A Practical Approach to the Simulation of Safety-critical Automotive Control Systems considering Complex Data Flows
Sébastien Dubé, Mesut Özhan, Achim Rettberg

To cite this version:
Sébastien Dubé, Mesut Özhan, Achim Rettberg. A Practical Approach to the Simulation of Safety-critical Automotive Control Systems considering Complex Data Flows. 8th European Congress on Embedded Real Time Software and Systems (ERTS 2016), Jan 2016, TOULOUSE, France. hal-01291352

HAL Id: hal-01291352
https://hal.archives-ouvertes.fr/hal-01291352
Submitted on 21 Mar 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
A Practical Approach to the Simulation of Safety-critical Automotive Control Systems considering Complex Data Flows

Sébastien Dubé  
Hella Engineering France  
Toulouse, France  
Email: sebastien.dube@hella.com

Mesut Özhan  
INCHRON GmbH  
Potsdam, Germany  
Email: oezhan@inchron.com

Achim Rettberg  
Hella Electronics  
Lippstadt, Germany  
Email: achim.rettberg@hella.com

I. ABSTRACT

Embedded systems highly contribute to the efficiency, safety, and usability of our present-day means of transport like cars and airplanes. Due to the possible hazards and risks involved with their operation, safety standards like DO-178C for avionics and ISO 26262 for automotive commend the application of methods and tools according to the state of the art. Functional safety requirements imposed on hardware and software imply the detection of malfunctions and taking corrective actions, before hazards actually occur. As described in [5] one of the key challenges thereby is the prediction and verification of the system’s timing behavior. In this paper we describe a model-based approach for real-time simulation focusing on complex end-to-end data flows typically encountered in safety-critical automotive control applications. Based on first-hand experiences gained during the development of an electrical power steering control system, we illustrate how real-time simulation models can be utilized to guide design decisions, and help to achieve safety goals defined at system level. Furthermore, we discuss the issues of response time analysis for dynamic state-dependent data flows considering different semantics for communication in the context of the AUTOSAR standard.

II. KEYWORDS

Automotive Control Applications, Electronic Power Steering, Fault Tolerant Time Interval, Response Time Analysis, Real-Time Simulation, AUTOSAR

III. MOTIVATION

Engineers developing electrical/electronic systems for the automotive or avionics domain have to manage a high degree of functional, technological, and organizational complexity. Design and implementation of highly dynamic safety-critical applications make a model-based approach with careful consideration of fault tolerance indispensable. A fault-tolerant design follows a safety strategy which defines the (worst) conditions that a system must cope with, and defines safety mechanisms for fault detection and error handling that must be implemented. Such a design ensures that a system experiencing malfunctions remains operational – possibly with reduced functionality – and transits into a safe error-free state.

Safety mechanisms built into a system are liable to hard real-time requirements. Hence, the transition into a safe state must be achieved under all possible conditions with respect to a limited time span, the so called fault tolerant time interval (FTTI) as defined in [23]. However, if one considers the many different factors that influence a system’s timing behavior, e.g. the number of possible faults and failure states, the complexity of the data and control flow, or the scheduling properties of different hardware and software variants, it turns out, that finding a robust and reliable dynamic architecture design is a very challenging task. The problem of verifying the schedulability of processes and messages in a distributed system is NP-hard [7], and besides hard real-time requirements, for safety-critical systems additional requirements must be considered in order to guarantee dependability.

During the past decades, since Liu and Layland in [19] presented their work about rate-monotonic scheduling (RMS), research in scheduling theory has developed a large number of concepts and mathematical proofs in order to take on the challenges presented by the analysis and optimization of embedded control applications. But, despite the many advances in extending their scope and applicability as described by Sha et al. in [22], most theoretical approaches remain limited to specific use cases as typically some of their constraints are violated in real industrial systems. Examples for those limitations and constraints are given by Cooling in [9] and Davis et al. in [10]. A more recent study taking into account precedence relations between tasks is provided by Kermia in [16].

The software standards developed by the Automotive Open System Architecture (AUTOSAR) partnership [3] have dramatically changed the way automotive systems are built today, and the transformation of processes, methods, and tools is not yet finished. Since AUTOSAR release 4.0 there is also a dedicated specification [4] for the formalized description of timing properties and constraints available. And although, this helps to identify and exchange timing-related information in a complex automotive supply chain, AUTOSAR does not address the issue of how this information can be obtained, or a timing analysis can be performed.

In order to evaluate a system’s performance, interoperability, robustness, and eventually its safety, the specific characteristics of the AUTOSAR operating system and run-time en-
environment (RTE) must be considered on implementation level. These comprise concepts like OS applications, IOC, shared (multi-core) resources, schedule tables, and extended tasks. For AUTOSAR compliant real-time systems, Anssi et al. in [1] presented a study, where the applicability of schedulability analysis is evaluated using an open source implementation [12] of Palencia’s and Harbour’s algorithm [21]. Additional evaluations are provided by Hladik et al. in [15].

Viewed from the system engineering perspective eventually one key issue remains: Even if state of the art scheduling analysis methods can be applied, they can only provide true or false statements on the feasibility of a given system configuration. However, in order to perform design modifications efficiently, engineers need more fine-grained information about their system’s dynamic behavior, e.g. obtained by statistical analysis of trace data coming from target measurements or real-time simulation.

IV. Case Study: Electronic Power Steering

The research results presented in the following are based on the observations and experiences made by the authors during the development of an electronic power steering (EPS) system at Hella Engineering in Toulouse, France. Due to the safety and hard real-time requirements for this system, a model-based approach using SysML at logical architecture level, and AUTOSAR methodology at technical architecture level was implemented. For the simulation, visualization, and exploration of various design alternatives methods and tools [14] provided by INCHRON were also used from the very beginning of this project.

A. System overview

The electronic power steering system in our case study as depicted in figure 1 uses a brushless motor to assist the driver in steering his vehicle. Position and torque of the steering column are permanently measured by sensors and processed by the steering control module (SCM), which calculates an assistive torque that is applied depending on different driving conditions. Advantages of an electronic power steering system over a comparable hydraulic solution are improved fuel efficiency and greatly simplified manufacturing and maintenance processes. Also, in combination with an electronic stability control it vastly contributes to improved driving safety.

B. Objectives

A major challenge during the development of embedded systems such as the electronic power steering is the verification of end-to-end latency requirements. The difficulty lies in the fact that these systems feature many different functional and dysfunctional modes of operation with corresponding hard real-time requirements for monitoring, error detection, and error handling. Depending on the required safety level, the implementation of these safety mechanisms in addition to the actual control functionality, drastically increases the complexity of the data and control flow. As a consequence for the EPS system development, following a classical integration and test approach solely based on measurements on the target hardware, seemed not feasible. In order to reduce the time and effort, that it takes to find resource bottlenecks, timing errors, and eventually to verify the real-time requirements, it was one of our main objectives in this case study to apply state-of-the-art methods and tools, aiming at a virtual integration of the system in earlier development phases. Furthermore, the verification of an embedded system requires that timing properties and constraints are specified in a formal, unambiguous way, matching with the semantics of the formalism which is used for the analysis. Thus, another objective was to check if all the system information, needed by either formal schedulability analysis or model-based scheduling simulation, is available in a real project environment. Finally, we wanted to achieve a seamless integration of all methods and tools avoiding redundant modeling of the same information as much as possible.

C. Design of the dynamic architecture

An important part of the dynamic architecture design for the SCM was planning the execution of tasks and interrupt service routines (ISR) on the main microcontroller. The initial design was created according to the strategy of deadline-monotonic scheduling (DMS) [18] where tasks are assigned priorities depending on their deadline with priorities being inversely proportional to the length of the deadline. Compared to RMS, deadline-monotonic priority assignment is an optimal strategy for tasks that can have a deadline equal or smaller than their activation period. Furthermore, Audsley and Burns [2] [7] showed that with additional schedulability tests, the deadlines of sporadic tasks can be guaranteed within the deadline-monotonic theory.

Table I shows the initial scheduling configuration for the SCM according to the deadline-monotonic priority assignment strategy. In order to preserve most of the scheduling characteristics guaranteed by DMS, the SCM design was restricted to only use time-triggered, basic tasks with no priorities shared between two different tasks.

For the model-based simulation with the INCHRON ToolSuite a project file (.ipr) of this configuration needed to be

---

1 Note, that due to intellectual property rights of the companies involved in the development of the SCM, only anonymized and simplified versions of the original system information can be documented.
created. As suggested by the TIMMO-2-USE project in [11], there are basically two different possibilities to create a model for the simulation: either top-down based on requirements and design specifications in an early project phase, or bottom-up by reverse-engineering the necessary information based on an existing (prototype) implementation of the system. In this case study the initial simulation model was generated by importing the AUTOSAR ECU configuration data of a prototype sample into the INCRON Tool-Suite.

Another very important input parameter required for the schedulability analysis and simulation is the core execution time (CET) of the scheduled processes. Obtaining actual and reliable execution time information in reality is not an easy task. Even if sophisticated measurement capabilities are available, it can be very difficult or nearly impossible to do measurements for all possible operation states and (error) conditions in many different variants. An alternative to measurements on the target hardware is to predict worst case execution times by using static code analysis, e.g. as suggested by Ferdinand in [13]. However, this approach also has its limitations: for example increasingly complex caching and pipelining solutions found in modern microcontrollers, make it very difficult to predict the time consumption of machine code instructions.

In the case study, the model-based simulation of different scheduling scenarios was performed by using execution time information which was measured on the target hardware. Measurements were taken for various operation states, whereas focus was laid on those configurations that covered the most complex execution paths (in case of the control functions). In order to further increase the confidence into the results, we applied scaling factors to some of the measured values. However, it should also be noted, that for the verification and optimization of end-to-end latencies in a distributed system, time delays caused by asynchronous process executions can be more important, than the deviation of individual execution times.

D. Event chains with hard real-time latency requirements

A temporally ordered sequence of correlated events, that can be observed or measured in a system, is referred to as a chain of events, or event chain. Applied to embedded real-time systems, the concept of an event chain can be used to specify a sequence of function executions and (communication) data flows between them, which are subject to safety and real-time requirements. Figure 2 shows an example for such an event chain in the SCM, starting with the sampling of the torque sensor and ending with the control of the motor realizing the steering assistance.

The event chain concept is a very useful abstraction in order to describe the scope of an end-to-end latency requirement from the perspective of a control or system engineer, considering the influence of both the hardware and the software. In the automotive industry it is a widely used concept, but only since its formalization by the AUTOSAR standard, the semantical pitfalls of ambiguous textual and visual descriptions could be mitigated.

A key issue in the formal semantics of an event chain concerns the definition of data flow properties: the data flow in a system results from the production, transmission, and consumption of data by different hardware and software functions. In a distributed system the deployment (location) of a function mostly determines the means by which this function is able to communicate with other local or remote functions, e.g. via messaging over a network, shared variables etc. As the interpretation of an event chain depends on the behavior of the functions in its scope, and again their behavior depends on the value, state, order, and age of the processed data, a formal semantics of the data flow needs to take the different communication means and their characteristics into account. At the same time, the mathematical model which is used by the analysis or simulation, must be able to handle that formalism.

Initially, at the beginning of the case study, the data flow between hardware and software components at system level was modeled by using the flow port concept of SysML [20] as depicted in the figure 3. This modeling view helps to understand the relationships between architectural blocks, differentiated between hardware and software. However, some essential data flow characteristics as described above, were not supported by that concept. Furthermore, we also tried to derive a communication model from the AUTOSAR meta-model, but that approach turned out to be too complex, as the specification of the communication and the dependencies between basic software modules is significantly different from the concepts used for the application software. A further restriction was the
difficulty to adequately describe the behavior and influence of
hardware functions.

Eventually, in the case study a rather simple but sufficiently
powerful formalism, derived from the co-design methodology
for embedded systems (MCSE) [8], was used for the specifica-
tion of data flows as follows:

- **Shared variable** (or permanent data): Data which
can be read / written at any time. In order to avoid
data inconsistencies, the access to shared variables
must be protected against simultaneous read and write
operations by asynchronous processes.

- **Event**: An (activation) event that triggers the execu-
tion of a process.

- **Queued**: Data is buffered in a queue with FIFO
mechanism where the oldest data in the queue is read
first. In control theory the queue size usually directly
depends on the execution periods (frequency) of the
producing and consuming processes.

Figure 4 shows the application of the co-design methodol-
ogy for the modeling of the SCM system architecture.

This model describes the main hardware and software
functions, and also the data and control flow connections
according to the MCSE semantics. Compared to the basic
SysML semantics (see figure 3), the most important differences
are as follows:

- **Rich data flow semantics**: MCSE supports different
types (shared variable, event, queued) for data flows,
which allows abstract modeling of the most common
communication mechanisms encountered in real sys-
tems.

- **Activation flow**: Event-triggered activation of pro-
cesses is explicitly supported by MCSE, but not the
basic SysML model.

- **Timing properties**: Task periods, interrupt inter-
arrival times, and other timing parameters can be
specified as properties of a (software) block.

- **Closer to AUTOSAR model**: MCSE is closer to
the AUTOSAR meta-model (the SCM is based on
AUTOSAR), and offers better support for the concepts
used in the specification of the software architecture.

In the case study, this model was used as a starting point for
the definition of the data and control flow in the INCHRON
Tool-Suite. Basically, the MCSE concepts could be mapped
one-to-one to semantically equivalent communication concepts
in the tool. A further advantage of this approach is that event
chain definitions for the functional and dysfunctional operation
states of the system can be described in a common model,
using the same data and control flow concepts. For this paper,
we have selected two (dysfunctional) event chains, in order to
demonstrate the application of our approach for the verifica-
tion of end-to-end FTTI requirements. Figure 5 shows the sequence
of process executions for each event chain similar to the
definition in the simulation tool.

They start with the detection of a failure at different
signal sources, but then follow the same sequence of function
executions until they end with the transition into a safe state
(here disabling the application of the motor torque to the
motor power stage). Arrows between the processes represent
a (typed) data flow connection in the model, which are then
traced and highlighted in the simulation (see figures 10, 11,
13, 14).

In general, a failure of the system can have different
underlying causes originating from either the environment
(e.g. the system operating under conditions different from its
specification) or the system itself (e.g. a malfunction of an electrical, electronic, or mechanical component). However, at software level for the verification of FTTI requirements, this difference can be neglected. For example, if the system needs to shutdown when sensors indicate a too high temperature, we do not need to care, if this indication is caused by a sensor malfunction, or the system temperature being actually too high. Regardless of the failure source, the dynamic architecture design must ensure, that monitoring, error detection and error handling processes are executed in a deterministic way, and meet the FTTI requirements.

In a real system the malfunction causing the system failure can occur at any time, and in order to verify the compliance of the dynamic software architecture with the FTTI requirements, it is necessary to recreate the different scheduling situations in the simulation. Therefore, it is not sufficient to induce an error just once, but it must be done repeatedly over the time of the simulation. For the SCM the stimulation generator of chron-SIM was used to define an environment model (scenario), that would randomly induce an error with respect to the activation period of the concerned monitoring or error detection process. For example, if the error detection is executed every 10ms, then a uniform random variation between 0 and 10ms was chosen for the error induction. Figure 6 shows the stimulation scenario defined for the different errors induced during the simulation.

Figure 6. Stimulation scenario defined with the INCHRON Tool-Suite

The main benefit of this approach is, that it allows to reach a high coverage of the various, relevant preemption situations in a very short time – much shorter than in a hardware-in-the-loop (HIL) or prototype test environment.

V. SIMULATION AND OPTIMIZATION

Using the INCHRON Tool-Suite we performed several simulation runs in order to compare alternative scheduling configurations for the SCM. The traces generated by the simulation were evaluated according to the following quality criteria:

- Deadline violations
- Response time distribution
- CPU peak load in certain averaging intervals
- Start-to-start jitter
- End-to-end latencies of dedicated event chains

According timing requirements were specified directly in the simulation tool, and evaluation statistics were automatically generated after each iteration. After analyzing these results, the (scheduling) configuration was modified manually, and a new iteration was started. The most important observations and conclusions made during the case study, are presented in the following.

A. Adjustment of start offsets

Figure 7 shows an excerpt from the simulation trace of the initial system configuration as specified further above in table I: process executions (indicated by green areas within the rectangular process box) and preemptions (white areas) by higher priority processes are depicted as a Gantt diagram.

Figure 7. Gantt chart view of processes inside the SCM

One observation made during the analysis of simulation results for the initial scheduling configuration was, that CPU load peaks can occur when processes with relatively large execution times are activated at the same time. For example the time-triggered processes ASIC_Manager and Analog_Manager both with a period of 1ms were affected. Furthermore, Com_Tx and Com_Rx with a period of 5ms also had this issue. An obvious solution to smooth the peak load, would be to introduce a time delay (offset) between the activations of the processes in question. For example for the ASIC_Manager task, an activation offset of 500µs against the Analog_Manager task seemed feasible. However, the granularity of the underlying OS counter tick was just 1ms, and thus shifts between the activations were only possible in steps of 1ms. For this reason, the OS configuration was modified, and the granularity of the OS counter tick was reduced to 500µs.

Table II. ADJUSTED START OFFSETS FOR ASIC_Manager AND Com_Tx

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Period (µs)</th>
<th>Deadline (µs)</th>
<th>Offset (µs)</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASIC_Manager</td>
<td>Task</td>
<td>1000</td>
<td>1000</td>
<td>500</td>
<td>95</td>
</tr>
<tr>
<td>Com_Tx</td>
<td>Task</td>
<td>5000</td>
<td>5000</td>
<td>3500</td>
<td>45</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The activation offsets for ASIC_Manager and Com_Tx were adjusted as shown in table II.

B. Deadline violations

In the case study deadline requirements were defined for all processes. These were monitored and checked during the

2The worst-case response time determined for the ASIC_Manager task was lower than 500µs.
simulation of the different scheduling configurations. The simulation tool shows the evaluation of all (response time and other) requirements in a dedicated requirements evaluation view, where the number of successful, critical, and failed checks for each requirement is summarized (see figure 8). A concrete response time check is considered critical, if a certain predefined margin relative to the actual deadline is exceeded.

For example, we can see that the response time of Com_Rx is considered critical, as it exceeds the safety margin. Overall, we observed no hard deadline requirement violations in the case study.

C. Execution jitter

In order to monitor the efficiency of a specific scheduling configuration, it may be necessary to monitor additional process statistics. A very useful indicator is the execution jitter. Figure 9 for example shows the time distribution of the start-to-start (purple bars) and terminate-to-terminate jitter (green bars) for the Com_Rx task.

In the histogram, we can see that the start time of Com_Rx fluctuates between 4.98 and 5.03 ms due to preemptions caused by interrupts and higher priority tasks.

D. Evaluation of event chain 'Temperature Error'

A Temperature Error is indicated by the task Temp_Manager when the temperature value measured by the temperature sensor exceeds a predefined threshold. If this is the case, the Mode_Manager task shall switch into a corresponding failure mode, and send an error notification PWM_Stop to the power stage which is driving the motor. According to the system specification, the end-to-end latency requirement goal for this event chain was 12 ms.

The simulation of the SCM with chronSIM predicted, that violations of the end-to-end latency requirement are possible, although no activation violations occur and all processes meet their deadline requirements. An example from the simulation which shows such a requirement violation is given in figure 10.

In this particular case, we can observe that the end-to-end latency mainly arises from the 10 ms period of the Temp_Manager task, and from the additional delay (almost 3 ms) between the execution of the Temp_Manager and the Mode_Manager task. One possible solution would be to increase the execution frequency of the Temp_Manager task, in order to reduce the time delay which occurs after the provision of new sensor values by the Analog_Manager task. However, this would also increase the CPU load.

Alternatively, a solution would be to reduce the time delay between the activation of the Temp_Manager and Mode_Manager task, by adjusting the start offsets. Furthermore, it is also necessary, that Temp_Manager gets a higher priority than Mode_Manager, so it is scheduled first. Although, this priority change contradicts the DMS strategy – Temp_Manager has a bigger deadline than Mode_Manager – it seems to be the most feasible solution to reduce the event chain latency.

Table III. Adjusted Scheduling with OS Counter Tick of 500us

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Period (µs)</th>
<th>Deadline (µs)</th>
<th>Offset (µs)</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>Torque_Manager</td>
<td>Task</td>
<td>1000</td>
<td>100</td>
<td>0</td>
<td>200</td>
</tr>
<tr>
<td>Analog_Manager</td>
<td>Task</td>
<td>1000</td>
<td>500</td>
<td>0</td>
<td>102</td>
</tr>
<tr>
<td>ASIC_Manager</td>
<td>Task</td>
<td>1000</td>
<td>1000</td>
<td>500</td>
<td>95</td>
</tr>
<tr>
<td>Temp_Manager</td>
<td>Task</td>
<td>10000</td>
<td>10000</td>
<td>2000</td>
<td>71</td>
</tr>
<tr>
<td>Mode_Manager</td>
<td>Task</td>
<td>5000</td>
<td>5000</td>
<td>2000</td>
<td>70</td>
</tr>
<tr>
<td>Com_Rx</td>
<td>Task</td>
<td>5000</td>
<td>5000</td>
<td>2000</td>
<td>50</td>
</tr>
<tr>
<td>Com_Tx</td>
<td>Task</td>
<td>5000</td>
<td>5000</td>
<td>3500</td>
<td>45</td>
</tr>
<tr>
<td>Memory_Manager</td>
<td>Task</td>
<td>10000</td>
<td>10000</td>
<td>9000</td>
<td>5</td>
</tr>
<tr>
<td>Diagnosis</td>
<td>Task</td>
<td>10000</td>
<td>10000</td>
<td>8000</td>
<td>2</td>
</tr>
</tbody>
</table>

After the adjustment of the scheduling as defined in table III, a rerun of the simulation shows, that the end-to-end latency of the event chain now remains under the deadline of 12 ms. An example for a successful evaluation of the requirement is shown in figure 11.
Figure 11. Successful evaluation of end-to-end latency requirement (Temperature Error)

The distribution of the end-to-end latency for the event chain Temperature Error predicted by the simulation is shown in figure 12.

Figure 12. End-to-end latency distribution for event chain (Temperature Error)

According to these results, the latency of the event chain ranges from 0.3ms up to 10.4ms.

E. Evaluation of event chain ‘Analog Input Error’

In addition to the main temperature sensor, the SCM processes temperature information from two additional temperature sensors in different locations. These are connected to the SCM via two multiplexed (switched) analog inputs. The multiplexing is controlled by an ASIC using synchronous SPI bus communication. Depending on the SPI pin selection controlled by the ASIC_Manager, the ADC reads the sensor signal from either Analog_Input 1 or Analog_Input 2.

As shown in figure 13, the switching between the analog inputs every 6ms can lead to a delay for the recognition of a potential Analog Input Error. In this situation, we can see that an input error propagating from Analog_Input 1 is not processed immediately by the Analog_Manager task, but only after an additional switching cycle, when Analog_Input 1 is selected again after Analog_Input 2 was read. Considering the simulation results we can see that the switching frequency is highly relevant for the optimization of the end-to-end event chain latency. If we increase the frequency (decrease the period) switching inputs every 1ms, thus accepting a slight increase of the CPU load, we can reduce the time delay for the error indication to propagate and reach the power stage in less than 12ms as depicted in figure 14.

Figure 13. Violation of end-to-end latency requirement (Analog Input Error)

Figure 14. Successful evaluation of end-to-end latency requirement (Switched Analog Input Error)

VI. COMPARISON OF CPU LOAD FOR SCHEDULING ALTERNATIVES

Finally, after the optimization of the end-to-end latencies for the different event chains, we compared the CPU load characteristics of the different scheduling alternatives, as described in table I and III.

Figure 15. Simulated CPU load average for initial configuration
Figure 15 shows the CPU load for the initial configuration of the SCM before the optimization, and figure 16 shows the CPU load for the final configuration. In both cases the CPU load curve was calculated for a smoothing interval of 1ms with a granularity of 10μs.

We can observe that the CPU load peaks in the initial configuration reach up to 64%, and are slightly higher than those observed in the optimized configuration. The decrease from 64% to 58% was achieved mainly by the re-adjustment of the start offset for the ASIC_Manager task.

VII. GENERIC DESIGN RULES LEARNED FROM THE CASE STUDY

A key question for the optimization of the scheduling in the SCM was to know which configuration parameters can be changed, and which changes are the most effective in order to fulfill the timing requirements. The theory for preemptive fixed-priority scheduling usually focuses on the optimization of process priorities and relative activation offsets. As we have shown in this paper, the optimization of offsets – even when it is done manually – is very effective to minimize the number of context switches\(^3\) and to smooth the CPU load in time intervals which show high CPU load peaks. At this, it is important to follow the precedence relations imposed by the data flow in the system, as otherwise the end-to-end latencies of event chains will be unnecessarily prolonged. However, we have also shown that for the optimization of event chains, it can be very useful to consider changing also other aspects of the system configuration, like activation periods, or the number and decomposition of processes:

- **Periods of time-triggered processes**: Usually, the activation period of a time-triggered processes is deduced from the system requirements of the functions allocated to this process. Nevertheless, in some cases the activation period can be relaxed without violating any constraints. For example in many control systems, processes concerned with the sampling and (pre-)processing of sensor data are usually scheduled with a higher frequency than necessary, in order to compensate for unpredicted scheduling effects.

- **Number and decomposition of processes**: The number of processes and their decomposition, as a matter of fact the allocation of basic and application software functions (or runnables) to processes, is guided by both functional and safety requirements. As the end-to-end latency of an event chain results from a predefined, sequential chain of function executions and data flows, and thus depends on the timely interaction of the involved processes, it can only be optimized under consideration of the underlying process architecture. In a complex control system, consisting of many event chains with different criticality, designers must find a trade-off between the separation of concerns driven by safety requirements, and the compliance with real-time requirements imposed by the functional domain.

- **Execution order**: In general, the execution order of functions within one process should follow the data flow between the functions. In some cases, additional design techniques must be employed to break up feedback loops in the data flow.

- **Core affinity**: On multi-core processors, the core affinity of a process defines on which core(s) this process is allowed to execute. In order to avoid expensive cross-core communication between processes, functions of the same event chain should not be allocated to processes which execute on different cores.

VIII. EXTENDED TIMING-AWARE CO-DESIGN METHODOLOGY

Based on the experiences made in the case study, we have enhanced our development process for the verification of (safety-critical) event chains with hard real-time requirements. A seamless workflow as depicted in figure 17 combining timing measurements on the target hardware with model-based timing simulation was defined, and feasibility of the approach was tested using the commercial tools chronVIEW and chronSIM developed by INCHRON.

\(^3\)A context switch in an AUTOSAR OS with memory protection may consume several microseconds.
• Specify the structural system architecture comprising the basic hardware and software elements using the MCSE modeling concepts.
• Identify the interactions between application and basic software (e.g. required services), and basic software and hardware peripherals (e.g. sensor data acquisition).
• Specify the dynamic system architecture comprising the basic execution and communication blocks (tasks, interrupts, messages) and their associated timing properties (BCET, WCET, period, offset, deadline, priority etc.).
• Specify the functional data and control flow considering all relevant modes of operation.
• Perform safety analysis and deduce the dysfunctional data and control flow for all relevant fault conditions.
• Create (using chronSIM’s model editor directly) or generate (using the INCHRON python API) a timing simulation for the chronSIM tool out of the various architecture models.
• Define simulation scenarios in chronSIM for the various modes and fault conditions and perform simulation runs.
• If available perform measurements of the integrated target and import and analyze the measured trace with the chronVIEW tool.
• Extract timing properties and update simulation parameters with measured values. This is already done automatically by the tool.
• Iteratively perform simulations and target measurements until all functional and safety requirements can be fulfilled by the current set of timing properties.

IX. Conclusion and Future Work
Following the practical approach described in this paper, we have shown how state-of-the-art model-based simulation techniques can be used to support the dynamic architecture design of complex automotive control systems. Although, it may appear that adjusting the scheduling configuration in the presented examples is not too complicated, and the proposed solutions may seem obvious, one should consider the number and complexity of the entire event chains in the SCM. In reality, system engineers and software architects responsible for integration and testing, have to achieve many different competing, and sometimes contradicting design goals, especially concerning the dynamic behavior of the system. Model-based simulation and statistical analysis tools as provided by INCHRON greatly help to detect possible real-time requirement violations, and furthermore offer guidance in order to adjust and optimize an existing system configuration. Eventually, they help to document and prove the feasibility of the dynamic architecture design or subsequently proposed design modifications.

Another goal of the case study was to explore possible options for the integration of the different tools used in this project, e.g. for modeling the system architecture, compiling the AUTOSAR configuration, performing the timing simulation, and debugging and tracing on the target microcontroller. Although, some tool-integrations already exist, for example the INCHRON Tool-Suite can generate a model out of an AUTOSAR configuration in .arxml format, not all the relevant information is represented adequately in each model. In some case, we used the Python programming language and the model API of the INCHRON Tool-Suite to automatically generate the simulation model, and to extract execution times from a measured trace in order to update the parameters of the simulation model. In other cases, e.g. for the modeling of event chains or real-time requirements, the transformation from the system architecture model into the simulation model was done manually, mainly due to issues with the interpretation of the data flow semantics discussed in section IV-D of this paper.

In the future, we intend to increase the efficiency of the integration between the simulation and system modeling tools by extending the UML/SysML meta-model profile with the semantics defined in the MCSE methodology. This will make it possible, to automatically generate the event chain and requirement definitions used by the simulation, directly from the system architecture model, and thus eliminate the need to re-model them manually. This solution would also use the existing model API of the simulation tool, and can be maintained without high effort.

After that, we also want to evaluate, if formal verification methods as described in [6] can be applied in a real project environment, in order to find possible inconsistencies concerning the data and control flow already in the structural architecture model, and thus reducing the number of required simulation iterations for the optimization of the system. Finally, we plan to migrate the case study to a multi-core platform, in order to analyze and verify our assumptions for multi-core architectures.

REFERENCES


