Analyzing Multi-core Timing Effects on Control Systems via Co-simulation

Philipp Wagner
Vector Informatik GmbH
Regensburg, Germany
philipp.wagner@vector.com

Thomas Wilhelm
Vector Informatik GmbH
Regensburg, Germany
thomas.wilhelm@vector.com

Andreas Sailer
Vector Informatik GmbH
Regensburg, Germany
andreas.sailer@vector.com

***Abstract*—In the automotive industry, the integration of different functions is a challenging and transdisciplinary task. The effects caused by the timing behavior of discrete controllers on the control plant need to be understood during development. In this paper we show how co-simulation of a discrete multi-rate control unit and the physical plant is used to provide integration insights early in the development process, i. e. during design. We highlight effects of the timing behavior of a single-core controller on a control system from an industrial use case. We discuss multi-core timing phenomena that further influence the timing behavior, to motivate the research and the application of co-simulation during the development of control systems.**

***Index Terms*—co-simulation; multi-core; timing behavior; control systems; cyber-physical systems; automotive**

I. Introduction

Cyber-physical systems (CPS) consist of a continuous plant and discrete, embedded electronic control unit (ECU). This fact must be respected during the design of a control system. There are numerous timing effects of complex embedded software that affect the performance of control systems. Predicting these effects is infeasible without supporting tools like timing simulation. Arguably, the most accurate and reliable results can be achieved when functional and timing simulation are coupled (co-simulation). Co-simulation can be applied in various stages of the automotive development process, such as design and implementation phases. Systems are safety-critical and complex, so timing requirements have to be fulfilled throughout all development phases.

The following section gives some background on control software development, followed by an industrial case study in section III. In section IV we present related work with a trend towards multi-core, for which specific timing effects are discussed in section V. We conclude with an outlook.

II. Background

Fig. 1 illustrates artifacts in the process of control software engineering for an automotive ECU as seen in the industry. Co-simulation is possible on different levels of granularity. Control system hardware is not considered here. There are two architectural realms: The system and software architecture. From a system architecture perspective, the vehicle consists of several sub-systems, like the wheels, drive-train and the engine. Each of these sub-systems is composed of specific sub-functions. For each sub-function a component architecture is defined. For example a controller component contains sensors, actors and the controller.

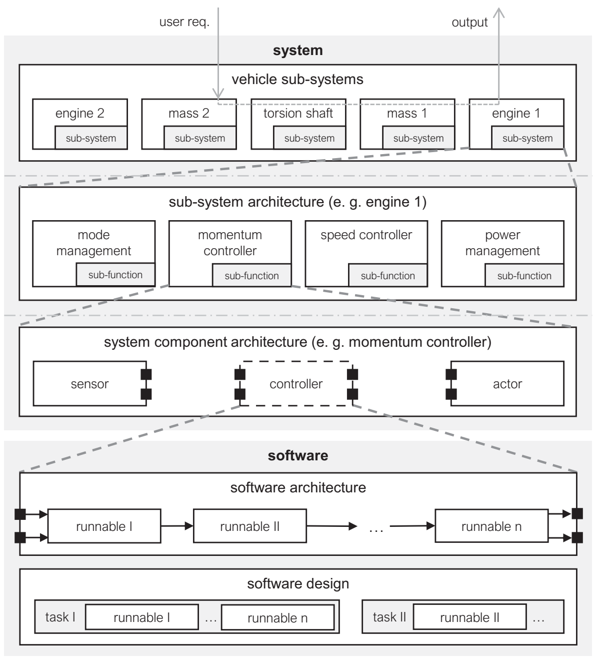

Fig. 1. Overview of the automotive control system development process

For all components, a software architecture and design are defined as part of the software realm. The software architecture contains information about input and output ports, signals, and control flow that represent the controllers effect chain. The software design contains the mapping of atomic runnable entities, which contain the control function code, onto tasks. On single-core ECUs task scheduling introduces effects that impact the control system performance, such as long preemption times due to execution of higher-priority tasks, leading to dropped task instances or missed deadlines.

Before developing the software of a new control system, an requirement-fulfilling controller is chosen, depending on the physical behavior of the controlled

This work is part of the EWORAM project with partial funding from the Bavarian Ministry of Economic Affairs, Regional Development and Energy under grant no. IUK499/001.

system as well its requirements and those of related sub-systems. Performance characteristics, like the acceptability of lasting deviations, compensation windows, and plant delays inform the selection process. The controller's parameters are calibrated to reach the required control quality, like high stability. This leads to implicit timing requirements for the controller software. Because budgets are limited, simulating the controller's functional performance under realistic constraints as early as possible, allows one to improve quality, which is enabled by our co-simulation approach.

After a controller is selected, the new control system is decomposed on different levels, as shown in Fig. 1. In that process, all of the steps described above are performed. Only then the actual functional code is implemented and the system can be integrated and tested. Yet, the decisions and trade-offs made during decomposition should be evaluated beforehand, e. g., through simulation. The validation against the individual requirements of the components, based on predicted timing behavior, is a way to uncover infeasible solutions early in the design process, before measurements are possible. In the following use case, we show this for the software architecture and design of a controller.

III. Industrial Case Study

In this section, we present an industrial case study that illustrates typical effects of timing behavior on a control system. The study is enabled by the developed co-simulation engine. It controls and facilitates communication between the functional simulation in MATLAB®Simulink®[1] and the ECU timing simulation in the Vector TA Tool Suite [2]. Results from the latter are used in Simulink®for discrete control of the plant. Both models are generated from a common data source.

A. Use case

In our use case, new anti-jerk functionality is added to a preexisting engine management system. The engine control unit runs on a multi-core platform and the existing software is distributed among the cores. However, the control function of the new anti-jerk system is designed to only use one core – parallel execution of its tasks is not required. Yet, the new controller in principle is subject to multi-core timing effects caused by the existing software. In the following example, we intentionally isolate the controller and force task preemption at regular intervals via priority assignments. This approach exemplifies the detrimental timing effects that appear in both single- and multi-core scenarios. Later, we show more timing effects specific to multi-core.

This use case is common in the automotive domain, in which solutions are developed incrementally and iteratively, as noted in [1]. Consequently, simulation models for legacy parts of the system under development exist during design and an early validation of new features through simulation is promising. Based on the requirements of the new sub-system and the environment, an engineer determines where the new control functions are placed. Timing simulation is a way to support this mapping process.

Valid solutions differ in terms of quality and the solution space is considerable. Data for a control system's performance is obtained through functional simulation. However, more accurate results depend on the timing behavior of the discrete controller. For example, research [2] shows that latencies in the controller software do not necessarily destabilize a control system, as long as they are constant. In complex embedded systems, especially on multi-core, such constant delays cannot be expected. Not activating the control task in some instances, as in our demonstration, is a drastic, yet realistic scenario.

B. Demonstration setup

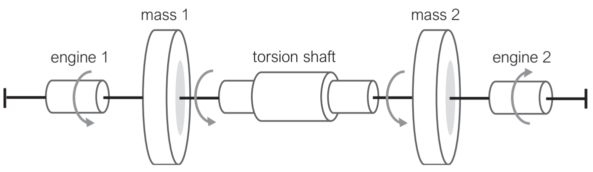

Fig. 2. Schematic of the demonstrator

For the following demonstration of the use case, the functional and timing models of a system with two degrees of freedom are used for co-simulation, shown in Fig. 2. The setup corresponds to the system architecture level in Fig. 1. Two rotating masses are attached to one engine each. These are connected by a torsion shaft, transmitting the torque of the first mass to the second. The shaft provides inertia to the setup. The engines control the rotational speed of the two masses, accelerating or decelerating them. The model can be used as a proxy for a variety of sub-systems. In our case it represents an engine-to-wheel-plant: The first rotating mass embodies the engine that transmits power to the wheel(s), signified by the second mass, via the power train (torsion shaft). Friction, slope, and other factors impacting speed, are provided by *engine 2* through deceleration of *mass 2*. The rotational speed of the second mass, the speed of the car, is a function of the power trains characteristics, material properties and the user input. This input to the controller is the desired speed of the car, which the *mass 2* sub-system converts to a torque request for the effect chain. The output (see Fig. 1) is the actuating variable for *engine 1*, resulting in torque. With the new anti-jerk controller, acceleration is supposed to be smooth and sufficiently quick, without overshooting. The user shall perceive immediate and consistent feedback. In Fig. 1, the desired speed (user requirement), resulting torque (actor output) and effect chain (dashed line) are depicted on the vehicle level.

[1]https://www.mathworks.com/products/simulink.html

[2]https://www.vector.com/us/en-us/products/products-a-z/software/ta-tool-suite/

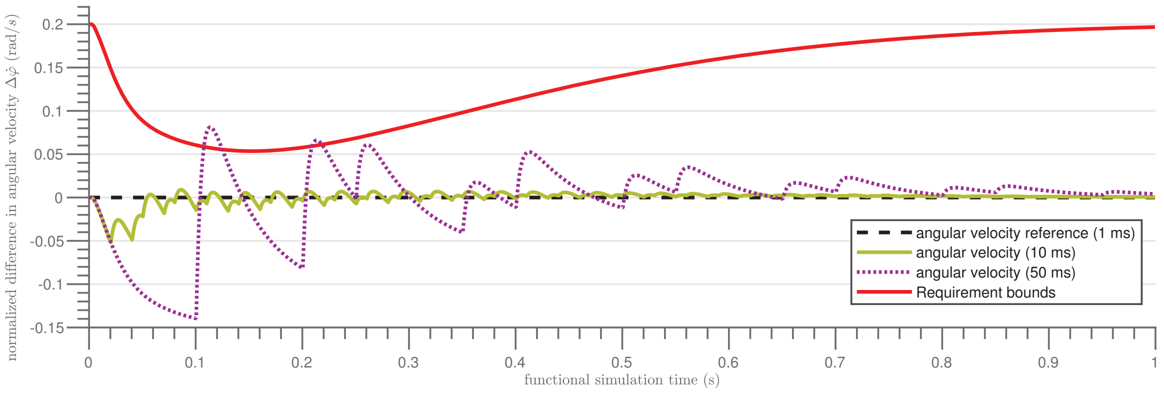

Fig. 3. Effects of regular task instance drops on control quality as delta of angular velocity

C. Results

The plot in Fig. 3 shows the difference in angular velocity $\Delta\dot{\varphi}$ of the two masses for two configurations: In the first configuration $\Delta\dot{\varphi}_{10}$ (solid green), the controller has an activation period of $10\,\mathrm{ms}$ and in the second $\Delta\dot{\varphi}_{50}$ (dotted violet) of $50\,\mathrm{ms}$. Jerk, the derivative of $\Delta\dot{\varphi}$, is less informative and therefore not shown. To avoid exceeding the permissible mechanical tension, it is required that $\Delta\dot{\varphi}$ shall not exceed $0.6\,\mathrm{rad/s}$ (solid red). For readability, the plot is normalized to $\Delta\dot{\varphi}_{\mathrm{ideal}}$ (dashed black), the results for an uninterrupted reference task with a period of $1\,\mathrm{ms}$ and near-optimal performance, such that $f(x) = x - \Delta\dot{\varphi}_{\mathrm{ideal}}$. This means that the reference curve is constantly zero. The constant requirement $\Delta\dot{\varphi} < 0.6\,\mathrm{rad/s}$ appears as a curve, with a value of $0.2\,\mathrm{rad/s}$ at $t = 0\,\mathrm{s}$, according to the scaled y-axis. At $t = 0\,\mathrm{s}$ the user wishes to accelerate, realized by a torque of $0.05\,\mathrm{N\,m}$. In both configurations, every third activation of the controller task is skipped. In that case, the controller does not compute an output for *engine 1*. For example, in the second configuration at $t = 0.1\,\mathrm{s}$ no value for the actuating variable is set for $50\,\mathrm{ms}$, resulting in a slower rotational speed of the first mass compared to the second. This shall not happen during an acceleration scenario: The second mass must reach and not exceed the rotational speed of the first mass (no overshooting). Consequently, the requirement for $\Delta\dot{\varphi}$ is not fulfilled. This represents jerking as sudden switches between relatively strong and no acceleration of the first mass. It is weaker for the configuration with a period of $10\,\mathrm{ms}$.

In this example, undesired effects are caused by intentionally dropping controller task instances. This is motivated by simulation results of a complex industry engine control model. In a realistic setting, this situation arises regularly or sporadically. With this design, we model a worst-case scenario that is independent from the underlying single or multi-core architecture.

Based on these results, several options to avoid the observed behavior exist. All of them require close examination of the system, using data provided by co-simulation. Under the given worst-case assumption, the required period of the controller task may be more than $10\,\mathrm{ms}$, but less than $50\,\mathrm{ms}$. Generally, either the timing behavior should be optimized, e. g., by improving the mapping of runnables to tasks, or by making the controller more robust via its parameters – or by both. Additionally, the initial time budget for the controller function can be validated. Co-simulation, applied in any of the relevant departments, facilitates this co-operative effort.

IV. Related Work

The modelling and simulation of control systems with real-time controllers is investigated in [3]. The authors' tool *TrueTime* is used to study the effects of networked embedded systems and software on control systems. Some early applications, including effects caused by preemptions in a networked system, are shown in [4]. Our experiments also feature preemptions as a means to not activate tasks, which degrades the performance of the controller. The capabilities of *TrueTime* and our timing simulator differ. For example, to the best of our knowledge, *TrueTime* currently does not support multi-core architectures. However, those are common in automotive and increase the complexity of timing behavior considerably.

In [5], a practical co-simulation application with a similar research interest is presented. The authors examine the effects of timing behavior, especially that of *adaptive variable rate (AVR)* tasks, on engine performance. AVR tasks can miss their deadlines depending on the engines revolutions per minute (rpm) and their execution time, among other scheduling-related reasons. They show that missed deadlines have substantial impact on system performance and suggest managing runtimes by adapting task complexity to rpm. To achieve this, motor and execution time data are exchanged between the simulators. The experiments are based on a co-simulation with MATLAB®Simulink®and *TRES* [6]. While we also use Simulink®for the functional simulation, timing behavior prediction is done using an existing model-based solution for multi-core embedded systems. According to the authors, *TRES* is inspired by *TrueTime*, albeit with the capability to simulate multi-core control units.

The benefits of co-simulation can only be reaped when properly integrated into standardized processes that are already in place. In [7] the authors map co-simulation to steps to some standard process models. We propose co-simulation steps for a different automotive implementation of the V-model. During three design phases co-simulation is used as a means to enable early verification and, consequently, iterations. A maturity model for simulation models for another implementation of the V-model is defined in [8]. They compare this process model to others in the industry, including Automotive SPICE. An approach for early verification and validation in that model is presented in [9]. A similar path is followed in [1], calling the idea *frontloading*. Early validation and verification, enabled e. g. by co-simulation, is an established concept regardless of the underlying process model, yet solutions are still under development, especially for multi-core timing behavior.

V. Multi-Core

The substantial impact of controller timing behavior on control system performance and quality are generally well understood. In the demonstration above, preemptions of the periodic controller task due to priority-based scheduling lead to execution delays and degrade the control system performance. Optimal task prioritization is one of the goals for both single and multi-core platforms. On multi-core architectures, additional sources for undue timing behavior exist and timing analysis is more complex. Due to growing interest in multi-core controllers for automotive control systems in research and practice [10], [11], the following design aspects need to be considered:

- Data exchange between tasks on different cores
- Parallel execution of tasks in the correct order
- Access to shared resources
- Allocation of tasks to cores

In practice, an effect chain of a controller consists of multiple tasks with data dependencies. When tasks that share data are allocated to different cores, communication takes longer compared to single-core ECUs. Co-simulation supports the evaluation of predefined data age constraints.

Furthermore, data dependencies in the effect chain impose an order of execution on the tasks. Consequently, some of them may run in parallel. While this is efficient, the correct order in which the tasks are executed must be guaranteed. Otherwise, e. g. outdated signal values are used for calculating the control function.

Additionally, access to shared resources, such as semaphores, memories, and buses can be blocked due to parallel execution of tasks, adding delays. Not only the tasks of the controller, but also other functions on the same ECU use and block these.

The design space for the allocation of tasks to cores is considerable. Since the design aspects above are inter-dependent, additional delays are introduced. Constraints exclude infeasible solutions. Timing and functional co-simulation enables earlier verification and data-driven decision making when dealing with these aspects. One approach to the challenges of multi-core timing behavior is the logical execution time (LET) introduced by Henzinger et al. [12]. In terms of data exchange, it guarantees time determinism at design time and during execution on the target.

VI. Conclusions and Outlook

The use case in this paper showed the effects of task preemptions on a control system. However, on multi-core platforms more sources for undesired timing behavior of the controller software exist. Therefore, the engineering process for control systems needs to be supported by co-simulation during design phases.

The timing simulation solution used supports multi-core and the specific effects introduced above. In the future, we will work on use cases that involve multi-core controllers to investigate their specific timing effects on control systems further. This is a matter of use case definition and modelling, because our co-simulation approach and tool chain have the required capabilities.

References

[1] Y. Jordan, D. von Wissel, A. Dolha, and J. Mauss, "Full virtualization of Renault's engine management software and application to system development," in *9th European Congress on Embedded Real Time Software and Systems*, 2018.

[2] B. Wittenmark, J. Nilsson, and M. Torngren, "Timing problems in real-time control systems," in *American Control Conference*, vol. 3. IEEE, 1995, pp. 2000–2004.

[3] A. Cervin, D. Henriksson, B. Lincoln, J. Eker, and K. Arzen, "How does control timing affect performance? analysis and simulation of timing using Jitterbug and TrueTime," *IEEE Control Systems Magazine*, vol. 23, no. 3, pp. 16–30, 2003.

[4] D. Henriksson, A. Cervin, and K.-E. Årzén, "TrueTime: Simulation of control loops under shared computer resources," in *15th IFAC World Congress*. International Federation of Automatic Control, 2002, pp. 417–422.

[5] P. Pazzaglia, A. Biondi, M. Di Natale, and G. Buttazzo, "A simulation framework to analyze the scheduling of AVR tasks with respect to engine performance," in *7th Int. Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems*, 2016.

[6] F. Cremona, M. Morelli, and M. Di Natale, "TRES: A modular representation of schedulers, tasks, and messages to control simulations in simulink," in *30th Annual ACM Symposium on Applied Computing*. ACM, 2015, pp. 1940–1947.

[7] J. Fitzgerald and K. Pierce, "Co-modelling and co-simulation in embedded systems design," in *Collaborative Design for Embedded Systems*, J. Fitzgerald, P. G. Larsen, and M. Verhoef, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 15–25.

[8] J. Bouquet, S. Faure, F. Fève, M. Foucault, U. Garcia, F. Guérin, T. Hubert, F. Levy, S. Louvet, P. Munier, P.-N. Paton, and A. Spiewek, "Model quality objectives for embedded software development with matlab and simulink," in *9th European Congress on Embedded Real Time Software and Systems*, 2018.

[9] J. Holtmann, J. Meyer, and M. Meyer, "A seamless model-based development process for automotive systems," in *Software Engineering 2011–Workshopband*. Gesellschaft für Informatik e.V., 2011, pp. 79–88.

[10] C. Brandberg and M. Di Natale, "A SimEvents model for the analysis of scheduling and memory access delays in multi-cores," in *IEEE 13th International Symposium on Industrial Embedded Systems*, 2018, pp. 1–10.

[11] S. Kehr, E. Quiñones, B. Böddeker, and G. Schäfer, "Parallel execution of AUTOSAR legacy applications on multicore ECUs with timed implicit communication," in *52nd Annual Design Automation Conference*. ACM, 2015, pp. 42:1–42:6.

[12] T. A. Henzinger, B. Horowitz, and C. M. Kirsch, "Giotto: A Time-Triggered Language for Embedded Programming," in *Embedded Software*, ser. Lecture Notes in Computer Science. Springer, Oct. 2001, vol. 2211, pp. 166–184.