End-to-End Timing Analysis of Cause-Effect Chains in Automotive Embedded Systems

Matthias Becker*, Dakshina Dasari†, Saad Mubeen*‡, Moris Behnam*, Thomas Nolte*
*MRTC / Mälardalen University, Västerås Sweden
{matthias.becker, saad.mubeen, moris.behnam, thomas.nolte}@mdh.se
† Robert Bosch GmbH, Renningen, Germany
dakshina.dasari@de.bosch.com
‡ Arcticus Systems AB, Järfälla, Sweden

***Abstract*—Automotive embedded systems are subjected to stringent timing requirements that need to be verified. One of the most complex timing requirement in these systems is the data age constraint. This constraint is specified on cause-effect chains and restricts the maximum time for the propagation of data through the chain. Tasks in a cause-effect chain can have different activation patterns and different periods, that introduce over- and under-sampling effects, which additionally aggravate the end-to-end timing analysis of the chain. Furthermore, the level of timing information available at various development stages (from modeling of the software architecture to the software implementation) varies a lot, the complete timing information is available only at the implementation stage. This uncertainty and limited timing information can restrict the end-to-end timing analysis of these chains. In this paper, we present methods to compute end-to-end delays based on different levels of system information. The characteristics of different communication semantics are further taken into account, thereby enabling timing analysis throughout the development process of such heterogeneous software systems. The presented methods are evaluated with extensive experiments. As a proof of concept, an industrial case study demonstrates the applicability of the proposed methods following a state-of-the-practice development process.**

I. Introduction

Automotive systems are getting complex with respect to traditional components like the Engine Management System (EMS) as well as modern features like assisted driving. While the increase in the EMS complexity is attributed to newer hybrid engines and stricter emission norms, assisted driving requires the perfect convergence of various technologies to provide safe, efficient and accurate guidance. This has led to software intensive cars containing several million lines of code, spread over up to hundred Electronic Control Units (ECU) [1].

Given this complexity, over the last decades, standards like AUTOSAR [2] have been proposed in order to conceive a common platform for the development of automotive software. These standards allow software components provided by different suppliers to be integrated on the same ECU, since they provide for a hardware agnostic software development. Such robust interfaces enable designers to design software in the early stages without knowledge of the concrete hardware platform on which it will be eventually executed. Thus, during the development it is often not known which other applications share the same execution platform.

Most of these automotive applications typically have strict real-time requirements – it is not only important that a computation result is correct, but also that the result is presented at the correct time. In addition to individual timing requirements on the response times of the tasks (i.e. the deadline of a task corresponds to the task's response time), these applications often have timing requirements on the end-to-end functionality of the task chains, so-called *end-to-end timing requirements*. Note that the task chains are commonly found in single-node as well as distributed real-time systems. On the one hand, the end-to-end timing requirement can be specified as an end-to-end deadline on a task chain, which corresponds to the end-to-end response time of the task chain. The end-to-end response time of a task chain is equal to the response time of the last task in the chain. The end-to-end deadline is considered satisfied as long as the response time of the last task in the chain is less than or equal to the specified deadline. It does not matter if the data from the input of the chain is transferred to the output of the chain within the specified end-to-end deadline or not. On the other hand, the end-to-end timing requirement on a task chain can also be specified by means of various timing constraints such as the age constraint [3], [4], [5], [6], [7]. The age constraint is specified on the data propagation through a chain of semantically related tasks, where it specifies the maximum time from reading an input value by the first task of the chain, until a corresponding output value is last produced at the end of the chain. It is of utmost importance that the data from the input of the chain is transferred to the output of the chain within the specified constraint. It is of utmost importance that the data from the input of the chain is only affecting the output of the chain within the interval specified by the constraint.

Many design decisions, such as task activations and communication semantics, have direct influence on the data age. Thus, bounding the data age of a chain early during the design process can potentially avoid costly software redesigns at later development stages. The analysis gets complex as a chain may consist of tasks with different periods leading to over- and under-sampling situations. Such systems are called multi-rate systems. Most of the available analysis methods for such systems focuses on the implementation level where detailed scheduling information is available [8] and thus they are not

applicable during early phases. In [9], a generic framework to calculate the data age of a cause-effect chain is presented, which targets single processor systems and is agnostic of the scheduling algorithm used.

Contributions: In this paper, we extend our earlier work on analyzing end-to-end delays among multi-rate effect chains [9] to utilize the information available at different levels of timing information. Specifically, we highlight the generic nature of the framework by showing how varied levels of information can be used to compute the maximum data age of systems with following characteristics:

1) Different activation patterns between tasks of one cause-effect chain, where individual tasks can be event triggered or periodically triggered.
2) Knowledge of the communication semantic: We extend the analysis to incorporate the explicit communication semantics as well as the Logical Execution Time (LET). Both being communication paradigms commonly used in the automotive domain.
3) Knowledge of offsets: The analysis for systems without knowledge of the schedule is extended to allow for task release-offset specifications.
4) Knowledge of the scheduling algorithm (like Fixed Priority Scheduling (FPS)): Most ECUs utilize operating systems which schedule tasks based on FPS. This allows to utilize existing analysis for such systems to determine worst-case response times of the individual tasks. It is then shown how the concepts of the analysis can be adapted to account for this information.
5) Knowledge of the exact schedule: Similar to most of the existing end-to-end delay analyses, we show how the exact schedule can be used to determine the exact delays with low computational overheads.

Finally, we compare these different scenarios with extensive evaluations, considering i) the tightness of the computed bounds and ii) the computation time for the analysis. We show that increased system information during the design process can thus be used to obtain tighter end-to-end latencies. An industrial case study is performed that demonstrates the applicability of the proposed methods in a state-of-the-practice tool-chain.

II. Related Work

The Timing Augmented Description Language (TADL2) [4] was conceived out of the need for a timing model in the automotive domain, and is the basis for the timing constraints that can be defined in AUTOSAR [3] and EAST-ADL [5]. The end-to-end timing constraints found in these automotive multi-rate systems were first discussed in [10]. Here, the authors describe the different design phases and link them to EAST-ADL [5] and AUTOSAR [2]. An increased level of system knowledge during the consecutive design phases is outlined. Note that these types of end-to-end delays focus on data propagation between independently triggered tasks, in contrast to the end-to-end response time considering the first response at the end of a chain of tasks, where tasks may trigger each other [11].

A method to compute the different end-to-end delays of multi-rate cause-effect chains is presented in [8]. In addition, the authors relate the reaction delay to "*button to reaction*" functionality and the maximum data age delay to "*control*" functionality. In this work the focus lies on the maximum data age and hence on control applications.

A model-checking based technique to compute the end-to-end latencies in automotive systems is proposed in [12]. The authors generate a formal model based on the system description which is then analyzed.

The end-to-end timing analysis in an industrial tool suite is discussed in [6]. Two different activation methods are discussed; trigger chains, where a predecessor task triggers the release of a successor task, and data chains, where tasks are individually triggered and hence over- and under-sampling may occur. These activation patterns are supported by several modeling technologies including the AUTOSAR standard [2]. The trigger activation pattern is also supported by middleware approaches, e.g., a distributable thread in Real-Time CORBA [13] closely resembles the trigger chain. In this work we consider the chains that can have any activation pattern.

End-to-end delays in heterogeneous multiprocessor systems are analyzed in [14]. Ashjaei et al. [15] propose a model for end-to-end resource reservations in distributed embedded systems, and also present the analysis, based on [8], for end-to-end delays under their model.

Additionally, several industrial tools implement the end-to-end delay analysis for multi-rate effect chains [16], [17], [18], [19]. However all of the discussed works require system information which is only available at the implementation level. In [9], a scheduling agnostic end-to-end delay analysis for data age is described, where only information about the tasks of the cause-effect chains is required. Additionally, this work shows how to add job-level dependencies to a task set, such that the data propagation between tasks in an effect-chain is restricted in a way that end-to-end delay constraints are met irrespective of the scheduling decisions.

The principle behind the job-level dependencies can be related to the *rate transition operation* of PRELUDE [20], which is a synchronous language for multi-rate real-time systems, based on the principles of LUSTRE [21]. LUSTRE is addressed in several works, [22] addresses such systems under fixed-priority scheduling [23], and [24] addresses systems under online priority-based scheduling. While these works address single-core systems, many-core target platforms are considered in [25], [26].

The LET communication paradigm was introduced with the time triggered programming language Giotto [27], [28]. Decoupling the execution and communication provides benefits for various areas of embedded systems where predictability and dependability are of most importance [29], such as the automotive industry [30].

In this work, we extend the results presented in [9], and show that by augmenting information available during the

different design phases, we can analyze the maximum data age with decreasing degree of pessimism. It is further shown how the existing framework can be used for different trigger schemes and communication paradigms. Hence, the input for the framework is provided in such a way that the underlying mechanisms and related works can be leveraged, while the set of systems that can be represented is significantly enlarged.

Relation to Authors' Previous Work

This work builds up on the timing analysis for the data age which was presented in [9], where early timing analysis is used to synthesize a partial ordering on the task's jobs to satisfy the corresponding data age constraints. This then significantly eases the synthesis process.

In [31] we extend this analysis by observing that adjustments of the read- and data-intervals can reflect the different levels of system information, while the main analysis engine remains the same. Additionally, in this work, the methods are extended to support mixed trigger chains. It is shown that the computed bounds are safe for the different communication semantics found in the automotive domain. Moreover, we show how these constraints can be addressed in EAST-ADL and AUTOSAR. Finally, the applicability of the proposed methods is demonstrated using a state-of-the-practice tool-chain in the automotive domain.

III. System Model

This section introduces the application model, inter-task communication mechanisms, and the notion of cause-effect chains as used in this work.

A. Application Model

We model the application as a set of periodic tasks Γ.Each task $\tau_i \in \Gamma$ is described by the tuple $\{C_i, T_i, \Psi_i\}$, where C_i is the task's Worst Case Execution Time (WCET), and T_i is the task's period. All tasks have implicit deadlines, i.e. the deadline of τ_i is equal to T_i. A task may also have an *offset* Ψ_i. An offset is an externally imposed time interval between the arrival of the task and its release for execution. For all tasks executing on a processor, the hyperperiod can be defined as the least common multiple of all periods, $HP = \mathrm{LCM}(\forall T_i, i \in \Gamma)$. Hence, a task τ_i executes a number of jobs during one HP, where the j^{th} job is denoted by $\tau_{i,j}$.

B. Inter-task Communication

In the automotive industry, three paradigms (explicit, implicit and LET communication) are typically applied and depending on the chosen paradigm, the communication predictability is affected. In these paradigms, inter-task communication is realized via shared registers wherein, a sending task writes an output value to a shared register, while a receiving task reads the current value of this register. Hence, there is no signaling between the communicating tasks, and a receiving task always consumes the newest value (i.e. last-is-best).

1) Explicit Communication: With this paradigm, tasks directly access the shared register to either read its value or to write a new value to the register. This means, anytime the code demands a read or write operation to the variable, the shared register is accessed. This results in uncertainty, since the exact point in time the register access is performed depends on the respective execution path of the task.

2) Implicit Communication: This communication paradigm is used in order to increase determinism and is based on the *read-execute-write* model. Here, a task reads all its input values into local copies before the execution starts. During the execution phase only those local copies are accessed. Finally, at the end of the execution the task writes the output values to the shared registers, making them available to other tasks. In short, reading and writing of input and output values is done at deterministic points in time, at the beginning and end of the tasks execution respectively. This is a common, and preferred communication paradigm found in several industrial standards (i.e. in AUTOSAR this model is defined as the *implicit communication* [32]. Also the standard IEC 61131-3 for automation systems defines similar communication mechanisms [33]).

3) Logical Execution Time: The LET communication paradigm [27], [28], [30] provides an abstraction to the system designer by temporally decoupling the communication among tasks from the tasks execution. In this model, the input values of a task are always read at the release of the task. The output values become available once the next period starts. This temporal decoupling of communication and execution has significant advantages. while implicit communication still suffers from execution jitter, the LET paves the way for fully deterministic communication.

C. Cause-Effect Chains in Single-Node and Distributed Real-Time Systems

For many systems it is not only important that the individual tasks execute within their timing constraints but also that the data propagates through a chain of tasks within certain time bounds. One example is the Air Intake System (AIS), which is part of the Engine Management System (EMS) in a modern car. For a smooth operation, the air and fuel mixture inside the engine must be controlled and the AIS is responsible for injecting the correct amount of air. To do so, an initial sensor task periodically samples the position of the pedal, followed by a number of control tasks that process this information, and finally an actuator task actuates the throttle to regulate the amount of air inside the engine. For the control algorithm, it is important that the sensed input data is fresh in order to reach the required control quality. Hence, the time from reading the data until the actuation is subject to delay constraints in addition to the task's individual timing constraints.

In AUTOSAR these constraints are described by the *cause-effect chains* [3]. For a task set Γ, a set of cause-effect chains Π can be specified. Where Π contains the individual cause-effect chains ζ_i. A chain ζ_i is represented by a Directed Acyclic Graph (DAG) $\{\mathcal{V}, \mathcal{E}\}$. The nodes of the graph are represented by the set $\mathcal{V}$ and contain the tasks involved in the cause-effect

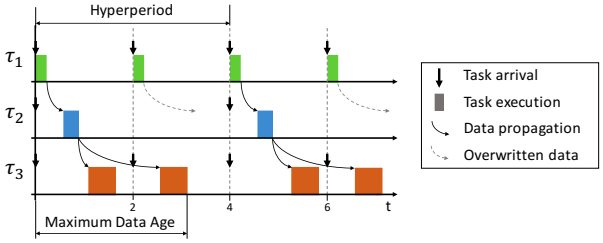


Fig. 1: Data propagation between tasks of a cause-effect chain on a single processor in a real-time system with maximum data age specified.

chain. The set $\mathcal{E}$ includes all edges between the nodes. An edge from τ_i to τ_k implies that τ_i has at least one output variable which is consumed by τ_k. A cause-effect chain can have forks and joins, but the first and the last task in the chain must be the same for all possible data paths. To simplify the analysis, chains with fork/join operations are decomposed into individual sequential chains. Hence, all cause-effect chains in Π are sequential.

End-to-End Timing Requirements: For each cause-effect chain, an end-to-end timing requirement can be specified that targets the data propagation through a chain of semantically related tasks in automotive systems. The end-to-end timing requirements can be defined for these systems by means of several timing constraints [4], [3], [5]. In this work the *data age*, one of the most important timing requirements for control systems, is examined. A detailed discussion of end-to-end delays is provided in [8]. The data age describes the maximum time which an input value can have effect on the output of the cause-effect chain. I.e. the maximum time from sampling an initial input value at the beginning of the cause-effect chain, until the last time this value has influence on the produced output of the cause-effect chain. Fig. 1 depicts an example with three tasks, τ_1, τ_2, and τ_3. All tasks are part of a cause-effect chain in this order. Note that τ_1 and τ_3 are activated with a period of $T = 2$, while τ_2 is activated with a period of $T = 4$. This leads to under-sampling between tasks τ_1 and τ_2, as τ_2 is activated with a slower rate than τ_1. Consequently, not all values that are written by τ_1 are read by τ_2. Similarly, over-sampling is observed between τ_2 and τ_3. τ_2 writes new values less frequently than τ_3 reads them, which leads to τ_3 reading the same value more than once.

While the output value of the first instance of τ_1 is consumed by the first instance of τ_2, the data produced by the second instance of τ_1 is overwritten before τ_2 has the chance to consume it. Similarly, the data produced by the first instance of τ_2 is consumed by the first instance of τ_3. Since no new data is produced before the second instance of τ_3 is scheduled the same data is consumed. In the example, this constitutes the maximum data age, from sampling of the first instance of τ_1 until the last appearance of the data at the output of the second instance of τ_3.

D. Job-Level Dependency

In many systems, data typically propagates between tasks having different periods (activation rates) [22], [20], [26], [9]. Such systems are difficult to analyze [8], especially during the early development phases where low level information, such as the employed scheduling algorithms, may not be available. This lack of information further leads to over-estimated end-to-end delay estimates. One way to add predictability, and thus reduce the pessimism, in such systems is to use the concept of *job-level dependencies* which can be specified already at early design phases [9]. A job-level dependency is a high level concept that can be used to specify a partial ordering on a pair of task instances, where the two tasks potentially execute at different periods.

A job-level dependency is defined by the expression $\tau_A \xrightarrow{(i,j)} \tau_B$. The index i refers to the i^{th} instance of τ_i, and j refers to the j^{th} instance of τ_B respectively. The dependency then imposes an ordering on these two instances, meaning the i^{th} instance of τ_i must be executed before the j^{th} instance of τ_B. The indices are chosen based on the respective occurrence of each tasks job within the hyperperiod of the two task's, e.g. $LCM(\tau_A, \tau_B)$, and repeat itself for each consecutive hyperperiod. Consequently, an execution order constraint can be defined between tasks of the same period, wherein the indices i and j are both set to 1. A repetitive execution order constraint results in a job-level dependency where $i \neq j$.

IV. Calculation of Data Propagation Paths

In this section we recapitulate the calculations of data propagation paths for systems without prior knowledge of the schedule, considering the specified job-level dependencies, as this is the basis for the work presented in this paper. For a more in depth explanation the reader is referred to [9].

A. Reachability Between Jobs

The notions of *read interval* and the *data interval* are key in deciding the valid span of communication. For a job $\tau_{i,j}$, the read interval is defined as the interval starting from the earliest time a job can potentially read its input data ($R_{min}(\tau_{i,j})$) until the latest possible time a job can do so without violating its timing constraints ($R_{max}(\tau_{i,j})$). Similarly, the data interval is defined as the interval from the earliest time the output data of a job can be available ($D_{min}(\tau_{i,j})$) up to the latest time a successor job of the same task overwrites the data ($D_{max}(\tau_{i,j})$). Hence, the read interval $RI_{i,j}$ is the interval $[R_{min}(\tau_{i,j}), R_{max}(\tau_{i,j})]$, and the data interval is $[D_{min}(\tau_{i,j}), D_{max}(\tau_{i,j}))$. These concepts are depicted in Fig. 2 for jobs of a task τ_i. For a system without any knowledge of scheduling decisions, one has to assume that a job may be scheduled anywhere, as long as it starts no earlier than its release and finishes no later than its deadline. In [9], the intervals for systems without offset are defined as follows:

$$R_{min}(\tau_{i,j}) = (j-1) \cdot T_i \quad (1)$$

$$R_{max}(\tau_{i,j}) = R_{min}(\tau_{i,j+1}) - C_i \quad (2)$$

$$D_{min}(\tau_{i,j}) = R_{min}(\tau_{i,j}) + C_i \quad (3)$$

$$D_{max}(\tau_{i,j}) = R_{max}(\tau_{i,j+1}) + C_i \quad (4)$$

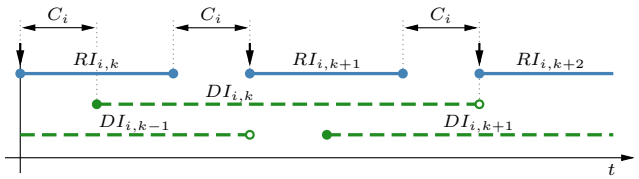


Fig. 2: Read and data intervals of consecutive jobs of τ_i if no scheduling information is available.

1) Deciding Reachability Between Jobs: In order for a job $\tau_{k,l}$ to consume data of a job $\tau_{i,j}$, the data interval of $\tau_{i,j}$ must intersect with the read interval of $\tau_{k,l}$. The function $\text{Follows}(\tau_{i,j}, \tau_{k,l})$ is defined to return $true$ if this is the case:

$$\text{Follows}(\tau_{i,j}, \tau_{k,l}) = \begin{cases} true, & \text{if } RI_{i,j} \cap DI_{i,j} \neq \emptyset \\ false, & \text{otherwise} \end{cases}$$

2) Adjusting the Data Interval for Long Chains: In order to capture the characteristics of data propagation in a cause-effect chain of length > 2, the data interval needs to be modified. Assume the first job of τ_i, as shown in Fig. 2 is followed by a job of a task τ_k. τ_k is released with the same period as that of τ_i, but the execution time of τ_i is shorter than the one of τ_k. $\text{Follows}(\tau_{i,1}, \tau_{k,1})$ returns true and indicates that $\tau_{k,1}$ can potentially consume the data of $\tau_{i,1}$. However, in order to decide reachability between the $\tau_{k,1}$ and a third task in the chain, the data interval of $\tau_{k,1}$ must be modified. This is the case because $\tau_{k,1}$ can consume the data of $\tau_{i,j}$ earliest at time $D_{min}(\tau_{i,j})$. Consequently, this data can earliest be available as output data of $\tau_{k,l}$ at time $D_{min}(\tau_{i,j})+C_k$. $D'_{min}(\tau_{k,l}, \tau_{i,j})$ defines the starting time of the data interval of $\tau_{k,l}$ if the data produced by $\tau_{i,j}$ shall be considered as well:

$$D'_{min}(\tau_{k,l}, \tau_{i,j}) = \max(D_{min}(\tau_{i,j}) + C_k, D_{min}(\tau_{k,l}))$$

Note that the data interval only needs to be adjusted if $D_{min}(\tau_{k,l})$ is smaller than $D_{min}(\tau_{i,j}) + C_k$. These modifications are local for the specific data path; hence, if another combination of jobs is involved then the original data interval must be used.

B. Calculating Data Paths

A recursive function is used to calculate all possible data propagation paths in a system. This function constructs all possible data propagation paths from a job of the first node in a cause-effect chain up to the job of a last node of the chain. Consequently this needs to be done for all jobs of the first task of a chain, within the hyper-period of the chain.

The function starts at the first level of the cause-effect chain; for the initial job, all jobs of the second task of the chain are found where $\text{Follows}()$ returns $true$. To these nodes, a logical data path is created from the initial job. The same principle is applied from each of these nodes to the jobs of the next lower level of the cause-effect chain. Once the last level is reached all possible paths are calculated and the function returns. Interested readers are referred to [9] for a detailed explanation.

C. Calculations for Maximum Data Age

For a given data path, the maximum end-to-end delay and thus the data age, can be computed as follows, where τ_{start} is a job of the first task of the cause-effect chain, and τ_{end} is a job of the last task of a cause-effect chain:

$$\text{AgeMax}(\tau_{start}, \tau_{end}) = (R_{max}(\tau_{end}) + C_{end}) - R_{min}(\tau_{start})$$

In order to compute the maximum data age for any possible path in the system, $\text{AgeMax}()$ must be computed for all computed data paths. The maximum of these values is the maximum data age of the cause-effect chain.

D. Calculations for Maximum Data Age with Job-Level Dependencies

The analysis framework presented in [9] further takes specified job-level dependencies into account. This is done by two modifications. 1) The read-intervals of the affected jobs need to be adjusted. Since a dependency $\tau_A \xrightarrow{(i,j)} \tau_B$ defines that the i^{th} job of τ_A needs to finish its execution C_B time units before the deadline of τ_B. Similarly, the j^{th} job of τ_B cannot start before the i^{th} job of τ_A had the chance to execute. 2) A logical boundary is introduced by a job-level dependency. Meaning a dependency $\tau_A \xrightarrow{(i,j)} \tau_B$ prohibits the propagation of data from the $(i-1)^{th}$ instance of τ_A to the j^{th} instance of τ_B. This is taken into account in the recursive calculation of the data paths.

V. Reachability between Jobs at Various Levels of System Timing Information

The basic computation of data age delays without prior knowledge of the schedule can result in pessimistic results. Many of the computed data propagation paths may not occur since scheduling algorithms impose a recurring order of jobs in each hyperperiod of the task set. In this section, modifications of the read and data interval are presented to reflect the behavior of the systems with more elaborate knowledge on the scheduling decisions. One key observation is that the presented method to calculate the different data paths and the maximum data age is independent of the concrete system model as long as the read and data intervals are adjusted accordingly. Table I depicts the required changes to the read and data interval for different levels of system information, if such a modification is required by the respective system. The remainder of this section discusses these required modifications in more detail, while example figures are provided.

A. Reachability in Mixed Trigger Chains

A mixed-trigger chain is defined as a cause-effect chain where different trigger events are used to release the tasks jobs.

The first trigger event is the periodic activation of tasks. Such a task is released with its specified period T_i, and can be executed anytime within the interval $[k \cdot T_i, (k+1) \cdot T_i)$, independent of other tasks in the effect-chain, where $k \in \mathbb{N}$.

Often it is important to impose an ordering to the execution of different tasks in the system. For example, if there is a

TABLE I: Description of the read and data interval for the system with different levels of timing information.

	No Knowledge	**Exact Schedule Known**	**WCRT Known**	**LET Execution Model**
$R_{min}(\tau_{i,j})$	$\Psi_i + (j-1) \cdot T_i$	$start_{i,j}$	$\Psi_i + (j-1) \cdot T_i$	$(j-1) \cdot T_i$
$R_{max}(\tau_{i,j})$	$j \cdot T_i - C_i$	$R_{min}(\tau_{i,j})$	$R_{min}(\tau_{i,j}) + WCRT_i - C_i$	$R_{min}(\tau_{i,j})$
$D_{min}(\tau_{i,j})$	$R_{min}(\tau_{i,j}) + C_i$	$end_{i,j}$	$R_{min}(\tau_{i,j}) + C_i$	$j \cdot T_i$
$D_{max}(\tau_{i,j})$	$R_{max}(\tau_{i,j+1}) + C_i$	$end_{i,j+1}$	$R_{max}(\tau_{i,j+1}) + C_i$	$(j+1) \cdot T_i$

relation between two tasks, one reading a sensor value and a second applying data processing as first step before the data is further used. Both functions can be implemented separately, however the underlying causal relationship allows a system designer to impose an execution order. Thus, the second trigger event is at the end of the execution of the predecessor tasks job in the cause-effect chain. Assume the partial cause-effect chain $\tau_A \rightarrow \tau_B$. If the two tasks are connected by such an activation pattern, it must hold that $T_A = T_B$. In order to impose the correct ordering of jobs, a job-level dependency is specified for the timing analysis. This is done between the two tasks: $\tau_A \xrightarrow{(1,1)} \tau_B$.

This example is visualized in Fig. 3, where an cause-effect chain of length 4 is shown. Tasks τ_A and τ_C are periodically triggered, whereas the task τ_B is triggered by the end of execution of a job of τ_A, and τ_D is triggered by the end of execution of a job of τ_C respectively. It is further shown how to represent such a trigger pattern using job-level dependencies. Modeling the two activation rates by the use of job-level

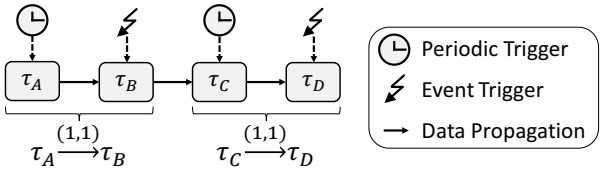


Fig. 3: Example of a mixed trigger chain and the required job-level dependencies for the timing analysis.

dependencies allows to directly apply the end-to-end timing analysis proposed in [9].

B. Knowledge of Task Offsets

In our earlier work [9], no task release offsets were considered in the analysis. In order to account for known offsets, the read interval needs to be adjusted. Given an offset Ψ, a job of a task can now read its input data only after Ψ time units with respect to the start of its period. The end of the read interval is unchanged at C time units before the next period starts. Since D_{min} and D_{max} are described by R_{min} and R_{max}, no direct changes are required in the formulation. With the adjustment of the input interval according to the offsets, the remaining calculations to determine the maximum data age can remain the same, as described in Section IV.

C. Reachability in Known Schedules

Many automotive real-time systems deploy time-triggered schedules in order to guarantee a deterministic timing behavior. In such a schedule it is known at design time when the different jobs of the different tasks are executed. Thus, a complete knowledge of the system is available. On the other hand, for dynamically scheduled systems it is often possible to compute the Worst-Case Response Time (WCRT). In that case the exact execution times of a task are not known but the earliest and latest time a task can execute is known.

1) Schedule is Available: Let's assume an offline schedule is available for the system. So for each job $\tau_{i,j}$ of the task set its exact start time is known as $start_{i,j}$, and similarly its finishing time is known as $end_{i,j}$, see Fig. 4. With this additional knowledge the read and data interval can be adjusted as shown in Table I.

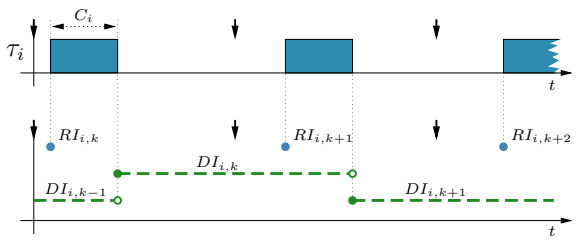


Fig. 4: Read and data intervals of consecutive jobs of τ_i if the exact schedule is available.

Since the start of the jobs execution, and hence the time it reads its input data, is known, the read interval collapses to a point. This also leads to smaller data intervals, resulting to no overlap between consecutive jobs.

2) Worst Case Response Time is Available: For systems where the WCRT of a task τ_i is known as $WCRT_i$, the read and data interval can be adjusted to account for this more accurate system information (see Fig. 5). The modifications of the read interval mainly reflect the possible execution of a job during its execution window (bounded by the WCRT).

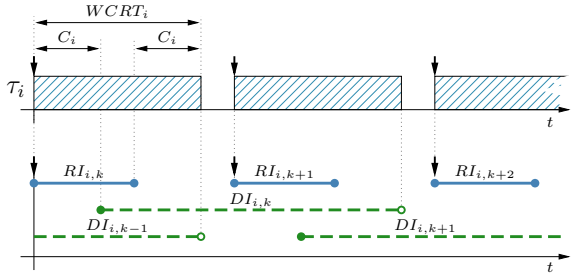


Fig. 5: Read and data intervals of consecutive jobs of τ_i if WCRT of the tasks are available.

D. Reachability in the LET model

In the LET model the read and write operations are confined to the boundaries of the execution window. In Fig. 6, these

points are highlighted by the thick-vertical dotted orange lines below the arrows marking the job releases. This temporal decoupling of communication and execution has significant advantages for the end-to-end delay calculations.

The periodic access to all input variables at the beginning of the period collapses the read interval to a point. The data interval is also defined, making the output data available for exactly the period after the jobs execution. These modifications are shown in Table I. As can be seen, all descriptions are independent of the actual execution time of the job.

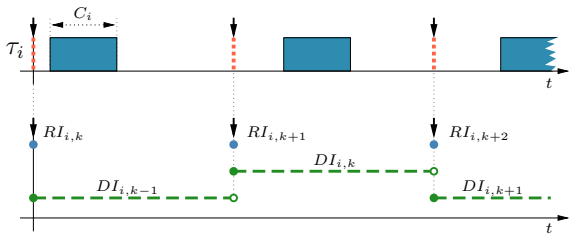


Fig. 6: Read and data intervals of consecutive jobs of τ_i if the system operates based on the LET model.

E. Discussion

All presented modifications affect solely the read and/or data intervals of the jobs. Hence, the existing calculations for the maximum data age, as presented in Section IV, can be applied without modification. This fact allows the system designer to perform the calculation of maximum data age based on various levels of system information.

A tradeoff between the required system knowledge and accuracy of the obtained maximum data age exists. For systems with exact knowledge, and for the LET systems, it can be observed that data intervals of different jobs of the same task never overlap with each other. This means that it is always certain which data is consumed by a job and thus, all data paths which are computed are observed when the system is executed.

Lemma 1. *If it holds for all tasks in a chain, that the read intervals of a task are reduced to a point (i.e. $R_{min}(\tau_{i,j}) = R_{max}(\tau_{i,j})$), then the calculated data age delays are exact. Here "exact" means that all calculated data age delays are observed during the execution of the real system.*

Proof. From the definition of the read and data intervals in Section V we can see that once $R_{min}(\tau_{i,j}) = R_{max}(\tau_{i,j})$ the resulting data intervals of consecutive jobs do not overlap. Given that data intervals do not overlap and read intervals are reduced to points, the function $\text{Follows}(\tau_{i,j}, \tau_{k,l})$ only returns $true$ for the jobs which actually consume the respective data during the execution of the real system. □

VI. Timing Constraints to Restrict Execution Order in Automotive Standards

While job-level dependencies are a useful notation to describe an ordering between jobs of different tasks, which possibly execute at different rates, they are not directly supported by the timing models found in the automotive standards such as EAST-ADL [5] or AUTOSAR [3]. In this section we address this problem first by showing how to represent a subset of such constraint using the EAST-ADL timing specifications. Later we show how to represent job-level dependencies using the AUTOSAR extensions for timing specifications [3]. In AUTOSAR such constraints can be represented using specific constraints that are not available at the higher abstraction levels that are defined by EAST-ADL.

A. Job-Level Dependencies and EAST-ADL

EAST-ADL allows to specify the so-called *Ordering Constraint* between two tasks. Such a constraint is specified between two events, the source and the target, where both need to appear at the same rate, i.e. $T_{\text{source}} = T_{\text{target}}$. The constraint then dictates that for each occurrence, the source needs to be executed before the target.

This can be mapped to a job-level dependency, where τ_A is the source and τ_B is the target: $\tau_A \xrightarrow{(1,1)} \tau_B$. Note, that a more general job-level dependency with instance indices $\neq 1$ cannot directly be represented by the timing constraints available in EAST-ADL.

B. Job-Level Dependencies and AUTOSAR

In AUTOSAR, a job-level dependency can be represented by an *Execution Order Constraint* [3]. In the basic form, such a constraint defines a trigger event that activates a group of tasks. The tasks inside this group are then executed sequentially, in the specified order. Hence, tasks that are connected by such a relation need to be triggered with the same rate. This then directly maps to a single-rate job-level dependency $\tau_A \xrightarrow{(1,1)} \tau_B$, since the ordering of the tasks in the group defines the execution order.

The same type of constraint can be used as *Repetitive Execution Order Constraint* [3], which allows to define relations between tasks that are *not* executed with the same period. Involved periods only need to be harmonic, i.e. the periods of the tasks need to be divisible. Compared to the basic execution order constraint, several task groups are related in this constraint and the active group is selected in a cyclic manner. Like the simple execution order constraint, this constraint type is activated by a trigger event with rate of $\min(T_A, T_B)$. The number of required cycles, and thus groups, can be computed by $\max(\frac{LCM(\tau_A,\tau_B)}{\tau_A}, \frac{LCM(\tau_A,\tau_B)}{\tau_B})$.

As an example in Fig. 7 , two tasks, τ_A and τ_B are scheduled where τ_A is activated with twice the periodicity of τ_B. While, in the general case, τ_B can execute anytime, it introduces pessimism since the execution order is unknown. To achieve correct functionality, the instance of τ_B must execute after the second instance of τ_A. This can be achieved by the repetitive execution order constraint shown in Fig. 7.

The constraint is triggered with the period of τ_A. The number of task groups that is needed is 2, since the hyper-period of the two tasks is twice the period of τ_A. In the first task group only τ_A is included, while both tasks are present in the second group in the order as specified by the

constraint. This dependency can be described by the job-level dependency $\tau_A \xrightarrow{(2,1)} \tau_B$. The resulting schedule of the described constraint is also shown in Fig. 7.

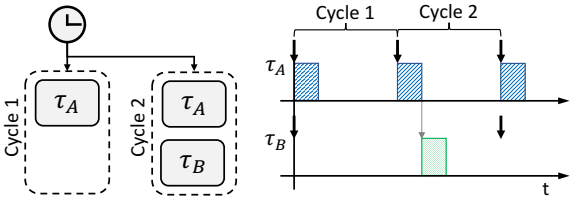


Fig. 7: Repetitive execution order constraint on the left and the resulting schedule on the right.

VII. Representation in Different Communication Paradigms

So far, all assumptions for the read and data interval are based on the implicit communication model, where it is assumed that all read operations happen at the start of the execution and all write operations happen at the end of execution. In this section, we are going to relax this restriction by showing that the model in fact covers all other communication models as well. In the remainder of this section, we first show that the read and data intervals are safe and that they capture all cases for the different execution paradigms.

A. Boundaries of the Read and Data Intervals

This section shows which boundary of the respective interval affects the maximum data age and is thus most important for the corresponding calculations.

Lemma 2. *The lower boundary of the read interval affects the maximum data age of a cause-effect chain.*

Proof. This can be shown by considering both corner cases. Lets assume the upper boundary of the interval ends at time R_{max} and overlaps with a predecessor instance $\tau_{j,i}$. If the upper boundary of the interval is extended by k time units to time $R_{max} + k$ it potentially overlaps with the data interval of $\tau_{j,i+1}$. This instance is released after the original instance $\tau_{j,i}$ and hence the consumed data is fresher and the data age becomes smaller. On the other hand, if the lower bound of the interval R_{min} is extended to $R_{min} - k$, the instance may read the data of $\tau_{j,i-1}$. This then leads to a larger value for the data age since $\tau_{j,i-1}$ is executed before $\tau_{j,i}$, thus the consumed data is older. □

Similarly we can address the data interval.

Lemma 3. *The upper boundary of the data interval affects the maximum data age of a cause-effect chain.*

Proof. This can also be shown by considering both corner cases. Assume that the first instance of the successor task τ_j that overlaps with the data interval with boundary D_{min} is instance i. By extending the lower boundary by k time units to $D_{min} - k$ an earlier instance $\tau_{j,i-1}$ may overlap with this extended data interval. Such a data propagation will lead to a reduced data age since $\tau_{j,i-1}$ is scheduled before $\tau_{j,i}$. On the other hand, if the upper bound of the data interval D_{max} is extended by k time units to $D_{max} + k$, the data may be read by a later instance of $\tau_{j,i}$ which will lead to a larger maximum data age. □

B. Explicit Communication

As described in Section III-B, explicit communication does not restrict the locality of memory access like implicit communication. Fig. 8 shows an example job of a task τ_i, where read and write access to the global variables happens at various points during its execution. In the figure, the variable L_k is read ΔL_k time units after the start of the execution. Note that the actual execution path of the code may lead to several other possible data access patterns and even the same execution path may lead to different values of ΔL_k, depending on the state of the processor and scheduling effects. To compute safe bounds for the maximum data age it is assumed that the data access happens as early as possible, with the start of the execution (see Lemma 2). Similarly, the write access to data happens as late as possible during the execution (see Lemma 3). Thus, the resulting boundaries for the intervals are equivalent to the intervals of the implicit communication model. However, differences between the two models do exist. The access to global variables is limited in the implicit communication model, compared to the explicit communication model, where expensive access to global variables may be accounted for multiple times during the execution. This then leads to different WCET estimates between the two models.

VIII. Evaluation

This section presents the evaluation of the proposed approaches to analyze the end-to-end delay based on various levels of system information. First the experimental setup is described, before the computed worst-case data age is compared based on various levels of timing information. Then the required computation time to perform the analysis at the presented information levels are evaluated. The applicability of the proposed methods to an industrial setting are presented in Section IX, using a case study.

A. Experimental Setup

The analyzed cause-effect chains for the experiments are generated according to the automotive benchmarks described in [34]. The task periods are uniformly selected out of the set $\{1, 2, 5, 10, 20, 50, 100, 200, 1000\}ms$. The individual task utilization is computed by UUnifast [35]. As stated in [34], an individual cause-effect chain is comprised of either 2 or 3 different periods, where tasks of the same period appear

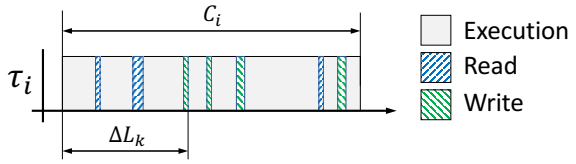
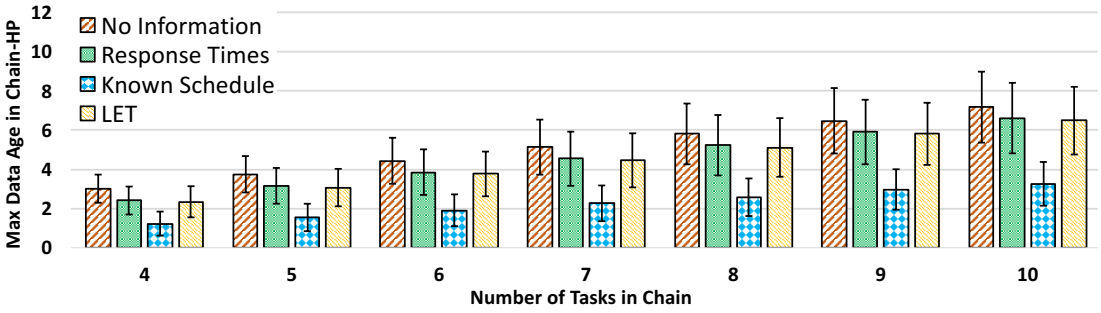
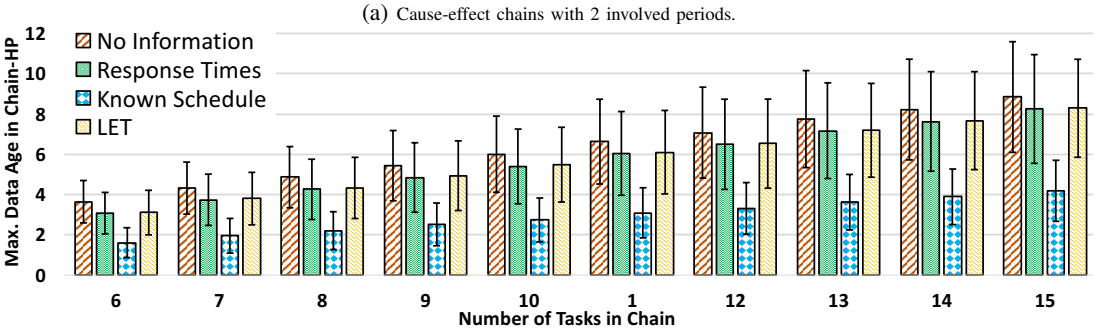


Fig. 8: Read and write operations in the explicit communication model.



(a) Cause-effect chains with 2 involved periods.



(b) Cause-effect chains with 3 involved periods.

Fig. 9: Comparison of the max. data age (normalized to the chains HP) for chains with 2 and 3 activation rates under various levels of system knowledge.

in sequence in the chain. Note that period-pairs are randomly generated in the cause-effect chain in conformance to [34]. For each of the presented data points, *1000 random cause-effect chains* are examined.

Fixed priority scheduling is used to generate the schedule and compute the task response times. Priorities are assigned as per the Rate Monotonic [36] policy, where priorities between tasks of the same period are assigned in arbitrary order. For the evaluation of the systems with required response times or known schedule, the response times are calculated by the classical response time analysis [37], and the schedule is generated by simulating the tasks' execution.

B. Analysis of Pessimism at Various Levels of Timing Information

The first experiment analyzes each cause-effect chain in a system given different information states – *no information*, *response times*, *known schedule*, and *LET communication*. The system contains 30 tasks while the generated cause-effect chains are comprised of 4 – 10 tasks in the case of two activation patterns (i.e. periods), and 6 – 15 tasks in the case of three activation patterns and the system utilization is set to 80%. The results are presented in Fig. 9. The calculated end-to-end latencies are normalized with respect to the hyperperiod of the chain and shown on the vertical-axis. The decreased pessimism in the analysis with increased system knowledge is visible in all results. The computed worst-case data age of the same scenario includes lesser pessimism from systems with no prior information to systems with known response times up to systems where the schedule is available. Additionally, we present the maximum data age under the LET model, which behaves close to the computed results based on response times in our setting. The difference of the execution semantic becomes visible when comparing with the results for known schedules. Both results are exact results under the respective execution semantic but the observed maximum data age for the LET model is larger than the value for the known schedule. This is because the LET model restricts the data communication to the period boundaries. While the maximum data age is larger, the data age experiences no jitter since it does not depend on the execution of tasks. This has significant benefits for control applications.

C. Analysis of the Computation Time

In this experiment we evaluate the required computation time for the analysis under the different levels of system knowledge, as shown in Fig. 10. The system contains 30 tasks while the cause-effect chain under analysis has a length of 4 – 10 tasks with two involved activation rates. All experiments were performed on a system containing an Intel i7 CPU (4 cores at 2,8 GHz), and 16 GB of RAM. The two scenarios with exact knowledge (i.e. the known schedule and the LET model) have very low analysis times with almost no increase with increasing length of the chain under analysis.

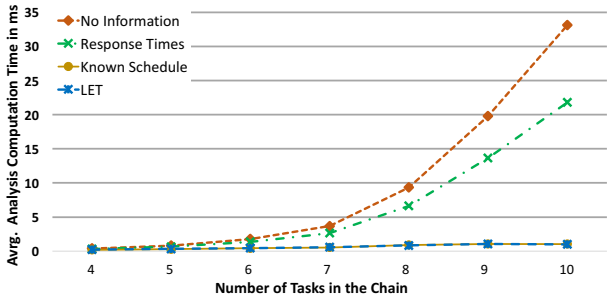


Fig. 10: Average analysis time for cause-effect chains with 2 involved periods.

On the other hand, the scenarios with less system information experience an exponential increase in analysis time. This is attributed to the increased uncertainty due to overlapping data intervals. Thereby increasing the possible successors that must be analyzed by the algorithm.

IX. Industrial Case Study

In this case study, we demonstrate the applicability of the proposed methods to one node of a Steer-by-Wire (SBW) application.

A. Prototype Setup

We integrate the proposed timing analysis into the RUBUS tool chain, which is a commercial tool chain [18], as seen in Figure 11. Rubus-EAST models the system at the design level with the EAST-ADL language, while Rubus-ICE uses this model to generate the implementation level model, compliant with the Rubus Component Model [38] containing the offline schedule to be executed at the target platform. We apply our proposed timing analysis on the artifacts generated by the respective tools (i.e. the design level model for Rubus-EAST and the implementation level model for Rubus-ICE). Currently, there is *no support in Rubus-EAST to perform timing analysis at the design level without additional expert knowledge*. At the implementation level however, it is possible to analyze the maximum data age of the cause-effect chains which is based on the timing analysis of [8].

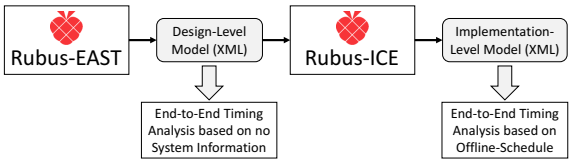


Fig. 11: Case Study setup showing different points to perform the timing analysis.

B. Steer-by-Wire Subsystem and its Timing Analysis

We base this case-study on the SBW system presented in [7], and analyze the ECU that handles the front wheel subsystem. The functionality implemented in this subsystem is part of a larger control loop, where control information and data is received and sent over a connected Controller Area Network [39] bus every 20 ms. The ECU reads from the angle and torque sensors, processes it and finally passes control to the actuator of the wheel. Internal processing includes filtering the sensor inputs and the main control functionality which is triggered every 10 ms. Detailed information about the tasks and their attributes can be found in Table II.

For the case study, we focus on two cause-effect chains shown in Fig. 12. The first cause-effect chain is the *Network Chain*, which spans from input In1, to the output Out1. The second cause-effect chain, the *Wheel Chain*, is related to the internal functionality and actuation and spans from In2 to Out2.

TABLE II: Timing properties of the tasks in the SBW subsystem.

	W_Angle	W_Torque	Pre_Filter	Control	Actuator	NW_In	NW_Out
T_i	10 ms	10 ms	N.A	10 ms	N.A	20 ms	20 ms
C_i	50 µs	50 µs	120 µs	200 µs	120 µs	100 µs	100 µs

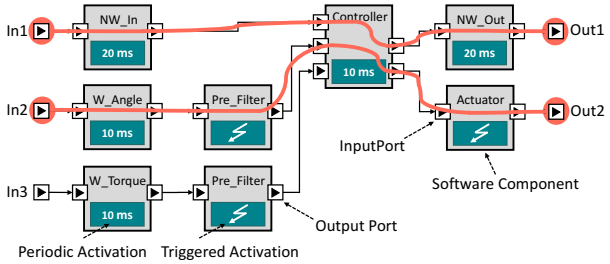


Fig. 12: Design-level software architecture of the SBW subsystem.

1) Analysis at the Design Level: At the design level, where no scheduling information is present, we analyze both cause-effect chains using the methods presented in this paper. For the Wheel Chain, the maximum data age is computed to be 40 ms if the basic analysis of [9] is applied. Since the Wheel Chain contains a mixed trigger pattern, the required job-level dependencies can be specified, as described in Section V-A. This consideration leads to a maximum data age of 20 ms. The Network Chain contains only independently triggered tasks, where the maximum data age is computed to be 60 ms.

2) Analysis at the Implementation Level: At the implementation level, in the Rubus tool chain, the exact schedule is computed offline and therefore we apply the proposed modifications of Section V-C. In the specific schedule, the Wheel Chain experiences a maximum data age of 10.22 ms, and the Network Chain experiences a maximum data age of 10.10 ms. It can further be shown that each observed data-path results in the same data age for the respective cause-effect chains, leading to jitter only on the data age due to the jitter of the last task in the chain. Hence, the experienced jitter is minimal, which is highly desirable in control systems.

3) Discussion: This case study demonstrates the applicability of the proposed methods in an industrial tool-chain. The existing system information can directly be used as input for the required analysis at the different abstraction levels, which is important, as the integration in the tool-chain needs to impose as few changes as possible to it. Being able to analyze the data age at both abstraction levels enables the engineers to detect design flaws already early on which can significantly reduce the development costs.

X. Conclusion and Outlook

Timing requirements specified for the end-to-end data propagation delays through a chain of independently triggered tasks are commonly found in the automotive domain. In this paper, we have shown how to utilize the different levels of system information available during the design of automotive systems in order to compute the maximum data age of a cause-effect chain. It is further shown how different communication semantics, that are typically used in these systems, can be addressed. This is done by extending the analysis method presented in [9] by adjusting the read- and data-intervals,

which are used as input values of the analysis, to reflect the increase in system knowledge. A clear trade-off can be observed between the required information for the analysis and the pessimism (overestimation) in the obtained results. Hence, the analysis presented in this work allows to compute the analysis results with high precision based on the various levels of available system information.

An industrial case study is performed to demonstrate the applicability of the proposed methods in using a state-of-the-practice tool chain. It is demonstrated that the data age computed at a high abstraction level (before the system synthesis) is less precise than the data age that is computed with full system information. The benefits emerge as system engineers can utilize these timing bounds to detect design flaws as soon as they become visible based on the available system information. Moreover, the early timing estimations also allow the system designer to perform early system refinements with respect to end-to-end timing. This can then significantly reduce development costs.

The analysis presented in this paper is applicable to single node real-time systems as well as distributed real-time systems, where the nodes are synchronized. Future work focuses on the analysis of maximum data age over cause-effect chains which are distributed over multiple nodes, where the nodes may or may not be synchronized.

ACKNOWLEDGMENT

The work presented in this paper is supported by the Swedish Knowledge Foundation (KKS) through the projects PREMISE, DPAC, and PreView. We thank our industrial partners Robert Bosch GmbH, Germany and Arcticus Systems, Volvo Construction Equipment and BAE Systems Hägglunds, Sweden.

REFERENCES

[1] M. Broy, "Challenges in automotive software engineering," in *the 28th ICSE*, 2006, pp. 33–42.
[2] *AUTOSAR*, last access October 2016, available at www.autosar.org.
[3] *AUTOSAR - Spec. of Timing Extensions*, AUTOSAR Std. 4.3, 2016.
[4] "Timing augmented description language (TADL) syntax, semantics, metamodel. Ver. 2, Deliverable 11," Tech. Rep., 2012.
[5] *EAST-ADL - Domain Model Specification*, EAST-ADL Association Std. V2.1.12, 2014.
[6] S. Mubeen, J. Mäki-Turja, and M. Sjödin, "Support for end-to-end response-time and delay analysis in the industrial tool suite: Issues, experiences and a case study," *Computer Science and Information Systems*, vol. 10, no. 1, 2013.
[7] S. Mubeen, T. Nolte, M. Sjödin, J. Lundbäck, and K.-L. Lundbäck, "Supporting timing analysis of vehicular embedded systems through the refinement of timing constraints," *Software & Systems Modeling*, pp. 1–31, 2017.
[8] N. Feiertag, K. Richter, J. Norlander, and J. Jonsson, "A compositional framework for end-to-end path delay calculation of automotive systems under different path semantics," in *the 1st CRTS*, 2008.
[9] M. Becker, D. Dasari, S. Mubeen, M. Behnam, and T. Nolte, "Synthesizing job-level dependencies for automotive multi-rate effect chains," in *the 22th RTCSA*, 2016, pp. 159–169.
[10] F. Stappert, J. Jonsson, J. Mottok, and R. Johansson, "A design framework for end-to-end timing constrained automotive applications," in *the 2nd ERTS*.
[11] J. C. P. Gutierrez, J. J. G. Garcia, and M. G. Harbour, "On the schedulability analysis for distributed hard real-time systems," in *Proceedings 9th Euromicro Workshop on Real Time Systems*, Jun 1997, pp. 136–143.
[12] A. C. Rajeev, S. Mohalik, M. G. Dixit, D. B. Chokshi, and S. Ramesh, "Schedulability and end-to-end latency in distributed ecu networks: Formal modeling and precise estimation," in *the 10th EMSOFT*, 2010, pp. 129–138.
[13] D. G. Schmidt and F. Kuhns, "An overview of the real-time corba specification," *Computer*, vol. 33, no. 6, pp. 56–63, June 2000.
[14] S. Schliecker and R. Ernst, "A recursive approach to end-to-end path latency computation in heterogeneous multiprocessor systems," in *the 7th CODES*, 2009, pp. 433–442.
[15] M. Ashjaei, S. Mubeen, M. Behnam, L. Almeida, and T. Nolte, "End-to-end resource reservations in distributed embedded systems," in *the 22th RTCSA*, 2016, pp. 1 – 11.
[16] Symtavision GmbH, "SymTA/S and TraceAnalyzer for ECUs," [Online] https://www.symtavision.com/products/ecu-timing/, last visited 25.10.2016.
[17] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst, "System level performance analysis - the symta/s approach," *Computers and Digital Techniques*, vol. 152, no. 2, pp. 148–166, March 2005.
[18] Arcticus Systems, "Rubus ICE," [Online] https://www.arcticus-systems.com/products/, last visited 25.10.2016.
[19] Timing Architects, "Timing Architects Inspector," [Online] https://www.timing-architects.com/ta-tool-suite/inspector/, last visited 25.10.2016.
[20] J. Forget, F. Boniol, D. Lesens, and C. Pagetti, "A real-time architecture design language for multi-rate embedded control systems," in *ACM Symposium on Applied Computing*, 2010, pp. 527–534.
[21] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud, "The synchronous data flow programming language LUSTRE," *Proceedings of the IEEE*, vol. 79, no. 9, pp. 1305–1320, 1991.
[22] J. Forget, F. Boniol, E. Grolleau, D. Lesens, and C. Pagetti, "Scheduling dependent periodic tasks without synchronization mechanisms," in *16th IEEE Real-Time and Embedded Technology and Applications Symposium*, 2010, pp. 301–310.
[23] N. Audsley, A. Burns, R. Davis, K. Tindell, and A. Wellings, "Fixed priority pre-emptive scheduling: an historic perspective," *Real-Time Systems*, vol. 8, no. 2/3, pp. 173–198, 1995.
[24] C. Pagetti, J. Forget, F. Boniol, M. Cordovilla, and D. Lesens, "Multi-task implementation of multi-periodic synchronous programs," *Discrete Event Dynamic Systems*, vol. 21, no. 3, pp. 307–338, 2011.
[25] W. Puffitsch, E. Noulard, and C. Pagetti, "Mapping a multi-rate synchronous language to a many-core processor," in *19th IEEE Real-Time and Embedded Technology and Appl. Symposium*, 2013, pp. 293–302.
[26] ——, "Off-line mapping of multi-rate dependent task sets to many-core platforms," *Real-Time Systems*, vol. 51, no. 5, pp. 526–565, 2015.
[27] T. A. Henzinger, B. Horowitz, and C. M. Kirsch, "Giotto: a time-triggered language for embedded programming," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 84–99, Jan 2003.
[28] C. M. Kirsch and A. Sokolova, *The Logical Execution Time Paradigm*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 103–120.
[29] S. Bensalem, K. Goossens, C. M. Kirsch, R. Obermaisser, E. A. Lee, and J. Sifakis, "Time-predictable and composable architectures for dependable embedded systems," in *9th ACM International Conference on Embedded Software (EMSOFT)*, 2011, pp. 351–352.
[30] J. Hennig, H. Hasseln, H. Mohammad, S. Resmerita, S. Lukesch, and A. Naderlinger, "Towards parallelizing legacy embedded control software using the LET programming paradigm," in *RTAS WiP*, 2016.
[31] M. Becker, D. Dasari, S. Mubeen, M. Behnam, and T. Nolte, "Analyzing end-to-end delays in automotive systems at various levels of timing information," in *IEEE 4th Int. Workshop on Real-Time Computing and Distributed systems in Emerging Applications (REACTION)*, 2016.
[32] *AUTOSAR - Specification of RTE*, AUTOSAR Std. 4.2.2, 2014.
[33] *IEC 61131-3*, International Electrotechnical Commission Std., 2003.
[34] S. Kramer, D. Ziegenbein, and A. Hamann, "Real world automotive benchmarks for free," in *the 6th WATERS*, 2015.
[35] E. Bini and G. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Systems Journal*, vol. 30, no. 1-2, pp. 129–154, 2005.
[36] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM*, 1973.
[37] M. Joseph and P. Pandya, "Finding response times in a real-time system," *The Computer Journal*, 1986.
[38] K. Hänninen et.al., "The Rubus Component Model for Resource Constrained Real-Time Systems," in *3rd IEEE International Symposium on Industrial Embedded Systems*, June 2008.
[39] ISO 11898-1, "Road Vehicles interchange of digital information controller area network (CAN) for high-speed communication, ISO Standard-11898, Nov. 1993."