Closing the Loop: Towards Control-aware Design of Adaptive Real-Time Systems

Tobias Klaus*, Florian Franzmann*, Maximilian Gaukler†, Andreas Michalka†, Peter Ulbrich*
*Chair of Distributed Systems and Operating Systems
†Lehrstuhl für Regelungstechnik (Chair of Automatic Control)
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Germany
{tobias.klaus, florian.franzmann, max.gaukler,
andreas.michalka, peter.ulbrich}@fau.de

***Abstract*—Current trends, such as mixed-criticality real-time systems, boost performance by leveraging system dynamics and adaptivity. This, however, also amplifies challenges for control engineering and real-time-system design. Experiments with control applications indicate problems with traditional assumptions such as 'better response times will not decrease the quality of the results'. Therefore, future real-time-system designs need to be application aware. Just respecting deadlines will not be enough. A thorough understanding of which design decision has what effect on the control application is mandatory.**

From the perspective of real-time-system design, control applications often appear as a single monolithic block that performs the steps of reading sensors, computing control laws and setting actuators. In truth, the control system as originally designed consists of a host of small, interdependent activities. Although the respective meta-information exists, it is repeatedly masked during the design process, making it impossible to use this information for control-aware coordination. Even worse, it is impossible to assess design decisions w. r. t. the Quality of Control.

In this paper we present a report on work in progress on an important step towards control-aware design. We extract and retain the original semantics and dependencies of control models throughout the whole development process. Thereby we allow for traceability of control activities and thus an assessment of real-time operating system decisions based on Quality of Control. This flow of information from the real-time back to the control domain closes the loop between the two disciplines, enabling future mechanisms for automated tailoring and evaluation of real-time control systems.

I. Introduction

In the future more and more control applications will be consolidated onto one processing node, sharing a set of sensors. Additionally, current developments, such as the mixed-criticality approach or dynamically reconfigurable systems, are expected to have massive impact on system dynamics and adaptivity. As automatic control is particularly sensitive to timing variations [1] due to its close connection to the outside physical world, these trends will amplify existing problems when it comes to the execution of control applications on real-time systems: On the one hand, we expect that control applications will behave much more adaptively and thus will have a greater volatility when it comes to CPU as well as general resource utilization. This adaptation may be dependent on the current internal criticality level [2] or on external influences such as a missing GPS signal that makes the application skip location evaluation. On the other hand the consolidation onto one computing platform means that each application must compete for resources with a lot more applications that also tend to behave dynamically. Thus the control application is faced with a more dynamic computing system whose timing properties will be less predictable than they used to be, which is detrimental to the resulting Quality of Control (QoC).

The currently prevailing point of view is that system quality should be optimized by tightening temporal bounds. For example, mixed-criticality scheduling provides different timing guarantees in each criticality level, which are typically associated with the expectation of a certain quality. This implicitly assumes that an application's quality, here especially the QoC, can be approximated as a static function of its current timing conditions, an assumption often underpinning embedded control systems design [2–4]. In the following, we want to raise caution about the validity of such assumptions, even if they sound convincing for non-control applications such as real-time data transfer, and highlight the necessity of application-aware design of real-time control systems.

II. Motivating Example

Summarizing the assumptions stated before, real-time engineers may be tempted to assume that 'shorter response times are better'. Let's take one step back and put this seemingly obvious statement to the test with an experiment: The controller of an inverse pendulum, an application often used for benchmarking control and scheduling methods, was subjected to a switched input delay, as it might occur from mixed-criticality scheduling. The QoC was assessed by means of a standard quadratic cost function. The higher the amplitudes of error and control signal are, the higher the cost function is. Therefore a high cost means low system performance and vice versa.

A criticality change from a high level to a low one and back should result in the step delay pattern as shown in Figure 1, assuming that the response times for the critical tasks are lower in a high-criticality level because fewer tasks are competing for the same resources. From the previous assumptions, it would be intuitively expected that the QoC matches the response time pattern, following the mantra of 'low delay is good and high delay is bad'. However, the experiment shows a more complicated picture: After switching to a higher delay, the cost

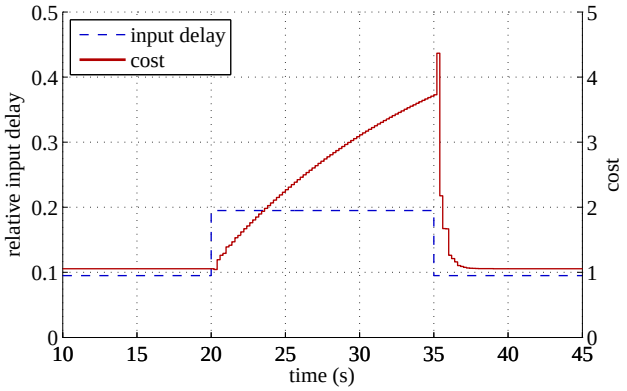

Figure 1. Performance of an inverse pendulum controller with discrete timing properties as anticipated for mixed-criticality scheduling. During low-criticality operation ($20 \leq t < 35$) the input delay was doubled, leading to a slow degradation of control performance, as can be deduced from an increase in cost. When the expected delay was reestablished ($t = 35$), the cost first unexpectedly overshot, then slowly decreased.

does not rise at once as expected. Instead, the control error slowly accumulates over the course of many seconds. When switching back to the lower input delay, surprisingly, the *cost rises at first*, and does not settle at a final low value until multiple control cycles have elapsed.

This counterintuitive behaviour shows that the relation between timing and control performance is non-trivial. In the context of applications that actually interfere with the environment, meeting deadlines is only part of the truth. Therefore, modern approaches to scheduling control applications e. g., in mixed-criticality systems, must incorporate application-specific knowledge and a close cooperation between computer science and control engineering seems to be of utmost importance [5].

Only with this knowledge application and real-time operating system (RTOS) can react correctly to changing criticality, performing mode switching as necessary while respecting all of the applications' requirements. Meeting these requirements mostly boils down to sharing resources more efficiently, which is a well-researched field. However, since mixed-criticality adds another dimension of complexity to this problem, a more thorough and detailed understanding of the applications' inner dependencies is crucial to meet their requirements according to criticality levels. In a first step towards our goal of improving the cooperation between RTOS and control applications we extract dependencies and semantic knowledge directly from Matlab/Simulink[1] models and allow for their full traceability throughout the whole development process. This allows for end-to-end assessment of execution conditions to determine the actual influence that RTOS decisions have on control systems, leading to new control-activity-aware coordination mechanisms.

III. Background

In some key aspects, the view control engineers have of a system is very different from the one of real-time-systems engineers. While control systems are designed in a model-based, data-flow-oriented fashion, computer scientists think in terms of jobs and tasks. Consequently, real-time engineers often perceive control applications as a single monolithic task, although these actually comprise lots of different activities which are connected by a data flow elaborately modelled by the control engineer in a domain-specific tool such as Simulink. Most of these activities are grouped by functional aspects e. g., Digital Signal Processing (DSP), state estimation or computing the controller's output, to keep the model maintainable and well arranged.

Moreover such data-flow models already include annotations about the anticipated execution period of sub-activities, since these are needed to correctly design discrete control systems. However, since the major criterion for grouping sub-activities is functionality, such blocks often are not modelled as individual top-level activities, but as members of larger multi-rate activities. For example a filter design for several sensors, each running at its own sampling time, will be mapped to one single-rate DSP activity instead of multiple single-rate ones by state-of-the-art code generators. It is obvious that this is bad for the utilization of the target real-time system, since the real-time engineer has to schedule the DSP's entire worst case execution time (WCET) with the rate of the highest-frequency sub-activity. In the past it was a cumbersome, yet beneficial manual task to mark each sub-activity as its own building block, generate its code and statically re-connect the data flow with RTOS mechanisms.

Since current generators generally only care about the top-level modular entities the architecture and data-flow modelled by the control engineer are hidden by mappings to static control flow like sequential function calls. Therefore these can neither be considered at design time nor be handled by RTOS primitives at runtime. Thus, the state of the art and currently the only option is to design the system under simplifying assumptions while overprovisioning the design to counter the negative impact of ignoring the system semantics.

Most of the activities just described behave exactly as anticipated by real-time engineers: They are activated periodically and have to be scheduled before their deadline which is normally located at the end of the current period. A totally different kind of module are actuators and sensors. Since these interact with the controlled object directly, the control engineer has to make certain assumptions about their timing properties. From the control engineer's point of view, meeting these requirements is what scheduling should be about. However, even in real-time control systems without mixed criticality the specific semantics of the different control blocks are mostly neglected by real-time engineers and scheduling algorithms. Even worse, the response times of the constituents remains unknown and cannot be provided to the control engineering design processes.

A tool already able to handle fine-grained dependency information is the Real-Time Systems Compiler (RTSC) [6], which is capable of analyzing, transforming and consolidating real-time applications. It accepts their source code and system

[1] http://mathworks.com/products/simulink/

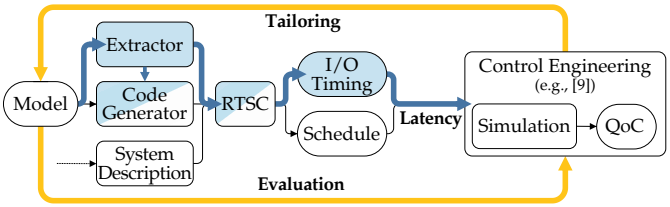

Figure 2. Continuous toolchain for the co-design of real-time control systems. Building upon previous work on the RTSC, we added extensions (shaded elements) to analyse and extract semantic information from control models (i. e., Simulink). Consequently, execution latencies can be traced back to their respective control activities, enabling simulation and evaluation of the resulting Quality of Control at compile-time. Ultimately, knowledge and traceability of I/O timing paves the way for automated co-design approaches.

description (i. e., additional information that cannot be extracted from the source) as input. The input is analyzed, tasks and subtasks are identified, and, as a result, graphs of Atomic Basic Blocks (ABBs) are generated. These trace data and control flows in an abstract manner, with individual ABBs spanning application code between synchronization points (i. e., syscalls). Consequently, ABB graphs serve as an RTOS and real-time-architecture agnostic intermediate representation of real-time systems. Important information such as task release times, periods, WCET, etc. are extracted from the real-time application. Annotating these in the ABB graphs allows the RTSC to rapidly reconfigure the system, to, for example, generate time-triggered multi-core systems from an event-triggered input [7]. For traditional real-time systems the RTSC already has the necessary abilities to resolve complex dependencies and generate tailored real-time systems. However, currently the RTSC does not grasp complex control applications and cannot identify the semantic properties of components such as sensors, actuators and computing blocks, which would allow it to trace the control system's data flow.

IV. Control-aware System Analysis and Transformation

To address these issues, we propose a continuous toolchain from the original control system model to the resulting schedules, the overall structure of which is illustrated in Figure 2. As a first step towards control-aware yet adaptive system design, we focused on a global, context-sensitive analysis of control and data flow across control-application layers and threads of execution. This step is of vital importance as these dependencies manifest differently on the various levels of abstraction. Since the fine-grained dependencies between sub-activities in the control model are usually masked by coarse-grained thread-based dependencies on the RTOS level we have to extract and retain this information as well as the semantic knowledge about control applications for further use. Consequently, we enriched the ABB graphs with additional semantic knowledge about control activities – i. e., what ABBs represent sensors and actors or what range of physical values they require – and adapted the RTSC's transformations to retain this information so that all real-time activities can be traced back to the control system's building blocks. As a next step we implemented an additional front-end, capable of extracting such semantic knowledge and data-flow graphs from Simulink models. This is the tool most control engineers use, and we wanted to avoid introducing yet another abstraction layer, which might mask the dependencies defined by the control application again. Therefore, we used Matlab and its Simulink scripting interface to examine the model itself. In addition to the extraction of ABB graphs this analysis also instruments the integrated Simulink Coder to generate dedicated C code for each activity. Here, the smallest objects of investigation are currently limited to subsystems due to the inability of the Simulink Coder to generate code for smaller units. Thus, we expect the model to be designed in a modular way, grouping functional blocks in corresponding subsystems that may include nested multi-rate subsystems. To remedy the already depicted problem of such multi-rate systems all elementary subsystems are determined and generated as separate activities. This is done using a depth-first search identifying all subsystems that can be executed in one continuous piece with a single period i. e., the model is decomposed along temporal borders instead of functional aspects.

Since we assume that the examined control model does not only consist of blocks that are part of the control system, and therefore relevant at run time, but also includes subsystems that are used only during system design for simulation purposes, these have to be identified automatically. First, all blocks that do not influence other blocks i. e., these do not have connections to output blocks via ports or other Simulink constructs, such as goto blocks, are classified as *debugging blocks* and therefore discarded. Subsystems flagged as *continuous-time* by Simulink are more important, since these represent the plant for simulation purposes. Though these blocks are not to be executed on the real-time system they denote start and endpoints for the control application. Blocks whose inputs are connected to such simulation subsystems are considered *sensor activities*. Likewise, blocks that connect to the environment simulation with output ports are interpreted as *actuator activities*. In order to maintain the model's structure the data flow from sensors to actuators is traced and cycles are identified and broken. In the RTSC the extracted activities and dependencies are used to connect the system's ABB graphs. In addition to the code generated by Simulink Coder, glue code capable of connecting the extracted control-activities, including some simple continuous blocks such as 'plus' or 'minus', between generated subsystems is added.

Figure 3 depicts first results of this automatic decomposition when applied to the Simulink model of the I4*Copter* [8], which is a multi-rate DSP and control system. It compares the utilization and modularity of different approaches for mapping control activities to RTOS mechanisms. The 'Simple' approach mapped the entire application to a single large software module, the 'Manual' approach identified six functional modules and our 'Automated' tool-based approach found all 36 atomic control activities. Scheduling these small modules with their individual periods led to a 55 % decrease in utilization compared to the

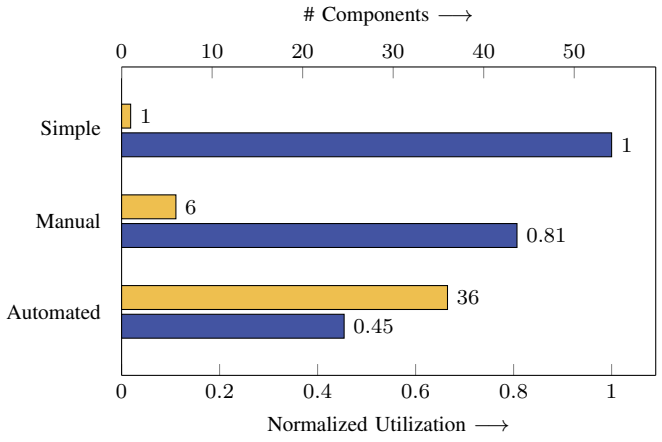

Figure 3. Comparison of the utilization of the monolithic vs. the manually and the automatically decomposed control model of the I4*Copter*. The fine-granular decomposition of the 'Automated' approach into 36 activities decreased utilization by 55 % in comparison to 'Simple' and by 36 % when compared to the 'Manual' approach.

'Simple' approach where one single module had to be scheduled with the highest rate present.

V. Outlook: Closing the Loop

As the motivating example at the beginning of this paper shows, classical timing properties such as met or missed deadlines are of limited use for the actually intended result: *the Quality of Control*. An all-temporal design and scheduling approach may lead to overprovisioning of resources or, for example, fail to provide the required QoC at runtime during mode switches in mixed-criticality settings.

In this paper we highlighted the need for computer science and control engineering to collaborate closely when designing and implementing control systems so as not to lose important information about the system along the way. Also, we emphasized the importance of end-to-end traceability of control-activities to determine the influence of RTOS decisions on the QoC.

A starting point on the way towards control-aware design is the toolchain presented in Figure 2. Here, dependency information acquired by analysis of the control model is used by the RTSC to generate schedules, from which the sensor and actuator timing can be extracted. Building upon this, the presented approach allows an assessment based on physical effects, facilitating the efficient design of future highly dynamic control systems.

In the future we intend to use the meta-information gleaned from the control engineering model to improve the suitability of scheduling decisions. Here the RTSC will show its full potential, performing deeds at compilation time that ordinarily are executed by the RTOS at runtime. The end-to-end traceability allows for a seamless flow of information between the domains of real-time and control engineering, allowing integration of a variety of already existing as well as future co-design approaches: by knowing the exact delay between input and output of each control system in a static schedule generated by the RTSC, it becomes possible to adjust the control design as proposed in [9]. Taking one more step, the RTSC can be used to simulate schedule switches by calculating different schedules for various criticality states and tracing the application's activities from Simulink to the scheduler and back to Simulink. This facilitates analysis of the QoC using Simulink's simulation or tools such as Jitterbug [1].

Vice versa, control knowledge could also be passed to future control-application-aware RTOSes, leveraging the QoC at runtime for adaptive scheduling and resource coordination.

Overall, the design process is no longer a one-way street from the control model to a generated real-time system in which important information is lost at every step. The flow of information from the real-time back to the control domain *closes the loop* between the two disciplines, enabling future mechanisms for automated tailoring and evaluation of real-time control systems.

References

[1] A. Cervin, D. Henriksson, B. Lincoln *et al.*, '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.

[2] G. Buttazzo, M. Velasco and P. Marti, 'Quality-of-control management in overloaded real-time systems', *IEEE Trans. on Computers*, vol. 56, no. 2, pp. 253–266, 2007.

[3] A. Cervin, J. Eker, B. Bernhardsson *et al.*, 'Feedback–feedforward scheduling of control tasks', *Real-Time Systems*, vol. 23, no. 1-2, pp. 25–53, 2002.

[4] F. Flavia, J. Ning, F. Simonot-Lion *et al.*, 'Optimal online (m,k)-firm constraint assignment for real-time control tasks based on plant state information', in *IEEE Intl. Conf. on Emerging Technologies and Factory Automation, 2008. ETFA 2008.*, 2008, pp. 908–915.

[5] D. Simon, A. Seuret and O. Sename, 'On real-time feedback control systems: Requirements, achievements and perspectives', in *1st Intl. Conf. on Systems and Computer Science (ICSCS), 2012*, Aug. 2012, pp. 1–6.

[6] F. Scheler and W. Schröder-Preikschat, 'The real-time systems compiler: Migrating event-triggered systems to time-triggered systems', *Softw. Pract. Exper.*, vol. 41, no. 12, pp. 1491–1515, 2011.

[7] F. Franzmann, T. Klaus, P. Ulbrich *et al.*, 'From intent to effect: Tool-based generation of time-triggered real-time systems on multi-core processors', in *19th IEEE Int. Symp. on OO Real-Time Distributed Computing (ISORC '16)*, Washington, DC, USA: IEEE, May 2016.

[8] P. Ulbrich, R. Kapitza, C. Harkort *et al.*, 'I4copter: An adaptable and modular quadrotor platform', in *26th ACM Symp. on Applied Computing (SAC '11)*, (TaiChung, Taiwan), New York, NY, USA: ACM, 2011, pp. 380–396.

[9] Y. Xu, K.-E. Årzén, A. Cervin *et al.*, 'Exploiting job response-time information in the co-design of real-time control systems', in *21st IEEE Int. Conf. on Embedded and Real-Time Comp. Systems and Applications*, 2015.