Towards Hierarchical Scheduling in AUTOSAR*

Mikael Åsberg, Moris Behnam, Farhang Nemati, Thomas Nolte
MRTC/Mälardalen University
P.O. Box 883, SE-721 23 Västerås, Sweden
mikael.asberg@mdh.se

Abstract

AUTOSAR [17] is a partnership between automotive manufactures and suppliers. It aims at standardizing the automotive software architecture and separating software and hardware. This approach makes software more independent, maintainable, reuseable, etc. Still there is much work to do in order for this standard to be usable. This paper focus on automotive software integration in AUTOSAR, with the use of hierarchical scheduling as an enabling technology. At this point, AUTOSAR components do not have any timing relation with its tasks [19, 20]. This causes an unpredictive runtime behavior which can only be analyzed and verified after integration phase. We will discuss how integration can be done in AUTOSAR, with runtime temporal isolation of components. This will enable schedulability analysis at the level of components rather than at the level of tasks.

1 Introduction

Our earlier work includes the development of a synchronization protocol for hierarchically scheduled subsystems [10] and techniques to minimize CPU system load when resources are shared between subsystems [22]. Moreover, we have implemented a hierarchical scheduler in the real-time operating system VxWorks [9]. It is equipped with a fixed priority and preemptive scheduling mechanism (at both global and local level). Also, we have measured and shown the scheduling overhead of the implementation. This paper is a continuation of our research in the area of hierarchical scheduling. It gives a summarized overview of the problem (and possible solution) of software integration in AUTOSAR, with respect to temporal behavior. The basic idea of AUTOSAR is to separate the software and hardware in subsystems (delivered by sub-contractors to car manifactures). Further, its initiative is to modularize the software into independent components, with well defined and standardized interfaces. In this way, competition in automotive software will increase, and software will become more portable and hardware-independent. Also, car functions will be composed of software parts (delivered by different vendors) in a higher degree. This will put more effort on software integration and make it more difficult. The reason for this is because functionality, in the form of Electronic Control Units (ECU) with integrated software (in a final product), are not longer delivered. Instead, car manifactures will most likely buy software components to a higher degree and integrate the components themselves, in order to get the functionality. The problem is that software integration will be more complex (in the temporal view) and AUTOSAR does not tackle this issue. Software components may share tasks (with other components), which at runtime, will create timing dependencies between components [19, 20] (Figure 1).

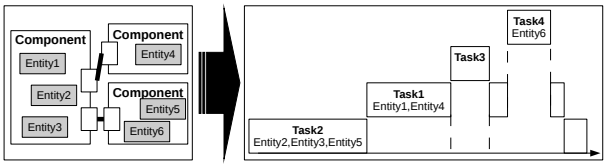

Figure 1. Hidden timing dependencies

There will be no relation between the high level component and its tasks' timing behavior. This makes component integration difficult, since schedulability analysis and timing verification can only be done once all the components have been mapped to tasks and assigned to ECUs. The idea is to use hierarchical scheduling, allowing for each component to be partitioned and scheduled individually. Thus, the Original Equipment Manufacturer (OEM) can then integrate components from different vendors into an ECU, without causing interference between components in the temporal domain (at runtime). Also, timing verification can be done at an earlier stage in the development process, even before the components are delivered to the OEM.

AUTOSAR The following briefly introduces the concept of AUTOSAR and the main properties of its infrastructure.

The AUTOSAR main structure (Figure 2) consists of Software Components (SW-C), at the application level, with standardized and well defined interfaces. They inter-

*The work in this paper is supported by the Swedish Foundation for Strategic Research (SSF), via the research programme PROGRESS.

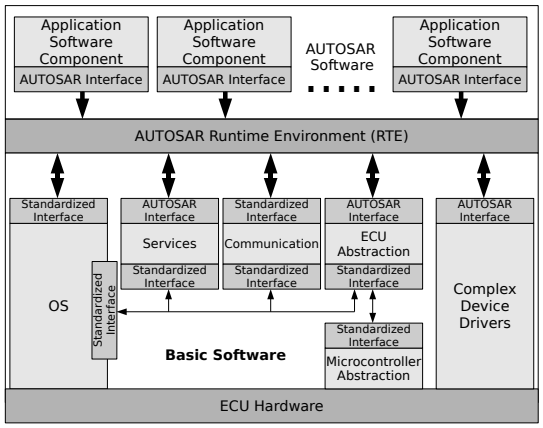

Figure 2. AUTOSAR infrastructural overview

act with the Runtime Environment (RTE) in order to communicate with other SW-Cs and Basic Software (BSW) modules [6]. The BSW modules lie beneath the application layer and provides basic services. They also have standardized interfaces which makes the whole architecture modular and hardware-independent. The RTE is a tailored (depending on SW-Cs, BSW modules and all types of communication) middleware for each ECU. It removes communication and hardware dependencies from SW-Cs.

An atomic component is a single component, whereas a composition of components is a collection of components [4] (Figure 3). An atomic component has ports (for communication) and schedulable parts called *runnable entities*. These can have timing properties, similar to those in the periodic task model (e.g., period, priority, etc.). During mapping from components to tasks, the runnable entities are mapped to tasks (many-to-one mapping). It is important to note that entities from different components can be mapped to the same task.

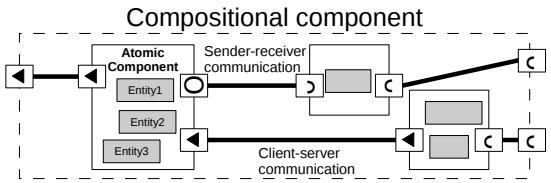

Figure 3. AUTOSAR component

The Operating System (OS) used in AUTOSAR, referred to as AUTOSAR OS, is OSEK/VDX [1] compliant. It has extensions like task/interrupt arrival rate enforcement (activations are monitored and restricted to a maximum amount per time frame) and schedule tables (task offset management) [5].

Hierarchical Scheduling Framework The Hierarchical Scheduling Framework (HSF) has been introduced to support hierarchical resource sharing among applications under different scheduling services. The HSF can generally be represented as a tree of nodes, where each node represents an application with its own scheduler for scheduling internal workloads (e.g., tasks). Further, resources are allocated from a parent node to its children nodes [23]. One of the main advantages of HSF is that it provides means for decomposing a complex system into well-defined parts (subsystems). In essence, the HSF provides a mechanism for time-predictable *composition* of coarse-grained subsystems. This means that subsystems can be independently developed and tested, and later assembled without introducing unwanted temporal behavior. Also, the HSF facilitates *reusability* of subsystems in time-critical and resource constrained environments, since the well defined interfaces characterize their computational requirements. The system contains a set of *subsystems* and a *global scheduler*. Each subsystem contains a set of *tasks* and a *local scheduler* (Figure 4).

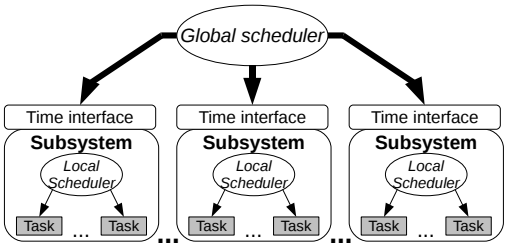

Figure 4. HSF

Subsystems are scheduled according to the scheduling policy of the global scheduler (for example FPS or EDF) and the parameters in the subsystem *time interface*. Subsystems can be represented as virtual tasks, where the parameters in the time interface corresponds to those in the periodic task model [16]. At runtime, subsystems are allocated a defined time (*budget*) every predefined *period* and they are executed based on their *priority*. This is similar to a hard task executing preemptably (depending on its priority) a defined time, Worst Case Execution Time (WCET), periodically at every period. When a subsystem is selected by the global scheduler, the subsystem tasks are executed and scheduled according to the scheduling strategy of the subsystem local scheduler. The global scheduler, and all the subsystem local schedulers, may all have different scheduling strategies (FPS, EDF .etc).

The outline of the paper is as follows: Section 2 presents related work. Section 3 focus on how the HSF can be integrated in AUTOSAR with respect to modeling, infrastructure, existing operating system, resource sharing and legacy applications. Section 4 gives an example of subsystem integration in AUTOSAR and finally, Section 5 concludes.

2 Related work

In [13], the authors discuss the issue of system-level integration of components from different suppliers in the aspect of timing isolation. The paper advocate for a

more extensive component interface which should include non-functional properties. In order to solve the integration problem of fault-isolation and error-containment, in both the temporal and logic domain, components could be placed in separate Intellectual Property (IP) cores (isolated by the help of hardware).

The TIMMO project [18] is focusing on how to validate timing behavior during the development process in distributed embedded automotive systems. The connection to AUTOSAR is how to insert useful time interfaces to the infrastructure, in order to be able to do proper validation with respect to time.

In [19, 20], the problem of lacking time interfaces are brought up. When mapping component runnable entities to tasks (basically component to task mapping), there are no restrictions when it comes to component boundaries. The component encapsulation is broken when it comes to scheduling, i.e., any runnable entity from any component can interfere with other components. This is possible because entities from different components can reside in the same task and any task can interfere with any other task. Thus, timing analysis can only be done after mapping since the component level does not reveal any timing behavior.

In [14], the authors show how schedulability analysis at the application level can be performed with relation to the AUTOSAR OS. Still, timing verification is done on task level after component integration but with consideration of AUTOSAR OS services.

3 HSF integration aspects

The use of hierarchical scheduling in AUTOSAR brings positive aspects such that it solves a part of the integration problem (component temporal isolation). The issue is how to integrate hierarchical scheduling in AUTOSAR (with minimum impact). The following sections discuss key issues when it comes to HSF integration in AUTOSAR.

3.1 AUTOSAR model

A subsystem in HSF typically encapsulates a big cohesive system, such as a seat-heater system, which in AUTOSAR would be modelled as a composition rather than an atomic component [4]. A subsystem should therefore represent a composition component (Figure 5) with the restriction that such a component is not allowed to map its tasks together with other components (other than the components in the same composition). The reason for this restriction is because it preserves temporal isolation and it facilitates schedulability analysis. In AUTOSAR however, components are allowed to share tasks with each other. This mismatch can be solved by having an optional type of component which would have a restrictive task mapping policy.

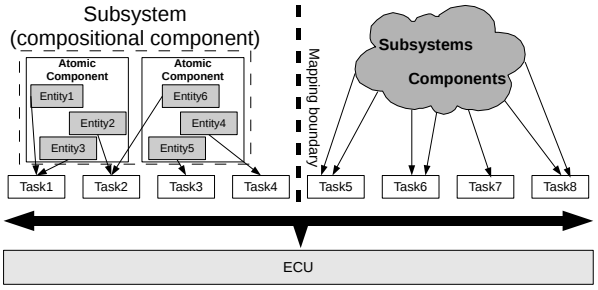

Figure 5. Subsystem mapping

3.2 AUTOSAR infrastructure

The implementation of HSF in AUTOSAR is a big challenge, since it should not impose any kind of restriction or dependency on OS nor the rest of the AUTOSAR framework. AUTOSAR OS and BSW scheduler [3] are the responsible parts of task scheduling in AUTOSAR. The BSW scheduler is responsible for mapping BSW module processing-functions to tasks as well as defining their triggers (cyclic or event triggered). The BSW module has an OS-independent configuration for which the BSW scheduler, together with OS primitives, use to form the glue-code between BSW modules and the AUTOSAR OS. Hence, the BSW module is independent of the OS scheduling mechanism. The RTE forms, in the same way, the scheduling specific glue-code between SW-C and AUTOSAR OS. The BSW scheduler does not influence scheduling during runtime, which means that the HSF schedulers (global and local) do not need to consider the function of the BSW scheduler.
The global scheduler (in HSF) can be integrated by putting it on top of the existing AUTOSAR OS and use the available operating system interface to accomplish hierarchical scheduling. This is similar to the technique used in [9], where our schedulers merely uses VxWorks primitives to perform scheduling. Our schedulers are loaded into the VxWorks kernel while the kernel itself is kept unmodified. In fact, in AUTOSAR it might be possible to define the global scheduler itself as a BSW module and to use the BSW scheduler to interface the AUTOSAR OS (Figure 6). In this way, the global scheduler module would be completely transparent and independent of the actual OS.

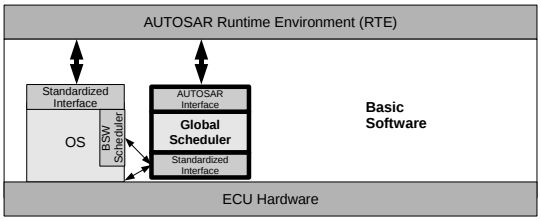

Figure 6. HSF in AUTOSAR infrastructure

The global scheduler could interface the AUTOSAR OS either directly or via the BSW scheduler. A connection between global scheduler module and the RTE could be useful in order to forward subsystem scheduling infor-

mation (e.g., period, budget, local scheduling algorithm etc.) from components to the HSF. An alternative solution is to let the global scheduler have a separate configuration file with subsystem parameters. A specification/implementation of a local scheduler could be done as a SW-C or as a BSW module itself with a standardized interface. This would add up to a more modular implementation of HSF in AUTOSAR by completely separating the subsystems from the global scheduler. This is the ultimate goal because it realizes *open real-time systems*. The idea is that subsystems are developed (and validated) in isolation, independently of each other, and later integrated together in the same environment without any modifications. At integration phase, schedulability analysis can be done, based on the subsystem interfaces. The subsystem local scheduler, with standardized AUTOSAR interface, could just be plugged into the infrastructure and interface the global scheduler BSW module which would then do the rest (scheduling of subsystems and running the subsystem local scheduler when it is active).

3.3 Compatability with AUTOSAR OS

AUTOSAR OS, as mentioned before, is an extension of OSEK/VDX. The OSEK/VDX standard defines that there could reside both preemptive and non-preemptive tasks. This is also true for *task groups* [1] which are used for tasks sharing a resource. A task group has a priority ceiling (same as the highest priority task within the group) and behaves non-preemptive within the group. The only tasks allowed to preempt are those that have higher priority than the group priority ceiling.

A problem arises if a budget depletes during the execution of a non-preemptive task, since the task is not allowed to be preempted. A solution to this problem could be to model non-preemptive tasks, such that they have a resource locked during the whole execution time, and use a hierarchical synchronization protocol such as SIRAP [10] or HSRP [12]. SIRAP uses a technique called skipping, where a task is not allowed to lock a resource if the maximum resource locking time (including the critical section and possible task preemptions) exceeds the remaining budget time. HSRP uses another technique called overrun. This method extends the budget until the task releases the resource. A hierarchical synchronization protocol can also be applied to task groups, in the case when the group is distributed across several subsystems. No problem arises if the task group resides in one subsystem.

One service that is not included in OSEK/VDX, but extended in AUTOSAR OS, is the notion of *schedule tables* [5]. A schedule table has a duration (period) which is the time between two consecutive schedule table activations. Further, a schedule table has expiry points expressed as offsets, with respect to the table duration, and each expiry point has a list of events and activations (e.g. activate a task). All tasks within a schedule table typically has the same period but individual offsets. Schedule tables can be single shot or repeated (cyclic) and they can be activated and deactivated during runtime. If a schedule table in a non-active subsystem would activate tasks, then they should not run (according to HSF) although they may be inserted into the task ready-queue. Setting an event would also occur. The only restriction is that there would be no re-scheduling since the active subsystem tasks are only allowed to run. The positive aspect is that there can be a mix of scheduling schemes (e.g., schedule tables, FPS, EDF etc.) separated by the subsystems, without them affecting each other with respect to scheduling.

3.4 Resource sharing

Hierarchical scheduling introduces more complexity when it comes to resource sharing. If several tasks within a HSF subsystem would share resources, then traditional resource sharing protocols can be used such as the Immediate Priority Ceiling Protocol (IPCP) (which is supported by the OSEK/VDX standard [1]). However, if tasks belonging to different subsystems would share resources, then resource sharing protocols such as SIRAP [10] or HSRP [12] should be used (in order to achieve mutual exclusive access to resources). Implementation of SIRAP and HSRP requires mutual exclusive access to kernel data-structures such as the task ready-, subsystem ready- and resource-queue [8]. So in a sense, the hierarchical synchronization protocol itself also need protection from interrupt routines and schedulers (which may also use scheduling specific data-structures). AUTOSAR specifies that the responsibility for this kind of resource sharing should be put on the BSW scheduler [2], even though IPCP can be used (which is part of OSEK/VDX). The BSW scheduler is in charge of resources, their users and the protection of these resources. BSW scheduler extracts resource/user information from AUTOSAR OS, which in turn get such information from the component configuration via the RTE. An implementation of a hierarchical synchronization protocol would typically be a user of such a resource that the BSW scheduler handles. The hierarchical synchronization protocol could be implemented as a BSW module, similar to the HSF (Figure 6), and interact with the BSW scheduler in order to get exclusive access to scheduling specific data-structures. SIRAP and HSRP both require SRP, which IPCP resembles, so it could be used instead. Thus, making hierarchical synchronization protocols easy to adapt to OSEK/VDX. The main goal is that the protocol should be a module that can be put on top of an existing OS, without any modifications.

3.5 Legacy applications

Development of automotive systems usually includes reusing of legacy code. Hence, when developing with AUTOSAR mechanism the legacy software should be mapped to the AUTOSAR standard. However, the existing legacy code is usually big and complex, so migration of existing code to AUTOSAR is a big challenge for automotive industry.

When migrating legacy applications, it is important that functional, as well as non-functional (e.g., temporal behavior), behavior of the application is not affected

by other applications in the new environment. Hierarchical scheduling provides the means for isolating applications (subsystems), so that they do not affect each other while scheduled on the processor. In this policy, the legacy application can be put into a subsystem and execute along with other applications without being aware of each other. However, the hierarchical scheduling policy needs the timing attributes of real-time tasks within the application. These attributes are used in order to generate subsystem time interfaces and derive schedulability test of subsystems. Timing attributes can be derived from static (source code) and/or dynamic analysis (run-time).

4 Illustrative example

In order to illustrate the benefits and applicability of hierarchical scheduling in AUTOSAR, we show a subsystem integration example in this chapter. The two subsystems, in this example, consists of an Anti-lock Brake System (ABS) and an Engine Control System (ECS). Both systems have the characteristics of being safety critical, i.e., incorrectness in their functionality and temporal behavior can be life threatening. It is not uncommon that these systems are delivered by the subcontractor as a whole product, including hardware (ECU) and software. After delivery, the ECUs are directly integrated without any changes or re-configurations. AUTOSAR however, tends to work against this by separating parts of the subsystem products. This, of course, puts more responsibility on the OEM in terms of subsystem integration. However, AUTOSAR has not yet defined how software integration can be done (with respect to temporal behavior) in an efficient manner. This is especially important for safety critical systems. If AUTOSAR realizes its purpose, then in near future one might vision software integration in a way that is not seen today. For example, integration of safety critical software from different vendors on the same ECU. This leads us to this example, where we illustrate how software integration can be done in AUTOSAR (efficient and safe) by the use of hierarchical scheduling.

4.1 Anti lock brake system

The purpose of ABS is to prevent wheel lock-up during braking by monitoring the wheel speed and, at detection of wheel lock-up, release the brake power [11]. At each wheel, there is a wheel speed sensor that reads the wheel rotation speed, and thus, gives the wheel deceleration. An ECU (with control algorithms) compares all wheels deceleration, in order to detect a wheel lock-up before it actually occurs. When the control algorithm detects an upcoming lock-up, the ECU manipulates the brake system. The manipulation is done by regulating two solenoid valves, belonging to a hydraulic pressure modulator (which is integrated in the ABS ECU). The two kinds of solenoid valves, input-valve and outlet-valve, are both installed at each of the four wheels. The input-valve is open when it is in normal braking state, i.e., normal braking. In this state, brake-fluid can flow between the master brake-cylinder (connected to the brake-pedal) and a wheels brake-cylinder. The brake-cylinder is under constant fluid-pressure when the input-valve is closed, no matter how much the driver pushes the brake pedal. The outlet-valve remains closed when input-valve is open. When the input-valve is closed, the outlet-valve can be opened. This will lead brake-fluid out from the brake-cylinder, thus releasing the brake-pressure at the affected wheel. The valve states and temporal properties are shown in Figure 7. During the build-up phase, the input-valve is kept open in order for the fluid to pass from master brake-cylinder to brake-cylinder without any resistance. When the input-valve is closed (can take up to 30 ms [21]), the pressure-holding phase begins until either the input- or the outlet-valve is opened. The next phase shown is the reduction phase, it can be required to be as short as 20 ms. The build-up phase however, can take up to 200 ms.

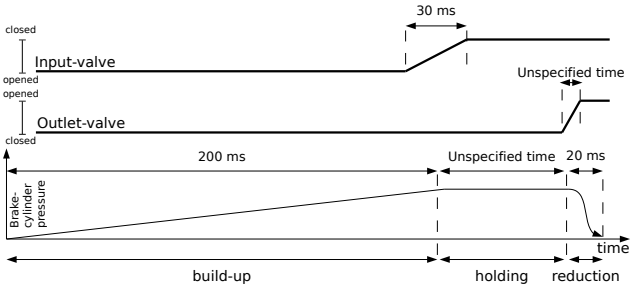

Figure 7. Valve states

Based on this limited temporal information, it would be suitable with at least three tasks (responsible for each wheel). One task for actuation of the solenoid valves, *Actuator* (τ_1), one for reading the speed sensor, *Sensor* (τ_2), and a task responsible for the control calculations, *Control* (τ_3). The latter task is assumed to include/share sensor data from the other wheels when calculating the braking power.

Since the reduction phase is required to be as short as 20 ms, the time between the control cycles should not be longer than this value. This implies that the tasks *Actuator*, *Sensor* and *Control* should have at least half of that value (10 ms), as period. If the reduction should be too long, then we would loose valuable braking distance. *Sensor* should of course always run before *Control*, in order for it to have access to fresh sensor values. *Actuator* should run after *Control*, so that the calculated data takes effect. This implies that *Sensor* has highest priority, followed by *Control*, and *Actuator* with lowest priority.

There is no information about the frequency of the control cycles in the specification, therefore, we disregard this aspect.

The input-valve may take 30 ms to close properly. We assume that manipulation of the outlet-valve (opening) within this time does not take affect.

The estimated WCET for *Control* is 0.5 ms and 0.1 ms for *Sensor* and *Actuator*. Table 1 concludes task parameters, where T represents task τ:s period and D the relative deadline. In this integration example we only include 3

tasks (one wheel) from the ABS subsystem.

Name	T	$WCET$	D	Priority
Sensor (τ_1)	10	0.1	10	0
Control (τ_2)	10	0.5	10	1
Actuator (τ_3)	10	0.1	10	2

Table 1. ABS task parameters

4.2 Engine control system

The next example is a one cylinder, four-stroke, ECS. It is comprised of 6 tasks, distributed on 3 processors [15]. The system is scheduled by the Local NonPreemptive Time-Triggered scheduler (LNPTT). The scheduler releases a new period for the tasks when the engine piston reaches its top, every second time (period of 720°). This is illustrated in Figure 8. The tasks, *Engine* (τ_4) and *Throttle* (τ_5), do engine speed and throttle opening acquisition respectively. This is needed by both the task *Ignition* (τ_7), as well as task *Injection* (τ_6), in order for them to calculate ignition angle and precise injection time. Task *Ignition* needs to finish before **Compression** phase. This is, in worst case, after 5 ms when engine speed is at its maximum, i.e., 6000 Revolutions Per Minute (RPM). Task *IG-Actuator* (τ_9) will actuate the ignition based on data from *Ignition*, preferably 1,5 ms before **Combustion** phase (there is a time delay when actual ignition occurs). Task *IN-Actuator* (τ_8) needs to inject gas before the next **Intake** phase. Task *Injection* needs to run before *IN-Actuator*, in order to calculate the gas injection time. In a worst case scenario, when engine is running at its maximum speed, all tasks will not meet their deadlines if they were to be executed on the same processor. The actuating tasks are extremely time sensitive, so they will run on their own processor. *IG-Actuator* cannot be released before *Ignition* has finished, the same goes for *IN-Actuator* with respect to *Injection*.

ECS tasks, within the same processor are released based on their WCET. The order is τ_4, τ_5, τ_7 and τ_6, corresponding to the release times: 0, 0.6, 1 and 3.5. Tasks scheduled by LNPTT on the same processor are non-preemptive mutually, but preemptive globally. In this integration example, we only include the tasks from the **Controller ECU** (Figure 8), within the ECS.

The tasks temporal properties are listed in Table 2. Note that the worst case period is 20 and the deadline is relative to this period.

Name	T	$WCET$	D	Priority
Engine (τ_4)	20	0.6	2.1	-
Throttle (τ_5)	20	0.4	2.5	-
Injection (τ_6)	20	2.5	8	-
Ignition (τ_7)	20	2.5	5	-
IN-Actuator (τ_8)	20	12	20	-
IG-Actuator (τ_9)	20	3.5	8.5	-

Table 2. ECS task parameters

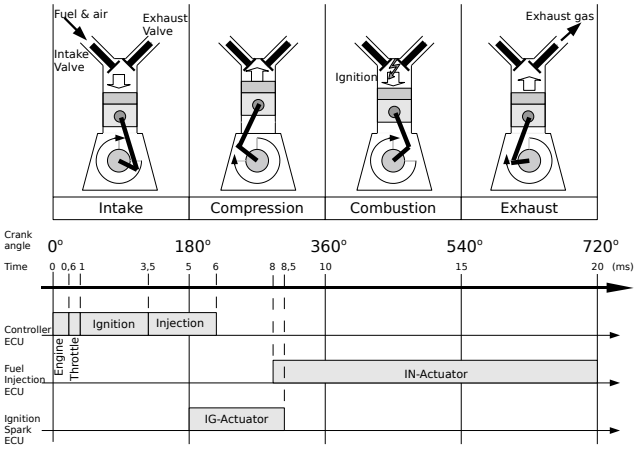

Figure 8. Four stroke engine cycle

4.3 Subsystem integration

There are (at least) three major concerns when it comes to subsystem integration and task mapping in the AUTOSAR scenario. **1) Schedulability analysis**, entities are mapped to tasks and schedulability analysis is done on task level. Analysis can be difficult since there may be many tasks and they may also be configured with different scheduling algorithms. **2) IP protection**, the inside of the component is exposed (because of task level schedulability analysis) which is bad if you want to hide the internal configuration. **3) Effective temporal isolation**, tasks can disturb each other if their execution time exceeds their WCET. The solution of having task deadline checking, as part of AUTOSAR OS, is expensive (frequent overhead).

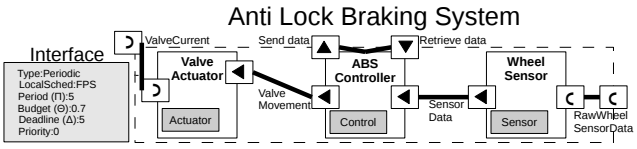

Figure 9. ABS component

Figure 9 and Figure 10 show how these two subsystems can be modeled according to AUTOSAR and how timing properties (time interface) can be expressed at component level instead of entity level.

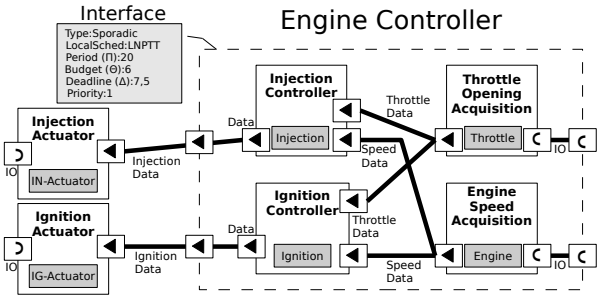

Figure 10. ECS component

ABS time interface For efficiency, the ABS subsystem period is set to half of the lowest task period [24]. The

budget is set, so that it satisfies Eq. (1) [23]. A subsystem task-set is defined as $\{\tau_1 ... \tau_n\}$ and a task, τ_i, has the parameters T_i, D_i and C_i (corresponds to WCET). In order for the subsystem to be schedulable, the following must hold:

$$\forall \tau_i \; \exists t : 0 < t \leq D_i, \;\; rbf_{\mathsf{FP}}(i,t) \leq sbf_{\Gamma}(t), \qquad (1)$$

where $rbf_{\mathsf{FP}}(i,t)$ denotes the request bound function of a task which is the sum of task τ_i:s WCET and the interference from higher priority tasks within the time period t. $sbf_{\Gamma}(t)$ denotes the supply bound function for CPU resource, which is the minimum amount of supplied budget within the time interval t. Γ represents the CPU resource (according to the periodic resource model) with period Π and budget Θ. Eq. (2) and Eq. (3) show how to calculate the request and supply bound functions, respectively:

$$rbf_{\mathsf{FP}}(i,t) = C_i + \sum_{\tau_k \in \mathtt{hp}(i)} \left\lceil \frac{t}{T_k} \right\rceil \cdot C_k, \qquad (2)$$

$$sbf_{\Gamma}(t) = \left\lfloor \frac{t - (\Pi - \Theta)}{\Pi} \right\rfloor \cdot \Theta + \epsilon_s \qquad (3)$$

where ϵ_s is defined as

$$\epsilon_s = max \left(t - 2(\Pi - \Theta) - \Pi \left\lfloor \frac{t - (\Pi - \Theta)}{\Pi} \right\rfloor, 0 \right) \qquad (4)$$

As an example, lets look at the analysis for task *Actuator*. This task is guaranteed to meet its deadline if the worst case holds. This will happen when all higher priority tasks are released at the same time as *Actuator*, and the budget is distributed so that the task requests budget precisely at the time when it depletes and the last required budget is supplied as late as possible (Figure 11). Calculating the (minimum) subsystem budget requires an iteration (while increasing Θ) of Eq. (1), until it satisfies the equation. Figure 11 intuitively shows that task *Actuator* will meet its deadline if Θ is at least 0.7, this also holds for task *Sensor* and *Control*.

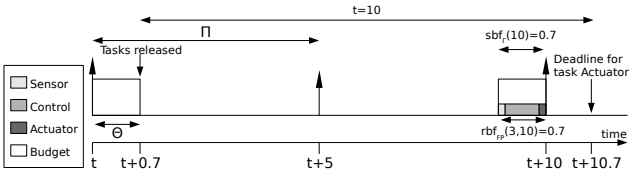

Figure 11. Task Actuator response time

ECS time interface As illustrated in Figure 12, if the ECS subsystem budget run as late as possible (according to its deadline), then all tasks meet their deadline (Eq. 5). Because the tasks and their subsystem are assumed to be released simultaneously (due to that the subsystem is event driven), the budget supply cannot be distributed in any possible worse case, with respect to its tasks arrival (as shown in Figure 12). Compared to Eq. 3, which accounts for that the tasks miss an entire budget and that the last one arrives as late as possible. In the ECS subsystem, only the latter can occur, according to our assumptions.

$$\forall \tau_i, \;\; prt_{\Gamma}(R_i) \leq D_i, \qquad (5)$$

where $prt_{\Gamma}(R_i)$ denotes the prolonged response time of task τ_i due to the periodic resource Γ and R_i represents the response time of task τ_i according to the Response Time Analysis (RTA) [7]. Eq. 6 show how to calculate the prolonged response time.

$$prt_{\Gamma}(R_i) = R_i + (\triangle - \Theta) \qquad (6)$$

where $\triangle$ denotes the deadline of the periodic resource Γ, or the period, in case of $\triangle = \Pi$.

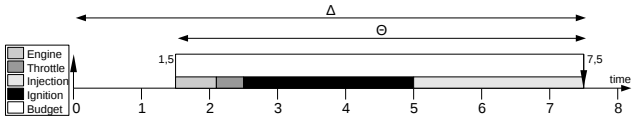

Figure 12. Deadline and budget for ECS subsystem

Conclusion It is the subcontractors responsibility that the local scheduling, within a subsystem, is sufficient with respect to the subsystem interface (illustrated in Figure 11). The OEMs responsibility is to map subsystem entities to tasks (and no sharing tasks between subsystems) and perform schedulability analysis at subsystem level. In this example, it is a matter of performing a simple RTA (assuming FPS as the global scheduling strategy) with the subsystem parameters (Figure 9 and Figure 10). Still there is one issue, how and who will define the subsystem parameters and how to 'contract' these parameters between OEM and the subcontractors.

Temporal isolation is kept due to the budgets, with minimal overhead. Concerning tasks that execute more than their WCET and the protection of this, it could be more efficient (and less overhead) to put such protection at subsystem level (by having budgets) rather than at task level. A missbehaving task inside a subsystem might not need a preventive action, because it is a critical task. It is better to let the subsystem handle this. For example, by decreasing non-critical tasks (within the same subsystem) execution time. In this way, the subsystem budget will remain the same and tasks outside of the subsystem will not get affected. Also, the subsystem critical tasks will still meet their deadlines.

Even though these subsystems are safety critical and are comprised of several tasks with different scheduling strategies (LNPTT and FPS), it is still safe (temporal) and easy (schedulability analysis) to integrate them, as shown in this example. Another positive aspect is that temporal information, needed for integration, can be placed as part of the component interface. This will make timing behavior, at runtime level, as encapsulated and predictable as on design level. These properties coincides well with the AUTOSAR paradigm.

5 Summary

We have shown how application integration, in AUTOSAR, can be achieved with the use of Hierarchical Scheduling Framework (HSF). The challenge addressed in the paper is component isolation at runtime [19, 20]. The components isolate functionality and communication nicely at design phase. However, when it comes down to task-mapping and running the tasks, then timing behavior crosses the component boundaries. Conclusively, there is a problem at both design/modeling, as well as in the runtime infrastructure. HSF solves the runtime issue. What is left is to incorporate the notion of subsystems/partitioning in the component model. This can be done with some restrictions on task-mapping. Another issue is how to integrate the HSF in the AUTOSAR infrastructure, with minimum impact. We have suggested a modular solution by integrating the the global scheduler, in HSF, as a BSW module. The global scheduler will interface either, the AUTOSAR OS itself, or the BSW scheduler, which will give it access to scheduling functions in a standardized way. Other related issues discussed are HSF compatability with AUTOSAR OS, how to solve additional synchronization issues (which HSF brings), legacy application integration and how to contract the subsystem parameters between OEM and the subcontractors. This last issue remains open. Hopefully, as our work progresses, we will focus more towards the practical side of HSF. Speculatively, AUTOSAR will open up the automotive software market and introduce more competition. This might lead to that OEMs can chose the best fitted software (ABS, seat-heating .etc) from a wide variety of software components (e.g. libraries), labeled with simple subsystem interfaces. In this view, it is up to the subcontractor to specify the subsystem parameters. The resource model introduced in [23] require only 2 subsystem parameters (period and budget). This is good because it decouples the OEM and the subcontractor in a higher degree.

Future work includes implementation of HSF and synchronization protocols (SIRAP and HSRP) in a OSEK/VDX compliant OS. A continuation of this could be an implementation in an AUTOSAR OS. Additional future work can hopefully lead to a cooperation with an AUTOSAR premium member, such as Volvo. This has the potential to open up the possibility of integrating HSF fully into the AUTOSAR infrastructure, and also to conduct real simulations and tests.

References

[1] OSEK VDX Portal. OSEK/VDX Operating System, February 2005. http://portal.osek-vdx.org/files/pdf/specs/os223.pdf.

[2] AUTOSAR GbR. Explanation of Interrupt Handling in AUTOSAR, August 2008. http://www.autosar.org/download/specs_aktuell/AUTOSAR_InterruptHandling_Explanation.pdf.

[3] AUTOSAR GbR. Specification of BSW Scheduler, August 2008. http://www.autosar.org/download/specs_aktuell/AUTOSAR_SWS_BSW_Scheduler.pdf.

[4] AUTOSAR GbR. Specification of the Virtual Functional Bus, August 2008. http://www.autosar.org/download/specs_aktuell/AUTOSAR_SWS_VFB.pdf.

[5] AUTOSAR GbR. Specification of Operating System, February 2009. http://www.autosar.org/download/specs_aktuell/AUTOSAR_SWS_OS.pdf.

[6] AUTOSAR GbR. Specification of RTE, February 2009. http://www.autosar.org/download/specs_aktuell/AUTOSAR_SWS_RTE.pdf.

[7] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. J. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8:284–292, 1993.

[8] M. Behnam, T. Nolte, M. Åsberg, and I. Shin. Synchronization protocols for hierarchical real-time scheduling frameworks. In *CRTS'08*, November 2008.

[9] M. Behnam, T. Nolte, I. Shin, M. Åsberg, and R. J. Bril. Towards hierarchical scheduling on top of vxworks. In *OSPERT'08*, July 2008.

[10] M. Behnam, I. Shin, T. Nolte, and M. Nolin. Sirap: A synchronization protocol for hierarchical resource sharing in real-time open systems. In *EMSOFT'07*, Oct. 2007.

[11] R. Bosch. *AUTOMOTIVE BRAKE SYSTEMS*. Warrendale, 1995.

[12] R. I. Davis and A. Burns. Resource sharing in hierarchical fixed priority pre-emptive systems. In *RTSS'06*, Rio de Janeiro, Brazil, December 2006.

[13] H. Heinecke, W. Damm, B. Josko, A. Metzner, H. Kopetz, A. Sangiovanni-Vincentelli, and M. D. Natale. Software components for reliable automotive systems. In *DATE'08*, pages 549–554, New York, NY, USA, 2008. ACM.

[14] P. Hladik, A. Dplanche, S. Faucou, and Y. Trinquet. Adequacy between autosar os specification and real-time scheduling theory. In *SIES'07*, pages 225–233, 2007.

[15] C. Li and Z. Jianwu. Wcet analysis for gasoline engine control. pages 2090–2095 vol. 4, July-1 Aug. 2005.

[16] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *ACM*, 20(1):46–61, 1973.

[17] AUTOSAR Partnership. http://www.autosar.org.

[18] TIMMO research project. http://www.timmo.org.

[19] R. Racu, R. Ernst, and K. Richter. The need of a timing model for the autosar software standard. In *WMAAS'06*, Rio de Janeiro, December 2006.

[20] R. Racu, A. Hamann, R. Ernst, and K. Richter. Automotive software integration. In *DAC'07*, pages 545–550, NY, USA, 2007. ACM.

[21] S. Rezeka. Modeling and sensitivity analysis of an abs hydraulic modulator. pages 826–830, June-1 July 1994.

[22] I. Shin, M. Behnam, T. Nolte, and M. Nolin. Synthesis of optimal interfaces for hierarchical scheduling with resources. In *RTSS'08*, December 2008.

[23] I. Shin and I. Lee. Periodic resource model for compositional real-time guarantees. In *RTSS'03*, Mexico, Dec. 2003.

[24] I. Shin and I. Lee. Compositional real-time scheduling framework with periodic model. *ACM*, 7(3):(30)1–39, April 2008.