CoreTAna: A Trace Analyser for Reverse Engineering Real-Time Software

Andreas Sailer*, Michael Deubzer**, Gerald Lüttgen† and Jürgen Mottok*
*Laboratory of Safe and Secure Systems, Ostbayerische Technische Hochschule Regensburg, 93053 Regensburg, Germany
Email: {andreas.sailer, juergen.mottok}@oth-regensburg.de
**Timing-Architects Embedded Systems GmbH, 93055 Regensburg, Germany
Email: michael.deubzer@timing-architects.com
†Software Technologies Research Group, Otto-Friedrich-University Bamberg, 96045 Bamberg, Germany
Email: gerald.luettgen@swt-bamberg.de

***Abstract*—With the availability of the AUTOSAR standard, model-driven methodologies are becoming established in the automotive domain. However, the process of creating models of existing 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 present CoreTAna, a novel tool that derives an AUTOSAR compliant model of a real-time system from a dynamic analysis of its trace recordings. This paper gives an overview of CoreTAna's current features and discusses its benefits for reverse engineering.**

I. Introduction

Since the release of the Automotive Open System Architecture (AUTOSAR) (http://www.autosar.org) V3.0 in 2007, the standard has experienced gradual acceptance in the automotive domain. One of its major goals is to define an open software architecture for the development of real-time systems, as well as a corresponding development methodology. Nevertheless, the process of deriving a model is often difficult and time consuming, especially when legacy code is involved [7]. The consideration of timing behaviour is an extra challenge that requires substantial effort [2], but is essential to the modelling process. Indeed, due to the current shift towards multi-core architectures in the automotive industry, OEMs are forced to gain detailed knowledge about their legacy software, like stimulation patterns of processes or the internal behaviour of functions. Besides this, most tools that tackle the challenges implied by this shift, such as finding an ideal task-to-core allocation or task priority assignment, work model-based.

This paper presents CoreTAna[1], a novel prototypic tool that reversely engineers an AUTOSAR compliant model, including the exact timing behaviour of a real-time, single- or multi-core system based on the dynamic analysis of a system's trace recordings. Such a model can either be derived automatically from scratch or by enriching an existing model. The fact that CoreTAna creates a standard compliant artefact, which enables further processing, e.g., system optimisation, sets our tool apart from existing solutions.

Technically, CoreTAna processes events that can be traced during the runtime of a system in a step-by-step manner. Every event indicates a change in the internal state of that system due to, e.g., function calls or data accesses. These pieces of information are taken from trace recordings and used by CoreTAna to deduce an abstraction of the system's structure and timing behaviour.

In combination with the TA Simulator (http://www.timing-architects.com), a commercial tool for model-based timing simulation used by many Tier-1's and OEMs in the automotive industry, CoreTAna generates an analysis report of how closely the synthesised model reflects the actual system.

All parts of CoreTAna, including the reverse engineering, are written in Java as an Eclipse plug-in. The implementation uses libraries and technologies like EMF (http://www.eclipse.org/modeling/emf/) for handling the model, Apache Commons Math (http://commons.apache.org) for the statistics and Collections for data structures, and the open-source constraint solver Choco (http://choco-solver.org) for solution finding.

Related Work

Although there are many tools available that analyse the timing behaviour of a software system, e.g., TraceAlyzer [5], these provide only a visual representation of the timing information. Consequently, they lack the possibility to directly synthesise a model of the actual system from an event trace, which is essential for further processing including model-based optimisation.

Kienle et al. [4] provide a good overview of software reverse engineering in the domain of embedded systems and present various techniques for obtaining specific artefacts, such as state machines, architecture models, and simulation models.

Closest to our work is that of Huselius [3], whose goal is to automatically create a probabilistic state machine model from observations of an implemented real-time system. However, the model allows only "a very high-level view of the system" [3, p. 57]. Consequently, concepts like functions, operating

This work was partially funded by the ITEA2 project AMALTHEA4public (ITEA2-13017) via the German Ministry of Education and Research (BMBF) under funding code 01IS14029L.

[1]A screeencast of CoreTAna is available at https://youtu.be/fbWEq9Odf9o and a version with limited functionality is included in the open-source tool chain AMALTHEA (http://amalthea-project.org).

system events, and semaphores are not supported. Since these concepts are essential for realising the use cases of interest to industry (see Sec. III-A), the developed algorithms turn out to be too limited to synthesise a model that can be used for system optimisation.

Similar approaches to ours can be found in other domains, such as that of distributed systems. For example, the Kieker framework [14] allows one to monitor and analyse the runtime behaviour of a web service. Although it supports dynamic architecture discovery and performance measuring, a combination of both at the same time is not possible.

Another related research area is that of process mining, where business processes are automatically discovered from event logs recorded by information systems. There is a large array of tools available for process mining including the de-facto standard, open-source framework ProM [13]. However, their biggest drawback is the missing notion of time.

II. Prerequisites

The following presents CoreTAna's prerequisites: a specification of the required input, the trace recordings, and a definition of the resulting model.

A. Trace Recordings

There are many sources from which information about a system can be acquired [1]. However, working on the basis of source code is infeasible in our case, because software functionality might be provided by vendors that do not hand over source code. Furthermore, static analyses lack the ability to determine dynamic information like execution times. Since the timing aspects of a system are of particular importance for the intended use cases presented in Sec. III-A, CoreTAna conducts a dynamic analysis based on trace recordings.

"Trace recording implies detection and storage of relevant events during runtime, for later off-line analysis" [6, p. 1]. It is a commonly used technique in the development of real-time systems, where the correctness of a system's behaviour depends not only on functional correctness but also on temporal accuracy. As a consequence, traditional debuggers cannot be used to observe the runtime behaviour of software since they require halting a system's execution, which would result in a significant alteration of timing behaviour [12]. Instead, the occurring events, which represent changes in the state of a system, are recorded during runtime. Possible state changes include, e.g., the start or termination of a task execution, the entry or return of a function, and the read or write access to a variable. CoreTAna uses the open-source trace format BTF and a complete list of events; a detailed specification can be found in [11].

B. Model Definition

Changes in the internal state of a system are, in general, caused by the application code under investigation interacting with the operating system. Let us consider the basis of real-time software, to which AUTOSAR refers as processes. Both tasks and interrupt service routines (ISRs) are manifestations of processes and represent the smallest units managed by a system's scheduler. In source code, processes are realised by dedicated programming language constructs. During compile time, such constructs become a sequence of instructions in a system's memory. Each of these sequences is bounded by a starting address, by which it can be called for execution, and a return instruction. Hence, each start and termination of a process is equivalent to the instant in which the starting address or, resp., the return instruction of this process is invoked. These changes can be detected during runtime, and corresponding events can be stored in a trace.

Since there is a close connection between system behaviour and recorded trace events, we may also deduce information the other way round, i.e., the inference of task priorities based on observed task pre-emptions. CoreTAna uses exactly the knowledge of this close relationship to describe the temporal behaviour of a real-time system based on events recorded in the traces.

The temporal behaviour of a system is influenced by multiple factors. For example, the execution time of a process depends not only on the number of instructions that have to be executed within a process, but also on the scheduling algorithm. If a process is pre-empted by another process, then the time between its start and termination becomes longer, since the instructions of the pre-empting process are executed in between. Based on those pieces of information and together with the recurrence of all processes, a high-level description of the system behaviour can be synthesised. We call this high-level description the *process level*, where processes are considered as black boxes. A low-level description is possible on the *function level*, where processes are considered by their side effects, such as data accesses, rather than as black boxes. Because functions are like processes, they have addresses in the system's memory, so their start and terminate events can also be recorded. Data accesses during that time can thus be associated with a specific function and their exact timings.

The control-flow within processes and functions may be sequential or branching. This is because different behaviours of the same process or function can be observed. In the model used by our tool, the control-flow is described by a so-called *call graph*, where the dynamic behaviour can be expressed either in a probabilistic or deterministic manner: the former expresses the likelihood that specific behaviour is performed, while the latter makes the behaviour dependent on specific variable values.

III. CoreTAna

This section elaborates on CoreTAna's use cases, which have been derived from industrial projects, and its internal workings.

A. Use Cases

The goal of CoreTAna is to automatically synthesise a model that contains detailed information about the timing behaviour of the software under investigation, in addition to a system description. Depending on the case of application, this

can be done from scratch or by enriching an existing model, e.g., created by static analysis from source code. Starting with the former, our tool can be used for the following purposes at process and function level.

- *Process Level*: CoreTAna takes only the process events from trace recordings into consideration. This means that each process is considered as a black box, and thus the details of its internal behaviour are unspecified. Elements generated in the model are, e.g., the processes and their priorities, activations, and execution times. The model can then be used to simulate the system's timing behaviour [8], or to optimise the task priorities or the process-to-core allocation [10].
- *Function Level*: Here, CoreTAna takes all pieces of information into consideration so that a detailed description of the whole system can be extracted. This includes, e.g., the call graphs, the called functions and their data and semaphore accesses, and their execution times. In doing so, the results of a simulation or optimisation conducted on a model typically become more accurate. Furthermore, additional characteristics of the system's performance, e.g., communication overheads and data dependencies, can be examined and considered for further system optimisation, like the splitting of tasks or the sequencing of functions within processes. For the latter, the generated model is used to rearrange function calls within processes to achieve better performance and without affecting the program logic.

CoreTAna can also enrich an existing model by analysing individual dynamic aspects of the system and make it more precise. Corresponding use cases, that have arisen from industrial projects, are the following.

- *System Overhead*: Based on the process events read from a trace, it is possible to model the scheduling overhead, i.e., the time that the system takes to switch contexts from one process to another. At function level, the overhead can be determined even more precisely by taking the time between the return of a function and the entry of the succeeding function into account, which is not consumed by the application itself but by the underlying system. In this way, a detailed performance specification of the system can be extracted automatically.
- *Stimulation Patterns*: Embedded systems react to external events from the environment. As a consequence, information on the appearing external events and their frequencies are needed to make a precise statement about the system's actual timing behaviour. For this reason, stimulations of processes are analysed, and temporal patterns, such as jitters or bursts, are derived.
- *Execution Times*: An exact picture of the execution times for both processes and functions can be derived with the help of dynamic analysis. These times may be modelled in a dynamic manner by fitting them to statistical distributions, e.g., uniform or normal distributions.
- *Internal Behaviour*: CoreTAna can synthesise a function's internal behaviour including the points in time when actions happen, like data or semaphore accesses, and the assigned values during those instants.
- *Call Graph*: The control flows of processes and functions are extracted by considering the observed behaviour of the individual instances, and creating a call graph structure with probabilistic or data-driven conditional branches.

B. Overview of Approach

CoreTAna defines an approach to handle the aforementioned use cases, which always starts with a system's captured hardware traces. Because vendors use their own trace format, the traces are first transformed into the BTF trace format [11], which provides a common interface for CoreTAna. Because reverse engineering individual pieces for the model requires only a specific subset of trace events, each trace is stored in a database, which makes selective processing faster.
Next is the reverse engineering: inputs are trace recordings from the system and already available pieces of information of the model, e.g., the program structure derived from static analysis. As a result, a model filled with this information is obtained. This model is exercised by a timing simulator, e.g., the TA Simulator (http://www.timing-architects.com), so that trace recordings that reflect the dynamic behaviour of the reverse engineered model can be generated.
Finally, to present to the user an indication of the quality of the resulting model, the original traces and the generated simulation traces are compared. Based on a multitude of different real-time metrics, such as response times and execution times, a visual comparison of the different inputs and outputs is offered to the user. Fig. 1 shows a screen shot of CoreTAna when plugged into the commercial TA Tool Suite.
An extension of the above approach can also be used to validate the algorithms of our reverse engineering [9].

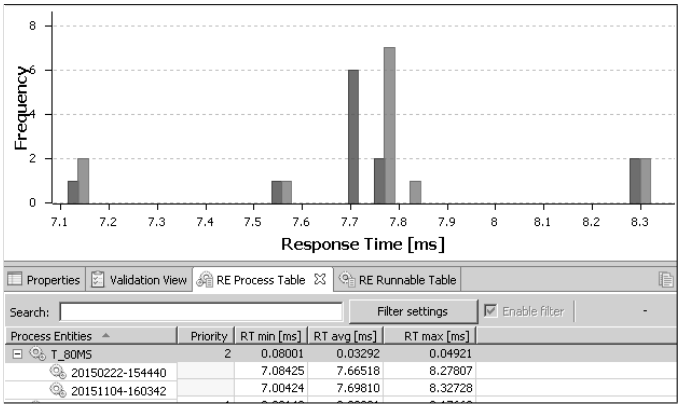

Fig. 1: CoreTAna's user interface shows, in its lower part, the metrics for processes and functions from the used sources, which are the original trace recordings and the simulation traces of the reverse engineered model. In the upper part, the values for the determined metrics are compared using a histogram.

C. Reverse Engineering

At the beginning of our reverse engineering approach and for each trace recording, all events are processed in chronological order. This allows us to reason about event sequences, such as context switches from one process to another. In order to achieve a fast analysis, only the events that are relevant

to the reverse engineering of a specific part of the model are considered. For example, only the activation events are important for the stimulation-pattern use case.

In the next step, the pieces of information collected from the trace events are processed. For example, each previously detected pre-emption event results in an integer linear programming (ILP) constraint that expresses the relation between the involved processes. An ILP solver is then employed to find a solution of the resulting constraint system, so as to make a statement about the priorities of the processes and of the scheduling algorithm employed by the system.

Finally, all obtained pieces of information are used to build the model. In cases where there are multiple ways to model the observed behaviour, such as the representation of periodic activations by a sequence of single activations, the alternatives are assessed in the order of their universality, starting with the most specific one. A detailed description of the algorithms employed by our reverse engineering approach can be found in [9] and [10].

IV. Exemplary Industrial Studies

The CoreTAna prototype is currently being evaluated within different industrial projects in the automotive domain. Two applications are briefly described here.

The scope of the first application is a model-based optimisation of AUTOSAR-compliant application software. The system under investigation is a simple braking system consisting of 28 tasks and almost 200 functions. At first, the system's structural information, including the tasks, their calling functions, and their data and semaphore accesses, was taken from AUTOSAR description templates and corresponding source code. CoreTAna was then used to refine this structural information with information about the system's dynamic behaviour, including the execution times and call probabilities of functions.

Because all of the system's tasks are periodic, a single trace recording spanning at least one hyper-period, consisting of roughly 500,000 events, was analysed for the refinement of the static model. A comparison of real-time metrics, e.g., activate-to-activates and response times of tasks determined from both simulation traces of the reverse engineered model and hardware traces, confirmed that the resulting model reflects the actual system well. The industrial partner has successfully used an automated design-space exploration tool on the synthesised model to optimise the system's task-to-core allocations, so as to meet the timing constraints demanded of the system.

The scope of the second application is the analysis of a software system that has been ported to a multi-core architecture. The system under investigation is a mature engine management system that consists of 75 processes. A trace covering 6.5 seconds of the system's runtime behaviour and containing roughly one million process events was recorded for our reverse engineering. CoreTAna was then used to automatically generate (from scratch) a model of the system's timing behaviour on the *process level*, which includes the system's processes and their priorities, activations, and execution times. In this application, the activations of ISRs were of special interest, since these depend on the dynamics of the system's environment, e.g., the engine speed. CoreTAna, infers recurring activation patterns for the processes based on the activations observed during runtime. Because of this, the industrial partner has been able to replay the recorded behaviour using a timing simulation and also analyse the presumed system behaviour.

V. Conclusions and Future Work

While there are many tools available that can acquire detailed knowledge about real-time software, CoreTAna stands out in that it reversely engineers an accurate, timed, and AUTOSAR-compliant model from given hardware traces. This is of benefit not only for comprehending legacy software, but in particular in the context of the challenges caused by the current shift towards multi-core architectures in the automotive industry. An evaluation of our prototypic tool is ongoing, but feedback from our industrial partners has so far been positive. Future research will focus on the detection of hardware-specific characteristics, such as the duration of data accesses, so as to obtain a more precise representation of the actual software behaviour.

References

[1] J. Andersson *et al.*, "Extracting Simulation Models from Complex Embedded Real-Time Systems," in *Intl. Conf. on Software Engineering Advances*. IEEE, 2006.

[2] S. Anssi *et al.*, "Enabling Scheduling Analysis for AUTOSAR Systems," in *14th IEEE Intl. Symp. on Object/Component/Service-oriented Real-time Distributed Computing*. IEEE, 2011, pp. 152–159.

[3] J. Huselius, "Reverse Engineering of Legacy Real-Time Systems: An Automated Approach Based on Execution-Time Recording," Ph.D. dissertation, IDT, Mälardalen Univ., 2007.

[4] H. M. Kienle *et al.*, "Software Reverse Engineering in the Domain of Complex Embedded Systems," in *Reverse Engineering - Recent Advances and Applications*. InTech, 2012.

[5] J. Kraft, "Enabling Timing Analysis of Complex Embedded Software Systems," Ph.D. dissertation, IDT, Mälardalen Univ., 2010.

[6] J. Kraft *et al.*, "Trace Recording for Embedded Systems: Lessons Learned from Five Industrial Projects," in *1st Intl. Conf. on Runtime Verification*, no. 3. Springer, 2010, pp. 315–329.

[7] D. Kum *et al.*, "AUTOSAR Migration from Existing Automotive Software," in *Intl. Conf. on Control, Automation and Systems*. IEEE, 2008, pp. 558–562.

[8] A. Sailer. (2014, August) Timing Simulation of Multi-Core Systems based on AUTOSAR Models. [Online]. Available: http://www.timing-architects.com/fileadmin/user_upload/Log-In_Miscellaneous/Whitepaper_Timing_Simulation_AUTOSAR.pdf

[9] A. Sailer, "Towards an Automated Reverse Engineering of Design Models from Trace Recordings," in *44. Jahrestagung der Gesellschaft für Informatik*. GI, 2014, pp. 2233–2245.

[10] A. Sailer *et al.*, "Optimizing the Task Allocation Step for Multi-Core Processors within AUTOSAR," in *Intl. Conf. on Applied Electronics*. IEEE, 2013, pp. 247–252.

[11] Timing-Architects Embedded Systems GmbH. (2014, April) BTF-Specification (Version 2.1.3). [Online]. Available: https://wiki.eclipse.org/images/e/e6/TA_BTF_Specification_2.1.3_Eclipse_Auto_IWG.pdf

[12] J. Trümper *et al.*, "Maintenance of Embedded Systems: Supporting Program Comprehension Using Dynamic Analysis," in *2nd Intl. Workshop on Software Engineering for Embedded Systems*. IEEE, 2012, pp. 58–64.

[13] W. M. P. van der Aalst, *Process Mining - Discovery, Conformance and Enhancement of Business Processes*. Springer, 2011.

[14] A. van Hoorn *et al.*, "Kieker: A Framework for Application Performance Monitoring and Dynamic Software Analysis," in *3rd ACM/SPEC Intl. Conf. on Performance Engineering*. ACM, 2012, pp. 247–248.