Available online at www.sciencedirect.com

ScienceDirect

IFAC Papers Online
CONFERENCE PAPER ARCHIVE

IFAC-PapersOnLine 49-6 (2016) 063–068

A Virtual Laboratory for the Prototyping of Cyber-Physical Systems

Alessandro Beghi [*], **Fabio Marcuzzi** [**], **Mirco Rampazzo** [*]

[*] *Department of Information Engineering, University of Padova, via G. Gradenigo 6/B, I-35131 Padova, Italy. E-mail: {beghi, rampazzom}@dei.unipd.it.*
[**] *Department of Mathematics, University of Padova, Via Trieste 63, I-35121 Padova, Italy. E-mail: marcuzzi@math.unipd.it*

Abstract:
Cyber-physical systems (CPS) refer to novel hardware and software compositions creating smart, autonomously acting devices, enabling efficient end-to-end workflows and new forms of user-machine interaction, in a wide range of application fields. Given their heterogeneous nature, CPS are naturally designed in the so-called simulation-centric process, where physical equipment design are translated into behavioral simulation models. Although many tools exist to ease the design phase in the different disciplines, their full integration is still an open problem. This fact holds particularly true in a control-design perspective, given that in a CPS the "plant" has a heterogeneous nature, and the design of model-based, advanced control techniques would strongly benefit from the availability of a common modelling environment. In this paper, we try to enhance this simulation-centric process by introducing a pure simulation kind of prototype, based on the co-simulation of the firmware and of the multi-physical controlled system. We introduce some innovative tools, implemented in μLab/CfL, and discuss upon their impact towards a better collaborative design and integration during the design of CPS. An example is given, taken from the HVAC (Heating, Ventilation, and Air Conditioning) field.

Keywords: Virtual Laboratory, Cyber-physical systems, Control, HVAC, LaTeX, Python

1. INTRODUCTION

Cyber-physical systems (CPS) refer to novel hardware and software compositions creating smart, autonomously acting devices, enabling efficient end-to-end workflows and new forms of user-machine interaction. In manifold emerging application domains such as health care, traffic management or energy supply, CPS carry a high potential for creating new markets and solutions to societal hazards, but impose highest requirements to quality in terms of resilience, safety, security and privacy, Kim and Kumar (2012), Derler et al. (2012), Baheti and Gill (2011). However, the heterogeneous, multi-physics, evolving and distributed nature of CPS bears major challenges to continuously assure these quality requirements employing state of the art methods and technologies. Foundational research efforts are needed to achieve a predictable quality level in an efficient, traceable and measurable way, coping efficiently with external and internal changes, supporting necessary transitions between mechanical, electrical and software engineering, as well as integrating management, design and deployment aspects.

CPS are naturally designed in the so-called simulation-centric process, where physical equipment design is translated into behavioral simulation models. These simulation models are then used to develop model-based control systems. These subsystem simulations and model-based code are then used to perform real-time hardware-in-the-loop (HIL) simulation for early integration testing, following the paradigm of Virtual Prototyping. Finally, the early-integration lab is transformed into the last stage integration facility as physical prototypes are delivered for final acceptance testing in scheduled releases.

The mark of a highly effective simulation-centric process is collaborative design and integration during each design simulation and real-time simulation stage. Such collaboration and integration requires the interaction among experts of different fields, each typically using their own simulation/design environment/tools, such as CAD Tools (e.g. Catia, ProEngineering, SolidWorks), behavioral modelling (e.g. ADAMS, Simulink, Dymola), and CACSD tools (e.g. Matlab/Simulink, Rhapsody), Junjie et al. (2012). Although some of such tools can operate in co-simulation (see for instance the coupling between Matlab/Simulink and ADAMS), their overall integration is still far from being satisfactory, Al-Hammouri (2012). Moreover, it would be preferable that all the experts from the different fields could exploit a common environment, so as to easily share and integrate their different skills. This fact holds particularly true in a control-design perspective, given that in a CPS the "plant" can have heterogeneous nature, as is typical in a multi-physics approach, and the design of model-based, advanced control techniques can strongly benefit from the availability of a common modelling environment.

In this paper, we try to enhance this simulation-centric process by introducing a pure simulation kind of prototype, based on the co-simulation of the firmware and of the multi-physical controlled system. This kind of prototype does not require any specific hardware, like HIL does, and can e.g. exploit emerging simulation paradigms like the cloud computing. To this purpose, we introduce in this simulation-centric process some innovative tools, implemented in μLab/CfL, Marcuzzi (2013), Crafa et al. (2014), and discuss upon their impact towards a better collaborative design and integration during the design of CPS.

More precisely: we describe an equation modelling framework and compare it with other approaches to physical systems modelling (section 2), we adopt an *alias/bindings* architecture to integrate in a single simulation model all the implementations done during the design cycle of the control system, from the initial control algorithm prototype to the final control firmware installed in the production equipment (section 3), we exploit the interoperability of firmware and Python scripts, in the co-simulation of the control logic, to achieve a better efficiency in prototyping the control algorithms and a smooth transition of their implementations, from the early prototype to the final product. In section 4 we discuss in general the possible impact of these tools on Virtual Prototyping of CPS and in section 5 we give an example taken from the HVAC (Heating, Ventilation, and Air Conditioning) field.

2. MODELLING APPROACHES

There is an existing modelling language which is common among engineers of different flavours: it is that of mathematical equations. What it is usually not shareable among them is the computer representation of these equations. Engineers more affine to computer science preferably use for modelling purposes text-based computer languages (C/C++, Fortran, Java, Matlab, etc.), while others use visual languages/tools specific for their applications (ADAMS for mechanics, FlowMaster for fluid dynamics, SPICE for electronic circuits, Simulink for control systems are only a few examples). In both cases, mathematical equations are hidden and the model can be understood only through a substantial knowledge of the computer language/tool adopted.

We consider here a tool, CfL Crafa et al. (2014), which converts automatically a mathematical model written in LaTeX into a Python script ready-to-use within a co-simulation environment like μLab, Marcuzzi (2013). More details about these tools can be found in Appendices A and B. In this way, we can apply model-based control techniques with a more efficient methodology, since it is based on a shareable model among mechanical, control, electrical, and software engineers.

Analytic models are a good compromise for control systems modelling, even if mechanical engineers often develop much more detailed multi-physical models. Here we consider a concentrated-parameter model, as usually done in the simulation of control systems, but in mechanical engineering distributed-parameters models, e.g. Finite Element models are often used, Hughes (2000). When a complex multi-physical model is available and a simpler analytical model could be more appropriate for control purposes, the integration between these different models can be made by means of System Identification techniques, e.g. a reduced-order model can be built from simulation data. Complex physical models, like that produced by state-of-the-art mechanical modelling tools, produce accurate simulation data that can be used to do system identification on reduced-order analytical models, Friswell and Mottershead (1995), retaining a good physical meaning. This enforces integration among control and mechanical engineers, rather difficult to obtain, instead, with black-box system identification, which is not a standard background of mechanical engineers and it is practically not present in the mechanical modelling tools.

3. FROM THE CONTROL ALGORITHM TO THE CONTROL FIRMWARE

As with models, the control logic is also initially conceived as a quite abstract algorithm, before implementing it e.g. in C language for a microcontroller. Again, mathematical equations are the common language that make this logic understandable also to non programmers. CfL can translate these equations in a Python script, which can be used in the co-simulation within μLab as if it were the control firmware, see Fig. 1. In this case the script instructions interact directly with the pins of the abstract microcontroller model.

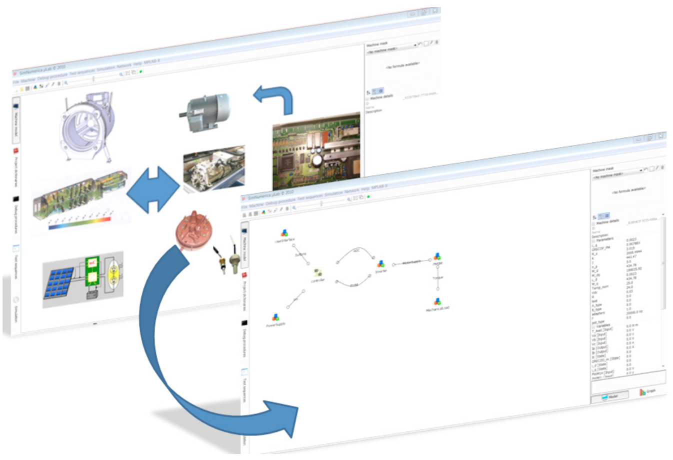

Fig. 1. In the co-simulation, physical system components are represented by blocks containing mathematical models, and the overall system as an interconnection of blocks.

At this stage, the co-simulation model can be set at an high level of abstraction, and can be e.g. debugged/analysed with the same simplicity as an usual PC program (like Matlab/Simulink, Rhapsody, etc.). The next prototyping steps will usually refine this model by adding details about the real-time control: microcontroller interrupt-driven architecture and peripherals, sensors and actuators physical models, communication devices, etc., and a control logic implementation that evolves by mixing firmware layers and Python scripts, and arrives at the production firmware, co-simulated in a real microcontroller model.

Thanks to the *alias/bindings* model architecture introduced in μLab, the engineers can define abstract project