Adequacy between AUTOSAR OS specification and real-time scheduling theory

Pierre-Emmanuel Hladik
LINA (FRE CNRS 2729)
École des Mines de Nantes
Nantes, France
Email: pierre-emmanuel.Hladik@emn.fr

Anne-Marie Déplanche, Sébastien Faucou and Yvon Trinquet
IRCCyN (UMR CNRS 6597)
Université de Nantes
Nantes, France
Email: firstname.name@irccyn.ec-nantes.fr

Abstract—AUTOSAR (AUTOmotive Open System ARchitecture) consortium is a development partnership between the main actors of the automotive manufacturing industry. It aims at defining an open standardized software architecture, in order to face the future challenges in automotive development. One of the important challenge concerns the development of time-critical systems, e.g. brake-by-wire or steer-by-wire. In order to master the development of such systems, one must be able to understand and analyze their real-time behavior. Responses to this problem can be found in the real-time scheduling theory, especially schedulability analysis techniques. In this paper, we propose a review of a subset of the AUTOSAR Operating System specification from a schedulability analysis point-of-view.

I. INTRODUCTION

AUTOSAR (AUTOmotive Open System ARchitecture) consortium is a development partnership between the main actors of the automotive manufacturing industry. It aims at defining an open standardized software architecture, in order to face the future challenges in automotive development. One of the important challenge is the design and implementation of safety-critical embedded systems. Among these systems, one can cite especially X-by-wire systems, the purpose of which is to command essential functions of a vehicle, for instance braking or steering. The main expected benefit of this technological evolution is increased safety of the driver and the passengers, by providing new driver assistance functionalities.

In order to be able to design such safety-critical systems, it is mandatory to master the complexity of the embedded system all along its development process. This has to be done by using jointly rigorous design principles and analysis tools that ease the process of detecting design flaws. This paper is a contribution to this problematic. It focuses on the real-time behavior of AUTOSAR systems. Within this context, it tries to give some responses to the following questions:

• Can we find in the real-time scheduling theory some results that could be used for the analysis of real-time applications built upon the scheduling service provided by an AUTOSAR OS implementation?
• If the response to the first question is affirmative, how can we adapt these results to the practical context of AUTOSAR OS? How can we restrict the usage of the scheduling services, so as to comply with the hypotheses assumed by the theory?

Notice that our goal is not to develop "from scratch" new schedulability analysis techniques, but rather to adapt existing theoretical results, so as to provide AUTOSAR OS user with some usable analysis techniques and corresponding design guidelines. We believe that it is necessary to develop such works in the AUTOSAR context. Hence, on the one hand, the analysis techniques can be used to build "intelligent" CASE tools for the system configuration activity (see [1] for an overview of the AUTOSAR design process), for instance by coupling schedulability analysis with dynamic constraint programming [2]. On the other hand, the design guidelines can be followed by the providers of RTE generator\(^1\) to ensure the predictability of the generated application.

In a previous paper [4], we have proposed a schedulability analysis technique for OSEK/VDX OS applications, together with the corresponding design guidelines. AUTOSAR OS being based on OSEK/VDX OS, we extend now our results to AUTOSAR OS applications. More specifically, the main contribution of this paper is to analyze the AUTOSAR OS specification from a schedulability analysis point of view, and to expose how our previous results can be adapted to this new context. The proposed extensions are based on state-of-the-art results established within the real-time scheduling community. Our analysis leads us to underline some points of the specification that are not clear. The second contribution of this paper is thus to underline these points and to suggest, for some of them, how they shall be made clear in order to comply with state-of-the-art real-time scheduling theory.

The paper is organized as follows: first, the scheduling services provided by AUTOSAR OS are presented; then, we recall the main principles of schedulability analysis techniques; next, we describe briefly our previous results concerning schedulability analysis of OSEK/VDX OS applications, before to expose how they can be extended so as to take into account AUTOSAR OS specificities; last, we give some conclusions and outline some future research directions.

\(^1\)The RTE, or Run-Time Environment [3], is the communication infrastructure of the AUTOSAR software architecture. The RTE generator is the tool responsible for generating the glue code between application code and the underlying operating system.
II. Overview of the AUTOSAR OS Specification

In the AUTOSAR software architecture [1], the operating system is the component responsible for scheduling all the active objects (tasks and ISRs) hosted on an ECU. We give below a very brief overview of the scheduling-related services that it provides. The material used to perform the work described in this paper is the specification of the AUTOSAR Operating System version 2.0.1 [5] and the specification of the OSEK/VDX operating system version 2.2.2 [6] on which AUTOSAR OS is based.

Let us recall that all the objects of an AUTOSAR OS application are static: they are configured off-line (during the system configuration activity), are created before the start-up of the application and are never destroyed (no dynamic allocation). Moreover, the present version of AUTOSAR OS specification targets only uniprocessor systems.

A. Task Management

AUTOSAR OS uses OSEK/VDX OS task management. In OSEK/VDX, two different task types are provided. On one side, a basic task is a sequential code without system call being able to block the task. The only synchronization points are at the beginning (the task is activated by the occurrence of a condition) and the end (the termination of a basic task can trigger the activation of another task by calling the service). It is possible to authorize the record of activation requests occurring while a basic task is already active. Every request is then recorded (no multiple instance) in the ready queue (by priority and arrival order).

On the other side, an extended task is composed of one or several blocks separated by invocations of synchronization services, which may result in a waiting state. Synchronization is based on the private event mechanism: only the owner task can explicitly wait for the occurrence of one or more of its events (logical OR). The setting of occurrences can be made by tasks (basic or extended) or ISRs (Interrupt Service Routine). For an extended task, it is impossible to authorize the record of activation requests occurring while it is already active.

B. ISR Management

AUTOSAR OS uses OSEK/VDX OS ISR management. Two different categories of ISR are provided: category 1, which are not allowed to call OS services, and category 2 (noted below ISR2), which are allowed to call some OS services. ISR are triggered by the occurrence of an external event resulting in an interrupt request on the host microcontroller. Each ISR has a static priority. The priority levels used for ISR are strictly greater than the priority levels used for tasks. According to the underlying micro-controller hardware, an ISR execution can be preempted by the activation of another ISR.

As OSEK/VDX OS allows resource sharing between task and ISR2, some implementations can treat ISR2 as tasks, i.e. by allowing context switching from an ISR2 to a task and back. This is for instance the case of Trampoline [7] (http://trampoline.rts-software.org), an academic open-source software project, which aims at implementing an OSEK/VDX OS compliant kernel.

As we will see later, schedulability analysis techniques do not make any difference between tasks and ISR.

C. Scheduling Policy

AUTOSAR OS uses and extends OSEK/VDX OS scheduling policy. Static priorities are assigned off-line to tasks and the Highest Priority First policy is used, with FIFO as a second criterion when many tasks share the same priority level. The specified scheduling policy addresses only uniprocessor systems. For an application, the scheduling can be: full non preemptive, full preemptive or mixed preemptive. In this last case, every task has its appropriate mode, preemptive or non preemptive (specified off-line during system configuration). There exists also a notion of group, for tasks that share a common internal resource. An internal resource is automatically locked by a task of the group when it gets the CPU and released when it terminates, waits for an event or invokes the Schedule service. Usual preemption rules are used for the tasks which are not in the group, according to their priority level. Inside the group the tasks cannot preempt among themselves.

AUTOSAR OS specific extensions concern timing protection service. The purpose of this service is to prevent the occurrence of timing faults. To do so, this service calls a user-provided function each time one of the following conditions occurs:

- the execution time (time spent in the running state) of a task or ISR2 is greater than the (offline specified) expected value;
- the execution time of a task or ISR2 while holding a shared resource is greater than the (offline specified) expected value;
- the arrival rate of a task or ISR2 is greater than its (offline specified) expected limit. For a task, this limit is the time that an instance of the task can spend in the running state in a given timeframe. As we will see later, this mechanism is closed to the aperiodic server concept of real-time scheduling theory. For an ISR2, this limit is the number of activations in a given timeframe. As we will see later, under certain conditions, this mechanism is closed to the sporadic task concept of real-time scheduling theory.

D. Shared Resources Management

AUTOSAR OS uses OSEK/VDX OS shared resources management policy. Hence, it coordinates the concurrent access to shared resources with the OSEK-PCP protocol (Priority Ceiling Protocol). OSEK-PCP protocol is more simple than original PCP [8], [9]. It is also known as "Immediate Priority Ceiling Protocol" (IPCP). When a task gets a resource, its priority is immediately raised to the resource priority, so that other tasks that share the same resource cannot get the CPU.
E. Alarm, counters and schedule tables

AUTOSAR OS uses OSEK/VDX OS alarm and counter related services with some minor modifications, and extends these services through the schedule table concept.

Alarms and counters allow the processing of recurring phenomena, for instance timer ticks, or signals from mechanical organs of a car engine (camshaft, crankshaft). When associated with a timer, they allow the management of periodic tasks. A counter is an object intended for the counting of “ticks” from a source. Of course, each counter has a maximal value: when this value is reached, the counter is reset to 0.

An alarm allows to link a counter and a task. It expires when the counter reaches a predefined value. Then, a statically defined action is taken: either the activation of the associated task, or the setting of an event of the task. An alarm can be defined to be single-shot or cyclic, the corresponding counter value being absolute or relative to the current counter value.

Schedule tables are an extension of the alarm concept. Like an alarm, a schedule table is linked to a counter. It is composed of a set of expiry points, whose corresponding counter value is relative to the activation of the schedule table. When an expiry point is reached, one or more actions (task activation or event setting) are taken. A schedule table can be defined to be single-shot or cyclic, the corresponding counter value being absolute or relative to the current counter value.

AUTOSAR being dedicated to the design of distributed embedded electronic architectures, it introduces the notion of global (network) time, and specifies mechanism to achieve resynchronization between local and global time for driving schedule tables.

F. Scope of the study

As this is not the main subject of this paper, we do not expose with details all the implementation constraints that must be fulfilled in order to conform to the schedulability analysis model. Nevertheless, in this paper, we assume the following hypothesis: applications are composed of basic tasks and ISR2 only. Communications between active entities are asynchronous. Active entities are activated either by a periodic alarm (tasks only), or by an external event (tasks or ISR2). Hence, the only synchronization means are resource sharing through the IPCP protocol. Moreover, it shall be possible to compute the WCET of each task of the application. We are aware that some of these hypothesis are restrictive for application designer, that is why we plan to relax some of them in future works.

III. BASIC PRINCIPLES OF SCHEDULABILITY ANALYSIS TECHNIQUES

The purpose of schedulability analysis is to ensure that timing requirements, especially real-time task deadlines, are met at design time. To do so, one has to model the system within a dedicated formalism and then to apply some mathematical analysis algorithm on the model. The system is usually modeled as a set of tasks, each task being characterized by different attributes, e.g. WCET, activation law, deadline, shared resources, etc. The analysis algorithm generally expresses a sufficient condition ensuring that all tasks of the system complete before their deadline. Notice that the analysis algorithms are dedicated to a specific formalism and its underlying semantics. Any change in the formalism, or its underlying semantics, implies to change the algorithm. The semantics captures (i) the behavior of the underlying execution platform and (ii) the behavior of the tasks executing on this platform. Concerning the first item, we must clearly comply with AUTOSAR OS specifications. Hence, we assume a uniprocessor platform; the priority of the tasks are set off-line and cannot change during system life; the scheduling policy is “Highest Priority First”, with FIFO as a second criterion for task sharing the same priority; tasks are not reentrant. For the second item, we have already given some constraints above (sectionII-F). We will give some supplementary constraints below, as we go along our statement.

In the field of fixed priority uniprocessor system schedulability analysis, two main categories of algorithms have been proposed: Response Time Analysis (RTA) [10] and Rate Monotonic Analysis (RMA) [11]. The principle of RTA is to compute the worst case response time (WCRT) of each task. Then, the system is schedulable if, for each task, its WCRT is lower or equal to its deadline. The principle of RMA is to search, for each task, any instant earlier than its deadline such that the time interval between the task activation and this instant is large enough to accommodate the computational requirement of the task itself and all the higher priority tasks. If such an instant exists for all tasks then the system is schedulable.

Both RMA and RTA need hypothesis on the activation laws of the tasks of the system. Hence, the earlier algorithms have been developed for periodic systems (i.e. all the tasks of the system are periodic). Of course, some extensions have been proposed, so as to take account non periodic tasks. In this paper, we will consider in a first time periodic systems, before to explain how the services offered by AUTOSAR OS can be used to exploit some extensions developed to handle non periodic tasks.

The algorithms developed for both approaches are known to have pseudo-polynomial complexity, and it is currently unknown whether the task set schedulability can be decided in polynomial time [12]. Despite this complexity, analytical techniques have very efficient implementations in practice that make them suitable for schedulability analysis of fixed priority policy [13]. In [2] a major result is to show that the pseudo-polynomial schedulability algorithms are sufficiently efficient in practice to be used repeatedly during an optimization process.

In this paper, we only consider the RTA approach. We classically define the response time of a task as the amount of time that elapses between the instant where the task is released for execution (activated) and the instant where it completes execution. The computation of the worst-case execution time of a given task is based on the well-known busy period analysis [14], [15]. A $\pi$-busy-period, where $\pi$ is a priority level, is
a time interval during which the processor is continuously processing tasks with priority $\pi$ or higher. By definition, an instance of a task $\tau_i$ with a priority level $\pi_i$ is necessarily executed during a $\pi_i$-busy-period. Furthermore, more than one instance of a task $\tau_i$ can be executed during the same $\pi_i$-busy-period. Hence, for any $\pi_i$-busy-period, we define $a_i(q)$, the date of the $q$th activation of $\tau_i$, $s_i(q)$, the date of its $q$th start time, and $f_i(q)$, the date of its $q$th finish time in the $\pi_i$-busy-period. Under fixed priority scheduling, $f_i(q)$ is commonly described as:

$$f_i(q) \overset{\text{def}}{=} s_i(q) + I_i(q) + C_i$$

where $I_i(q)$ is the interference, i.e. the sum of the execution times of higher priority tasks that may execute between the activation and the completion of the $q$th instance of $\tau_i$; $C_i$ is the worst-case execution time of $\tau_i$.

For any $\pi_i$-busy-period, we also define $r_i(q)$, the response time of the $q$th instance of $\tau_i$ in a $\pi_i$-busy-period. By definition of the response time, we have:

$$r_i(q) \overset{\text{def}}{=} f_i(q) - a_i(q)$$

Hence, the worst-case response time of $\tau_i$ is defined by:

$$R_i \overset{\text{def}}{=} \max_{\forall \pi_i \text{-busy-period}, \forall q \geq 1} \{r_i(q)\}$$

Computing the response time of each instance of $\tau_i$ in each $\pi_i$-busy-period in the system life can be computationally intractable. To overcome this problem, the idea is to identify the so-called critical instant for $\tau_i$. This instant marks the beginning of a $\pi_i$-busy-period where $\tau_i$ meets its worst-case response time. Let us call this busy-period $W$. To compute the worst-case response time of $\tau_i$, we just need to compute each $r_i(q)$ in $W$:

$$R_i \overset{\text{def}}{=} \max_{\pi_i \text{-busy-period}=W, \forall q \geq 1} \{r_i(q)\}$$

Depending on the formalism, it is not always possible to define the critical instant for a task. In this case, the solution consists in computing an upper bound on $R_i$. To do so, a "fake" critical instant has to be defined. This "fake" critical instant is followed by a busy period $W$ such that the following inequality must hold:

$$R_i < \max_{\pi_i \text{-busy-period}=W, \forall q \geq 1} \{r_i(q)\}$$

Of course, this introduces pessimism in the results and leads to an under-utilization of the hardware resources. Hence, it is interesting to explore the specificities of an RTOS, and to impose (realistic) constraints on the utilization of its services. This allows to design applications that can be accurately modeled within a formalism for which there exist an algorithm the pessimism of which is "reasonable". This is the type of work that we develop in the next section, within the context of AUTOSAR OS.

IV. Schedulability Analysis of AUTOSAR OS Applications

A. A basic model

We start here by defining a schedulability analysis algorithm for basic AUTOSAR applications (without taking into account schedule tables and timing protection mechanism), inspired from our previous work on OSEK/VDX [4]. AUTOSAR OS being based on OSEK/VDX OS, the analysis can be used "out-of-the-box". We will explain in the next section how the model and the algorithm can be adapted to the specificities of AUTOSAR OS.

1) Informal presentation of the formalism: We consider a formalism that allows to model a subset of AUTOSAR OS applications. Here is an informal presentation:

- all tasks are periodic (activated through an alarm) and "non-concrete", i.e. the date of their first activation is unknown;
- the scheduler is mixed preemptive;
- tasks can share resources with the IPCP protocol;
- tasks can share a same priority level;
- there can be groups of tasks in the system;
- there are no precedence constraints between tasks;

We want to point out that the considered scheduling behavior is not strictly a fixed priority one, in the sense that the priority of a task may vary during its execution. Hence:

- while it is holding a shared resource, the task priority is raised to the ceiling priority of the resource;
- once elected for execution, a non-preemptible task has an "infinite" priority;
- once it has been elected for execution one time, a task member of a group has its priority raised to the ceiling priority of the group internal resource.

Such a behavior can be captured with the notion of pre-emption threshold introduced by Wang and Saksena in [16]. The preemption threshold is the priority level that the task has during its execution, i.e. the priority considered by the scheduler for the task after its first start event, and before its terminate event. In our formalism, we refine this notion: each task is composed of a sequence of blocks and each block has a preemption threshold attribute.

2) Formal presentation of the formalism: Let $\Gamma = \{\tau_i\}$, $1 \leq i \leq n$ be the set of the $n$ tasks of the application under analysis. Each task $\tau_i$ is a tuple $\langle p_i, d_i, \pi_i, e_i \rangle$ where: $p_i$ is its period; $d_i$ is its relative deadline (we allow either $d_i \leq p_i$, or $d_i > p_i$); $\pi_i$ its user-declared priority level; and $e_i = \{\langle e_{ij}, \gamma_{ij} \rangle \}$ is a set of execution times with their corresponding preemption threshold. A couple $\langle e_{ij}, \gamma_{ij} \rangle$ describes a code block of $\tau_i$, where $e_{ij}$ is the worst-case execution time of the block, and $\gamma_{ij}$ is the preemption threshold associated to the block (see Fig. 1). Of course, we always have $\gamma_{ij} \geq \pi_i$.

Here, for each task, $e_{i0}$ is its worst-case execution time and $\gamma_{i0}$ is deduced from its characteristics: for a non-preemptive task, $\gamma_{i0}$ is considered as infinite (or the highest priority level); for a task belonging to a task group, $\gamma_{i0}$ is equal to the priority...
of the group, i.e. a priority higher than all priorities of the tasks in the group; and for other tasks $\gamma_{i0} = \pi_i$.

Each other couple $(e_{ij}, \gamma_{ij})$, $1 \leq j$, corresponds to a critical section in $\tau_i$, with $e_{ij}$ the time needed to execute the code between the lock and unlock operations, and $\gamma_{ij}$ its ceiling priority.

3) Schedulability analysis algorithm: We point out that the computation of the worst-case response time developed is an exact one in the sense that: (i) the critical instant describes a worst-case scenario that may occur; (ii) the equation used to compute response times do not introduce approximations.

In [16], Wang and Saksena develop a schedulability analysis for a formalism close to ours. Beside our refinement of the notion of preemption threshold at the block level, notable differences are that our formalism allows to model shared resources, as well as systems with task sharing the same priority.

Wang and Saksena identify the critical instant for $\tau_i$:

- $\tau_i$ is activated together with an instance of each higher priority task,
- the task that contributes to the maximum blocking time has just started executing. Blocking time is defined as the maximal time where a task can be delayed by the execution of a lower priority task that it cannot preempt.

They prove that when computing the blocking time for a task $\tau_i$, one needs to consider blocking from only one lower priority task with a preemption threshold higher than or equal to $\pi_i$. Such a critical instant specification can easily be extended to our formalism by taking into account resource sharing.

Let us consider first resource sharing. From a scheduling point-of-view, IPCP exhibits the same worst-case performance as "original" PCP [17]. Thus, the properties of PCP shown by Shu et al. in [8] hold. Especially, the worst-case blocking time of $\tau_i$ caused by a lower priority task holding a resource the ceiling priority of which is greater than $\pi_i$ is reduced to the duration of at most one critical section of such a task. Concerning our formalism, it is obvious that a task $\tau_i$ can be blocked by at most one critical section of a lower priority task with a preemption threshold higher than $\pi_i$.

Concerning task sharing the same priority level, the critical instant for $\tau_i$ occurs when all tasks of priority higher or equal to $\pi_i$, except $\tau_i$, are activated simultaneously. Indeed, for tasks of equal priority, FIFO scheduling is used and [18] show that the worst-case response time for $\tau_i$ is found considering the first activation of $\tau_i$ at time $t \in [0, p_i]$. This leads to consider more activations of interfering tasks than only those made at time 0 (i.e. the date of the critical instant).

We can now define the critical instant for a task $\tau_i$. It is an instant where:

- $\tau_i$ is activated,
- each higher priority task is activated,
- the longest code section with a preemption threshold greater than $\pi_i$ but belonging to a lower priority task starts.

In the following, the date of the critical instant is 0. The date of the first activation of $\tau_i$ is $t (a_i (1, t) = t)$, where $t \in [0, p_i]$. We consider a discrete time semantics ([19] proves that this does not cause any loss of generality with respect to feasibility test of schedules), hence the number of values that can be taken by $t$ is finite. Of course, the date of the $q^{th}$ activation is:

$$a_i (q, t) = t + (q - 1)p_i$$

According to this definition of the critical instant for $\tau_i$, we obtain the following expression for $B_i$ (blocking time):

$$B_i = \max_{\forall t \leq \gamma_{ij} \leq \pi_i} \{ e_{ij} \}$$

We now derive the expression to compute $s_i(q, t)$, the date where the $q^{th}$ instance of $\tau_i$ in the busy period starts its execution. Obviously, it can start at last when: the blocking time is elapsed; all the previous instances of $\tau_i$ in the busy period have completed (non reentrancy); all higher priority tasks activated before $s_i(q, t)$ have completed; all task of priority $\pi_i$ activated before $a_i(q, t)$ have completed. We obtain the following fix-point equation:

$$s_i(q, t) = B_i + (q - 1)e_{i0} + \sum_{\forall j, \pi_j > \pi_i} \left( 1 + \frac{s_i(q, t)}{p_j} \right) e_{j0} + \sum_{\forall j \neq i, \pi_j = \pi_i} \left( 1 + \frac{a_i(q, t)}{p_j} \right) e_{j,0}$$

Between the start time of the $q$th instance of $\tau_i$ and its finish time, the scheduler can select for running tasks with priority higher than $\gamma_{i0}$. Thus, if $\tau_i$ is preemptible, its finish time is computed by solving the fix-point of:

$$f_i(q, t) = s_i(q, t) + e_{i0} + \sum_{\forall j, \pi_j > \gamma_{i0}} \left( \left[ \frac{f_i(q, t)}{p_j} \right] - \left[ \frac{s_i(q, t)}{p_j} \right] - 1 \right) e_{j0}$$

Notice that equation 4 holds for non-preemptive tasks, thanks to the preemption threshold concept. Indeed, for a non-preemptive task, its preemption threshold $\gamma_{i0}$ is equals to $+\infty$ and the last term of the equation (interference of higher priority tasks) is null.

In order to compute the worst-case response time of $\tau_i$, we must compute the response time of each instance of $\tau_i$.
appearing during the busy period, for each possible $t$ (date of the
first activation of $\tau_i$ in the busy period). Hence, we need
to compute $L_i(t)$, the length of the $\pi_i$-busy period

$$L_i(t) = B_i + \sum_{\psi,j \neq i, \pi_i \geq \pi_j} \left[ \frac{L_i}{p_j} \right] e_{j0} + \left[ \frac{L_i - t}{p_i} \right] e_{i0} \quad (5)$$

The number of instances of $\tau_i$ in the busy period being $N_i(t) = \left[ \frac{(L_i(t) - t)}{p_i} \right]$, we finally obtain:

$$R_i = \max_{t \in [0, \min\{L_i - 1, p_i\}], q \in [1, N_i(t)]} \{ f_i(q, t) - a_i(q, t) \} \quad (6)$$

In the following sections, we give some hints on how these results can be extended so as to take into account AUTOSAR OS specifities, especially (i) the timing protection mechanism and (ii) the schedule tables.

B. AUTOSAR OS timing protection mechanism and non periodic tasks

AUTOSAR OS introduces a protection service, so as to prevent
the occurrence of timing faults. This service triggers
a user-defined function when a task or ISR2 execution time
overruns its WCET, holds a resource for too long, or violates
its arrival rate. In this paper, we develop algorithms for high
level analysis of AUTOSAR OS applications. Hence, we will
suppose that tasks cannot overrun their WCET or hold
a resource for too long because of some hardware or software
failure. Similarly, we will suppose that timers are perfect (no
clock drift) and periodic tasks cannot violate their arrival rate.
Hence, we will only explore the timing protection mechanism
as a solution to the problem of correctly handling tasks with
non-periodic activation laws within a time-critical application.

Real-time scheduling theory classically consider two non-
periodic activation laws [20]: sporadic and aperiodic. The spo-
radic law applies to recurrent time-critical task, such that one
can give a lower bound between two consecutive activations.
This lower bound is sometimes called pseudo-period because,
if the task is activated at its maximal rate, it behaves like a
periodic task, the period being equal to the pseudo-period. An
aperiodic task is a non time-critical task whose activation law
is not constrained.

From a schedulability analysis point of view, the execu-
tion policy of sporadic and aperiodic tasks shall allow
the computation of worst-case response time for all the time-
critical tasks of the system. In [4], we outline that OSEK/VDX
OS does not natively provide the required mechanisms to
implement such a policy. Under certain hypothesis, the arrival
rate enforcement mechanisms of AUTOSAR OS allows to
overcome this limitation.

Let us start with ISR2 arrival rate enforcement. The AU-
TOSAR OS specification states that “The Operating System
shall limit the number of interrupt occurrences within a
configured timeframe (...).”. To the best of our knowledge,
the specification does not clearly state if the timeframe is a
static or a sliding one. As illustrated by figure 2, depending
on the interpretation, a same configuration can produce two
different scenarios.

Fig. 2. Two possible interpretation of ISR arrival rate enforcement with a limit of one activation per timeframe and discard of other requests: static timeframe (above) and sliding timeframe (below)

The second interpretation (sliding window) corresponds to
the classical definition of sporadic tasks. Hence, if it is the case
that it is the correct interpretation, it is easy to adapt existing
theoretical results: when computing the worst-case response
time of another task, each sporadic ISR is treated as if it is
actually a periodic one; to compute the worst-case response
time of a sporadic ISR, the algorithm must be adapted. Such
a problem is for instance studied by Bernat in [21]. If the
correct interpretation is the first one (static window), we are
not aware of any work dealing with such a problem. Original
algorithms have to be developed.

Let us consider now the task arrival rate enforcement
mechanism. It is very similar to the aperiodic server concept
(except that in AUTOSAR OS, the server can be used by only
one task). The idea of an aperiodic server is to bound the
amount of interference an aperiodic (or even sporadic) task
can produce on lower priority time-critical tasks. A server task
is described by: $C_s$, its maximum capacity, i.e. the amount
of time where the server task can be in the running state until
it is replenished; and $T_s$ its replenishment period.

As for ISR, to the best of our best knowledge, AUTOSAR OS specification does not clearly state if the timeframe is a
static or a sliding one. The first case (static window) is similar
to the Deferrable Server (DS) [22], which has its capacity
refilled to the maximum capacity periodically. Whenever a non
periodic tasks arrives and there is capacity available, it starts
executing until it finishes or the capacity is exhausted. If it does
not consume all the capacity, the remaining time is available
for the next instance of the task that could arrive in the same
server period. The maximum interference a DS can produce
on lower priority tasks in interval $[0, w]$ was shown to be [23]:

$$\left(1 + \left[ \frac{w - C_s}{T_s} \right] \right) C_s \quad (7)$$

The second case (sliding window) is equivalent to the
Sporadic Server (SS) [24], where the capacity of the server is
replenished to its maximal capacity exactly $T_s$ time units after
the acceptance of an activation request for a non periodic task.
If the task consumes $c$ time units, then the server will have its
capacity increased of $c$ time units, exactly $T_s$ time units after
the start time of the task. The maximum interference a SS can produce
on lower priority tasks in interval $[0, w]$ is [23]:

$$\left[ \frac{w}{T_s} \right] C_s \quad (8)$$

Equations (7) and (8) can be directly used with Equations
(2), (4) and (5) to compute the worst-case response times.
of task with priority lower or equal to the server priority.

The problem of computing the worst-case response time of a sporadic task the execution of which is controlled by a server has also been studied (for instance in [24]). However, in order for such results to be usable, it is mandatory to study more deeply the theoretical execution model and to see if it can be easily simulated with AUTOSAR OS services. In the theoretical work, when the capacity of a server becomes empty, it goes to a waiting state until it replenishes. In AUTOSAR OS, when the capacity of a server becomes empty, a protection function is called. From this function, it is possible to kill or start tasks. We don’t know yet if it is possible to block the server until it replenishes.

C. Taking into account schedule tables

1) Modeling schedule tables: In the real-time scheduling theory, the term “schedule table” is usually associated to offline scheduling techniques. In this context, a schedule table describes completely the activation points of all the tasks of the system. When a task is activated, it runs until it completes (no preemption). Obviously, except their name, AUTOSAR OS schedule table do not share many characteristics with this brief description. Notable differences include: an AUTOSAR OS schedule table describes only the activation points of a subset of the tasks of the system; it can be started at any time, thus it does not provide a complete description of the task activation times; different schedule tables can be attached to different counters, and thus can be asynchronous; the scheduling policy is potentially preemptive; etc.

For short, an AUTOSAR OS schedule table allows to specify the offsets between activation dates of a set of tasks. Thus, we have to consider the work dealing with schedulability analysis of tasks with offsets.

First, we can extend our formalism so as to be able to model schedule tables. A system is now a couple \( \langle \Gamma, T \rangle \), where \( \Gamma \) is the set of tasks and \( T \) is the set of schedule tables. A schedule table \( T_i \in T \) is a couple \( \langle P_i, T_i \rangle \) where:

- \( P_i \) is the period of the schedule table;
- \( T_i \subseteq \Gamma \) is the set of tasks activated by table \( T_i \);

A task \( \tau_i \in \Gamma \) is now a tuple \( \langle p_i, d_i, \pi_i, e_i, o_i \rangle \) where \( o_i \) is the offset of \( \tau_i \) relative to the activation of its schedule table (if \( \tau_i \) is member of a schedule table). Of course, for all tasks \( \tau_i \) member of a schedule table \( T_j \), AUTOSAR OS specification imposes \( o_i < P_j \) and \( p_i = P_j \). If a task is activated more than once in a schedule table, or member of more than one schedule table, it can be splitted in different tasks (the non-reentrancy of tasks being insured by the FIFO scheduling policy used within priority levels). Notice that we consider that all the time related attributes (WCET, period, offsets, deadline) are expressed in the same time base: the physical time. This implies that we consider an ideal case where the drift between the timers of the system is negligible.

Let us underline that the schedulability analysis algorithm developed in section IV-A.3 can still be used with the formalism extended to schedule tables. Indeed, in the basic formalism, all tasks are non-concrete, which means that the date of their first activation is unknown (a similar formal proof can be found in [25]). However, it is reasonable to think that exploiting the new information should allow to improve the accuracy of the results. We explore this way below.

2) Schedulability analysis: In a first time, let us consider the simple (and most predictable) case where we know the relative activation dates of all the schedule tables of the system. It means that:

- all the schedule tables are driven by the same counter;
- all the schedule tables are started with the StartScheduleTableAbs service, and each time a schedule table is activated, it is at the same absolute counter value (but two different tables can be started at two different values).

A model meeting these hypotheses can easily be translated to a more simple formalism, obtained by extending our original formalism with offsets. For instance, consider two schedule tables \( T_1 = \langle P_1, \{\tau_1, \tau_2\} \rangle \) and \( T_2 = \langle P_2, \{\tau_3, \tau_4\} \rangle \). \( T_1 \) is always activated at the counter date \( c_1 \), while \( T_2 \) is always activated at the counter date \( c_2 \). To conduct the schedulability analysis, we can consider the set of tasks \( \{\tau^1_1, \tau^2_2, \tau^1_1, \tau^4_4\} \) where \( o_1 = o_1 + c_1, o_2 = o_2 + c_1, o_3 = o_3 + c_2 \) and \( o_4 = o_4 + c_2 \). From a schedulability analysis point-of-view, both model are equivalent.

The model obtained after the transformation exposed above is similar to the usual model of task with offsets studied in the real-time scheduling theory. Thus, in the case where the offsets are not all equal (tasks are said asynchronous), the work of Redell and Törngren [26] can be used to improve the accuracy of the analysis. Due to space limitation, we simply explain the idea of their algorithm. All tasks are assumed periodic, with offsets, and independent. Many tasks can share the same priority level, deadlines can be arbitrary large, and shared resources are managed with PCP. In order to compute the worst-case response time of a task \( \tau_i \), all the task instances that may interfere with \( \tau_i \) are splitted into different sets. The construction of the set is function of (i) the activation time of the included instances and (ii) the start time of \( \tau_i \) (as for the computation presented in section IV-A.2, where we consider interference before \( s_i(q, t) \), and after). The complexity is linear with the number of instances within the hyperperiod (the least common multiple of all periods).

To extend their method to tasks with preemption threshold (the main difference between their formalism and ours), we just have to modify the interference term that they use (Eq(12) in [26]), so as to take into account preemption threshold: after the start time of a task \( \tau_i \), interference can only be caused by interfering tasks, the priority of which is higher than \( \gamma_{i0} \).

Now, if we relax our hypothesis concerning schedule table relative activation times, it is impossible to perform the proposed transformation. The only information that we can use to reduce the pessimism of the analysis is the relative activation dates of the tasks of a same schedule table. This problem can be related to the problem studied by Tindell in [27]. In its paper, Tindell develops an algorithm to compute an upper bound on the worst-case response time for a set
of task grouped into "transactions". A transaction is periodic and contains a set of tasks. Each task of the transaction is activated with an offset relative to the activation of the transaction. Lastly, transactions are asynchronous. To reduce the pessimism of schedulability analysis, this method considers a critical instant where all tasks are not released together. Here, to compute the worst-case response time with schedule tables, we can consider each schedule table as a transaction.

Palencia et al. in [28] extend Tindell’s algorithm and propose an exact response-time analysis under a full preemptive scheduling policy with shared resource. The price to pay for computing exact values is an increase of the complexity of the algorithm. For this reason, Palencia et al. also propose a simpler algorithm that computes an upper bound approximation. To extend Palencia’s results to mixed scheduling, we have to take into account preemption threshold (as above) and use Equation (1) to compute the blocking time.

V. CONCLUSION

As stated in the introduction, the purpose of this paper is to give some partial answers to the following questions:

- Can we find in the real-time scheduling theory some results that could be used for the analysis of real-time applications built upon the scheduling service provided by an AUTOSAR OS implementation?
- If the response to the first question is affirmative, how can we adapt these results to the practical context of AUTOSAR OS? How can we restrict the usage of the scheduling services, so as to comply with the hypotheses assumed by the theory?

The answer to the first question is obviously "Yes". We did already answer this question in the context of OSEK/VDX OS in [4]. Here, we prove that the extensions proposed by AUTOSAR OS, mainly timing protection and schedule tables, correspond more or less to pre-existing concepts developed in the last 25 years within the real-time scheduling theory community. Two surprising points reward to be noticed:

- AUTOSAR OS specification sometimes use confusing terms, especially "schedule tables". This could lead the reader to a misunderstanding of the specification.
- Despite the fact that it targets safety-critical system, there exist some ambiguous points (or is it our understanding of the specification which is to blame?) in AUTOSAR OS 2.0.1 specification, for instance the precise behavior of the arrival rate enforcement of ISR and tasks.

Concerning the second question, partial answers are given in section III. We have proposed different formalisms allowing to model the schedulability problem associated with some AUTOSAR OS applications. We have also explained how existing analysis techniques shall be modified so as to fit with these formalisms.

Of course, some complementary works still have to be done before to obtain a mature proposition. Today, we believe that the two most urgent working directions are the following:

- we need to confront our work to an industrial case-study, so as to know if the design constraints imposed by the analysis techniques proposed so far must be relaxed or not.
- we need to perform a quantitative comparisons of the different analysis methods. The idea is to be able to choose the right tradeoff between results accuracy and algorithms complexity. Such a choice is crucial if the analysis algorithm is to be called, for instance, from an iterative optimization process.

ACKNOWLEDGEMENT

The authors would like to thank the reviewers for their helpful and judicious remarks.

This work is supported by DGE/FCE n06.2.93.0619, project O4A.

REFERENCES


