Comparing Trace Recordings of Automotive Real-time Software[*]

Andreas Sailer
Timing-Architects Embedded Systems GmbH
Franz-Mayer-Straße 1
93053, Regensburg, Germany
andreas.sailer@timing-architects.com

Michael Deubzer
Timing-Architects Embedded Systems GmbH
Franz-Mayer-Straße 1
93053, Regensburg, Germany
michael.deubzer@timing-architects.com

Gerald Lüttgen
Otto-Friedrich-University Bamberg
Software Technologies Research Group
An der Weberei 5
96047, Bamberg, Germany
gerald.luettgen@swt-bamberg.de

Jürgen Mottok
Ostbayerische Technische Hochschule (OTH) Regensburg
Laboratory of Safe and Secure Systems (LaS^3)
Seybothstraße 2
93053, Regensburg, Germany
juergen.mottok@oth-regensburg.de

ABSTRACT

The process of engineering models of existing real-time system components is often difficult and time consuming, especially when legacy code has to be re-used or information about the exact timing behaviour is needed. In order to tackle this reverse engineering problem, we have developed the tool CoreTAna. CoreTAna derives an AUTOSAR compliant model of a real-time system by conducting dynamic analysis using trace recordings.
Motivated by the challenge of assessing the quality of reverse engineered models of real-time software, we present a novel mathematical measure for comparing trace recordings from embedded real-time systems regarding their temporal behaviour. We also introduce a benchmark framework based on this measure, for evaluating reverse engineering tools such as CoreTAna. This considers common system architectures and also includes randomly generated systems and three systems of industrial automotive projects. Finally, an industrial case study demonstrates other use cases of our measure, such as impact analysis.

KEYWORDS

Real-time systems, tracing, profiling, timing model, reverse engineering, AUTOSAR, CoreTAna

1 INTRODUCTION

Since the release of the AUTomotive Open System ARchitecture (AUTOSAR) (http://www.autosar.org) standard V3.0 in 2007, AUTOSAR has experienced gradual acceptance. One of its major goals is the definition of a software architecture for the development of real-time systems and a corresponding development methodology. For this purpose, a model containing an abstract description of a system is created on which standard software, such as the operating system or runtime environment, is then configured. However, the process of deriving a model is often difficult, error-prone, and time consuming, especially when legacy code is reused [11]. The consideration of timing behaviour is an extra challenge that requires substantial effort [2], but is essential to the modelling process. Over the past years, a variety of tools have been developed to address these difficulties by automatically analysing different aspects of a real-time system, including its exact timing behaviour, based on the system's trace recordings [16].
This paper discusses the assessment of tools such as CoreTAna (Sec. 2), which tackle this reverse engineering problem. The crucial task of the assessment is to determine how closely a synthesised model reflects the actual system. To meet this challenge, we introduce a novel measure (Sec. 3), which compares trace recordings from embedded real-time systems regarding their temporal behaviour. We also introduce a synthetic benchmark (Sec. 4), which consists of different system descriptions that cover common system architectures in the real-time software domain, for assessing the quality of reverse engineering tools. To highlight the applicability of our measure and to show the validity of this benchmark, we provide trace recordings of these system descriptions to CoreTAna. CoreTAna deduces an abstraction of the underlying real-time system and synthesises a model in order to enable simulations of the system's timing behaviour. We then use such a model and our measure to assess the quality and performance of the reverse engineering offered by CoreTAna. For evaluating our benchmark (Sec. 5), trace recordings of randomly generated systems are applied to CoreTAna, which also demonstrate CoreTAna's capabilities and performance for actual automotive projects. Additionally, we conduct an industrial case study (Sec. 6), in which our measure is employed to highlight the differences between products of the same product family. This shows that our measure can also be applied to other use cases.

2 CONTEXT

The current shift towards multi-core architectures in the automotive industry is forcing OEMs and Tier-1's to gain detailed knowledge about their legacy software. Most tools that tackle the challenges implied by this shift, such as finding an ideal task-to-core allocation or task priority assignment, are model-based and work in compliance with the AUTOSAR standard. Deriving, or reverse engineering, such a standard compliant model is often difficult and time consuming.

[*]This work was partially funded by the research project FORMUS^{3}IC via the Bavarian Research Foundation (BFS) under funding code AZ-1165-15.
Many thanks to the reviewers for their valuable feedback.

2.1 Reverse Engineering Real-Time Software

In order to tackle the challenge of reverse engineering an AUTOSAR compliant model of a real-time, single- or multi-core system, including its exact timing behaviour, via a dynamic analysis based on the system's trace recordings, we have developed CoreTAna [16]. This tool derives such a model either automatically from scratch or by enriching an existing model. The fact that CoreTAna creates a standard compliant artefact, which not only allows one to analyse the system's behaviour via timing simulation but also enables further processing such as system optimisation [17], sets this tool apart from existing solutions [7, 9, 14].

Technically, CoreTAna processes events that can be observed and recorded during the runtime of a system in a step-by-step manner. Every event indicates a change in the internal state of the system due to, e.g., function calls or data accesses. These pieces of information are taken from trace recordings of the real-time system under study and used by CoreTAna to deduce an abstraction of the system's structure and timing behaviour. Due to space constraints, we refer the reader to [16] for a detailed description of CoreTAna.

2.2 Problem Definition

Because CoreTAna performs a dynamic analysis, the only artefact that can be used for the evaluation of CoreTAna's quality of reverse engineering are the specific moments in time at which certain events are observed during system execution. As the synthesised model ideally reflects the same temporal behaviour, the most obvious evaluation approach is to compare the trace recordings of the actual system with those generated by simulating the synthesised model [15]. As depicted in Fig. 1, CoreTAna employs the Timing-Architects (TA) Simulator [20] for this purpose. This commercial model-based tool is used by many Tier-1 suppliers and OEMs in the automotive industry and allows one to simulate the timing behaviour of AUTOSAR compliant real-time systems.

As presented in the next section, a lot of research has been conducted regarding the comparison of trace recordings. Most, however, focuses on statistical methods, yielding results that suggest an insufficient correspondence. This is due to the fact that a model represents just an abstraction of a real system, where characteristics are summarised at the available level of detail. Thus, a model describes rather a similar system than the same exact system that underlies the model.

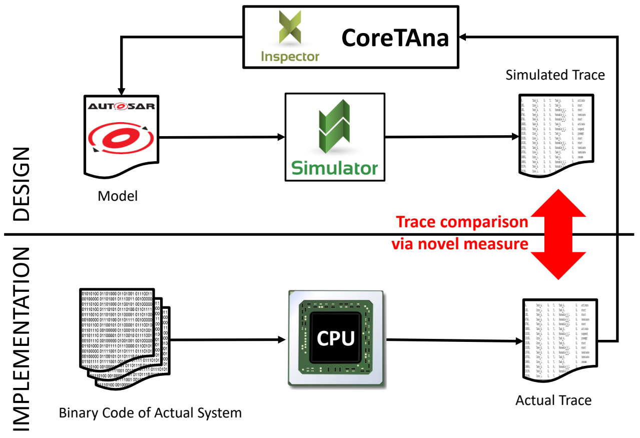

Figure 1: Schematic approach for analysing how closely an AUTOSAR model synthesized by CoreTAna reflects the actual system.

2.3 Related Work

The problem of comparing trace recordings from embedded real-time systems regarding their temporal behaviour is similar to the problem of validating simulation models [13]. Many different techniques are available that tackle the problem of simulation model validation [4, 12, 18]: some are based on statistical methods such as goodness-of-fit tests [9, 13], while others use mathematical procedures. Because of the aforementioned shortcoming of statistical methods, only mathematical procedures are considered in the following.

Anderson discusses in [1, p.108ff] the equivalence of observable attributes such as response times, patterns and resource utilizations in trace recordings. Nevertheless, no explicit tolerances are suggested for comparing these attributes.

Huselius introduces in [7, p.122ff] an objective measure called "Sum of Divergence" which establishes a bijective mapping between two response time samples in order to summarise the differences in their distributions. Disadvantages of this solution are that the *Least Common Multiple* (LCM) is used to obtain equally-sized sampled distributions and also that the measure is normalised by the maximum difference between samples in the distributions. A single scattered outlier can thus have a big impact on the quality of the result, because the outlier can, e.g., be multiplied by the LCM or yield an improper difference for normalisation.

Nemati et al. describe in [14] an algorithm that first divides trace recordings into equally sized time windows and then calculates the differences in resource-consumption attributes, such as the execution times of tasks, between correlating time windows. However, not only is it difficult to determine in advance a leeway for difference, but the traces also have to start from exactly the same state and have to contain the same sequences of states. Especially, the latter cannot be guaranteed for probabilistic models.

Due to the absence of a suitable assessment in the quality of reverse engineered real-time software, we have developed a new measure.

3 OUR MEASURE

Before we present details of our measure for comparing the temporal behaviour of real-time systems, the foundations of trace recordings and their representation of system behaviour are discussed.

3.1 Foundations

The temporal behaviour of a real-time system is primarily determined by the employed scheduling policy, by which the operating system decides which task can run on what processing unit. Consequently, tasks evolve through the following states during a system's execution according to the BTF trace specification [19]:

ACTIVE The task is ready to be executed and waits for allocation of a processing unit for the first time.

NOT INITIALIZED The task is passive and can be activated.

PARKING The task has requested a resource that is not available and has been preempted while waiting for its release.

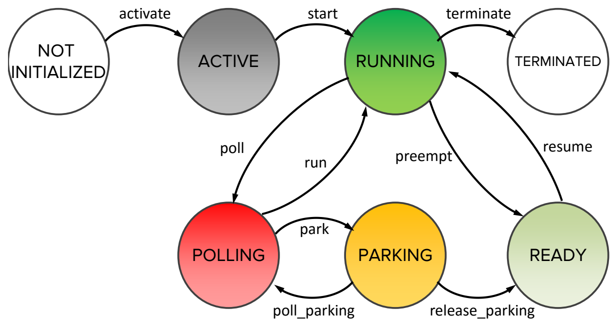

Figure 2: AUTOSAR task state model, extended by states for active (POLLING) and passive (PARKING) waiting for a resource.

POLLING The task has requested a resource that is not available and waits actively for its release.

READY The task fulfils all functional prerequisites for execution and waits to be allocated a processing unit.

RUNNING The task has been assigned to a processing unit, and its instructions are being executed.

TERMINATED The task has finished executing its instructions and is passive.

These states and their transitions to each other are depicted in Fig. 2. In this figure, the basic task state model used in AUTOSAR [3], which is adopted from the OSEK standard, is refined in order to distinguish some task states more precisely. Originally, the OSEK basic task state model specifies the following three states [8, p. 18]: running, ready and suspended. This means that the states *NOT INITIALIZED* and *TERMINATED* in Fig. 2 are equal to the OSEK state *suspended*, the states *ACTIVE*, *PARKING* and *READY* all correspond to the OSEK state *ready*, and the state *RUNNING* together with the state *POLLING* describe the OSEK state *running*. State *WAITING*, which represents another state in AUTOSAR and BTF, is omitted here because no example in Sec. 4 and Sec. 5 requires this state. The states *PARKING* and *POLLING* matter mainly in multi-core systems and, thus, they are only relevant for the examples in Sec. 5.

The transitions of a task from one state to another during a system execution can be observed and logged. This process of detecting and storing relevant events during runtime for later off-line analysis is called trace recording or, for short, *tracing* [10, p. 1]. Possible sequences of observed state transitions for a task *T* are shown by the traces in Listings 1 and 2.

Listing 1: Trace sample 1

```
 0, T, activate
 0, T, start
 5, T, terminate
10, T, activate
10, T, start
16, T, terminate
```

Listing 2: Trace sample 2

```
 1, T, activate
 1, T, start
 5, T, terminate
12, T, activate
12, T, start
17, T, terminate
```

Both traces respect the BTF trace format [19], where each line represents an observed event. An event is defined by three attributes, all separated by comma. The first attribute indicates the moment in time when the event occurred. The next attribute identifies the entity that caused the event, in this case a task called *T*. Finally, the action that triggered the event is stated, which is, in this example, a state transition of task *T*. These pieces of information can then be used to visualise system execution in a Gantt chart, as shown in Fig. 3.

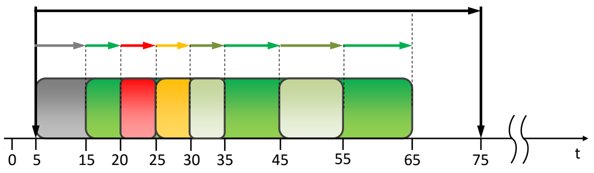

Figure 3: Visualisation of a trace example in a Gantt chart showing (via colours) a task in different states over time.

Another way to objectively analyse a system's temporal behaviour is to calculate metrics from a trace recording [5]:

A2A *Activation-To-Activation*: The distance between two successive activations of a task. In Fig. 3, this is indicated by the horizontal black arrow.

NET *Net Execution Time*: The actual execution time of a task, which spans from the starting point to the termination point and excludes the time during which the task is interfered. In Fig. 3, this is indicated by the sum of the lengths of all dark green arrows.

Parking *Parking Time*: The timespan of a preempted task waiting for a requested resource. In Fig. 3, this is indicated by the length of the orange arrow.

Polling *Polling Time*: The timespan of a task actively polling for a requested resource. In Fig. 3, this is indicated by the length of the red arrow.

Ready *Ready Time*: The timespan of a task between its start and termination in which it is not executed on any processing unit. In Fig. 3, this is indicated by the sum of the lengths of all light green arrows.

SD *Start Delay*: The time from the activation moment of the task to the moment of its start. In Fig. 3, this is indicated by the length of the grey arrow.

Because these metrics comprehend the temporal behaviour of a real-time system, we apply them to compare trace recordings. So far, only a trace recording at task level has been considered by us, which means that only the state transitions of tasks are detected and stored. Besides tasks, also other entities, e.g., functions, which are called *runnable* in automotive terminology, and data signals, as well as their corresponding events, such as function calls and data accesses, can be observed during a system's runtime. Thereby, it is possible to get a more detailed insight into the system's behaviour, which not only allows to assess its real-time performance, but also to compare different behaviour by defining a distance measure for trace recordings.

3.2 Definition of our Measure

To use the aforementioned metrics for comparing the temporal behaviour of real-time systems, we initially determine the existence of corresponding entities in the comparative trace recordings. A correspondence between individual entities can be established based on similarities, such as unique identifiers, identical memory locations or similar behaviour.

Definition 3.1. Amount Distance Δ_A: Let $P(s_i)$ be the task entities within a trace sample s_i, and let $|X|$ denote the cardinality of a set X. The Amount Distance Δ_A is defined by

$$\Delta_A(s_1, s_2) = 1 - \frac{|P(s_1) \cap P(s_2)|}{|P(s_1) \cup P(s_2)|}. \tag{1}$$

To compare the amount of same entities in two trace samples, Eq. 1 considers only task entities and not all observable entities. This is because technical limitations rule out the recording of all entities at once, whereas a reversely engineered model can contain additional entities to reproduce correct behaviour. Nevertheless, the temporal behaviours of tasks in both samples have to match, which then also indirectly reflects a correct representation of other entities, e.g., functions.

Definition 3.2. Entity Distance Δ_E: Let $E(s_i)$ be all task and function entities within a trace sample s_i, and let w_m be the balanced weight for a metric m. The Entity Distance Δ_E is defined by

$$\Delta_E(s_1, s_2) = \sum_{e \in E(s_1) \cap E(s_2)} \frac{\sqrt{\sum_{m \in M} w_m \cdot (m_{s_1} - m_{s_2})^2}}{|E(s_1) \cap E(s_2)|}, \tag{2}$$

where

- $P(s_i) \subseteq E(s_i)$ and $|E(s_1) \cap E(s_2)| > 0$
- $w_m = \frac{1}{|M|}$
- $M = \{min_t, max_t, \bar{x}_t, Q_{1,t}, Q_{2,t}, Q_{3,t}, IQM_t \mid \forall t \in \{NET, A2A, SD, Ready, Parking, Polling\}\}$; here, min_t is the minimum, max_t is the maximum, $\bar{x}_t$ is the mean, $Q_{1,t}$ is the lower quartile, $Q_{2,t}$ is the median, $Q_{3,t}$ is the upper quartile, and IQM_t is the interquartile mean of a time metric t.

The Entity Distance Δ_E determines the differences between all entities, including functions and tasks, that are available in both samples. These entities are compared by calculating the weighted Euclidean distance between the temporal behaviours recorded in the one sample and in the other sample. Due to the fact that each trace recording usually contains multiple observations of the same entity and, thus, also a variety of temporal characteristics, measures of descriptive statistics are used to quantitatively summarise the general behaviour for each entity. Measures of spread or shape are not considered because they would lead to inconsistent results, e.g., if the variances of two comparable entities are exactly the same but their individual times are far apart. In this case, the temporal behaviour is completely different, but an alignment in the variance would presume otherwise. Instead, multiple measures of location (minimum, maximum, mean, etc.) are used to capture differences in variability. The weight of the Euclidean distance is chosen in such a way that each metric measure contributes to the same extent, because none outranks the others in importance.
To achieve a distance measure within the range [0,1], each metric value is scaled to the maximum value for the metric in all samples: $x'_t = \frac{x_t}{max_t(s_1, s_2, \ldots, s_n)}$. As an example, Table 1 lists two metrics including their scaling, which comprehend the temporal behaviour of the trace samples s_1 and s_2 in Listing 1 and, resp., Listing 2. Applying the scaled metrics to Eq. 2 then yields a distance of task T

Table 1: Exemplary real-time metrics t and their scaled means $\bar{x}'_t$, determined for task T in trace samples s_1 (Listing 1) and s_2 (Listing 2).

t	s_1			s_2		
	X_t	X'_t	$\bar{x}'_t$	X_t	X'_t	$\bar{x}'_t$
A2A	{10}	$\{\frac{10}{11}\}$	$0.\overline{90}$	{11}	$\{\frac{11}{11}\}$	1
NET	{5,6}	$\{\frac{5}{6}, \frac{6}{6}\}$	$0.91\overline{6}$	{4,5}	$\{\frac{4}{6}, \frac{5}{6}\}$	0.75

between the two trace samples of approximately 7.7 %:

$$\Delta_E(s_1, s_2) = \frac{\sqrt{\frac{1}{6} \cdot (0.\overline{90} - 1)^2 + \frac{1}{6} \cdot (0.91\overline{6} - 0.75)^2 + 0}}{1} \approx 0.077.$$

Finally, the equality in the amount of same entities in both samples and the differences of each individual entity are combined in our Distance Measure Δ, to obtain the distance between two sample trace recordings.

Definition 3.3. Distance Measure Δ: Let s_1, s_2 be two sample trace recordings, let Δ_A denote the Amount Distance, and let Δ_E be the Entity Distance. Then, the Distance Measure Δ is defined by

$$\Delta(s_1, s_2) = 1 - [1 - \Delta_A(s_1, s_2)] \cdot [1 - \Delta_E(s_1, s_2)]. \tag{3}$$

4 SYNTHETIC BENCHMARK

One motivation for us to propose a novel measure is to assess the quality of our reverse engineering tool, CoreTAna, with the help of a benchmark. The goal of this benchmark is to provoke realistic challenges of the real-time development process, e.g., blocking situations, which then have to be managed by a reverse engineering tool under evaluation. The basic idea for our synthetic benchmark is inherited from the work of Huselius [7, p. 111ff], where so-called *Archetypes*, which represent common architectural patterns in the real-time software domain, and feasible variations for each pattern, so-called *PICs*, are described. Unfortunately, Huselius only gives a general description of each Archetype, which makes it impossible for us to reproduce them precisely.
Because reproducibility constitutes the foundation of any scientific benchmark, we have made all systems that are described in this section public, as example models in the Eclipse APP4MC Release Version 0.7.2 (https://www.eclipse.org/app4mc/), which is an open-source platform for engineering embedded multi- and many-core software systems. Our models cover all Archetypes originally introduced by Huselius, but have been extended by additional variations to consider AUTOSAR-specific aspects and to cover the increased functionality provided by CoreTAna. The benchmark is designed in a consecutive way such that each variation within a pattern alters the previous one by a single aspect. To highlight the impact of the changes made by each variation and to show the expressiveness of our measure, we state in brackets how traces of each variation differ from traces of the previous one using our distance measure Δ and Huselius' *Sum of Divergence* (SoD).
To be able to conduct an exemplary evaluation on measuring the performance and the quality of CoreTAna's reverse engineering, we generate a simulation trace that covers a specified number of time units for each system using the TA Simulator [20]. If not indicated otherwise, the generated trace recordings are not limited to specific

BTF events [19], so as to provide a detailed insight into a system's behaviour. The traces are then analysed by CoreTAna, which is included in the TA Tool Suite Release 16.3 [20]. This analysis is performed on a workstation containing an Intel Core i7-4930K hexa-core CPU, where each core is clocked at a frequency of 3.4 GHz, with 32 GB of RAM and running the 64-bit version of Windows Server 2012.
To manifest a high quality of the performed reverse engineering, our measure has to yield results close to zero percent when comparing the trace recordings of the actual system with those generated when simulating the synthesised model. Mismatches in the recorded behaviour are revealed by our measure due to the fact that the system's scheduling propagates each disparity on and on, and due to the quadratic characteristic of the used Euclidean distance. We determine Huselius' Sum of Divergence not only to compare it with our measure but also to substantiate its shortcomings mentioned in Sec. 2.3.

4.1 Purely Periodic without Communication

This system architecture pattern consists of a task set of seven tasks, in which each task is activated periodically and no data accesses are performed. The execution time for each task is determined by so-called *runnable entities*. All tasks contain just one runnable, except for T_7 that calls at first $R_{7,1}$ and after that $R_{7,2}$. The variations applied to this system pattern are:

1) **Initial Task Set:** The tasks T_4, T_5, T_6 and T_7 are active and scheduled according to fixed-priority preemptive scheduling.
2) **Increase of Task Set Size I ($\Delta_{1,2} \approx 28.6$ %, SoD$_{1,2} \approx 32.1$%):** The tasks T_3, T_4, T_5, T_6 and T_7 are active. Thereby, the utilisation of the system is increased.
3) **Increase of Task Set Size II ($\Delta_{2,3} \approx 27.3$ %, SoD$_{2,3} \approx 65.5$ %):** The tasks T_1, T_3, T_4, T_5, T_6 and T_7 are active, i.e., the utilisation of the system is further increased.
4) **Increase of Task Set Size III ($\Delta_{3,4} \approx 22.7$ %, SoD$_{3,4} \approx 41.5$ %):** From this variation through to variation 9, all tasks (T_1 - T_7) are active. This increases the utilisation of the system again.
5) **Accuracy in Logging ($\Delta_{4,5} \approx 0.0$ %, SoD$_{4,5} \approx 0.0$ %):** A trace containing just task events is used (see [19]). Thereby, only a limited insight into the system's runtime behaviour is available for reverse engineering.
6) **Schedule ($\Delta_{5,6} \approx 15.4$ %, SoD$_{5,6} \approx 22.6$ %):** From this variation through to variation 9, T_7 is set to non-preemptive. Hence, the system's timing behaviour is changed, which results in ineffective activations, because the maximum number of queued activation requests is exceeded.
7) **Activation ($\Delta_{6,7} \approx 0.6$ %, SoD$_{6,7} \approx 9.4$ %):** From this variation through to variation 9, the maximum number of queued activation requests is set to 2 for all tasks. This solves the problem of queue overflows for activation requests in the previous variation.
8) **Schedule Point ($\Delta_{7,8} \approx 5.2$ %, SoD$_{7,8} \approx 18.0$ %):** A scheduler call is added to T_7 between the calls of $R_{7,1}$ and $R_{7,2}$. This changes the timing behaviour.
9) **Scheduling Algorithm ($\Delta_{8,9} \approx 11.2$ %, SoD$_{8,9} \approx 24.0$ %):** The scheduling algorithm is set to *Earliest Deadline First*, so that the timing behaviour is changed completely.

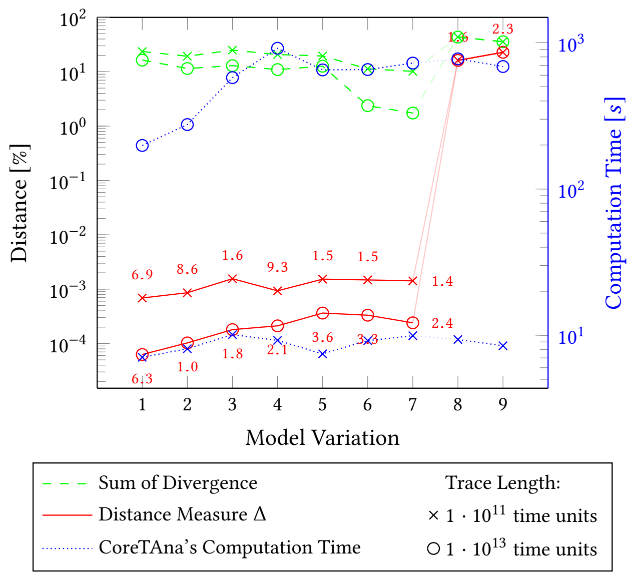

Figure 4: Results of CoreTAna for the variations of system architecture pattern 'Purely Periodic without Communication'. Each red marker, resp., green marker denotes the result of our measure Δ, resp., Huselius' Sum of Divergence for comparing a trace from a variation of this pattern with one generated when simulating CoreTAna's reversely engineered model. A blue marker indicates the time it took CoreTAna to synthesise the model from the trace. The meaning of markers and coloured lines defined in the legend is identical throughout all further result visualisations shown in Figs. 6, 8, 10 & 12. The reduced opacity in the lines isolates variations that feature characteristics that are not supported by CoreTAna (i.e., Schedule Points and EDF Scheduling).

For each of these systems, a trace recording is applied to CoreTAna and the difference to those generated when simulating the synthesised model is determined using our measure Δ. The results are shown in Fig. 4 and turn out to be twofold. If all system characteristics are supported by CoreTAna's underlying reverse engineering approach, the synthesised model reflects the actual system behaviour very well. This is demonstrated by the high similarity between the trace recordings of Variations 1 to 7 in the figure. However, if at least one system characteristic, such as the scheduling algorithm, is not considered in CoreTAna, then the difference rises rapidly. This is due to the fact that, even if only one task is affected by a variation, this change can have an impact on all tasks because of the scheduling. Moreover, the Euclidean distance used in our measure has a quadratic characteristic.
As shown by the blue dotted line in Fig. 4, the time it takes CoreTAna to generate a model grows linearly with the trace length, resp., with the amount of events contained within the trace.

4.2 Client-Server without Reply

This system architecture pattern extends the previous one by adding one-way communication between tasks. The implemented task set is depicted in Fig. 5. The variations applied to this system pattern are:

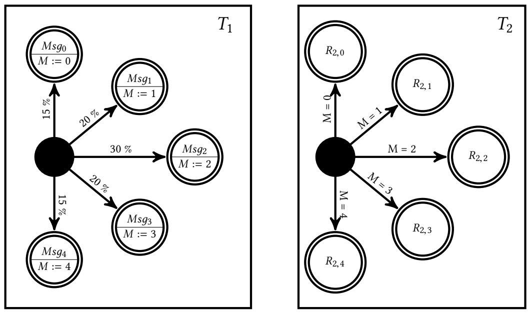

Figure 5: State diagram implemented by the system architecture pattern 'Client-Server without Reply'.

1) **Initial Task Set:** All tasks as defined above are scheduled according to fixed-priority preemptive scheduling.
2) **Exclusive Area ($\Delta_{1,2} \approx 32.3$ %, SoD$_{1,2} \approx 9.4$ %):** For this variation, all data accesses are protected by a mutex and the priority ceiling protocol is applied. Hence, blocking situations appear.
3) **Inter-Process Activation ($\Delta_{2,3} \approx 34.7$ %, SoD$_{2,3} \approx 21.0$ %):** From this variation onwards, task T_2 gets activated by an inter-process activation from task T_1, so that a direct connection between T_1 and T_2 is established.
4) **Priority Ordering ($\Delta_{3,4} \approx 14.2$ %, SoD$_{3,4} \approx 57.71$ %):** From this variation onwards, the priority relation between tasks T_1 and T_2 is reversed. Thereby, a switch from asynchronous to synchronous communication is realised.
5) **Event Frequency Increase ($\Delta_{4,5} \approx 3.0$ %, SoD$_{4,5} \approx 0.1$ %):** From this variation onwards, the periodicity of T_1 is shortened so that system utilisation is increased.
6) **Execution Time Fluctuation ($\Delta_{5,6} \approx 0.6$ %, SoD$_{5,6} \approx 0.2$ %):** From this variation onwards, the execution time distribution is widened for both tasks. Hence, the system utilisation is increased further, which results in ineffective activations, because the maximum number of queued activation requests is exceeded.
7) **Activation ($\Delta_{6,7} \approx 4.5$ %, SoD$_{6,7} \approx 0.1$ %):** From this variation onwards, the maximum number of queued activation requests (see [3, p.227]) for both tasks is set to 2. Thereby, the problem with activation requests resulting from the previous variation is solved.
8) **Accuracy in Logging I ($\Delta_{7,8} \approx 0.0$ %, SoD$_{7,8} \approx 0.0$ %):** For this variation, a trace containing just task and runnable events is used. Hence, only a limited insight into the system's runtime behaviour is available for reverse engineering.
9) **Accuracy in Logging II ($\Delta_{8,9} \approx 0.0$ %, SoD$_{8,9} \approx 0.0$ %):** For this variation, a trace containing just task events is used. This means, again, that only a limited insight into the system's runtime behaviour is available for reverse engineering.

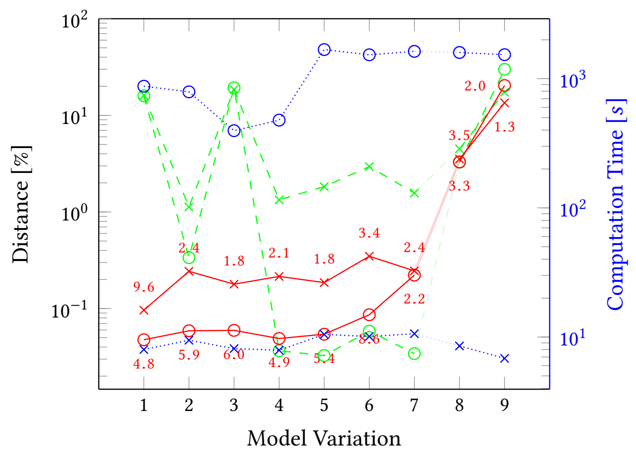

Figure 6: Results of CoreTAna for the variations of system architecture pattern 'Client-Server without Reply'. The meaning of markers and coloured lines is according to the legend defined in Fig. 4. The reduced opacity in the lines distinguishes variations that do not vary within the system but instead limit the level of detail available for reverse engineering.

On closer examination of the results visualised in Fig. 6, the consequences of the increased complexity of this system pattern when compared to the previous one stand out. This manifests itself in the fact that the values resulting from our distance measure Δ differ by a factor of 100 (see the red markers around 10^{-1} % in Fig. 6 vs. those around 10^{-3} % in Fig. 4). In addition, a rise of the red lines towards Variations 8 and 9 is clearly visible. This is due to the reduced amount of information contained within the trace, which is caused, e.g., by undersampling or a decrease of detail.

4.3 State Machine

In this system architecture pattern, the previous one is extended in such a way that task T_2 that receives messages varies its dynamic behaviour and, consequently, its execution time not only according to the transmitted content but also according to its current internal state, i.e., the previously transmitted contents. The implemented state machine is depicted in Fig. 7.

The variations applied to this system pattern are equal to those described in Sec. 4.2. Only the sequence in which the underlying system is modified by each variation is slightly changed, in order to provoke realistic challenges such as exceeding the maximum number of queued activation requests.

The results for this pattern, which are visualised in Fig. 8, show a noticeable similarity to those for the previous pattern. In particular, the differences between the peaks and valleys of the results for the

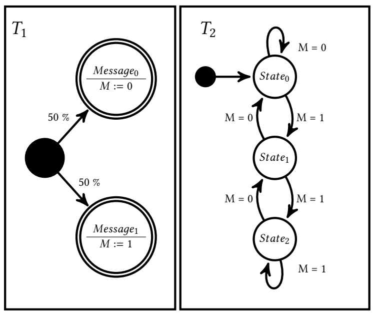

Figure 7: State diagram implemented by the system architecture pattern 'State Machine'.

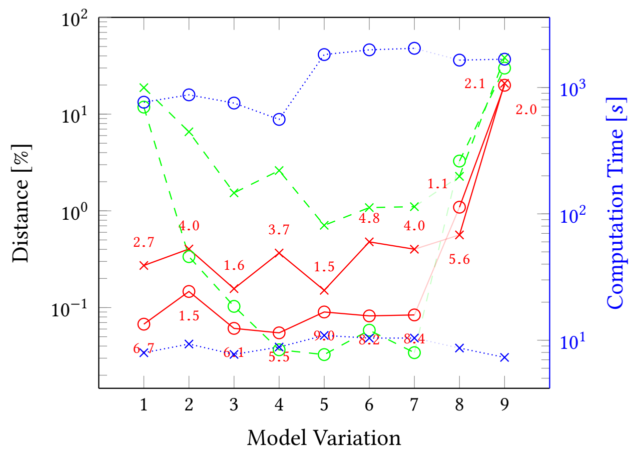

Figure 8: Results of CoreTAna for the variations of system architecture pattern 'State Machine'. Coloured markers and lines are used as in Fig. 6.

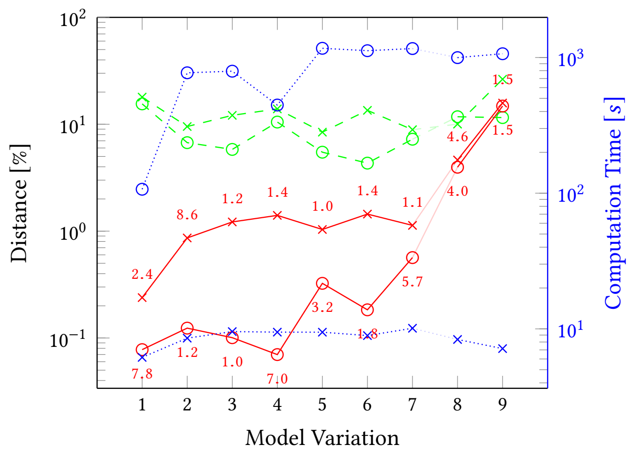

Figure 10: Results of CoreTAna for the variations of system architecture pattern 'Feedback Loop'. Coloured markers and lines are used as in Figs. 6 & 8.

shorter trace recordings become greater. This is due to the characteristics of the probabilistic model used in this system architecture pattern. Recording the system's runtime behaviour for a longer period and employing it to CoreTAna yields steadier results, because shorter trace recordings have a smaller chance to cover all possible behaviour of the probabilistic model.

4.4 Feedback Loop

The task set of the previous system architecture pattern is expanded further, so that messages are exchanged in a loop instead of just in one way. In addition to the feedback loop as depicted in Fig. 9, other system architecture patterns are added to be executed concurrently, in order to increase complexity. Tasks T_5 and T_6 represent a client-server without reply, and task T_7 is a periodically activated task without any communication.

The variations for this system pattern are equal to those applied to the previous patterns described in Secs. 4.2 & 7. However, the changed characteristics of this task set in comparison to the previous ones required us again to slightly change the sequence in which the variations are applied.

Fig. 10 visualises the resulting quality of the performed reverse engineering. This time also the results of the longer trace recordings show significant spread, which can once again be explained by the underlying probabilistic model. Because the complexity of the system increased in comparison to the previous system architecture, the chance to cover all possible behaviour decreased to such an extent that even large trace recordings do not cover everything.

4.5 State Machine Feedback Loop

Finally, the previous system architecture pattern is expanded by combining the ideas behind patterns *State Machine* and *Feedback Loop*. This means that messages are exchanged in a loop, and each sender/receiver is also a state machine. In addition to the state machine feedback loop as depicted in Fig. 11, other system architecture patterns are added to be executed concurrently: tasks T_3 and T_4

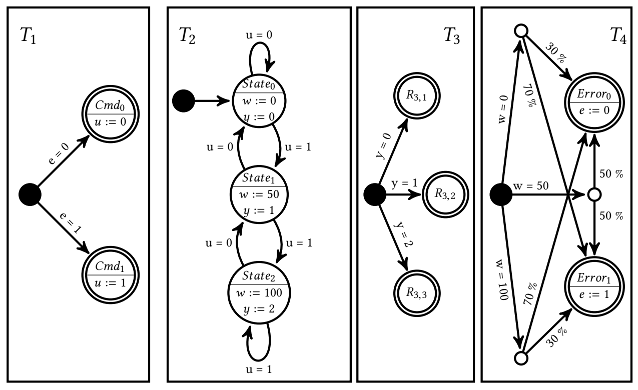

Figure 9: State diagram implemented by the system architecture pattern 'Feedback Loop'.

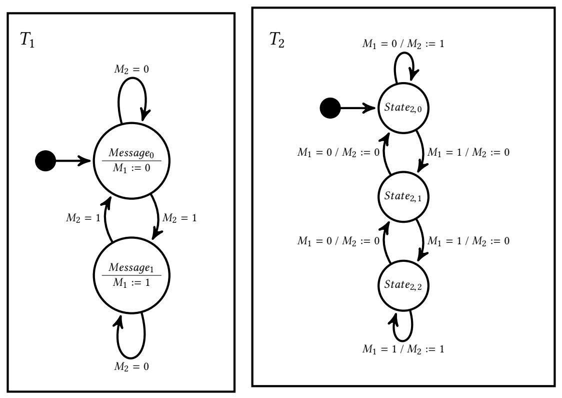

Figure 11: State diagram implemented by the system architecture pattern 'State Machine Feedback Loop'.

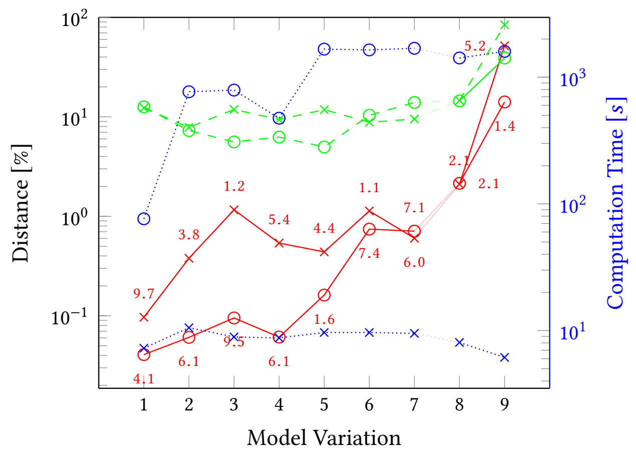

Figure 12: Results of CoreTAna for the variation of system architecture pattern 'State Machine Feedback Loop'. Coloured marks and lines are used as in Figs. 6, 8 & 10.

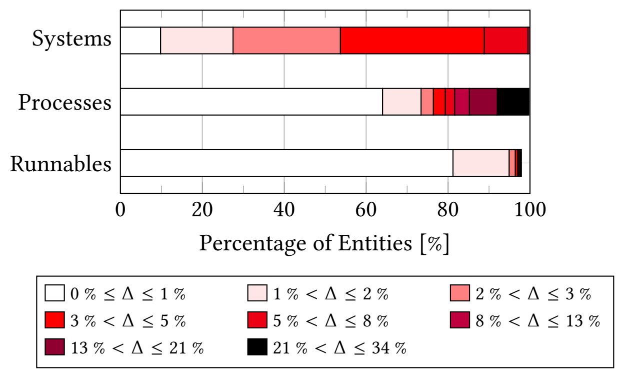

Figure 13: Stacked bar chart of CoreTAna's results for reversely engineering the randomly generated models, based on a trace recording that covers 1 second of a system's execution. The coloured bars mark the percentage of systems, processes, and runnables in which their reversely engineered behaviour differs from the original one by a certain distance value Δ.

represent a client-server without reply, and task T_5 is a periodically activated task without any communication.

The variations and the sequence in which these are applied to this system pattern are identical to those used for pattern 'Feedback Loop' in Sec. 4.4, which allows a comparison of results.

The results for this pattern, which are visualised in Fig. 12, show once again a significant spread. Noticeable is, furthermore, the fact that, for both trace lengths, the gap between the results is smaller than before. Moreover, omitting the data accesses from the logging (Variation 8) has only a small impact on the results. All this indicates a high amount of probability in this pattern.

5 BENCHMARK EVALUATION

So far, only isolated system architectures have been applied for evaluating CoreTAna. Because actual industrial systems are more diverse, we validate the scope and practical significance of the proposed benchmark by assessing CoreTAna's capabilities in processing trace recordings of randomly generated systems and traces from actual automotive projects.

5.1 Randomly Generated Systems

The randomly generated systems are not totally random, but their generation follows configurable settings. This allows us to influence the possible design space such that a desired mixture of system architectures can be ensured. The configuration used for this evaluation is designed in such a way that a total of 1000 different systems are generated. The number of tasks in each system is chosen randomly between 9 and 20. All tasks are activated harmonically by periodic alarms, with an offset between 0 and 50 ms and a period between 1 and 1000 ms. Their execution times are determined based on both their load, which can be between 1 and 20 %, and the number of called runnables, which each cover between 30,000 and 100,000 instructions. This results in roughly 5 to 100 runnables per task, which are bundled in 6 to 13 subsequent call sequences (a list of calls that are executed by a process; definition see 'https://www.eclipse.org/app4mc/help/app4mc-0.8.1/index.html#section3.12.10.3'). Conditional statements are added to each task in such a way that the execution of any number of these call sequences depends on random values within the domain of one of the 1 to 20 generated data signals. This is repeated again with the budgeted call sequences, so as to enable nested control flows. Data dependency is then established by adding, with a probability of 50 %, a signal access to each runnable, which writes a random value within the domain of that data signal.

A purely periodic task, which is roughly defined as a task without any conditional statements, is consequently generated with an approximate probability of at least $\frac{1}{9}$ or $11.\bar{1}$ %. This emerges from the fact that roughly 5 to 100 runnables per task are bundled into 6 to 13 coherent call sequences, which yields 1 to 9 call sequences per task. Therefore, the probability that a task representing a server in a 'Client-server without Reply' architecture is generated, is equal to the probability that one conditional statement is present, which is at least $\frac{1}{9}$, and assumes that there is no write access to that data signal within the task. Because 11 data signals are generated on average per system, this results in a probability of at best $\frac{1}{99}$ or $1.\overline{01}$ %. In contrast to this, a task representing a state machine has to modify the data signal on which it depends. Because a runnable has write access to each signal in 50 % of the cases, the resulting probability is roughly 5 %.

Fig. 13 visualises how capable CoreTAna is to reverse engineer our randomly generated models. The stacked bar chart depicts that 96 % of the runnables differ by less than 3 % (sum of yellow, blue and red bars). However, there are also scattered outliers, which indicate differences in the runnable behaviour as high as 57 %. Due to the fact that multiple runnables are mapped to a single task, the task's difference is added up by the difference of each runnable called, which consequently yields worse results. Nevertheless, the fact that all resulting differences for the overall systems are lower than 10 % (shown in the stacked bar chart by the absence of accordingly coloured bars) demonstrates that CoreTAna performs well in capturing the systems' dynamic behaviour.

5.2 Industrial Case Studies

To also establish a connection to real-life problems, we describe our experiences with using CoreTAna in actual projects within the automotive domain.

5.2.1 Engine Management System (EMS). In this industrial project, an engine management system (EMS) consisting of 75 tasks is analysed. To do this, a trace covering 6.5 seconds of the system's execution was provided in a 40 MB file with $90 \cdot 10^3$ events. Because of the high complexity of the system under investigation and technical limitations, it is only possible to observe and record task state transitions. Thus, no detailed insight into the system's behaviour is available.
The visualisation in Fig. 14 of the differences between each task shows that the quality of the reverse engineering is inconsistent. Some tasks contain only a small amount of variability, which is why their behaviour is reflected quite well with a difference of only 4 %. Other tasks, however, show a difference of 35 %, which indicates a lack of detail in the synthesised model. Nevertheless, the overall difference of 19.88 % is in line with the results of the synthetic models in Sec. 4, where evaluations of CoreTAna's capability to handle the limited accuracy in logged data yielded differences between 13 % and 52 %.

5.2.2 Steering System. For this industrial case study a model of a steering system software is generated. The system under investigation consists of 18 tasks and interrupt service routines. Altogether, they call 130 different runnables. The used hardware platform is a dual core processor with a frequency of 120 MHz for each processing unit.
The starting point for our reverse engineering is a trace recording that covers 30 seconds and contains all task and runnable calls performed during that time. This resulted in roughly $27 \cdot 10^6$ events and a file of 1.7 GB, i.e., the system produces roughly one million events per second. This example shows that it is currently not technically possible to observe the system behaviour in detail, e.g., all data accesses.
We evaluate CoreTAna's performance and the achieved quality of the reverse engineering by us using our proposed measure; the results are visualised in Fig. 14. In contrast to the previous industrial case study, where only task events have been recorded, the details added by also observing function calls lead to less spread. The difference for each task lies just between 14.93 % and 26.23 %. However, the overall difference of 15.54 % is slightly lower than in the previous case study, which is probably due to the trace containing more information. Nevertheless, the trace is missing knowledge about data accesses, so that the varying internal behaviour due to data dependencies still cannot be determined in full detail.

5.2.3 FMTV Challenge 2016. This industrial case study is inspired by the *Formal Methods for Timing Verification* (FMTV) Challenge 2016 [6], where a model of an industrial real-time system has been published in order to discuss solutions to concrete timing verification problems. Although no trace recordings from the actual system are provided, we have used this model and the TA Simulator [20] to generate a simulation trace. Because this model-based timing simulation is a commercial tool that is employed by many

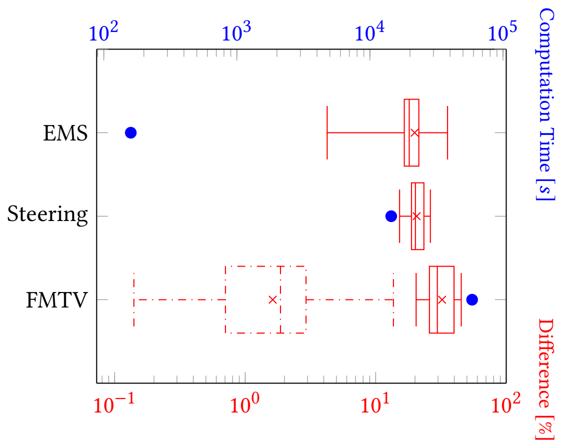

Figure 14: Box plots with whiskers from minimum to maximum, summarising the differences between each task/ISR in the individual industrial case studies. The crosses mark the result of the difference measure for the overall system. The dashed box plot visualises the results of the FMTV Challenge with the given hardware model.

Tier-1s and OEMs in the automotive industry and because the provided model is very detailed, it is reasonable to assume that the generated trace recording corresponds very closely to the actual system behaviour.
The model describes a full-blown engine management software that consists of 10 periodic tasks and 11 interrupt service routines (ISRs) that interact with the system sporadically. The functionality is provided by 1250 runnables, and roughly 10000 different data signals are accessed for communication. On the hardware side, a micro-controller architecture with four symmetric cores is available for processing. Each core has access to its local RAM and to the shared global RAM via a crossbar. Based on this model, the system's execution has been simulated for 30 seconds. During that time, roughly $341 \cdot 10^6$ events occurred and were recorded in a trace, which resulted in a file size of roughly 17 GB.
Fig. 14 visualises the differences between each task and ISR. Although the trace contains all details of the system's behaviour, the results are still around 20 %. Motivated by this rather disappointing outcome, a closer examination showed that the hardware limitations of the crossbar together with cumulative data accesses caused tasks to wait. Because the correct modelling of the hardware properties are not subject of the reverse engineering, the reconstruction was repeated with the hardware model as given input, so as to evaluate CoreTAna's performance regarding how closely the software model reflects the actual behaviour. The results are shown in Fig. 14 as a dashed box plot. With the differences mainly being around 1 %, these results correspond more to the expected outcome by being in line with the synthetic benchmark.

6 IMPACT ANALYSIS

Although our measure was developed to assess how closely a synthesised model reflects the timing behaviour of the underlying system, it is beneficial in multiple applications. For example, it assists with hardware tracing, in particular, with the evaluation of

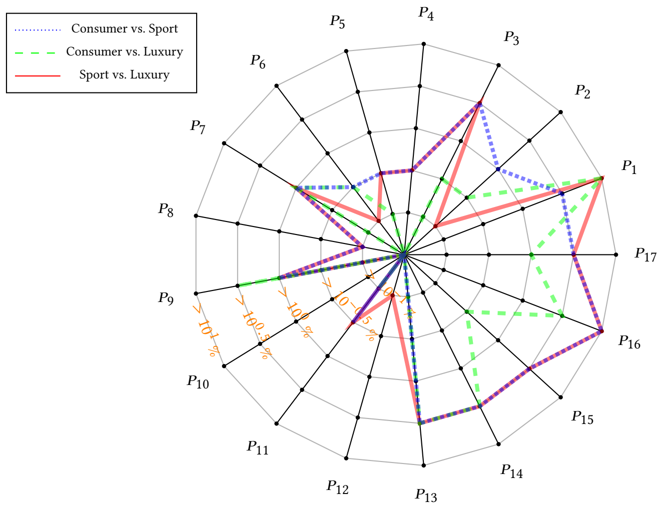

Figure 15: Radar chart showing the differences of each process (tasks and ISRs) in case of comparing the products 'Consumer', 'Sport' and 'Luxury' with each other. Each process of the common software product platform is represented by a spoke on which the change is plotted based on our measure, and each colour indicates the products of the product family that are compared.

the error-prone process of configuring the on-chip debug units by highlighting inconsistencies in the recorded system behaviour.

Another use case for which we have successfully applied our measure is determining effects on timing behaviour due to system changes. For example, the software of the steering system described in Sec. 5.2.2 represents not a single product but a whole product family. This means that it is possible to create software for the three distinct products 'Consumer', 'Sport' and 'Luxury', which all have the same architecture but different functionalities. To analyse the impact of the varying functionalities to the timing behaviour of the system's processes (tasks and ISRs), we have compared, in pairs, trace recordings of the products using our measure. The results of this comparison are depicted in Fig. 15 and show that not all processes are affected to the same extent. For example, process P_{10} has exactly the same timing behaviour in all three products, whereas the timing behaviour of processes P_1, P_3 and $P_{13} - P_{17}$ differ widely, with differences partially over 10 %.

7 CONCLUSIONS AND FUTURE WORK

Making a justified statement about the quality of reversely engineered models of real-time system components is often difficult. With our novel measure it is now possible to determine differences between the temporal behaviour of the actual system and that of its synthesised model by comparing corresponding trace recordings. In particular, CoreTAna's evaluation results via the proposed benchmark and the industrial projects confirm that our measure is capable of assessing reverse engineering tools. Our experience with using the measure for impact analysis also shows that it can be applied to other use cases in a very helpful way.

It is now possible to gradually extend CoreTAna with the help of our measure, e.g., by considering hardware-specific characteristics such as the duration of data accesses, and to continuously obtain feedback on how well the actual software behaviour is reflected in a reverse engineered model.

Currently, we are also gaining more practical experience by applying both CoreTAna and our measure within an industrial environment.

REFERENCES

[1] Johan Andersson. 2005. *Modeling the Temporal Behavior of Complex Embedded Systems: A Reverse Engineering Approach.* Licentiate Thesis. Mälardalen University, Sweden.

[2] Saoussen Anssi, Sara Tucci-Piergiovanni, Stefan Kuntz, Sebastien Gerard, and Francois Terrier. 2011. Enabling Scheduling Analysis for AUTOSAR Systems. In *Intl. Symp. on Object/Component/Service-oriented Real-time Distributed Computing.* IEEE, 152–159.

[3] AUTOSAR. 2016. *Specification of Operating System.* AUTOSAR Standard SWS OS Release 4.3.0. AUTOSAR.

[4] Jerry Banks (Ed.). 1998. *Handbook of Simulation: Principles, Methodology, Advances, Applications, and Practice* (1 ed.). Wiley, Chapter 10: Verification, Validation and Testing.

[5] Giorgio Buttazzo. 2011. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications* (3 ed.). Springer.

[6] Arne Hamann, Dirk Ziegenbein, Simon Kramer, and Martin Lukasiewycz. 2016. 7th Int. Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems – Verification Challenge. (2016). Retrieved March 24, 2017 from https://waters2016.inria.fr/challenge/ .

[7] Joel Huselius. 2007. *Reverse Engineering of Legacy Real-Time Systems: An Automated Approach Based on Execution-Time Recording.* Ph.D. Dissertation. Mälardalen University, Sweden.

[8] ISO. 2005. *OSEK/VDX Operating System.* ISO Standard 17356-3:2005. International Organization for Standardization.

[9] Johan Kraft. 2010. *Enabling Timing Analysis of Complex Embedded Software Systems.* Ph.D. Dissertation. School of Innovation, Design and Engineering (IDT), Mälardalen University, Sweden.

[10] Johan Kraft, Anders Wall, and Holger Kienle. 2010. Trace Recording for Embedded Systems: Lessons Learned from Five Industrial Projects. In *Intl. Conf. on Runtime Verification (LNCS)*, Vol. 6418. Springer, 315–329.

[11] Daehyun Kum, Gwang-Min Park, Seonghun Lee, and Wooyoung Jung. 2008. AUTOSAR Migration from Existing Automotive Software. In *Intl. Conf. on Control, Automation and Systems.* IEEE, 558–562.

[12] Averill M. Law. 2009. How to Build Valid and Credible Simulation Models. In *Winter Simulation Conf.* IEEE, 24–33.

[13] Yue Lu, Thomas Nolte, Iain Bate, Johan Kraft, and Christer Norström. 2011. Assessment of Trace-Differences in Timing Analysis for Complex Real-Time Embedded Systems. In *Intl. Symp. on Industrial and Embedded Systems.* IEEE, 284–293.

[14] Farhang Nemati, Johan Kraft, and Christer Norström. 2008. Validation of Temporal Simulation Models of Complex Real-Time Systems. In *Intl. Computer Software and Applications Conf.* IEEE, 1335–1340.

[15] Andreas Sailer. 2014. Towards an Automated Reverse Engineering of Design Models from Trace Recordings. In *Jahrestagung der Gesellschaft für Informatik.* GI, 2233–2245.

[16] Andreas Sailer, Michael Deubzer, Gerald Lüttgen, and Jürgen Mottok. 2016. CoreTAna: A Trace Analyzer for Reverse Engineering Real-Time Software. In *Intl. Conf. on Software Analysis, Evolution, and Reengineering*, Vol. 1. IEEE, 657–660.

[17] Andreas Sailer, Stefan Schmidhuber, Michael Deubzer, Martin Alfranseder, Matthias Mucha, and Jürgen Mottok. 2013. Optimizing the Task Allocation Step for Multi-Core Processors within AUTOSAR. In *Intl. Conf. on Applied Electronics.* IEEE.

[18] Robert G. Sargent. 2007. Verification and Validation of Simulation Models. In *Winter Simulation Conf.* IEEE, 124–137.

[19] Timing-Architects Embedded Systems GmbH. 2016. BTF-Specification (Version 2.1.5). (January 2016). Retrieved July 13, 2017 from https://www.eclipse.org/app4mc/docu/standards/TA_BTF_Specification_2.1.5.pdf .

[20] Timing-Architects Embedded Systems GmbH. 2016. TA Toolsuite Version 16.03.0. TA Academic & Research License Program. (2016). Retrieved March 24, 2017 from http://www.timing-architects.com .