Enabling Scheduling Analysis for AUTOSAR Systems.

CONFERENCE PAPER · JANUARY 2011

DOI: 10.1109/ISORC.2011.28 · Source: DBLP

2011 14th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing

Enabling Scheduling Analysis for AUTOSAR Systems

Saoussen Anssi[1], Sara Tucci-Piergiovanni[2], Stefan Kuntz[1], Sébastien Gérard[2], François Terrier[2]
[1]*Continental Automotive France SAS, PowerTrain E IPP*
1 Avenue Paul Ourliac - BP 83649, 31036 France
{saoussen.ansi, stefan.kuntz}@continental-corporation.com
[2]*CEA LIST, Laboratory of model driven engineering for embedded systems,*
Point Courrier 94, Gif-sur-Yvette, F-91191 France
{sara.tucci, sebastien.gerard, francois.terrier}@cea.fr

***Abstract*- AUTOSAR (Automotive Open System Architecture) is enjoying increasing interest and broad acceptance in the automotive domain. AUTOSAR aims at defining an open standardized software architecture to face future challenges in automotive development including the development of time-critical systems (e.g. brake-by-wire or steer-by-wire). Mastering the development of such systems requires being able to analyze their real-time behavior. Scheduling analysis is the theory that studies how far a real-time system may satisfy its real-time requirements against its real-time properties. In this paper, we will study to what extent it is possible to apply some of those scheduling analysis techniques on real-time systems deployed on AUTOSAR-compliant architectures. The paper focuses on scheduling analysis techniques implemented in one open source tool. A concrete case study shows the feasibility of the approach and shows scheduling analysis results.**

I. INTRODUCTION

Automotive software systems are characterized by increasing complexity, tougher safety requirements and ever-tighter timing constraints. To face these challenges, major OEMs and tier-1 suppliers founded the AUTOSAR development partnership [1]. The main goal of the initiative is to define a methodology that supports a distributed function driven development process and to create a standard for the software architecture of automobile Electronic Control Units (ECUs). This standard includes basic software, application software structure and components interfaces. Through this standardized architecture, AUTOSAR is expected to bring several benefits to industry concerning the increasing development complexity such as smooth integration of third party software, easier reuse of software/hardware components and seamless application of diverse development tools [2].

However, in addition to the complexity issue, automotive applications often have to fulfill stringent timing constraints to function properly. For example, power train and chassis applications include complex (multi-variable) control laws, with different sampling rates, for use in conveying real-time information to distributed devices. One hard real-time constraint is ignition timing, which varies with engine position. The latter is defined by a periodic event characterizing the flywheel zero position. End-to-end response times must also be bounded, because a too long control loop response time may not only degrade performance, but also cause vehicle instability. As these constraints have to be met in every possible situation, there is a strong need to perform timing analysis and verification on these applications.

AUTOSAR as a formal system architecture model can provide much more benefits for automotive real time development than usually promoted high-level benefits. However, for that statement to be true we claim that the AUTOSAR system model must allow easy and accurate timing verification for automotive software.

One timing verification approach to ensure that a system meets its timing requirements is scheduling analysis. This type of analysis allows designers to detect timing faults and hence avoid costly design mistakes. It may also be used to evaluate the impact of possible platform migrations or modifications on scheduling parameters. In this paper, our focus is made on the AUTOSAR system model; we study to what extent the AUTOSAR system model allows specifying necessary information to make the model analyzable from a scheduling point of view. To this end, we firstly propose a categorization of scheduling relevant information organized in a *scheduling analysis model.* The model includes all architectural aspects having an impact on system schedulability. Four dif-

1555-0885/11 $26.00 © 2011 IEEE
DOI 10.1109/ISORC.2011.28

ferent categories, are identified, namely: (i) application workload, i.e. the system load in terms of functions and their worst-case execution time, (ii) application timing behavior, i.e. end-to-end computations and end-to-end deadlines (iii) resource platform, i.e. processing and communication resources application functions are supposed to run in, and (iv) allocation, i.e. the mapping of functions on software resources (tasks) and of software resources on hardware resources (ECUs).

As a next step, the paper explains in details how to model for each scheduling analysis model's category the respective information using AUTOSAR constructs.

Let us note that constructing the scheduling analysis model is a necessary step for schedulability analysis. However, before applying schedulability analysis, an adequate schedulability test able to analyze the system under consideration must be identified. Many scheduling analysis tools exist offering a wide range of schedulability tests covering different classes of systems (e.g. single-processor or distributed, with fixed-priority or EDF schedulers, with preemptive or non preemptive tasks, etc.). Once the test identified, a schedulability analysis model transformation, towards the identified test's model, is the last necessary step for schedulability analysis application.

In this paper, we illustrate how to apply schedulability analysis on a concrete automotive case study. Starting from a cruise control system, we build the corresponding AUTOSAR scheduling analysis model and we identify an adequate schedulability test to analyze the obtained model with the MAST scheduling tool [3]..A transformation of the AUTOSAR model to a MAST input model is then presented along with scheduling analysis results.

As a global result, this study shows how to concretely perform scheduling analysis on AUTOSAR models using scheduling theory techniques.

The paper is then organized as follows: In section II related work is presented as a general overview about tests, tools and methodologies developed in the context of real time verification. Section III is dedicated to the characterization of a common analyzable system model. In section IV, we show to what extent the AUTOSAR system model complies with this characterization. In section V, the study is illustrated by performing scheduling analysis on a concrete case study. In this section, we highlight the challenges met to apply scheduling theory techniques on our use case.

II. RELATED WORK

Schedulability Tests and scheduling analysis Tools. The first exact schedulability test for the preemptive monoprocessor scheduling of a set of periodic tasks, each with its deadline equal to its period, was introduced by Lehoczky et al. [4]. The test determines whether a set of tasks is schedulable using the rate monotonic algorithm of Liu and Layland [5]. The response time of each task is calculated, and checked against its deadline. Later, other tests were developed relaxing a number of assumptions: Audsley et al. [6] developed a feasibility test for sets of tasks in which deadlines can be less than periods, and which are scheduled using the deadline monotonic algorithm [7]. Lehoczky [8] provided a feasibility test for periodic tasks with arbitrary deadlines. For distributed systems a number of tests have also been developed, e.g. [9], [10] and [11]. In [12] authors extended existing tests to special heterogeneous architectures: fixed priority-scheduled CPU connected via a TDMA-scheduled bus. However, the analysis proposed in [12] is not an exact analysis, because it makes the assumption that tasks are independent. To take into account dependencies between tasks, Tindell proposed in [10] a test for fixed priorities in which offsets among release times of dependent tasks can be taken into account. The test has been later extended to distributed systems by Palencia and González [11] greatly reducing the pessimism of the approach presented in [12]. This test is available in the MAST [3] tool and, as we will see in section V, this test will be selected for our case study.

The development of scheduling analysis tools lies at the very core of the schedulability analysis issue. While the number of such tools is constantly increasing, they also vary widely in terms of analysis capabilities and supported features. MAST [3] and Cheddar [13], are open source tools offering classical feasibility tests allowing schedulability analysis of fixed-priority and EDF-based monoprocessor and distributed systems. Cheddar gives also the possibility to specify new schedulers and task models that cannot be described by classical approaches. However, Cheddar focuses only on tasks and does not support a function-level characterization as MAST does. Rapid-RMA [14] and SymTA/S [15] are commercial tools performing scheduling analysis for monoprocessor and distributed systems. Rapid-RMA is based on rate monotonic and deadline monotonic algorithms assuming tasks to be independent, SymTA/S enhances classical schedulability tests to allow analysis for automotive systems. For example, this tool extends the Tindell's technique [10] to take into account

OSEK cooperative tasks (task that can be preempted only in predefined points). Unfortunately, this tool is not freely available.

Real-time methodologies and specifications. A number of methodologies for the development of real-time systems propose to integrate scheduling validation in the design phase.

In Saksena et al. [16], a methodology for schedulability validation of object-oriented models is proposed. The methodology starts from a design model where specification of active and passive objects, message semantics and object interaction is available. End-to-end computations are defined to identify jobs, and a mapping of jobs into tasks is proposed. The obtained task set is validated by an ad-hoc test. The same approach for schedulability validation is also followed for component based models [17].

A body of work is concentrated on architectural specifications enabling scheduling analysis. The focus is on modeling information compliant to above- mentioned scheduling methodologies, e.g. end-to-end computations, jobs to tasks mapping, and timing constraints, in a specific model. In this context, [19] and [18] focus on UML. [18] proposes an ad-hoc profile for scheduling analysis extending the OMG SPT profile [21]. This paper, however, was antecedent to MARTE, [22] issued to upgrade SPT to UML 2. In [19] a MARTE-based methodology enabling scheduling analysis is proposed. In the AUTOSAR context, the only other attempts to enable timing verification of AUTOSAR models are [23] and [24]. However, while [24] focuses only on the AUTOSAR OS specification, [23] considers a former version of AUTOSAR where timing extensions were not yet introduced. AUTOSAR models were then completed with Timing Augmented Description Language (TADL) [25] and analysed using the SymTA/S tool [15]. In this paper we work on the recent version of AUTOSAR which includes timing extensions.

III. SCHEDULING ANALYSIS MODEL

This section characterizes a necessary set of features that should be present in a system model to make scheduling validation possible. This characterization is the result of the study of several schedulability validation methodologies, schedulability tests and tools described in Section II. Let us remark that we limit the characterization to the features that are essential to make the model analyzable. Optional features required by specific analysis tests and tools will not be mentioned here, but we will show that the essential features here identified are sufficient in the context of the presented case study.

The features contained in an analyzable system model may be organized in four main categories:

A. Application workload

The application workload represents the processing load of the system. It represents the different operations (functions) executed in the system and contending for use of processing resources and other shared resources. An operation may represent a small segment of code execution as well as the sending of a message through a communication medium. Operations are generally organized in processing flows (set of related operations/functions). To make the analysis possible, scheduling analysis requires the specification of the execution /transmission time (worst, best or average) for operations/messages.

B. Application timing behavior

The application timing behavior represents the timing information of the different operations or processing flows involved in the system under analysis. Timing information contains both timing description (timing properties) and timing constraints. Timing description contains the specification of the triggering of system operations or processing flows (recurrence, activation jitters, etc.). Most studied scheduling analysis tools allow analyzing systems with various triggering patterns such as periodic, sporadic, singular, etc. For those activation patterns, it is necessary to specify the period or the mean inter-arrival time of the triggering events. Timing constraints must be met by the system operations or flows. They are represented essentially by operation deadlines, output jitter bounds and end-to-end deadlines.

C. Resource Platform

It represents the concrete architecture and capacity of hardware (e.g., CPU or buses) and software (e.g. tasks) resources. For hardware resources such as processors, the model should contain the description of the scheduler used. For a more accurate analysis, it may be also necessary to specify the processor overheads (e.g. context switch overhead). For software resources such as tasks, it is necessary to specify the task nature (preemptive, non-preemptive, etc.) as well as its priority. Involved shared resources should also be described.

D. Allocation

To be analyzable, the model specifies the allocation of the operations to software resources (e.g.

tasks) and the allocation of software resources to hardware resources (e.g. processors).

IV. SCHEDULING ANALYSIS AWARE MODELING IN AUTOSAR

In this section, we show how AUTOSAR system model meets the characterization of Section III.

A. Application workload

The system workload is described through two categories of elements: *runnable entities* and the *basic software entities*. Runnable entities are the smallest code-fragment that are provided by a software component and are subject to scheduling by the underlying operating system. Runnable entities are specified in the system model as a part of the *internal behavior* of software components.

Basic software entities are also subject to scheduling and contend for use of processing resources. Basic software entity represents the smallest code fragment that can be described for a *basic software module* or cluster.

In AUTOSAR, it is possible to specify the execution time for both runnable entities and basic software entities as *resource consumption* when describing respectively the corresponding *software component implementation* or *basic software implementation*. The resource consumption element provides information about memory and time consumption for each software component implementation or basic software implementation. Maximum, minimum or nominal execution times can be specified

B. Application timing behavior

AUTOSAR allows the modeling of the application timing behavior features through its *timing extensions*. Timing extensions allow specifying the timing description and the timing constraints of the system. They are used to describe the timing behavior at different levels: the virtual functional bus level (*VFB timing*), the software components level (*Swc timing*), the basic software module level (*Bsw module timing*), the system level (*system timing*) and at the ECU level (*ECU timing*). On each level, processing flows are described through the *event chain* concept. An event chain describes the causal order of a set of functionality dependant timing description events. Every event chain describes a causal relationship between two events. The first is called *stimulus* (e.g. event representing the activation of a runnable entity) and the second is *response* (e.g. event representing the termination of a runnable entity). Furthermore, event chains can be hierarchically decomposed into an arbitrary number of sub-chains called *event chain segments*.

The triggering of operations is supported through the *event* concept. Timing constraints can be attached to both event chains and events. For an event, timing constraints specify its arrival pattern as well as its activation jitter. Supported arrival patterns are: periodic, sporadic, burst, concrete and arbitrary. For event chains, it is possible to specify their latencies. A latency timing constraint restricts the time duration between the occurrence of the stimulus and the occurrence of the corresponding response of that chain.

C. Resource platform

AUTOSAR allows specifying the system hardware resources when describing the system topology at the system level. The *ECU instance* concept allows defining the ECUs used in the topology. Communication networks can be specified through the *communication cluster* concept that represents the main element to describe the topological connection of communicating ECUs. For each communication cluster, we can define one or more *physical channel*s that describe the transmission medium that is used to send and receive information between two communicating ECUs, as well as the protocol used for the communication.

AUTOSAR allows describing the software resources involved in the system when defining the OS configuration. Tasks are specified through the *Os task* concept that represents an OSEK task. Task priority can be specified using the attribute *Os task priority*. The attribute *Os task schedule* allows specifying if the task is preemtible or not. Interrupts are supported through the *Os ISR* concept that represents an OSEK interrupt service routine.

Shared resources may be specified using the *Os resource* concept, used to coordinate the concurrent access of tasks and ISRs to shared resources. The attribute *Os task resource ref* of the *Os task* element allows listing the shared resources accessed by the specific task.

D. Allocation

The allocation of tasks to hardware resources is performed in AUTOSAR during the ECU configuration process. The configuration of a particular ECU used in the system involves the configuration of the OS and of the runtime environment *RTE*. The OS configuration contains among others the definition of the different OS tasks involved. Hence, this indicates

that the defined tasks are allocated to the ECU which is subject to configuration.
The mapping of runnable entities and basic software entities to OS tasks is part of the RTE configuration. The mapping of runnable entities to OS tasks is based on the mapping of the *RTEEvents* that activate those runnable entities to OS tasks. In a similar way, basic software entities are mapped to OS tasks by mapping the *BswEvents* that activate them.

V. SCHEDULING ANALYSIS OF A CONCRETE AUTOSAR SYSTEM

In this section, we illustrate the conducted case study by presenting how AUTOSAR models are developed for an automotive application and how to apply scheduling analysis.

A. Use case presentation

The application considered is a cruise control system. This system consists of a switch sensor that acquires the driver inputs (set cruise, cancel cruise, increase speed, decrease speed, etc.) and a control system that processes the inputs and maintains the vehicle speed according to a given speed set point. The cruise control system is composed of eight elementary functions: input acquisition (responsible for acquiring the sensor data), input interpretation (responsible for interpreting the acquired sensor data to determine the driver's desire), diagnosis (to detect errors or inconsistencies in acquired data), limp home (decides which action to take in case of detected error), speed set point (responsible for calculating the speed setpoint desired by the driver), application condition and basic function (responsible for calculating cruise control states and transitions to decide whether to carry out specific cruise control activities) and the controller (PI controller that maintains the vehicle speed).
The cruise control functions are distributed over two ECUs: the Body Controller ECU and the Engine Management ECU communicating via CAN bus.
Table 1 summarizes the timing information of the cruise control functions, task allocation and task priorities.
The acquisition and failure management tasks are executed on the Body Controller ECU. The setpoint and control tasks are executed on the Engine Management ECU.
In addition to function deadlines contained in Table 1, the cruise control should satisfy the following end-to-end constraints:

1*) In normal operating mode (no failure), the duration from the acquisition of sensor inputs until the controller delivers the corresponding torque request should not exceed 500ms.
2*) The failure management (from the diagnosis start until the limp home orders a state change) should be performed within 100ms
Figure 1 shows the system architecture where arrows depict function dependencies for the normal operating mode and failure management end-to-end flows.

Table 1. Timing information of the cruise control functions

Functions	WCET (ms) **	Period (ms)	Deadline (ms)	Allocated to	Task priority
Input acquisition	2.5	10	20	Acquisition Task	1
Input interpretation	2.32	40	80		
Diagnosis	1.52	10	20	Failure Management Task	4 (highest priority)
Speed setpoint	3.5	40	80	Setpoint Task	2
Limp home	1.03	10	20	Control Task	3
Application condition	3.92	40	80		
Basic function	2.08	40	80		
Controller	1.4	40	80		

** The WCETs used in this example were measured using internal methods and tools that for confidentiality reasons cannot be presented here

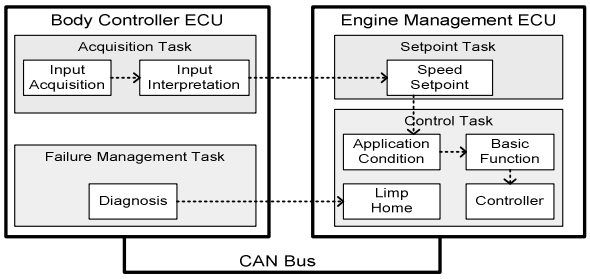

Figure 1. Cruise Control architecture

B. Cruise control AUTOSAR scheduling analysis model[1]:

a) Application workload

The cruise control system is modeled as a *composition* formed by two *application software components*. For each application software component, we describe the *internal behavior*. In this behavior, each elementary function of the cruise control is modeled

[1] The AUTOSAR model for the cruise control is developed using the CESSAR-CT tool that is based on the ARTOP framework [27].

as a *runnable entity*. The first application software component contains the input acquisition, the input interpretation and the diagnosis runnable entities. The second application software component is formed by the limp home, the speed setpoint, the application condition, the basic function and the controller runnable entities. For each runnable entity, we specify the activating *RTEEvent.*

To specify the execution time of each runnable entity, in each application software component implementation, we specify a *resource consumption* to describe the needed resource concerning the execution time of each runnable entity.

In parallel, to take into account the impact of the basic software on the processor utilization and the system scheduling, we model a *basic software module* for which we describe the *basic software entities* and their execution times.

b) Application timing behavior

To specify the application timing behavior for the cruise control, we define a Swc timing that allows describing the cruise control timing properties and constraints. For each runnable entity, we create an event chain. Its stimulus is the timing description event "runnable entity activated" of the runnable entity and its response is the timing description event "runnable entity terminated". To specify the runnable entity deadline, we define a latency timing constraint on the event chain defined. To describe the triggering of the runnable entity, we define an event triggering constraint on the stimulus event. In this timing constraint, we describe the arrival pattern of this event.

To model the end-to-end constraints 1*) and 2*), we model two event chains: the first (control event chain) is formed by the following event chains: input acquisition, input interpretation, speed setpoint, application condition, basic function and controller. For this event chain, we define a latency timing constraint of 500ms. In a similar way, we define an event chain (failure management event chain) formed by the event chains diagnosis and limp home for which we define a latency timing constraint of 100ms.

c) Resource platform & allocation

As stated before, the cruise control is a distributed system. To model the CAN bus involved in the system, we defined a *CANPhysicalChannel* that allows referencing the ECUs connected to the CAN bus. To model ECUs, we define for each ECU an *ECU instance* element. For each ECU instance, we define the software resources as well as the mapping of the runnable entities or basic software entities to tasks.

The description of the tasks allocated in each ECU is performed in two steps. The first step is the definition of the OS configuration. In this configuration definition, the OS is modeled by an *ECU configuration module definition* element. For this module, we define an *ECU parameter configuration container* called *OsTask.*

Once this definition is done, the second step is the modeling of the concrete configuration of the OS. For this, we define an *ECU module configuration value*. In this module configuration value, we define the corresponding tasks as *ECU container values* having *OsTask* as a definition.

Mapping the runnable or basic software entities to OS tasks is done in two steps following the RTE configuration for each ECU. In the first step, which is the definition of the RTE configuration, we create an *ECU module definition*. To this module definition, we associate a container definition called *RteSwComponentInstance* (or *RteBswModuleInstance* if we consider the basic software entities mapping) in which we create another container called *RteEventToTaskMapping* that allows referencing the mapped *RTEEvent* and the OS task. The second step is the specification of the concrete mapping value of the cruise control runnable entities (or basic software entities). This is done by creating container values for which we specify the elements created in the first step as definitions.

The mapping of the basic software entities to OS tasks is done in a similar way.

C. *Performing scheduling analysis for the cruise control system:*

We performed scheduling analysis using the MAST tool. MAST offers a suite of scheduling analysis tests, ranging from classic RMA for fixed priority monoprocessor systems to more sophisticated analyses for EDF schedulers and distributed systems [26]. To perform scheduling analysis for the cruise control system, we firstly need to select an adequate schedulability test. As shown in section A, the system presented here is a distributed system using fixed priority schedulers (the two Controllers are managed by an OSEK operating system). Moreover, constraint 1* and 2* show that tasks are dependant. For such system, the Tindell test [10], later improved in [11] and available in MAST as "offset_based" test is chosen.

The AUTOSAR model is transformed in a MAST model according to the mapping detailed below.

Each runnable/basic software entity is represented by an *operation* in MAST (an operation may represent a

small segment of code execution as well as the sending of a message).

The OS tasks along their priorities are modeled as *scheduling servers*. A MAST scheduling server represents a schedulable entity in a processing resource.

An elementary event chain corresponds to an *activity,* which represents the execution of an operation. The MAST activity allows also specifying the allocation of the operation (runnable/basic software entity) to the scheduling server (OS task) and the input/output events, i.e. the input event triggering the activity and the output event generated when the activity execution completes.

Timing requirements can be attached to output events; in particular, each runnable deadline specified in table 1 is transformed in a *hard local deadline*. For instance, for the input interpretation operation, a hard local deadline of 80ms is specified. Note that such local deadlines are relative to the arrival of the event that activates the activity.

Event chains of the cruise control (control and failure management event chains) are represented by *transactions* in MAST. A transaction is a succession of interrelated activities that are executed in the system. The transaction is characterized by an external event and a sequence of activities. The output event of each activity in the transaction is the input event of the subsequent activity. For the first activity, the input event is the external event of the transaction.

The end-to-end constraints 1* and 2* are captured by *hard global deadlines* that represent deadlines relative to the external event that activates the transaction. Note that in the control event chain, the first runnable (input acquisition) is activated by a periodic event of 10ms while the other runnables of the event chain are activated every 40ms. We define then an external event in the MAST model for the transaction control with period equal to 10ms. However, we have to capture also the fact that following runnables are not executed in each occurrence of the transaction. To model this situation in MAST, we use a *rate divisor event handler.* Rate divisor is a kind of activity that only generates one output when a number of input events equal to the rate factor have arrived. In our case, a rate divisor with rate factor equal to 4 is then placed between the input acquisition and input interpretation related activities.

The distributed nature of the system must also be captured. According to [11], the offset-based test takes into account a distributed system with a CAN Bus if (1) the CAN Bus is modeled as a processor and (2) each message sent over the CAN bus is modeled as a task assigned to this processor with a priority equal to the sending task priority.

In our case, this is ensured by creating in the transactions, representing respectively the control and the failure management event chains, a special activity representing the sending of a message generated at the end of the input interpretation activity (send/receive interpretation) and at the end of the diagnosis activity (send/receive diagnosis). As table 2 shows, these two new activities are allocated to tasks (respectively acquisition message task and diagnosis message task) executing on the CAN bus and having the same priority as the sending task (respectively acquisition task and failure management task). Table 2 summarizes the characteristics of the MAST model obtained. MAST results show that the cruise control system is schedulable. Table 3 shows the slack of each processor/transaction. A processor/transaction slack is the percentage by which we can increase execution times of operations executing

Table 2. MAST model for the cruise control

Transaction	External Event period (ms)	Activity	Rate divisor	Local deadline (ms)	Global deadline (ms)	Scheduling server	Priority	Processor
Control	10	Input acquisition	1	20		Acquisition task	1	Body controller
		Input interpretation	4	80		Acquisition task		
		Send/receive interpretation				Acquisition message task	1	CAN bus
		Speed setpoint		80		Setpoint task	2	Engine management
		Application condition		80		Control task	3	
		Basic function		80				
		Controller		80	500			
Failure management	10	Diagnosis	1	20		Failure management task	4	Body controller
		Send/receive diagnosis				Diagnosis message task	4	CAN bus
		Limp home		20	100	Control task	3	Engine management

in the processor/transaction without jeopardizing system schedulability. Slacks results show that the system configuration could be improved to use processors in a more efficient way as a big amount of the body controller capacity is unused

Table 3. Cruise control processors and transactions slacks

Processors	**Slack (%)**
Engine Management	71.20
Body Controller	205.25
CAN bus	629.14
Transactions	**Slack (%)**
Failure management	260.94
Control	89.84

Table 4 and 5 give the worst response times obtained for the cruise control activities and transactions output events. As the tables show, all the worst response times are lower than the specified deadlines, i.e. the system is schedulable.

Table 4. Cruise control activities worst response times

Activity	**Worst response time (ms)**	**Deadline (ms)**
Input acquisition	4.84	20
Input interpretation	34.84	80
Speed setpoint	49.82	80
Controller	59.29	80
Diagnosis	1.52	20
Limp home	10.95	20

Table 5. Cruise control transactions response times

Transactions	**Worst end-to-end response time (ms)**	**End-to-end deadline (ms)**
Failure management	10.95	100
Control	59.29	500

VI. CONCLUSION

In this paper, we have presented the basic features of an analyzable model as required by scheduling analysis. We have evaluated to what extent the AUTOSAR system model matches such features.

This study shows that it is possible to perform scheduling analysis implemented in common open source tools for AUTOSAR systems.

REFERENCES

[1] AUTOSAR Partnership (www.autosar.org)

[2] H. Fennel, S. Bunzel, und H.H.E. al, "Achievements and exploitation of the AUTOSAR development partnership," *Proceedings of Convergence 2006*, 2006.

[3] MAST website (Mast.unican.es)

[4] Lehoczky, J., L. Sha and Y. Ding, "The Rate Monotonic Scheduling Algorithm: Exact Characterisation and Average Case Behaviour," Proceedings of the Real-Time Systems Symposium (1989).

[5] Liu, C. L., J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," Journal of

[6] Audsley, N., Burns, A., Richardson, M., Tindell, K., and Wellings, A. Applying new scheduling theory to static priority preemptive scheduling. Software Engineering Journal 8, 5 (1993), 285–292.

[7] Leung, J., and Whitehead, J. On the complexity of fixed-priority scheduling of periodic, real-time tasks. Performance Evaluation 2 (1982), 237–250.

[8] Lehoczky, J. "Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines," Proceedings 11th IEEE Real-Time Systems Symposium (5-7 December 1990) pp.201-209.

[9] K. Tindell and J. Clark. Holistic schedulability analysis for distributed real-time systems. Microprocessing and Microprogramming - Euromicro Journal (Special Issue on Parallel Embedded Real-Time Systems), 40:117–134, 1994

[10] K. Tindell, "Adding Time-Offsets to Schedulability Analysis", Technical Report YCS 221, Dept. of Computer Science, University of York, England, January 1994.

[11] J.C. Palencia Gutiérrez and M. González Harbour, "Schedulability Analysis for Tasks with Static and Dynamic Offsets". Proceedings of the 18th. IEEE Real-Time Systems Symposium, Madrid, Spain, December 1998.

[12] K. Tindell and J. Clark. Holistic schedulability analysis for distributed real-time systems. Microprocessing and Microprogramming - Euromicro Journal (Special Issue on Parallel Embedded Real-Time Systems), 40:117–134, 1994

[13] Cheddar website (Beru.univ-brest.fr)

[14] Rapid-RMA website (tripac.com/html/prod-fact-rrm.html)

[15] SymTA/S website (www.symtavision.com/symtas.html)

[16] M. Saksena and P. Karvelas Designing for Schedulability Integrating Schedulability Analysis with Object-Oriented Design , Proceedings of the 12th Euromicro Conference on Real-time Systems (EUROMICRO2000), June 2000.

[17] Z.Gu and Z.He, Real-time scheduling techniques for implementation synthesis from Component-based Software Model, Proceedings of the 8th International Symposium on Component Based Software Engineering (CBSE2005), Springer, 2005

[18] C. Bartolini, et al, A UML Profile and a Methodology for Real-Time Systems Design, 32nd EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO'06), 2006

[19] C. Mraidha, S. Tucci-Piergiovanni, S. Gérard. Optimum: A MARTE-based Methodology for Schedulability Analysis at Early Design Stages. Third IEEE International workshop UML and Formal Methods. November 2010, Shangai, China.

[20] H. Espinoza, J. Medina, H. Dubois, S. Gérard, and F. Terrier.Towards a UML-Based Modeling Standard for Schedulability Analysis of Real-Time Systems. MARTES Workshop at MODELS Conference. 2006.

[21] OMG, UML Profile for Schedulability, Performance, and Time, Version 1.1. 2005.

[22] OMG, UML Profile for Modeling and Analysis of Real-Time and Embedded systems. 2009.

[23] Klobedanz et al., Timing modelling and analysis for AUTOSAR-based software development – a case study. In Design, Automation & Test (DATE 2010), March 2010.

[24] P. E. Hladik, A. M. Deplanche, S. Faucou, Y. Trinquet. "Adequacy between AUTOSAR OS specification and real time scheduling theory", *Industrial Embedded Systems 2007*, 2007

[25] TIMMO (TIMing MOdel). ITEA2 IST. website https://www.timmo.org/

[26] J. M. Drake, M. G. Harbour, J. J. Gutiérrez, P. L. Martinez, J. L. Medina, J. C. Palencia. "Modeling and analysis suite for real time applications (MAST 1.3.7), Description of the MAST model," Report, Universidad De Cantabria, SPAIN, 2008.

[27] ARTOP User Group website (www.artop.org)