MECHAniSer - A Timing Analysis and Synthesis Tool for Multi-Rate Effect Chains with Job-Level Dependencies

Matthias Becker*, Dakshina Dasari†, Saad Mubeen*, Moris Behnam*, Thomas Nolte*
*MRTC / Mälardalen University, Sweden {matthias.becker, saad.mubeen, moris.behnam, thomas.nolte}@mdh.se
† Research and Technology Centre, Robert Bosch, India dakshina.dasari@in.bosch.com

Abstract—**Many industrial embedded systems have timing constraints on the data propagation through a chain of independent tasks. These tasks can execute at different periods which leads to under and oversampling of data. In such situations, understanding and validating the temporal correctness of end-to-end delays is not trivial. Many industrial areas further face distributed development where different functionalities are integrated on the same platform after the development process. The large effect of scheduling decisions on the end-to-end delays can lead to expensive redesigns of software parts due to the lack of analysis at early design stages. Job-level dependencies is one solution for this challenge and means of scheduling such systems are available. In this paper we present MECHAniSer, a tool targeting the early analysis of end-to-end delays in multi-rate cause effect chains with specified job-level dependencies. The tool further provides the possibility to synthesize job-level dependencies for a set of cause-effect chains in a way such that all end-to-end requirements are met. The usability and applicability of the tool to industrial problems is demonstrated via a case study.**

I. Introduction

Many application domains for embedded systems are subject to timing constraints in order to fulfill their requirements. Such real-time systems are well studied and several tools are available to analyze these properties. However, for many systems it is not only important that the individual tasks execute within their specified deadlines, but also that data propagates through a chain of tasks within a specified end-to-end delay constraint. In the automotive industry such chains are called cause-effect chains [1], [2]. The tasks in such a chain can have different activation periods which makes the calculation of such end-to-end delays a challenging task since over and undersampling effects need to be considered.

Currently it is left to the discretion of the system designer to guarantee that all end-to-end delay constraints are met in the system. While this is viable in small applications, the growing complexity of industrial applications renders this approach increasingly difficult. Automotive applications for example contain several multi-rate cause-effect chains [3]. Additionally, one task can be part of several chains which increases the problem complexity further.

This highlights the need for tool support during the system design, giving the designer viable input during early stages of the development where only limited or even no concrete knowledge of the schedule is present. This need is further increased since applications of several suppliers may be integrated on the same Electronic Control Unit (ECU) during the system integration which is usually done by the Original Equipment Manufacturer (OEM). Changes in the system design can be very expensive at this stage. Having means to obtain end-to-end delay bounds for the data propagation through a chain of tasks before the system integration can thus provide valuable information and reduce the risk of costly design changes in the later development phases.

One way to reduce the possible data propagation among tasks of different rate is the use of job-level dependencies [4]. A job-level dependency introduces a constraint in the data propagation between two tasks and is specified on job-level. Several works address the scheduling problem of systems with specified job-level dependencies. These works cover fixed-priority and dynamic priority scheduled systems [5], [6], as well as time triggered schedules [7], [8]. The problem of analyzing such systems and to synthesize job-level dependencies is addressed in [9].

A. Contributions

Several available tools support the end-to-end delay analysis of cause-effect chains, which are primarily based on the principles proposed in [10]. They however assume that knowledge of the task schedule is available when the system is analyzed. In contrast, the proposed tool MECHAniSer can be helpful in the early design phases where the exact task schedule is unknown. Its key features include analysis to i) compute bounds on the end-to-end delays ii) synthesize job-level dependencies when specified timing constraints are violated iii) compute end-to-end analysis in the systems where job-level dependencies are specified. To facilitate a faster system design, the tool implements a heuristic to place job-level dependencies in a system consisting of several, possibly interconnected, cause-effect chains. This is done in a way such that the maximum data age delay of each cause-effect chain is met.

B. Paper Layout

The rest of the paper is organized as follows, in Section II the system architecture and background information is provided. In Section III the calculations to obtain the data age delay are described before the tool itself is discussed in Section IV. The tool is evaluated based on a case study in Section V, followed by a discussion of related tools in Section VI and the conclusions and future work in Section VII.

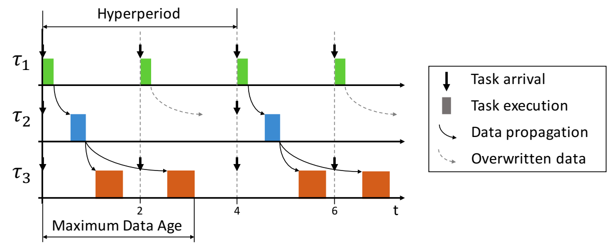


Fig. 1: Data propagation between tasks of a cause-effect chain in a real-time system with maximum data age specified.

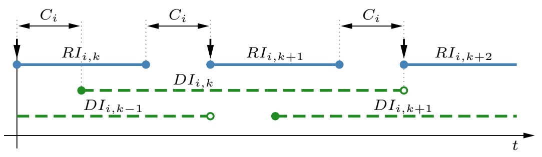


Fig. 2: Read and data intervals of consecutive jobs of τ_i if no scheduling information is available.

II. System Architecture and Background

A. System Model

The system is comprised of a set of periodic tasks Γ. Each task $\tau_i \in \Gamma$ can be described by the tuple $\{C_i, T_i\}$, 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. For all tasks executing on a processor, the hyperperiod can be defined as the least common multiple of all periods, $HP = \text{LCM}(\forall T_i, i \in \Gamma)$. Hence, a task τ_i executes a number of jobs during one HP, where its j^{th} job is denoted by $\tau_{i,j}$.

B. Communication Model

In this work inter task communication is realized via shared registers, a model commonly used in the industrial domain [10], [11]. With this, a sending task writes an output value to a shared register, which is then read by the receiving task without the need for any signaling between the communicating tasks. Also, the receiving task always consumes the newest value present in the shared register.

In order to facilitate determinism, a *read-execute-write* semantic is followed in which a task reads all its input values into *local copies* before the execution starts. It then executes by acting on these local copies and writes the output values after the execution back 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, i.e. at the beginning and end of the tasks execution respectively. This is a common communication mechanism found in several industrial standards (i.e. in AUTOSAR this model is defined as *implicit communication* [12], the standard IEC 61131-3 for automation systems defines similar communication mechanisms [13]).

C. End-to-End Timing Requirements

A cause-effect chain is typically specified by an end-to-end timing requirement, as defined for automotive systems in [1], [2]. In this work the *data age*, the most important timing requirement in control systems, is examined. A detailed discussion of corresponding end-to-end delays is provided in [10]. For data age, 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 is of interest. 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 over- and under-sampling between the different tasks. 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, 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 by τ_3 again. 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

A job-level dependency is similar to the rate transition operator of PRELUDE [4]. Defined between two tasks, a job-level dependency specifies which job of a task needs to finish its execution before a certain job of the successor task can start.

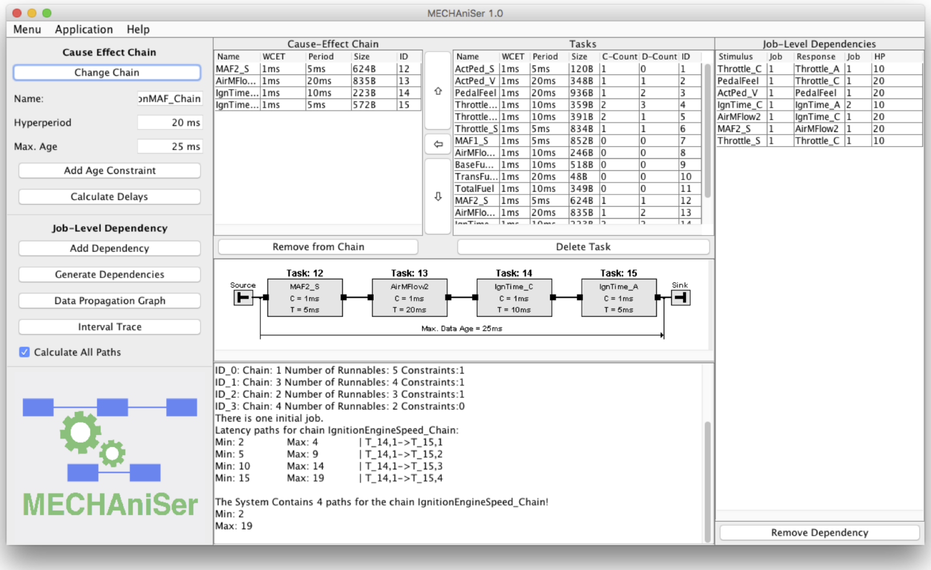
A job level dependency is described as $\tau_i \xrightarrow{(k,l)} \tau_j$, meaning that the k^{th} job of τ_i needs to proceed the l^{th} job of τ_j. This also implies that the dependency between the two jobs applies for the duration of the hyperperiod of the two jobs only, e.g. $LCM(\tau_i, \tau_j)$.

III. Calculating Latencies

In this section, we recapitulate the calculation of data propagation paths for systems without prior knowledge of the schedule. For a more in depth explanation a reader is referred to [9]. Several properties of tasks under register communication are observed to determine reachability between jobs. Based on this the different data propagation paths of the cause-effect chain can be calculated.

A. Reachability between Jobs

The concepts of *read interval* and *data interval* are central to decide if data can be propagated between two distinct jobs. For a job $\tau_{i,j}$, the read interval is defined as the interval starting from the earliest time $\tau_{i,j}$ can potentially read its input data ($R_{min}(\tau_{i,j})$) until the last possible time $\tau_{i,j}$ 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 $\tau_{i,j}$ can be available ($D_{min}(\tau_{i,j})$) up to the latest time a predecessor job of the same task overwrites the data ($D_{min}(\tau_{i,j})$). 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 the scheduling decisions, one has to assume that a job can be scheduled anywhere, as long as it starts not



Name:	ɔnMAF_Chain
Hyperperiod	20 ms
Max. Age	25 ms

[✓ Calculate All Paths]

Cause-Effect Chain

Name	WCET	Period	Size	ID
MAF2_S	1ms	5ms	624B	12
AirMFlo...	1ms	20ms	835B	13
IgnTime...	1ms	10ms	223B	14
IgnTime...	1ms	5ms	572B	15

Tasks

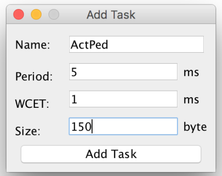
Name	WCET	Period	Size	C-Count	D-Count	ID
ActPed_S	1ms	5ms	120B	1	0	1
ActPed_V	1ms	20ms	348B	1	1	2
PedalFeel	1ms	20ms	936B	1	2	3
Throttle...	1ms	10ms	359B	2	3	4
Throttle...	1ms	10ms	391B	2	1	5
Throttle_S	1ms	5ms	834B	1	1	6
MAF1_S	1ms	5ms	852B	0	0	7
AirMFlo...	1ms	10ms	246B	0	0	8
BaseFu...	1ms	10ms	518B	0	0	9
TransFu...	1ms	20ms	48B	0	0	10
TotalFuel	1ms	10ms	349B	0	0	11
MAF2_S	1ms	5ms	624B	1	1	12
AirMFlo...	1ms	20ms	835B	1	2	13

Job-Level Dependencies

Stimulus	Job	Response	Job	HP
Throttle_C	1	Throttle_A	1	10
PedalFeel	1	Throttle_C	1	20
ActPed_V	1	PedalFeel	1	20
IgnTime_C	1	IgnTime_A	2	10
AirMFlow2	1	IgnTime_C	1	20
MAF2_S	1	AirMFlow2	1	20
Throttle_S	1	Throttle_C	1	10

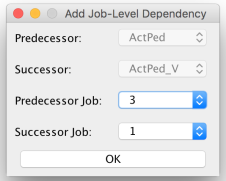
```
ID_0: Chain: 1 Number of Runnables: 5 Constraints:1
ID_1: Chain: 3 Number of Runnables: 4 Constraints:1
ID_2: Chain: 2 Number of Runnables: 3 Constraints:1
ID_3: Chain: 4 Number of Runnables: 2 Constraints:0
There is one initial job.
Latency paths for chain IgnitionEngineSpeed_Chain:
Min: 2          Max: 4          | T_14,1->T_15,1
Min: 5          Max: 9          | T_14,1->T_15,2
Min: 10         Max: 14         | T_14,1->T_15,3
Min: 15         Max: 19         | T_14,1->T_15,4

The System Contains 4 paths for the chain IgnitionEngineSpeed_Chain!
Min: 2
Max: 19
```

Fig. 3: Main view of the tool.



Name:	ActPed	
Period:	5	ms
WCET:	1	ms
Size:	150	byte

(a) Window to add a task.



Predecessor:	ActPed
Successor:	ActPed_V
Predecessor Job:	3
Successor Job:	1

(b) Window to add a dependency.

Fig. 4: Windows to add new elements.

before its release and finishes not after its deadline. In [9], the notations to define the intervals are as follows:

$$\begin{aligned} R_{min}(\tau_{i,j}) &= (j-1)\cdot T_i \\ R_{max}(\tau_{i,j}) &= R_{min}(\tau_{i,j+1}) - C_i \\ D_{min}(\tau_{i,j}) &= R_{min}(\tau_{i,j}) + C_i \\ D_{max}(\tau_{i,j}) &= R_{max}(\tau_{i,j+1}) + C_i \end{aligned}$$

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 same period as τ_i, but its execution time is shorter than the one of τ_k. $\text{Follows}(\tau_{i,1},\tau_{k,1})$ returns true and indicates that $\tau_{i,1}$ can potentially consume the data of $\tau_{k,1}$. 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 the original data interval must be used.

B. Calculating Data Paths

To calculate all possible data propagation paths in a system, a recursive function is used. 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, inside the hyperperiod of the chain.

As a result a set of data propagation paths is provided, where each path comprises an ordered list of involved jobs.

C. Constructing Data Propagation Paths and Max. Data Age

For a given data path, the maximum end-to-end latency and the data age, is computed. Given τ_{start} is a job of the first task of the cause-effect chain, and τ_{stop} is a job of the last task of a cause-effect chain:

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

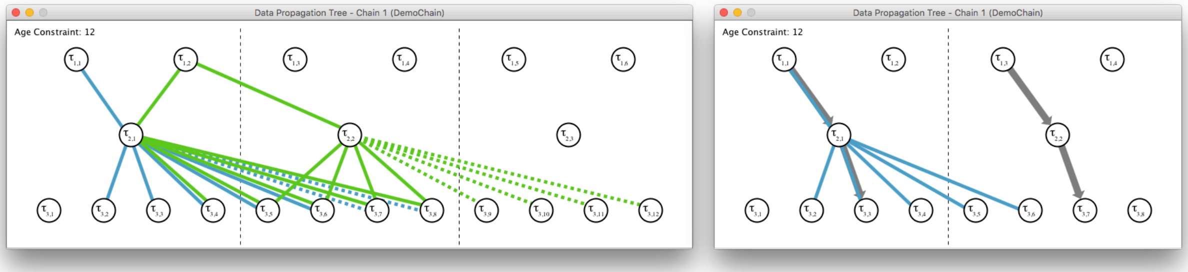
In order to compute the maximum data age for any possible path in the system, $\text{AgeMax}()$ must be computed for all data paths. The maximum of these values is the maximum data age of the cause-effect chain.

IV. Tool Layout and Usage

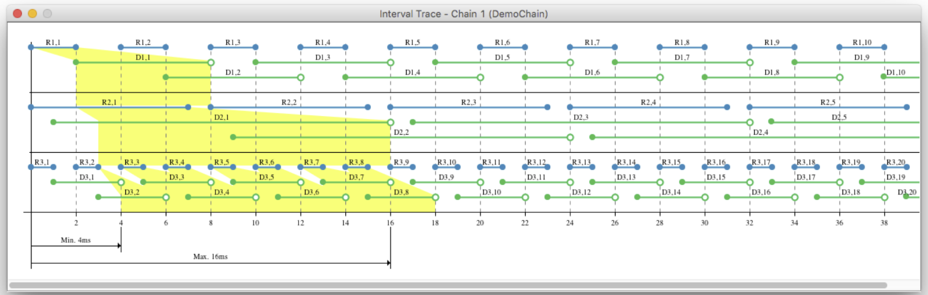
This section briefly outlines the different forms of data input to the tool. Further the tool layout and its usage are discussed and a closer look is provided into the different visualization options.

A. Input Formats

The tool specifies its own XML format to save a current project. Additionally it is possible to import projects designed with AMALTHEA V1.0[14]. AMALTHEA is an open tool platform for the design of multi-core systems in the automotive domain. The implementations for the support of additional


Data Propagation Tree - Chain 1 (DemoChain)
Age Constraint: 12
Data Propagation Tree - Chain 1 (DemoChain)
Age Constraint: 12

(a) The graph view in MECHAniSer. Edges between nodes depict possible data propagation while dashed edges show paths leading to larger data age then specified with the age constraint. The same chain with generated job-level dependencies is shown in the right window.



(b) The trace-view of MECHAniSer depicts the read- and data-interval of each involved job and visualizes the minimum and maximum data age of initial jobs as well as their possible data propagation range (in yellow).

Fig. 5: The two different visualization options for a cause-effect chain.

tools (i.e., AMALTHEA V1.1, Rubus ICE [15]) are currently ongoing and will be made available in the future.

B. Layout and Usage

The tool is built around a main panel which is shown in Fig 3. The panel depicts the chain under analyis and also provides clickable interfaces to additional features of the tool.

1) The Main Panel and its Parts: The main window displays information about all tasks of the system, in the *"Tasks"*-table, as well as on all specified job-level dependencies in the *"Job-Level Dependency"*-table. The selected chain is graphically visualized, as shown in Fig. 3, while the *"Cause-Effect Chain"*-table describes the different parameters of the chain. This chain can further be analyzed and modified. The left column also provides means to manage job-level dependencies. The additional views can also be opened here via the button *"Data Propagation Graph"* and *"Trace View"*. Output for the user is provided in the text-box at the bottom part of the window.

A user can add or delete a task over the *"Application"*-menu (see Fig. 4a) with the *"Add Task"* and *"Delete Task"* buttons. Note that the tool also displays the number of chains and the number of job-level dependencies that a task is part of. In order to keep the system consistent, a task must first be removed from all cause-effect chains and from all job-level dependencies before it can be removed from the system.

The chain which needs to be analyzed is selected via the button *"Change Chain"*. This action pops up a window wherein a user can select the desired cause-effect chain. Once approved, the tool updates the related views. A new task can be added to the chain by selecting the respective task in the task table and then clicking the left-arrow button which appends the task to the chain. The correct position of a task is set by selecting the task in the chain table and then clicking the up- and down-button which alter the tasks position. A task can be removed from the chain by selecting the task followed by the button *"Remove from Chain"*.

Finally a maximum data age constraint can be specified on the chain by clicking on the button *"Add Age Constraint"*. This pops up a window where the age constraint can be specified. Note that this new input overwrites any previously specified constraint. A constraint can be removed by specifying a maximum data age of 0.

2) Calculating Minimum and Maximum Data Age: The minimum and maximum data age of the currently selected cause-effect chain under consideration of all specified job-level dependencies can be computed by clicking on the button *"Calculate Delays"*. This action computes delays by applying the analysis presented in [9]. All data propagation paths are calculated, implying all possible paths that the data can propagate, when read from any of the initial jobs of the chain.

An initial job is defined as any job that the first task of the cause-effect chain releases during the first hyperperiod of the chain. Since the number of possible paths depends on the number of involved tasks as well as on the involved periods, a large number of data propagation paths might be generated. A user has hence the possibility to uncheck the option *"Calculate All Paths"* which will only calculate the data propagation path for the minimum and maximum job at each chain level. Hence this reduces the complexity of the calculation and simplifies the post processing by the system designer.

3) Adding and Synthesizing Job-Level Dependencies: The second strength of the tool is to handle job-level dependencies.

The left column of the main window provides means to add a job-level dependency manually as well as to synthesize job-level dependencies for all cause-effect chains in the system. The button *"Add Dependency"* opens a new window (see Fig. 4b) which allows to select the two involved tasks and the dependent instances. Note that first the two tasks need to be selected before the menu for the involved jobs becomes active. This is the case since, depending on the selected tasks, the available job instances change.

The button *"Generate Dependencies"* triggers a heuristic [9] which adds job-level dependencies to the system in a way that all specified age-constraints are met. Already specified dependencies are not affected. The main intuition behind the heuristic is that a placement of a job-level dependency can prune a branch of the data propagation tree. Hence the heuristic adds dependencies in a way such that all branches which lead to larger end-to-end delays than specified are removed.

4) The Graph View: The graph view, as shown in Fig. 5a, depicts the data propagation tree of the currently selected chain. Each data path originating from the different initial nodes is colored differently for a more effective visual presentation. The different jobs of the involved tasks are drawn in a way that the data always propagates from top to bottom, i.e. the beginning of the chain is at the top and the last task of the chain is at the bottom. Branches which lead to end-to-end delays larger than the specified constraint are shown in dashed lines. These branches need to be removed in order to meet the specified constraints. Note that this representation depicts no time information, the execution of the jobs depends on the exact path a data propagates and hence cannot be shown in this overview. However, jobs are grouped such that jobs of the same hyperperiod are arranged together and separated by the vertical dashed lines. A user can obtain further information of the different nodes by clicking on them which then displays an information box.

5) The Trace View: The trace view is shown in Fig. 5b. This view visualizes the read- and data-interval of all jobs of one chain (see Fig. 2 for a description). Initially the first initial job is selected and the propagation of the calculated data paths is visualized via yellow overlay. Additionally the minimum and maximum data age of these data paths are shown. A user can change this view to any other initial job by clicking on the respective read interval.

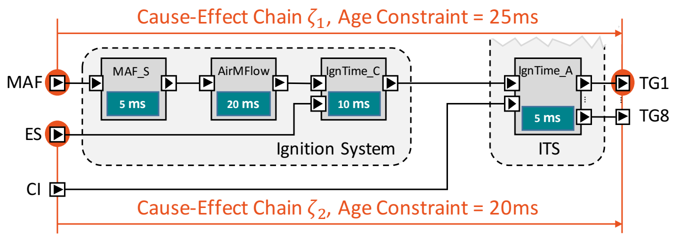
C. Implementation and Distribution

To be platform independent, the tool is developed in Java. The main development is performed under OSX which might cause a diverging visual appearance on other platforms. The tool is freely available online[1]. A user documentation and examples are provided under the same link.

V. Case Study

The applicability of the presented tool is demonstrated on a case study of an Engine Management System (EMS). This case study is adapted from the results presented in [11]. The EMS consists of several subsystems which control the air

[1]http://www.mechaniser.com


Cause-Effect Chain ζ1, Age Constraint = 25ms
MAF
ES
CI
MAF_S
5 ms
AirMFlow
20 ms
IgnTime_C
10 ms
Ignition System
IgnTime_A
5 ms
ITS
TG1
TG8
Cause-Effect Chain ζ2, Age Constraint = 20ms
Fig. 6: Tasks and specified cause-effect chains of the IS and ITS.

and gas mixture which is injected into the cylinders. The Air Intake System (AIS) controls the amount of air via the throttle position, while the Fueling System (FS) controls the amount of gas which is injected per stroke. The Ignition System (IS) controls the exact time of the ignition, both FS and IS feed into the Injection Time and Ignition Time Actuation System (ITS). For a smooth and energy efficient operation of the vehicle, several age constraints must be met. The complete EMS of this case study comprises 16 different tasks which three different periods (5 ms, 10 ms, and 20 ms)

Due to space limitations, we discuss only part of the complete EMS. The case study includes two cause-effect chains, ζ_1, and ζ_2, which are specified from the Mass Air Flow (MAF) input to the output for Ignition Time of cylinder 1 to 8 (TG1-8). The cause-effect chain ζ_2 is specified from the Engine Speed (ES) input up to TG1-8. Both chains span from the IS to the output of the ITS. 4 tasks with 3 different periods are involved (see Fig. 6) and WCETs of all tasks are set to 1 ms. We refer to [9] for a case study of the AIS.

A. Analysis of Latencies using MECHAniSer

Both specified cause-effect chains contain a number of runnables which are triggered at different periods. For the chain ζ_1 all four tasks are involved. The calculation of all data propagation paths results in 70 different paths, a minimum data age of 4 ms, and a maximum data age of 55 ms. The maximum possible data age exceeds the specified age constraint of 25 ms and the chain is not directly schedulable by the system. In the next step we will show how the tool generates job-level dependencies to remove the data propagation paths which exceed the constraint.

The second chain ζ_2 consists only of two tasks. Hence, the number of data propagation paths is smaller. Four paths are identified, with a minimum data age of 2 ms and a maximum data age of 20 ms. Here the specified age constraint of 20 ms is met without the need to specify job-level dependencies.

The required computation time for the analysis of the two chains is 5 ms and 2 ms for ζ_1 and ζ_2 respectively.

B. Synthesizing Job-Level Dependencies

The initial analysis of the two cause-effect chains revealed that, while ζ_2 meets its age constraint, ζ_1 does not. Hence job-level dependencies need to be generated in order to meet the constraint.

The tool generated three different job level dependencies in order to meet the constraint of the cause-effect chain. One job-level dependency was generated between each consecutive pair of tasks. This successfully reduces the maximum data age to 25 ms, allowing the cause-effect chain to meet its constraint.

The presence of the job-level dependencies further reduces the number of data propagation paths to 13. The required computation time is $19\,\mathrm{ms}$. Since ζ_2 is subset of ζ_1, the specified job-level dependency between the last two tasks of the cause-effect chain can have influence on ζ_2, hence ζ_2 needs to be revalidated as well. The job-level dependency specified for the two tasks is defined between the first job of task *IgnTime_C* to the second job of task *IgnTime_A*, the parameters are not influenced and the latency stays at $20\,\mathrm{ms}$ with 4 different data propagation paths.

VI. Related Tools

Many industrial standards specify constraints for the propagation of data through a chain of tasks [1], [2]. A detailed discussion of end-to-end delays is provided in [10]. The authors formally specify age- and reaction delays in multi-rate systems which communicate via register-communication and further develop a method to calculate end-to-end delays in such systems.

Several commercially available tools support the analysis of end-to-end delays in cause-effect chains. Examples are SymTA/S TraceAnalyzer for ECUs [16], Rubus ICE [15], and Timing Architects Inspector [17].

To the best of our knowledge the analysis presented in [10] is implemented in these tools [18], [19]. EELAP [20] is an open source end-to-end analyzer for the ProCom [21] real-time component model. The tool is built on the analysis of [10]. All these tools however require an existing schedule in order to analyze the system. Hence, the calculation of end-to-end delays in early design phases is not supported.

Several works address systems where job-level dependencies are specified [5], [6], [7], [8]. The application model in these works is specified by the *prelude* language [4] which specifies the rate-transition operation. On task level this operation is equivalent to a job-level dependency. The prelude compiler is available [22] and can generate synchronized multi-task C-code which then can be executed by the supported target OS. To the best of our knowledge, no tool exists that can automatically generate job-level dependencies in order to meet the end-to-end timing constraints.

VII. Conclusion and Future Work

In this paper we presented MECHAniSer, the first tool for the analysis and synthesis of multi-rate cause-effect chains with specified job-level dependencies. The tool allows to analyze systems at early design phases, where detailed scheduling knowledge is not available. Further, the tool synthesizes job-level dependencies for a set of cause-effect chains in a way that all their end-to-end timing constraints are met. This allows such systems to be scheduled on any platform which supports these concepts [5], [6], [7], [8].

The tool provides its own XML format to store the project configurations but it also provides the possibility to import projects from existing tools and hence eases the design process. Multiple graphical views are provided to support the system designer and to ease the understanding of the data propagation in multi-rate cause-effect chains. Several extensions to the tool are possible. One limitation of the current implementation is the time granularity. Future versions of the tool will allow to specify time values in smaller granularity than ms. Besides data age, many industrial applications specify reaction constraints. Analysis for this type of constraint is currently not supported but will be part of future work.

Acknowledgment

The work presented in this paper is supported by the Swedish Knowledge Foundation (KKS) through the projects PREMISE and DPAC; and the Swedish Foundation for Strategic Research (SSF) through the projects PRESS.

References

[1] *AUTOSAR - Spec. of Timing Extensions*, AUTOSAR Std. 4.2.2, 2014.

[2] *EAST-ADL - Domain Model Specification*, EAST-ADL Association Std. V2.1.12, 2014.

[3] S. Kramer, D. Ziegenbein, and A. Hamann, "Real world automotive benchmarks for free," in *6th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems*, 2015.

[4] 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.

[5] 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.

[6] 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.

[7] W. Puffitsch, E. Noulard, and C. Pagetti, "Off-line mapping of multi-rate dependent task sets to many-core platforms," *Real-Time Systems*, vol. 51, no. 5, pp. 526–565, 2015.

[8] ——, "Mapping a multi-rate synchronous language to a many-core processor," in *19th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2013, pp. 293–302.

[9] M. Becker, D. Dasari, S. Mubeen, M. Behnam, and T. Nolte, "Synthesizing job-level dependencies for automotive multi-rate effect chains," in *Proceedings of the 22th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, [Online] http://www.es.mdh.se/publications/4368-, 2016.

[10] 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 *Int. Workshop on Compositional Theory and Technology for Real-Time Embedded Systems*, 2008.

[11] P. Frey, "Ulmer Informatik Berichte Nr 2010-03 - Case Study: Engine Control Application," University Ulm, Tech. Rep., 2010.

[12] *AUTOSAR - Specification of RTE*, AUTOSAR Std. 4.2.2, 2014.

[13] *IEC 61131-3*, International Electrotechnical Commission Std., 2003.

[14] AMALTHEA, "An Open Platform Project for Embedded Multicore Systems," [Online] http://www.amalthea-project.org/index.php/contact, last visited 16.05.2016.

[15] Arcticus Systems, "Rubus ICE," [Online] https://www.arcticus-systems.com/products/, last visited 16.05.2016.

[16] Symtavision GmbH, "SymTA/S and TraceAnalyzer for ECUs," [Online] https://www.symtavision.com/products/ecu-timing/, last visited 16.05.2016.

[17] Timing Architects, "Timing Architects Inspector," [Online] https://www.timing-architects.com/ta-tool-suite/inspector/, last visited 16.05.2016.

[18] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst, "System level performance analysis - the symta/s approach," *IEE Proceedings - Computers and Digital Techniques*, vol. 152, no. 2, pp. 148–166, 2005.

[19] 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.

[20] J. Kuncar, R. Inam, and M. Sjödin, "End-to-end latency analyzer for ProCom - EELAP," Tech. Rep. ISSN 1404-3041 ISRN MDH-MRTC-272/2013-1-SE, 2013.

[21] R. Inam and M. Sjödin, "Implementing and evaluating communication-strategies in the procom component technology," *SIGBED Rev.*, vol. 9, no. 4, pp. 41–44, 2012.

[22] Prelude, "programming critical real-time systems," [Online] http://www.lifl.fr/%7Eforget/prelude.html, last visited 16.05.2016.