Mechatronics 43 (2017) 99–111

Contents lists available at ScienceDirect

Mechatronics

journal homepage: www.elsevier.com/locate/mechatronics

Virtual prototyping of embedded control software in mechatronic systems: A case study☆

Alessandro Beghi[a], Fabio Marcuzzi[b], Paolo Martin[a], Fabio Tinazzi[c,*], Mauro Zigliotto[c]

[a] Department of Information Engineering, University of Padova, Italy
[b] Department of Mathematics, University of Padova, Italy
[c] Department of Engineering and Management, University of Padova, Vicenza, Italy

ARTICLE INFO

Article history:
Received 8 September 2016
Revised 14 February 2017
Accepted 17 March 2017

Keywords:
Co-simulation
Mechatronic systems
Embedded control

ABSTRACT

The large majority of technological devices can be seen as the sum of components of heterogeneous nature (electrical, mechanical, thermal, software, etc.) so that their analysis and design calls for the use of tools from different engineering disciplines. Integration among the different tools is particularly relevant at control system design level, since it is at this stage that it is required to analyze the behavior of both each single component and the system as a whole, with different levels of detail. In this paper *mechatronic systems* are considered, that exhibit a strong interplay between mechanical and electrical components, and the issue of modeling and designing embedded control algorithms and architectures for such systems is addressed. In particular, an integrated virtual prototyping approach for analyzing the system behavior down to embedded software level is proposed, that can be used in a wide number of situations that can represent the actual real-world operational conditions of consumer products. This approach can be used for system design (in particular at control systems level), embedded software design, and virtual testing so as to optimize and reduce the costs of late stage software development, physical prototypes, and their testing. The case study is based on some recently derived software tools that perform the co-simulation of the firmware execution and the multi-physical controlled system dynamics. The actual control software implemented in the final product can be entirely developed and tested inside the virtual prototype. To prove the validity and potentialities of the proposed approach, a real case study is presented, regarding a very common, though complex to simulate, mechatronic system such as an electric sliding gate. It turns out that the proposed environment goes beyond hardware-in-the-loop tools, since it does not require the use of specific hardware (the hardware itself is simulated in detail) but allows to analyze in detail the status of the microprocessors and peripherals at arbitrary time-scales and allows the designer to study at the earliest design stages the dynamics between the multi-physical controlled system and the firmware, without committing to a given hardware structure.

1. Introduction

In the design of advanced control systems it is common practice to make extensive use of Computer Aided Control Systems Design (CACSD) software tools. At a very basic level, such tools allow to simulate the relevant system dynamics, for a first assessment of the different control strategies. However, when very complex systems are to be controlled, more sophisticated, integrated tools come into play. Such tools are at the core of what is sometimes called the simulation-centric process. In many cases new components are modeled in software first [1]. The designer runs simulations of a new component in conjunction with models of the rest of the system to study the behavior of the overall system and to optimize the algorithms and routines used in the new component before building any physical prototype. This type of application is commonly referred to as a *co-simulation*: all parts of the system, both continuous-time processes and discrete-time events, are being simulated in software [2,3]. The resulting virtual prototypes are then validated in a Hardware-In-the-Loop (HIL) application that includes the effects of the hardware-system interaction (sampling, time lags, etc.). HIL tools allow the testing of new hardware components and prototypes while communicating with software models that simulate the rest of the physical system. The physical components being tested respond to the electrical signals, computed by the simulation model running on a computer, as they were operating in the real system, since these signals are gener-

☆ This paper was recommended for publication by Peter Hehenberger.

* Corresponding author.
E-mail addresses: alessandro.beghi@unipd.it (A. Beghi), fabio.marcuzzi@unipd.it (F. Marcuzzi), paolo.martin@unipd.it (P. Martin), fabio.tinazzi@unipd.it (F. Tinazzi), mauro.zigliotto@unipd.it (M. Zigliotto).

http://dx.doi.org/10.1016/j.mechatronics.2017.03.004
0957-4158/

ated compatibly with those generated by the real controlled system. Additionally, HIL systems are commonly used for fault analysis studies (shorted/open circuit signals, etc.), such as the unintentional islanding in distribute energy resources in [34], reliability (endurance) tests of new components in large plants [36], and the occurrence of anomalous behaviors in complex mechanical systems [16]. To setup a simulation-centric control system design project, there are therefore some basic, preliminary steps to be taken, namely: 1) derivation of a detailed mathematical model of the system to be controlled, as the basis for the application of modern, model-based control system design techniques, and 2) definition of some integration strategies, that allow the actual co-simulation of the different software components, as well as the coupling with hardware components via the HIL tools.

Nowadays, the approach described above is challenged by the widespread appearance of embedded systems, that is complex, software-rich products that have to manage a very large and heterogeneous set of constraints coming from the physical world [4]. In fact, traditional development approaches are often mono-disciplinary in style, in that separate mechanical, electronic and software engineering groups handle distinct aspects of product development and often do that in sequence. This is commonly referred to as the main issues in the development process of a complex system [6,7]. However, cross-cutting system-level requirements that cannot be assigned to a single discipline, such as performance and dependability, can cause great problems, because their impact on each discipline is exposed late in the development process, usually during integration. Embedded systems design, in which the viability of the product depends on the close coupling between the physical and computing disciplines, therefore call for a more multidisciplinary approach and the development of new tools. Some examples of integration of physical and computational systems are reported in [8,9]. They both refer to automotive applications, namely the control of an electronic throttle and a gasoline fuel pump. The validity of the co-simulations are proven by fair comparisons between simulated and experimental results. However, they are carried out by using several simulation tools (Saber, CoMET, Simulink) and different programming languages (C/C++, Matlab).

In the light of the previous considerations, it appears that simulation-based design processes have to meet new requirements, with stronger emphasis on the interaction among the various systems components at the lowest possible level, so as to include extensive validation of the embedded software component. This paper proposes an integrated and extensive design approach, that explicitly takes into consideration such requirements. In particular, the integrated virtual prototyping environment can be used for analyzing the system behavior down to embedded software level in a wide number of situations that can represent the real-world operational conditions of devices. Such environment can be used for system design (in particular at control system level), embedded software design, and virtual testing so as to optimize and reduce the costs of late stage software development, physical prototypes, and their testing.

The aforementioned higher degree of integration is achieved by using new software tools that allow to couple dynamical simulation of multi-physics systems with simulation of the embedded control software running on microcontrollers/DSPs (usually called “control firmware” by the designers of consumer products with embedded microcontrollers onboard). This capability constitutes a step forward with respect to present day available technologies, where firmware development tools exist, but yet, they are not directly integrated in the dynamical systems simulation environments. Notably, the proposed virtual prototyping environment makes use of an equation modeling framework, CfL, based on open-source software languages, Latex and Python [10]. Therefore, engineers with different backgrounds are able to work on the same platform, describing the system with mathematical equations that are automatically translated into a simulation model of the system. From the control design point of view, the approach enabled by the use of the framework CfL eases the communication between engineering groups, [4,5]. The model is written in Python, which is becoming a standard for scientific/engineering computing. Moreover, Python is very well tied with the C language, which represents the standard for microcontrollers/DSPs programming. Existing tools available for developing embedded control software, although at an advanced level of user-friendliness and applicability, are based on proprietary languages (the user cannot freely exchange models between different tools), domain-specific, mono-disciplinary (and rather unknown to non-specialists) and sometimes far from C language, see e.g. Matlab/Simulink, dSpace [28] etc. A notable tool is Modelica that embodies much of the spirit of the CfL equation modeling framework. Its programming language (C++) requires a sophisticated compiler to translate the textual model into an executable simulation, problem partially mitigated by PyModelica. An integration of the CfL equation modeling framework with these existing tools is reasonably possible, but out of the scope of this paper. The same applies to a rigorous assessment of whether it would be better a textual modeling language like LaTeX or visual modeling tools.

In the proposed μLab environment, an alias/bindings architecture is used 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 control firmware installed in the final product. These implementations are created and stored by the user in *Project Dictionaries* [11]. The μLab co-simulation engine permits a quite general interoperability of firmware and Python scripts at runtime. This feature is exploited in the co-simulation of the control logic to achieve a better efficiency in prototyping of the control algorithms and a smooth transition from the early prototype to the final product. At the earliest stages, the μLab environment allows the designer to study at simulation level the dynamics between the multi-physical controlled system and the firmware. During the final refinements, an exact model of the microcontroller can be included in the co-simulation. These advantages are important in the development of advanced control algorithms, (e.g. Model Predictive Control (MPC) [12–15]). These algorithms represent a challenge for the integrated simulation architecture and they usually require a keen use of the microprocessors resources.

This paper presents a real case study, namely, the virtual prototyping and embedded control software design for an electric sliding gate, a very common and quite complex to simulate mechatronic system. The paper is organized as follows. In Section 2, the adopted co-simulation software is introduced. In Section 3 the modeling of the electrical gate is derived, considering its various components (electric motor, mechanical transmission, and electrical motor drive). In Section 4 the experimental and co-simulation results are discussed, while in Section 5 the results are compared with those obtainable with state-of-the-art tools. Conclusive remarks are reported in Section 6.

2. The adopted co-simulation environment

A mechatronic system, i.e. a mechanical system whose operations are controlled by electrical and electronic equipments [32], can be simplified as sketched out in Fig. 1. The electrical parts are the peripherals, such as the electric motors, related converters, and sensing devices, i.e. the group of sensors and transducers with their conditioning circuits, which can also be considered part of the electric area. The sensor interface of Fig. 1 describes the software part related to the measurement information acquisition.