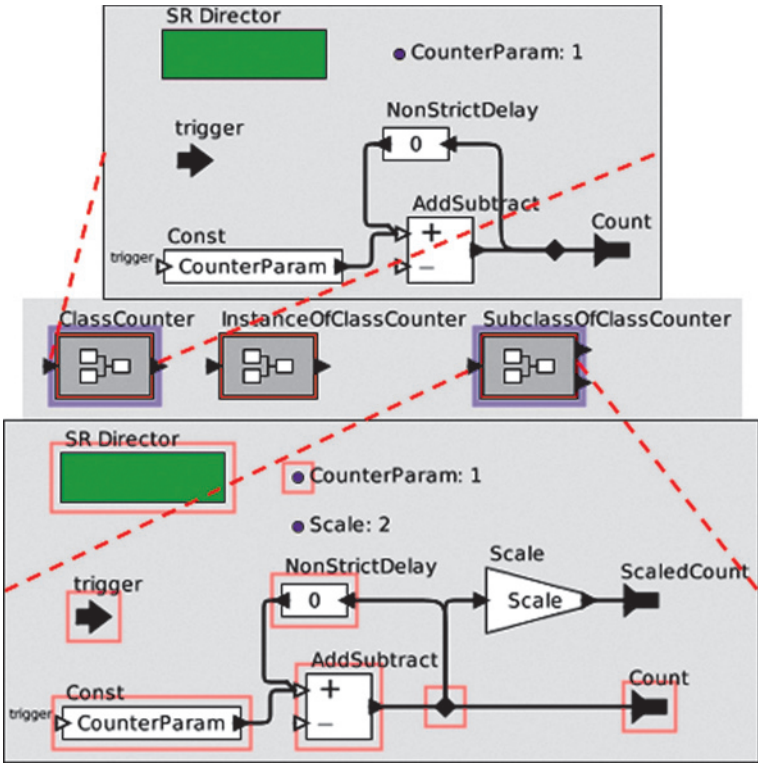

Figure 2: OO modeling in Ptolemy II.

class. The inherited components are outlined with a pink line, indicating visually that they are inherited components. The subclass extends the base class by adding some additional actors, connections, and ports. These additions do not have the pink outline.

Classes and subclasses in Ptolemy II are purely syntactic objects and play no role in the execution of a model: they are not visible to the director and their ports cannot be connected to other ports. Therefore, instances must be created. Instances inherit actors, ports, and parameters from the base class and are visible to the director.

4 Modeling of MTSs through AO and OO

This section illustrates how horizontal and vertical refinements can be implemented through AO and OO modeling. In particular, we propose to use AO modeling for cross-cutting concerns. We define as cross-cutting concerns non-functional behaviors which do not modify the mathematical model of an actor. Whereas, different phases of the design process requires different mathematical models for the same actor and different configuration parameters for design-space exploration. We propose OO modeling for them. Next, we illustrate these concepts by applying them to typical elements of MTSs.

4.1 Software-based controllers

A software-based controller is a device for computing processes. In many real-time systems today, processes are executed periodically. A controller consists of a hardware architecture (e. g. multicore computers etc.) which implements algorithms for scheduling processes (e. g. EDF etc.) and strategies for computing single processes. Two computation strategies can be adopted depending on the instant in which outputs are written:

- *Read-compute-write* (RCW): every time a process is be computed, inputs are read, then control logic is executed and after that, outputs are written. The time the outputs are written depends on the execution time of the process.
- *Write-read-compute* (WRC): outputs computed during the previous cycle are written, inputs are read and then, computation is performed. The time in which outputs are written is independent from the execution time. Thus, IO operations are deterministic, given that the execution time is less than the period.

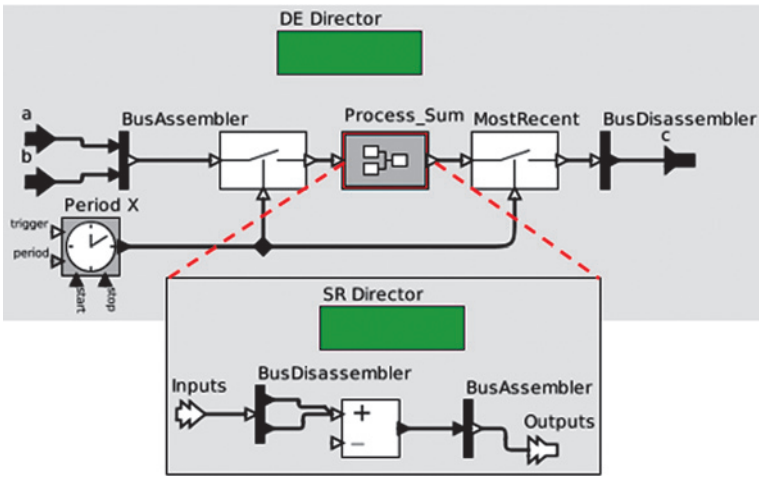

Figure 3: Model of a process computed with the WRC strategy.

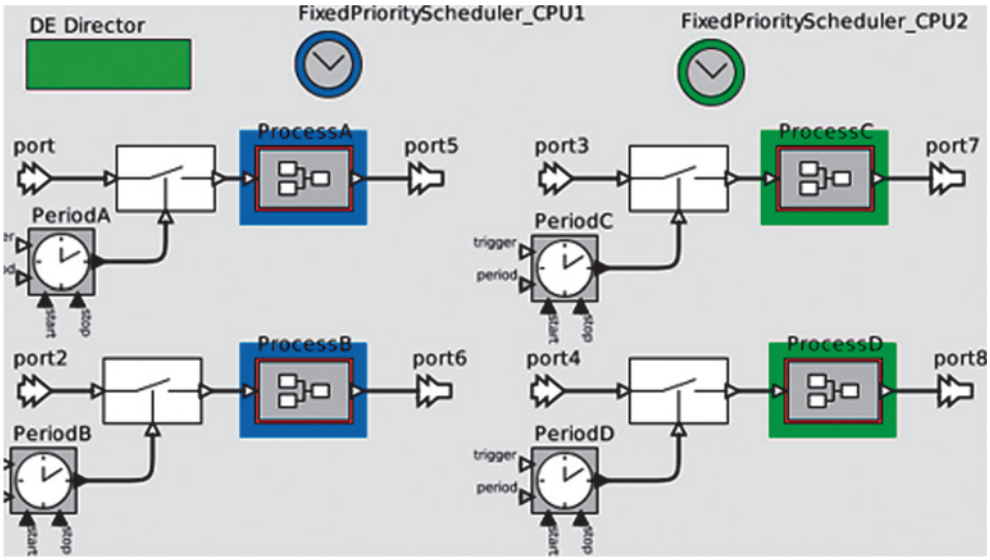

Figure 4: Example of an RCW software-based controller.

Control logic is written in processes which are computed by hardware on the basis of the selected scheduling algorithm and computing strategy. For example, a dual core controller may have static assignment of processes to CPUs. Every CPU schedules processes with an EDF algorithm and processes are computed with the WRC strategy. Next, we show how a generic real-time controller can be modeled.

– *Process*: a composite actor is created for every process. This actor contains the implemented control logic. It is computed with the Synchronous Reactive (SR) director [8] (i. e. zero execution time) and is triggered by a discrete clock. Figure 3 illustrates the implementation of a process which sums two inputs every "X" seconds. `BusAssembler` and `BusDisassembler` actors are used for grouping process I/Os. The left hand side `MostRecent` actor is necessary for making process be computed at the ticks of the discrete clock. If port `a` and `b` had been directly connected to process input ports, a computation would be performed every time a token was received from either port `a` or `b`.
– *Computation strategy*: the right hand side `MostRecent` actor of Figure 3 is used for implementing an WRC strategy. In fact, if the process has a non zero execution time, this actor is triggered before the computation has been performed. The removal of the actor would result in an RCW strategy.
– *Hardware architecture and scheduling algorithm*: execution aspects are created on the basis of the chosen configuration. Figure 4 illustrates a two processors controller in which processes are statically assigned to CPUs. `ProcessA` and `ProcessB` are assigned to `CPU1`, while `ProcessC` and `ProcessD` to `CPU2`. Every CPU implements a fixed priority scheduling algorithm with preemption. Priority and execution time are set as parameters of the association

of aspect with actors. The utilization of a scheduler with/without preemption is configured as parameter of the `FixedPriorityScheduler`[1].

The defined aspects model software deployment in a clean way without increasing model complexity. Aspects are reusable and can be employed in different models. Removing aspects from models is done by simply deleting the aspect; no other changes to the model are required. The utilization of aspects for modeling software-based controllers facilitates design-space exploration. In fact, different deployment configurations such as hardware components and scheduling strategies can be quickly modeled and simulated.

4.2 Physical elements

During a design process, physical plant, sensors and actuators may have different mathematical models for modeling different abstractions of the system and different configuration parameters for design-space exploration. OO is used for tracing the different abstraction models and easily modeling alternatives of physical means.

For example, we can consider a rotational incremental encoder. Two classes can be created for modeling two layers of abstractions: ideal and quadrature encoder. Both classes have angular velocity as input, but the ideal encoder outputs motor position, while the quadrature encoder outputs two quadrature waves. The two classes can be saved in a user library. Then, a class can be dragged and dropped into a model and instances can be created. Configuration parameters of instances can override the ones of the base class. When an instance overrides one parameter, the constraint for that parameter to have the same value of the class one is broken. Whereas, if configuration parameters of instances do not override the ones provided in the class, changes in the class configuration parameters will automatically propagate to all instances. Again, this feature facilitates modeling during design-space exploration.

4.3 Communication buses and networks

Many MTSs include multiple computing platforms, which communicate via networks to control plants with large physical extent. Even when the plant is not physically distributed, networked solutions may be used to distribute computational load, provide physical partitioning of the application, enable more timely local control, or to provide redundancy. The inclusion of networks into MTSs requires temporal characteristics of networks to be included in the MTS model, since network latency will negatively affect the timing of communication between platforms [6].

Communication aspects are proposed for modeling network effects as shown in Figure 1. These aspects may implement *communication protocols* and compute the latency of messages by including congestion effects.

5 Case study

In this section we show how aspects and objects can be used during a design process for simulating all domains of MTSs through different abstraction layers via simulations.

We consider a load that must perform a periodic trajectory with a certain tracking error. A DC motor is selected as actuator. A cascade control of position, velocity and current is implemented. Position is sensed through a rotational encoder, and current through a current transducer. Velocity is obtained numerically from position.

The first abstraction layer that can be performed is *functional simulation*. Here, a model implements the dynamic equations of motor and load, and control law is computed with a continuous controller: SR director which interacts every time step of the solver with the physical model. A cascade control of position, velocity and current is implemented from the continuous controller. An ideal encoder is used for sensing position, while velocity is directly obtained from the motor equations. Target trajectory is computed through a state machine which is reset when the final position has been reached.

Then, *software in the loop* (SIL) simulation can be performed [10]. Control law implemented in the continuous controller is discretized and necessary sample rates are identified. After that, deployment strategy is selected: control law is written in processes computed with the identified sample rates, processes are assigned to CPUs, and a scheduling strategy is defined for single CPUs. Deployment strategy includes also communication means and protocols. We think this layer should be utilized for estimating acceptable values for deployment means (e. g. tolerated worst case execution time, WCET, maximum communication latency etc.). The definition of methodologies for this layer will be matter of future investigations.

In the case study, SIL is implemented starting from the functional model through these steps:

1 FixedPriorityScheduler: the aspect implements java control logic which can be found at: https://chess.eecs.berkeley.edu/ptexternal/src/ptII/doc/codeDoc/ptolemy/actor/lib/aspect/FixedPriorityScheduler.html

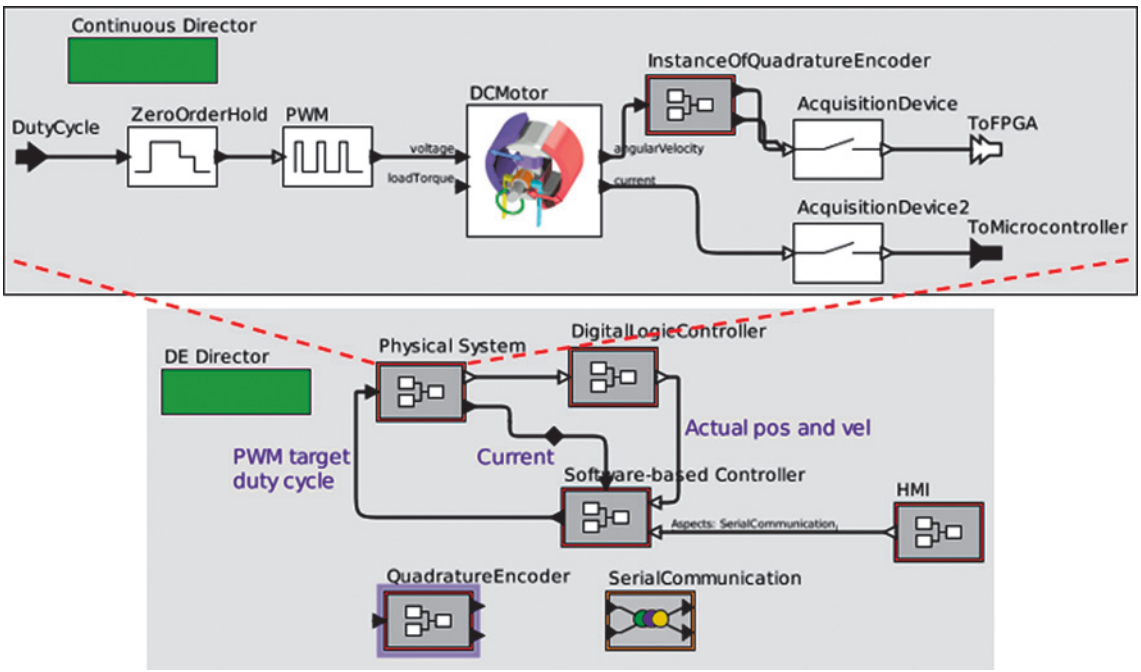

Figure 5: Software in the loop model.

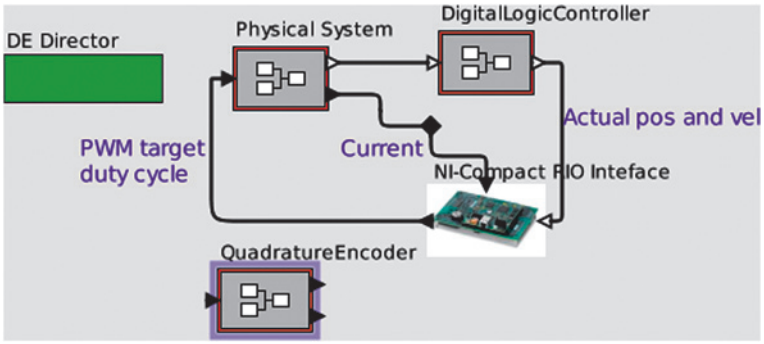

Figure 6: Hardware in the loop model.

- ideal encoder is replaced with a two quadrature waves encoder;
- encoder and current are sensed through acquisition devices with certain sample rates;
- DC motor is commanded through PWM;
- a digital logic controller is used for computing velocity and position from the encoder output signals and for regulating the PWM duty cycle. Digital logic controllers implement control logic directly through hardware digital gates. Therefore, their execution time can be neglected in comparison with software-based controllers. Their logic is computed through SR directors in the simulation model. For the reason that a PWM actor is defined in Ptolemy II within the continuous domain, we can abstract the functionality and consider target duty cycle directly realized from that actor.
- Target trajectory, and position, velocity and current control laws are computed in processes running in a software-based controller. Processes are assigned to one CPU which utilizes a fixed priority scheduling; as the controller in which code will be deployed. Execution time of processes is arbitrary assigned in order to identify acceptable solutions.
- An HMI is implemented for communicating with operators;
- digital and analog I/Os are used for all the connections (i. e. negligible latency). The only exception is the communication with the HMI which is performed serially.

SIL simulation model is represented in Figure 5. Aspects and objects are created from domain-specific engineers and then assembled generating a complete model of the

MTS. Interoperation semantics is granted from the utilization of Ptolemy II simulation software.

Eventually, a *hardware in the loop* (HIL) simulation is implemented [16]. Target controllers are selected, code is generated and controllers run with a real time model of the application. It is verified that controllers fulfill the specifications identified in the SIL phase (e. g. WCET etc.). HIL simulation model is shown in Figure 6. The `NI-CompactRIO` actor implements an interface which communicates with the real controller. Generally, ethernet communication is used which allows communication time not lower than few milliseconds. Digital logic controllers can be executed at much faster rates. For this reason, we decided to leave this object "virtual" because the communication with physical world would have corrupted the results. Moreover, digital logic controllers are deterministic and there is no need to simulate them through HIL.

6 Conclusion and future work

This paper has shown how AO and OO can support designers for modeling non-functional behaviors of MTSs without excessively increasing the complexity of functional models. These tools allow segregating domains of expertise because aspects and objects can be implemented by domain-specific expert and then easily integrated in a unique model of the solution. Moreover, design-space exploration is faster due to the modularity and interchangeability of aspects and objects. Next, we discuss potential future work.

SIL is the most critical phase of the design process. Many degrees of freedom (DoFs) must be selected as tolerated WCET of controllers, control law types (e. g. lead/led, PI, PID etc.) and parameters (e. g. gains, sample rate etc.), communication means and protocols etc. Generally, physical prototypes are built and "try-and-error" iterations are performed until working solutions are met. There is the need of optimization-based methodologies for identifying optimal combination of the DoFs before selecting physical components and building prototypes.

References

1. G. Agha. *Actors: A Model of Concurrent Computation in Distributed Systems*. MIT Press, Cambridge, MA, USA, 1986.
2. I. Akkaya, P. Derler, S. Emoto, and E. A. Lee. Systems Engineering for Industrial Cyber-Physical Systems using Aspects. *Proceedings of the IEEE*, 2015.
3. D. M. Auslander and C. J. Kempf. *Mechatronics: Mechanical System Interfacing*. Prentice Hall, 1996.
4. D. Broman, L. Greenberg, E. A. Lee, M. Masin, S. Tripakis, and M. Wetter. Requirements for Hybrid Cosimulation. In *18th International Conference on Hybrid Systems: Computation and Control, HSCC*, April 2015.
5. C. Brooks, C. Cheng, T. H. Feng, E. A. Lee, and R. von Hanxleden. Model Engineering using Multimodeling. In *International Workshop on Model Co-Evolution and Consistency Management (MCCM)*, 2008.
6. J. Cardoso, P. Derler, J. C. Eidson, and E. A. Lee. Network latency and packet delay variation in cyber-physical systems. *IEEE Network Science Workshop*, 2011.
7. F. Cremona, M. Morelli, and M. Di Natale. TRES: a modular representation of schedulers, tasks, and messages to control simulations in simulink. In *Annual ACM Symposium on Applied Computing (SAC)*. ACM, 2015.
8. S. A. Edwards and E. A. Lee. The semantics and execution of a synchronous block-diagram language. *Science of Computer Programming*, 2003.
9. S. A. Edwards and E. A. Lee. The Case for the Precision Timed (PRET) Machine. In *Proceedings of the 44th Annual Design Automation Conference*. ACM, 2007.
10. S. Han, S.-G. Choi, and W. H. Kwon. Real-Time Software-In-the-Loop Simulation for Control Education. *International Journal of Innovative Computing, Information and Control*, 2011.
11. F. Harashima, M. Tomizuka, and T. Fukuda. *"Mechatronics: What is it, Why and How?"*. IEEE/ASME Transactions on Mechatronics 1, 1996.
12. D. Henriksson, A. Cervin, and K. Årzèn. TrueTime: Real-time Control System Simulation with MATLAB/Simulink. In *Proc. of the Nordic MATLAB Conference*, 2003.
13. C. Hewitt. Viewing control structures as patterns of passing messages. *Artificial intelligence*, 1977.
14. E. Lee and S. Neuendorffer. Classes and subclasses in actor-oriented design. In *Formal Methods and Models for Co-Design*, 2004.
15. E. A. Lee. Constructive Models of Discrete and Continuous Physical Phenomena. Technical report, EECS Department, University of California, Berkeley, Feb 2014.
16. D. Maclay. Simulation gets into the loop. *IEE Review*, May 1997.
17. Modelica. Functional mock-up interface for model exchange and co-simulation. Technical report, 2014.
18. A. Pinto, L. Carloni, R. Passerone, and A. S. Vincentelli. Interchange Format for Hybrid Systems: Abstract Semantics. In J. Hespanha and A. Tiwari, editors, *Hybrid Systems: Computation and Control*. 2006.
19. C. Ptolemaeus, editor. *System Design, Modeling, and Simulation using Ptolemy II*. Ptolemy.org, 2014.
20. S. Tripakis and D. Broman. Bridging the Semantic Gap Between Heterogeneous Modeling Formalisms and FMI. Technical report, EECS Department, University of California, Berkeley, 2014.
21. R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström. The Worst-case Execution-time Problem – Overview of Methods and Survey of Tools. *ACM Trans. Embed. Comput. Syst.*, May 2008.

Bionotes

Giacomo Barbieri
University of Modena and Reggio Emilia, Via Amendola 2, 42122 Reggio Emilia, Italy
giacomo.barbieri@unimore.it

Ing. Giacomo Barbieri received the master degree in Mechanical Engineering in 2012. From 2013, he is attending a Ph.D. at the University of Modena and Reggio Emilia in collaboration with Tetra Pak Packaging Solutions. His main expertise is life-cycle design, and simulation and verification of production systems.

Patricia Derler
National Instruments, 2168 Shattuck Ave., 94704 Berkeley, USA
patricia.derler@ni.com

Dr.-Ing. Patricia Derler received the degree in Software Engineering (2002) from the University of Hagenberg, Austria and Ph.D. in Computer Science (2006) from the University of Salzburg, Austria. She was postdoctoral researcher at the University of Berkeley for 4 years and joined National Instruments in 2015. Her research is focused on timed models of computation for distributed embedded real-time systems, as well as the modeling and simulation of timing behavior.

David M. Auslander
UC Berkeley, 5120 Etcheverry Hall, 94720 Berkeley, USA
dma@me.berkeley.edu

Prof. Ing. David M. Auslander completed undergraduate studies at the Cooper Union and graduate studies at MIT, both in mechanical engineenng. He is Professor of the Graduate School, Mechanical Engineering Department, University of California at Berkeley. He has interests in dynamic systems and control. His research and teaching interests include CPSs and real time software, and mechanical control. Current projects in these areas are building energy control, design methodology for real time control software for mechanical systems, satellite attitude control, simulation methods for constrained mechanical systems, and engineering curriculum development. He consults in industrial servo control systems and other control and computer applications.

Roberto Borsari
Tetra Pak Packaging Solutions SpA, Via Delfini 1, 41123 Modena, Italy
roberto.borsari@tetrapak.com

Dr.-Ing. Roberto Borsari received the degree in Nuclear Engineering (1989) and Ph.D. in Nuclear Engineering (1994) from the University of Bologna. He is presently manager for Forming and Virtual Verification within Tetra Pak Packaging Solutions. His major area of expertise is computational physics and his activities focus mainly on the definition and application of model-based systems engineering approaches to the development of complex packaging systems, with special emphasis on modeling and simulation of package forming and related manufacturing processes.

Cesare Fantuzzi
University of Modena and Reggio Emilia, Via Amendola 2, 42122 Reggio Emilia, Italy
cesare.fantuzzi@unimore.it

Prof. Dr.-Ing. Cesare Fantuzzi received the master degree in Electronic Engineering (1990) and Ph.D. in System Engineering (1995) from the University of Bologna. From 1996 to 2001 he was Assistant professor at Department of Engineering of University of Ferrara, and from 2001 to 2006 he was Associate Professor of Automatic Control at Department of Science of Method of Engineering of University of Modena and Reggio Emilia. Since 2006, he is Full Professor of Automatic Control and Industrial Automation at the University of Modena and Reggio Emilia. His research interests include modeling and control of production systems, fault diagnosis, human-centered machine interfaces and service robotics.