Applying Holistic Distributed Scheduling to AUTOSAR Methodology

Ahmed DAGHSEN, Khaled CHAABAN, Sébastien SAUDRAIS, Patrick LESERF

ESTACA, Rue Georges Charpak BP 76121 53061 Laval Cedex 9

Abstract: AUTOSAR initiative continues to envoy success in the automotive software domain by providing a common framework for efficient software development and by the integration and reuse of software components. However, additional work is needed to consider non-functional properties in the development cycle. Recent release of AUTOSAR has defined a common language to define timing-related information for the automotive embedded system across all development layers.

In this paper, we propose an approach to consider the AUTOSAR timing model. Then we aim to transform it to a classical scheduling model in order to apply directly fundamentals scheduling theories for timing analysis. The approach is applied on a realistic case study, a steer-by-wire system.

Keywords: real-time scheduling, AUTOSAR, timing analysis.

1. Introduction

Nowadays, the number of software and processor embedded in vehicles is growing fast. The diversity of components providers also increase the complexity of their integration and do not ease the reuse of these components because of their heterogeneity. To answer this problem, AUTOSAR (AUTomotive Open System ARchitecture) has been introduced by automobile manufacturers, suppliers and tool developers as the future standard of automotive Electricals/Electronics (E/E) engineering. By breaking up the cohesion between hardware infrastructure and the application software, embedded automotive system complexity can be managed and software reuse is promoted [4]. AUTOSAR development methodology is based on a model-driven development style. The software architecture, as well as the Electronic Control Unit (ECU) hardware and the network topology, are modelled in a formal way defined in a metamodel supporting the software development process from architecture up to integration. All available modelling elements are specified by the "AUTOSAR metamodel" [4].
Moreover, many AUTOSAR applications are considered as time-critical or at least time-dependent. Thus, precise timing and prioritisation of functions are essential for both safety and comfort of in-vehicle applications and also for the continuous deployment of the standard within the automotive industry. Many improvements and extensions to the current AUTOSAR system model have been developed recently to handle all timing-related information during the development process [4]. Thus, complexity and development cost cycle are reduced significantly while reliability is improved. Furthermore, AUTOSAR allows an easy integration of timing information using existing system's software model and hardware topology following a model-driven approach. However, there are few works that use these timing properties and constraints to make a global timing analysis of the system. Timing analysis means verifying if the given timing properties fulfil the given timing constraints. We can distinguish between local and global timing analysis. Where local timing analysis addresses tasks scheduling regarding a processor or an ECU, global scheduling considers the global distributed system where communication bus and gateways must be analysed together with ECUs tasks.

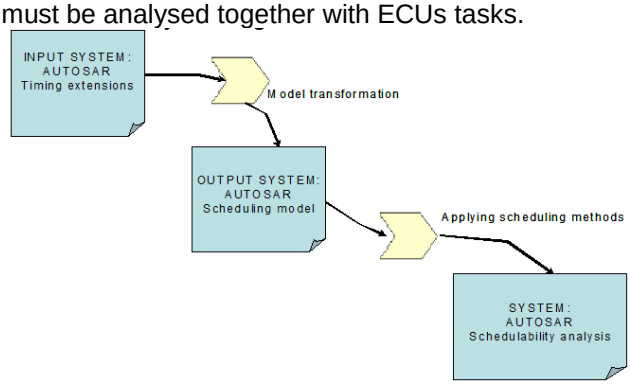

Figure 1 : using AUTOSAR timing model to apply scheduling analysis.

In this paper we propose a holistic scheduling analysis of the distributed system. The proposed approach is illustrated in two main steps in the Figure **1**: first applying a model transformation to convert AUTOSAR timing model to a scheduling model, then applying scheduling techniques for timing analysis of the whole AUTOSAR system. We apply the method to a case study, a steer-by-wire application, and simulate it using TrueTime Matlab toolbox [6].

The remaining of the paper is organized as followed. Section 2 presents an overview of AUTOSAR methodology and scheduling analysis. Section 3 presents the system model and explains the application of the scheduling approach to AUTOSAR. Section 4 shows the steer-by-wire case study. Section 5 concludes and presents future works.

2. AUTOSAR

We present by this section a short description of AUTOSAR methodology and then some related works concerning real-time scheduling in AUTOSAR.

2.1 Methodology

According to AUTOSAR approach, the development process of an E/E system has the general structure shown in Figure **2**.

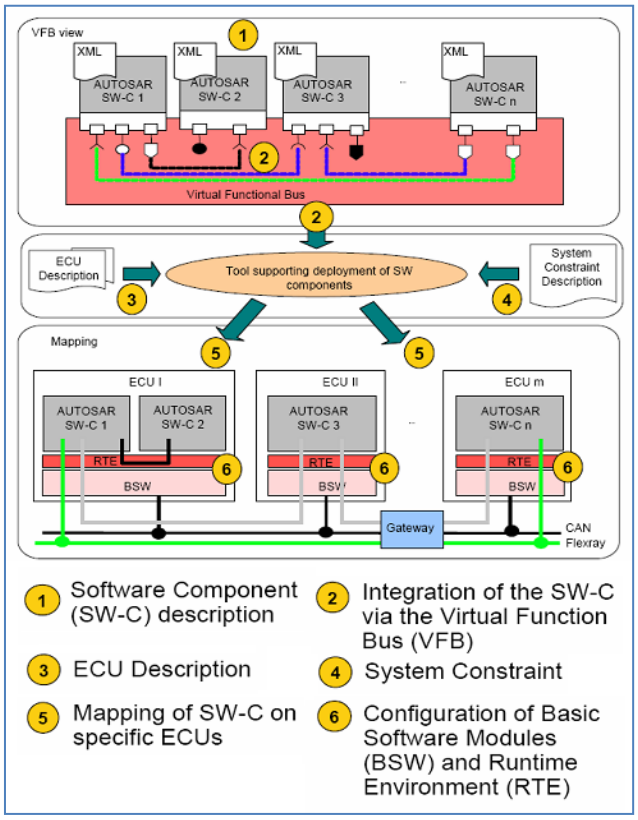

Figure 2 : AUTOSAR Methodology

Step 1: the first step in AUTOSAR development process is to define the set of software components (SWCs) constituting the user software applications. SWCs communicate using ports through their interfaces. According to release 4.0 of AUTOSAR, an interface may be of sender/receiver, client/server or calibration type. Further, a SWC may be one of the three types: sensor/actuator, application or calibration type.

Now that the SWCs types and external interfaces are defined, we must describe the internal behaviour of each SWC. The internal behaviour decomposes a SWC into runnable entities which represents the C code that will be executed at runtime.

A runnable is triggered using an event. An event may be of timing or data type. Timing event runnables are triggered periodically when data event runnables are triggered at the arrival pattern of data from connected sensors or other components of the system (inter ECUs). Runnables exchange data via specific AUTOSAR variables called inter-runnables variables instead of global variable that is prohibited in AUTOSAR specifications.

Step 2: at the Virtual Functional Bus (VFB) level, SWCs are defined without consideration of the underlying hardware on which these SWCs will run on later. So, two software components might run on the same ECU or on different ECUs and this is completely transparent to software developers. The communication between the components is then either an intra-ECU communication or an inter-ECU communication and is routed via the VFB bus which allows a virtual integration of the system independently of underlying software and hardware.

Steps 3, 4, 5: now that the SWCs and their internal behaviours are defined at the high level, we must proceed by the mapping of SWCs to available ECUs. This phase requires some information about system and ECU constraints. For example, an application SWC may be mapped on any ECU. In contrast sensor and actuator SWCs are bound to a particular ECU where the sensor or the actuator is connected to. At this stage, we may have also some engineering constraints (e.g. ECU load, SWCs resources consumption, ECU hardware requirements, etc.) that may influence the mapping operation.

Step 6: after the mapping of SWCs to ECUs, we can proceed by the development and integration of each ECU. The software architecture of an ECU is composed of three main layers: (1) the SWCs layer, (2) the Run Time Environment (RTE) layer and then (3) the Basic Software (BSW) layer.

The SWCs contain the application's functional code. RTE represents an instance of the VFB bus per ECU. It is the "glue code" between the application SWCs and the BSW layer. It provides standardized interfaces to communicate with the BSW layer and to communicate between SWCs themselves. Data exchange between SWCs themselves and between SWCs and the underlying BSW layer is performed exclusively via RTE. Thus, depending on SWCs locations, the RTE allows data exchange either directly via a shared memory or by sending messages via a network bus.

BSW layer makes the link between RTE layer and all hardware features of the microcontroller (µC). BSW is composed of 80 modules abstracted by 3 layers: the service layer, the ECU abstraction layer and the µC abstraction layer. The service layer provides µC

and ECU independent services like Operating System (OS) and communication services.

2.2 Scheduling analysis

There are few works dealing with the exploitation of the AUTOSAR timing extensions for timing analysis. In the scope of TIMMO project, [3] [4] give a general framework for the relation between AUTOSAR concepts and timing constraints. They have proposed also an extension of AUTOSAR standard towards the possibility to specify the system's timing constraints. Thus, a scheduling analysis of an AUTOSAR application can be performed at the low-level. But the resulting task timing reveals hardly any direct and intuitive timing-relation with high-level software components to which timing information shall finally be attached. Another mismatch at the communication bus level is the lack of rules describing the AUTOSAR tasks due to the absence of clear rules describing the activation of the tasks within the software components. At the bus communication level, the frame generation modes and the buffering strategy complicate the timing behavior of the transmitted frames and implementation dependency.

Many real-time scheduling theories have been developed recently in the field of embedded systems. In this paper, we present a method of applying scheduling works of J. Sun et al. [2] to AUTOSAR methodology. The proposed method deals with an end-to-end approach to schedule tasks that share resources in a distributed system. The method considers only static scheduling analysis of tasks with fixed priorities and deadlines.

A task in a distributed real-time system is called an end-to-end task if it consists of a chain of subtasks and has an end-to-end deadline. Each subtask is assigned a proper priority and its worst-case response time can be bounded. From the end-to-end scheduling point of view, a task that needs remote resources is viewed as a chain of subtasks in the following way. Each critical section associated with a remote resource is a subtask executed on the synchronization processor of the remote resource. A segment that requires no resources is also considered as a subtask and this subtask is executed on the local processor. The schedulability analysis of this approach is of four steps: first step is the mapping of a given task set to an end-to-end task. Second step is the assignment of priorities to subtasks composing the end-to-end task. Several methods can be used to assign priorities like the global-deadline-monotonic and effective-deadline-monotonic assignment. Third step is to determine the worst case response time for each subtask. The worst case of a subtask is thus determined by applying a specified equation and using tasks model. From the results obtained in the previous steps, the worst case response time for a parent task or end-to-end task is obtained by adding the response times of its subtasks. Thus if the sum obtained was less than the relative deadline of this task in all the parent tasks, then the system is schedulable using this algorithm. Let's note that it is sufficient to have only one task of the parent tasks which doesn't fulfill the condition of schedulability to conclude that the whole system will not be schedulable.

3. Applying scheduling approach to AUTOSAR

We present in this section the system model assumptions and then we show how to apply it to AUTOSAR.

3.1 System model

In this part, we describe the end-to-end system model used as the basis of the work. As defined before, a task in a distributed real-time system is called an end-to-end task if it consists of a chain of subtasks and has an end-to-end deadline. We call a real-time system an end-to-end system if it consists of more than one processor and a set of end-to-end tasks [2]. The system model is defined as follows:

- The system consists of a set $\{P_i\}$ of processors and a set $\{T_i\}$ of tasks.
- Each task T_i consists of a chain of n_i subtasks; $T_{i,1}$, $T_{i,2}$, …,$T_{i,n}$. T_i is referred as the parent task to its subtasks and subtasks are referred to as sibling subtasks to each other if they have the same parent task.
- Each request for execution of a subtask is called an instance of that subtask and the corresponding instances of all subtasks are collectively called an instance of their parent task.
- Subtask $T_{i,j}$ is a predecessor (successor) of subtask $T_{i,k}$ if $j < k$ ($j > k$), and $T_{i,j}$ is the immediate predecessor (successor) of $T_{i,k}$ if they are also adjacent ($| j-k | = 1$).
- Each task T_i is a periodic task with a period pi. For simplicity, we will consider that all subtasks of T_i are of the same period.
- The release time of the first instance of $T_{i,1}$ is the phase f_i of task T_i .
- An instance of $T_{i,j}$ cannot start to execute before the complete execution of $T_{i,j-1}$.
- Each task T_i has a relative end-to-end deadline Di.
- Each subtask $T_{i,j}$ has a maximum execution time $Y_{i,j}$ and a fixed priority $O_{i,j}$.
- Subtasks are statically assigned to processors.

The system model imposes strong restrictions on tasks properties. So the following assumptions are considered in our system model.

We consider both preemptive and non-preemptive tasks and subtasks. We also assume the establishment of a common time base for all processors in the network. We neglect the jitter of

periodic tasks and we also consider that the system subtasks and message subtasks are synchronized.

3.2 The algorithm

The PTTDF (Per Task Time Demand Function) algorithm allows computing the tighter upper bounds on the response times of the end-to-end tasks in static systems. The structure of the PTTDF algorithm is defined as follows:

1. **Require :** maintain a task set $\{T_i\}$ with period p_i and execution time t_i Maintain a subtask set $\{T_{i,j}\}$ associated with the task $\{T_i\}$. Each subtask $\{T_{i,j}\}$ has the execution time $t_{i,j}$, the priority $o_{i,j}$ and the processor $T_{i,j}$ executes on.
2. For each subtask $T_{i,j}$:
   a. Compute $H_{i,j}$ and $N_{i,j}$;
   b. For every task in H_{ij} compute the PTTD function $M_k^{i,j}(t)$;
   c. Compute $SH_{i,j}$ and $\Delta_{i,j}$;
   d. Compute $W_{i,j}(t)$;
   e. Compute $C_{i,j}$ using the iteration method.

For each task T_i, calculate $C = \sum_{i=1}^{ni} C_{i,j}$

For each subtask $T_{i,j}$, $H_{i,j}$ denotes the set of subtasks that are on the same processor as $T_{i,j}$, are of different parent tasks and have priorities higher than or equal to $T_{i,j}$. Let $N_{i,j}$ denote the set of tasks that have subtasks in $H_{i,j}$.

The PTTD function $M_k^{i,j}(t)$ is an upper bound of $M_k^{i,j}(t_0, t)$ for any time instant t_0. This upper bound is:

$$M_k^{i,j} = \sum_{T_{k,l} \epsilon H_{i,j}} \left\lceil \frac{t}{p_k} \right\rceil t_{k,l} \qquad [1]$$

The first equation, Eq.[1], gives a loose bound of $M_k^{i,j}(t_0, t)$ with k the index of parent tasks of $H_{i,j}$ subtasks.

Let $SH_{i,j}$ denote the set of subtasks each of which is a sibling subtask of $T_{i,j}$, executes on the same processor as $T_{i,j}$, and has priority higher than or equal to $T_{i,j}$. $\Delta_{i,j}$ is equal to the sum of the execution times of the subtasks in $SH_{i,j}$:

$$\Delta_{i,j} = \sum_{T_{i,k} \epsilon SH_{i,j}} t_{i,k} \qquad [2]$$

The time demand function is given by this equation:

$$W_{i,j}(t) = t_{i,j} + \Delta_{i,j} + \sum_{T_k \epsilon N_{i,j}} t_{i,k} \qquad [3]$$

The upper bound $C_{i,j}$ of the response time of T_{ij} is the first time instant when the time demand $W'_{i,j}$ is met by time supply t:

$$C_{i,j}(t) = \min\{t > 0 | t = W_{i,j}(t)\} \qquad [4]$$

$C_{i,j}$ is calculated using an efficient iterative method as follows:

$$S_k = W_{i,j}(S_{k-1}) \qquad [5]$$

With $S_0 = W_{i,j}(0)$ and $C_{i,j}$ should be ≤ p_{ij} .

Sometimes this iterative method does not converge after a finite number of iterations then we put $C_{i,j}$ to ∞ and by consequence the system is not schedulable.

3.3 Incorporating algorithm to AUTOSAR

As mentioned before, the approach presented in this paper is to perform a transformation of AUTOSAR timing properties and constraints into a complete scheduling model. By using this model, we can apply directly existing scheduling theories to the AUTOSAR application.

Let's recall that an AUTOSAR system consists of software components (SWCs) communicating with one another and interacting on a Virtual Functional Bus (VFB). SWCs are then mapped to specific control units (ECU) distributed over a network. Recall also that an end-to-end task is composed of multiple subtasks running on multiple processors. This task is the chain of subtasks which are subject to precedence constraints. We suppose that a task is subject to an end-to-end deadline and we don't care about the response time of a particular subtask.

In AUTOSAR context, an end-to-end AUTOSAR task corresponds to the activity from the reception of the data on the R-Port (required port) of a sensor SWC to the P-Port (provider port) of the actuator SWC. This end-to-end-task can be executed on different SWCs that belong to the same ECU or on different ECUs using communication bus.

Moreover, we consider in our model that each sensor or actuator SWC contains only one runnable; while an application SWC may contain several runnables. Each runnable in our model corresponds to a subtask of the end-to-end AUTOSAR task. Furthermore, the runnables inside an application SWC have precedence constraints. Each ECU represents one processor and the communication bus represents a link processor. Each subtask executing on the bus is a message transmitted by a given ECU to another one. The transmission of each message is modeled as a "message" non-preemptive subtask on the link processor (e.g. FlexRay). The maximum execution time of a "message" subtask is equal to the maximum time needed to deliver the message when it is alone on the bus. Thus the execution time of each subtask on the bus is known.

The model considers also the delays introduced by the communication between high-level SWCs and the underlying BSWs and RTE by modeling them as subtasks. The time delay taken from the start to the end of a runnable execution is also supported by the model.

Now that the scheduling model of the AUTOSAR application has been constructed from the system timing information, we can apply directly the scheduling algorithm of J. Sun et al in order to verify

the system schedulability and to compute worst end-to-end delays.

AUTOSAR model	System model
Timing chain	End-to-end task
Subchain	Subtask
ECU	Processor
Communication bus	Link processor
Latency	Release time
Runnable	Subtask

Table 1: relationship between scheduling system model and AUTOSAR 4.0 concepts

Table **1** illustrates some relationship between scheduling system model and the AUTOSAR release 4.0 one.

4. Case study

In this section, we show how to apply the scheduling algorithm previously presented on a steer-by-wire system developed using the AUTOSAR methodology.

4.1 System description

As depicted in Figure **3**, a basic steer by wire system is composed of three main blocks: the hand wheel (i.e. steering), controllers and the road wheels.

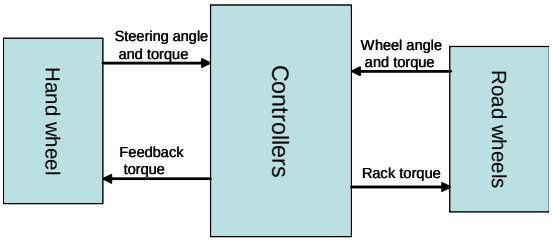

Figure 3: basic architecture of a steer-by-wire system

When the driver operates the hand wheel to turn the vehicle, a steering angle signal will be sent to the controller. Two kinds of sensors are necessary to acquire the steer angle and the torque applied by the driver. The controllers will process all acquiring signals and also perform some control functions associated with the vehicle's steering function and output an actuator angle for the road wheels that in turn will turn the wheels through an actuator. The feedback signals (actuator feedback and wheel feedback) involve some kind of force or torque sensors and are necessary so that the driver get the feeling of turning a traditional steering wheel and feel the effect of turning the wheels on a certain type of road.

The Steer-By-Wire system may be composed of two main functions: (1) the feedback torque function and (2) the rack torque function (Figure **4**).
The feedback torque function is essential for the system operation. It computes the feedback force applied to the steering wheel so that the driver feels the effect of tuning the wheels on a certain type of road.

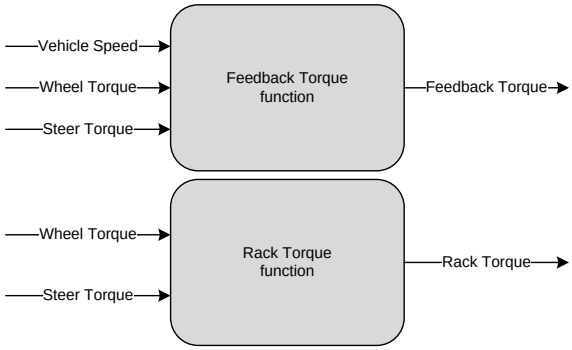

Figure 4: two main functions of the steering system

The rack torque function is the main system function that permits to control the front axle actuator.
The distributed steer-by-wire architecture involves several components: ECU's, communication lines (e.g., FlexRay bus) and appropriate sensors and actuators.
Figure **5** illustrates the implementation of the rack torque function according to AUTOSAR approach. At the VFB level, the signal path involves four components. The "Steer Sensor" component acquires the sensor physical data and passes it to the application software component "Steer Manager" for treatment. Afterwards the signal is sent to the application software component "Wheel Manager" for order computation until it is finally send to the actuator via the "Wheel Actuator" component.
At the system level, we map SWCs to available ECUs and then we configure RTE and BSW modules. In our case study we have only two ECUs, steer ECU and wheel ECU.

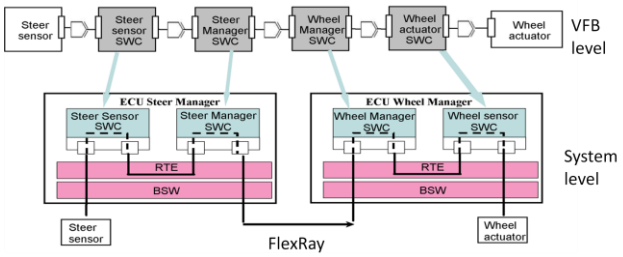

Figure 5: implemntation of the rack torque function in AUTOSAR

4.1 Applying scheduling algorithm

In order to apply the scheduling algorithm to our case study, each function is represented as an end-to-end task. So, we have only two end-to-end tasks. Note that in AUTOSAR, and end-to-end task passes by 3 stages: from hardware to software represented by the transformation of data from the physical sensor to the sensor SWC (e.g. steer sensor or wheel sensor SWC), the second stage is all the actions that pass between the sensor SWC till the

software control represented by the actuator SWC. The last stage is the interface that is done between the actuator SWC and the physical actuator (as shown in **Erreur ! Source du renvoi introuvable.**).
Let's note T_1 as the rack torque end-to-end task and T_2 as the feedback torque end-to-end task:

T_1: has 17 subtasks: $T_{1,1}$, $T_{1,2}$, $T_{1,3}$, $T_{1,4}$ …$T_{1,17}$.
T_2: has 17 subtasks: $T_{2,1}$, $T_{2,2}$, $T_{2,3}$, $T_{2,4}$ … $T_{2,27}$.
Each end-to-end timing chain segment in AUTOSAR model corresponds to a subtask.

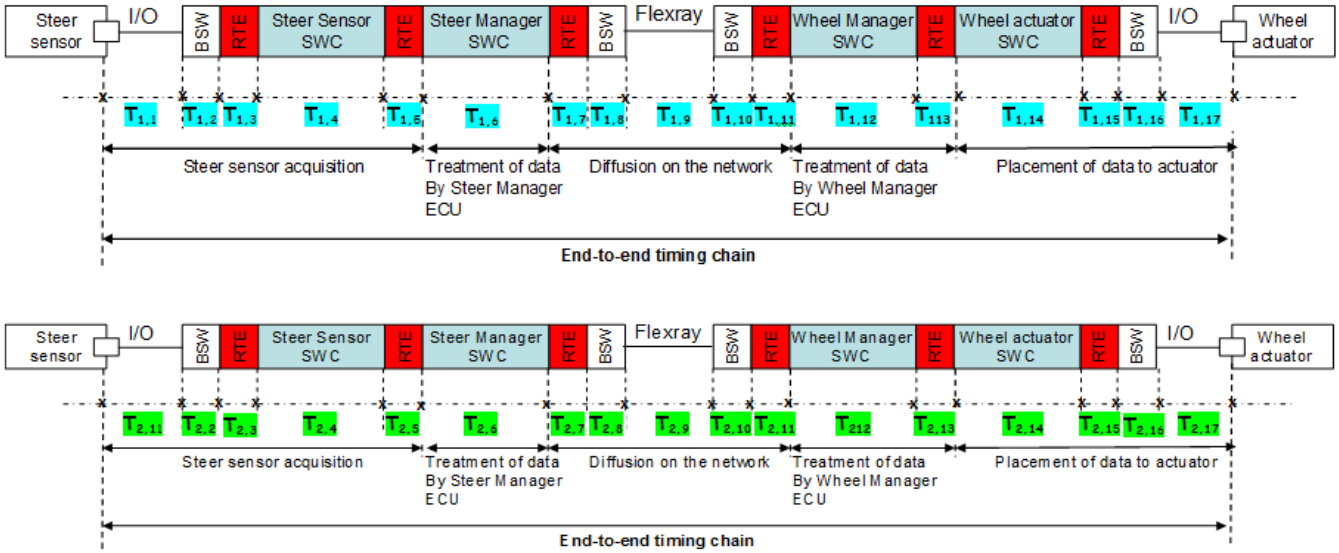

Figure 6: End-to-end timing representation of AUTOSAR methodology

Subtasks of each function are executed on a specific ECU. As noted above, we have two processors (P_1 at the steer side and P_2 at the wheel side) and the communication bus represents a link processor P_3.
The table below summarizes the two tasks T_1 and T_2 with their subtasks properties. Priorities are fixed with respect of precedence constraint.

T_1					T_2				
$T_{i,j}$	Proc	$O_{i,j}$	$p_{i,j}$	$t_{i,j}$	$T_{i,j}$	Proc	$O_{i,j}$	$p_{i,j}$	$t_{i,j}$
$T_{1,1}$	P_1	1	50	1	$T_{2,1}$	P_3	20	50	1
$T_{1,2}$	P_1	2	50	1	$T_{2,2}$	P_3	21	50	1
$T_{1,3}$	P_1	3	50	1	$T_{2,3}$	P_3	22	50	1
$T_{1,4}$	P_1	4	50	4	$T_{2,4}$	P_3	23	50	4
$T_{1,5}$	P_1	5	50	2	$T_{2,5}$	P_3	24	50	2
$T_{1,6}$	P_1	6	50	4	$T_{2,6}$	P_3	25	50	4
$T_{1,7}$	P_1	7	50	1	$T_{2,7}$	P_3	26	50	1
$T_{1,8}$	P_1	8	50	1	$T_{2,8}$	P_3	27	50	1
$T_{1,9}$	P_2	8	50	8	$T_{2,9}$	P_2	27	50	8
$T_{1,10}$	P_3	8	50	1	$T_{2,10}$	P_1	27	50	1
$T_{1,11}$	P_3	9	50	1	$T_{2,11}$	P_1	28	50	1
$T_{1,12}$	P_3	10	50	4	$T_{2,12}$	P_1	29	50	4
$T_{1,13}$	P_3	11	50	2	$T_{2,13}$	P_1	30	50	2
$T_{1,14}$	P_3	12	50	4	$T_{2,14}$	P_1	31	50	4
$T_{1,15}$	P_3	13	50	1	$T_{2,15}$	P_1	32	50	1
$T_{1,16}$	P_3	14	50	1	$T_{2,16}$	P_1	33	50	1
$T_{1,17}$	P_3	15	50	1	$T_{2,17}$	P_1	34	50	1

Applying the algorithm:
Here we show just some results for each step of the algorithm:

a) For each subtask $T_{i,j}$ we compute $H_{i,j}$ and $N_{i,j}$:
For example we have:
$H_{1,1}= \{\}$; (idem for all subtasks of T_1)
$H_{2,1} = \{T_{1,10}, T_{1,11}, T_{1,12}, T_{1,13}, T_{1,14}, T_{1,15}, T_{1,16}, T_{1,17}\}$;
And $N_{1,1}= \{\}$; (idem for all subtasks of T_1)
$N_{2,1} = \{T_1\}$; (idem for all subtasks of T_2)

b) For every task in $H_{i,j}$ we compute the Per-Task Time Demand (PTTD) function $M_k^{i,j}(t)$:
For example:
$M^{1,1}(t) = 0$; (idem for all subtasks of T_1)

$$M_1^{2,1}(t) = \frac{t}{50} + \frac{t}{50} + \frac{4\,t}{50} + \frac{2t}{50} + \frac{4\,t}{50} + \frac{t}{50} + \frac{t}{50} + \frac{t}{50} = \frac{15\,t}{50};$$

c) For each subtask T_{ij} we compute $SH_{i,j}$ and $\Delta_{i,j}$. As an example, for task T_1 we have:
$SH_{1,1}= \{\}$; $SH_{1,7}= \{T_{1,10}, T_{1,11}, T_{1,12}, T_{1,13}, T_{1,14}, T_{1,15}, T_{1,16}\}$;
$\Delta_{1,1}=0$; $\Delta_{1,17}=14$
And for task T_2 we have:
$SH_{2,1}= \{\}$; $SH_{2,7}= \{T_{2,10}, T_{2,11}, T_{2,12}, T_{2,13}, T_{2,14}, T_{2,15}, T_{2,16}\}$;
$\Delta_{2,1}=0$; $\Delta_{2,7}=14$

d) For each subtask $T_{i,j}$ we compute $W'_{i,j}(t)$ given by Eq.[3]:
For example, for task T_1 we have:
$W_{1,1}(t) = 1$; $W_{1,17}(t) = 15$;
And for task T_2 we have:
$W_{2,1}(t) = 1 + \frac{15t}{50}$; $W_{2,17}(t) = 15 + \frac{15t}{50}$;

e) We compute $C_{i,j}$ using the iterative method based on Eq.[5]:
Computing C_{11}:

$$S_0 = W_{1,1}(0) = 1; S_1 = W_{1,1}(S_0) = 1;$$
$$S_2 = W_{1,1}(S_1) = 1;$$

$C_{1,1}$ is the sum of the terms of a geometric progression. In this case the progression is a constant sequence. Then $C_{1,1} = 1$.
Similarly we obtain $C_{1,2}=2,\ldots,C_{1,17}=15$
Computing $C_{2,1}$:
$S_0 = W_{2,1}(0) = 1;$
$S_1 = W_{2,2}(1) = 1+\frac{15}{50}$; $S_2 = W_{2,3}(1+\frac{15}{50}) = 1+\frac{15}{50}(1+\frac{15}{50}) = 1+\frac{15}{50}+\frac{15}{50}*\frac{15}{50}$;
$C_{2,1}$ is the sum of the terms of a geometric progression with first term 1 and common ratio $\frac{15}{50}$.
Then $C_{2,1} = \frac{1}{1-\frac{15}{50}} = 1.42$.
Hence C_1 = 1+2+…+15 = 133 > 50 then T_1 is not schedulable.
And C_2 = 1.42 + 2.84 +…21.3= 196.6 >> 50 then T_2 is not schedulable. Finally, the system is not schedulable.

The obtained bounds are very pessimist since the used algorithm does not take into account the precedence constraints of subtasks when computing PTTD function $M_k^{i,j}(t)$ which is the sum of all subtasks' execution time in $H_{i,j}$.

According to [2], an improvement of computing the PTTD function is possible in order to obtain tighter bounds of $M_k^{i,j}(t)$ and thus tighter bounds C_i of the response time of T_i.

The PTTD function $M_k^{i,j}(t)$ computed by Eq.[1] assumes that the subtasks of each task in $N_{i,j}$ are independent, but the actual time demand may be less than the sum in the right hand side of Eq.[1] because of the precedence constraints among subtasks. Then the PTTD function can be considered as the maximum of $\{M_{k,l}^{i,j}(t)\}$ for all $T_{k,l}$ in $H_{i,j}$.

Applying this modification, we obtain a new value of the upper bound C_2 =155.32, which is tighter than the first one. Despite this improvement, the upper bound $C_{i,j}$ still very pessimistic. Another improvement axe may be the calculation of an end-to-end response time of each subtask and then to consider the response time of the last subtask of an end-to-end task.

4.2 Test by simulation

Using TrueTime tool [6] we simulate the architecture of the steer-by-wire system. The objective of the simulation is to verify the adequacy between the scheduling results and the simulation one.

TrueTime is a Matlab toolbox that permits to simulate controllers interconnected by a network. It contains two main blocks: kernel block and network block.

Kernel block allows the simulation of a complete controller with full OS (Operating System) primitives, e.g. preemptive/non-preemptive tasks, overheads, execution time, priority-driven scheduling, etc. It allows also the interaction with the external environment using I/O ports.

Network block simulates the network of the system. It supports several communication protocols (CAN, Ethernet, TDMA, etc.).

TrueTime permits to take into account the clocks drifts between networked-computers. But in our simulation, we consider a global time base for all networked nodes.

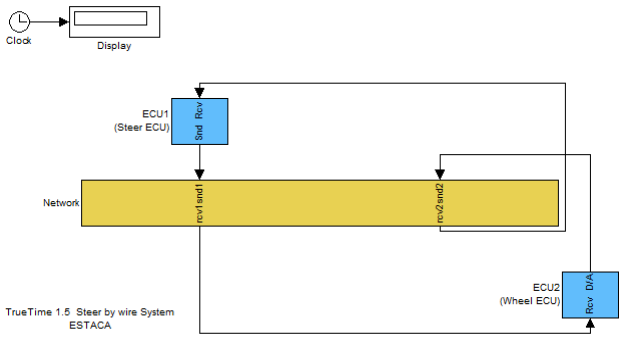

Figure 7: true time model of the steer by wire system

As shown in Figure 7, the steer-by-wire system is composed of two kernel blocks: wheel ECU and steer ECU and one network block (for the link processor).
For kernel blocks, we consider a preemptive OS with priority-driven scheduling. For the network block we use a TDMA protocol which is suited for such critical application.

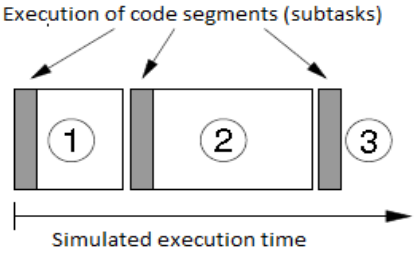

Figure 8: sequence of segments code

In TrueTime simulator, the execution code of a task may be divided on several segments code (Figure 8). We use this feature in order to implement each subtask of our system model on a segment code. Subtasks of the same task have the same period and priorities are assigned respecting precedence constraint.
Using periodic tasks, we are able to simulate the system at all levels, e.g. BSW, RTE, SWC, etc. Sensor and actuator are also simulated using dedicated periodic tasks. We neglect the overhead of the OS scheduler and we consider that the execution time of each subtask is given *a priori* as well as its deadline and phase.

Now that the set of tasks is defined, we can perform a simulation of the system. The expected results of the simulation correspond to a Rate Monotonic Analysis (RMA) analysis of the system. Thus, we have obtained 38 ms of end-to-end delay for tasks T1 and T2.

5. Conclusion

The support of timing requirements in AUTOSAR has received a wide attention recently. The main goal is to perform an early verification and analysis of the system performance at the design level and before implementation. We have proposed in this paper an approach to transform an AUTOSAR timing model to a scheduling model. By this transformation, we can apply directly the scheduling techniques from the real-time system community to the AUTOSAR system. In this paper, we have applied an end-to-end scheduling algorithm proposed by J. SUN in order to bound end-to-end latencies. We have shown how to apply it on a steer-by-wire system. However, the proposed algorithm computes loose bounds of end-to-end latencies and it makes several simplifications on the system model. A more accurate system model must be considered in order to reflect a realistic AUTOSAR system. Moreover, more accurate and optimized real-time scheduling techniques may be adapted to our approach such as [7] or [8].

6. References

[1] K. Chaaban, P. Leserf, S. Saudrais: “Steer-By-Wire System Development Using AUTOSAR Methodology”, ETFA, SPAIN, 2009.

[2] J. Sun, J. Liu, R. Bettat: “An End-To-End Approach to Schedule Tasks with Shared Resources in Multiprocessor Systems”, Department of Computer Science- University of Illinois.

[3] O. Scheickl, M. Rudorfer: “Automotive Real Time Development Using a Timing-augmented AUTOSAR Specification”. BMW Car IT, Munich.

[4] K. Richter, R. Racu, R. Ernst: “The Need of a Timing Model for the AUTOSAR software standard”, Braunschweig, Germany.

[5] S. Fürst et al.: “AUTOSAR – A Worldwide Standard is on the Road”, 14th International VDI Congress Electronic Systems for Vehicles 2009, Baden-Baden.

[6] A. Cervin, D. Henriksson, B. Lincoln, J. Eker, K. Årzén: "How Does Control Timing Affect Performance? Analysis and Simulation of Timing Using Jitterbug and TrueTime". IEEE Control Systems Magazine, 23:3, pp. 16--30, June 2003.

[7] K. Tindell: “Adding time-offsets to schedulability analysis”. Technical Report YCS 221, Department of Computer Science, University of York, UK, 1994.

[8] J. C. Palencia and M. G. Harbour: “Exploiting precedence relations in the schedulability analysis of distributed real-time systems”. In Proceedings of the IEEE Real-Time Systems Symposium, pages 328–399. IEEE Computer Society, 1999.