Schedulability Analysis for Distributed Heterogeneous Time/Event Triggered Real-Time Systems

Traian Pop, Petru Eles, Zebo Peng
Department of Computer and Information Science, Linköping University, Sweden
{trapo,petel,zebpe}@ida.liu.se

Abstract

This paper deals with specific issues related to the design of distributed embedded systems implemented with mixed, event-triggered and time-triggered task sets, which communicate over bus protocols consisting of both static and dynamic phases. Such systems are emerging as a new standard for automotive applications. We have developed a holistic timing analysis and scheduling approach for this category of systems. Three alternative scheduling heuristics are presented and compared. We have also identified several new design problems characteristic to such hybrid systems. An example related to bus access optimization in the context of a mixed static/dynamic bus protocol is presented. Experimental results prove the efficiency of such an optimization approach.

1. Introduction

Embedded systems very often have to satisfy strict timing requirements. In the case of such hard real-time applications, predictability of the timing behaviour is an extremely important aspect. Frequently, such applications are implemented as distributed systems. This is the case, for example, with many applications in the automotive industry. Predictability of such a system has to be guaranteed globally, considering both the task schedules determined for the particular processing units as well as the timing of the communication between different components of the system.

Task scheduling and schedulability analysis has been intensively studied for the past decades. The reader is referred to [2],[3] for surveys on this topic.

A few approaches have been proposed for a holistic schedulability analysis of distributed real-time systems, taking into consideration both task and communication scheduling. In [23], Tindell provided a framework for holistic analysis of event-triggered task sets interconnected through an infrastructure based on a generic TDMA protocol. In [15], the authors improve Tindell's analysis by allowing dynamic task offsets, which produced tighter bounds for task response times. Work in the area of scheduling and schedulability analysis diversified significantly by considering particular communication protocols, like the Token Ring protocol [20][22], the ATM protocol [8][11], CAN bus [7][24], or TTP bus[12]. In [17] and [18] we have developed a holistic analysis allowing for either time-triggered or event-triggered task sets communicating over a TTP bus. In addition to schedulability analysis, this work has also addressed the optimization of the TTP based bus configuration in order to fit the particular application.

Two basic approaches for handling tasks in real-time applications can be identified [13]: the event-triggered (ET) and time-triggered (TT). There has been a long debate in the real-time and embedded systems communities concerning the advantages of each approach and which one to prefer [1], [13], [26]. Several aspects have been considered in favour of one or the other approach, such as flexibility, predictability, jitter control, processor utilization, testability, etc.

The same duality is reflected at the level of the communication infrastructure, where communication activities can be triggered either dynamically, in response to an event (as is the typical case with the CAN bus [4]), or statically, at predetermined moments in time (as in the case of TDMA protocols and, in particular, the TTP [13]).

An interesting comparison of the TT and ET approaches, from a more industrial, in particular automotive, perspective, can be found in [14]. Their conclusion is that one has to choose the right approach depending on the particularities of the scheduled tasks. This means not only that there is no single "best" approach to be used, but also that inside a certain application the two approaches can be used together, some tasks being time-triggered and others event-triggered.

The fact that such an approach is considered for future automotive applications is also indicated by the recent activities related to the development and standardisation of bus protocols which support both static (ST) and dynamic (DYN) communication. Such a protocol has been suggested in [16] and [21]. A mixed protocol has been also proposed by a consortium, to be used as a standard in automotive applications [10]. In [6], the authors describe the so called Universal Communication Model (UCM), a framework for modelling at a high level of abstraction the communication infrastructure in automotive applications.

Efficient implementation of new, highly complex distributed automotive applications entails the use of TT task sets together with ET ones, implemented on top of a communication infrastructure with a mixed ST/DYN protocol. Given its flexibility, such an approach has the potential of highly efficient, fine-tuned, and optimised implementations.

Our main contribution in this paper is related to the scheduling and schedulability analysis of distributed embedded systems implemented with both ET and TT task sets, which are communicating through mixed ST/DYN bus protocols. Such an analysis and scheduling procedure constitutes the fundament for any synthesis approach aiming at an efficient, highly optimised implementation of a distributed application which is also guaranteed to meet the timing constraints.

We also identified several design problems which offer the potential of significant optimization and which can be solved by efficient design space exploration, based on the timing analysis mentioned above. In order to illustrate the potential of such optimizations, we have looked more closely at one particular communication synthesis problem.

In the next section we present the architecture of the distributed systems and the application model that we are studying. Section 3 describes the holistic scheduling and schedulability analysis we have developed. Some specific optimization issues are presented in Section 4. Section 5 describes a particular optimization problem related to the bus access, while Section 6 presents some experimental results. The last section presents our conclusions.

2. System Architecture and Application Model

2.1 Hardware Architecture and Bus Access

We consider architectures consisting of nodes connected by a unique broadcast communication channel. Each node consists of a communication controller, a CPU, memories (RAM, ROM), and an I/O interface to sensors and actuators (see Figure 1).

We model the bus access scheme using the Universal Communication Model [6]. The bus access is organized as consecutive cycles, each with the duration T_{bus}. We consider that the communication cycle is partitioned into static and dynamic phases (Figure 1). Static phases consist of time slots, and during a slot only one node is allowed to send ST messages; this is the node associated to that particular slot. During a dynamic phase, all nodes are allowed to send DYN messages and the conflicts between nodes trying to send simultaneously are solved by an arbitration mechanism based on priorities assigned to messages. The bus access cycle has the same structure during each period T_{bus}. Every node has a communication controller that implements the static and dynamic protocol services. The controller runs independently of the node's CPU.

2.2 Software Architecture

For the systems we are studying, we have designed a software architecture which runs on the CPU of each node. The main component of the software architecture is a real-time kernel which supports both time-triggered and event-triggered activities. An activity is defined as either the execution of a task or as the transmission of a message on the bus. For the TT activities, the kernel relies on a static schedule table which contains all the information needed to take decisions on activation of TT tasks or transmission of ST messages. For the ET tasks, the kernel maintains a prioritized ready queue in which tasks are placed whenever their triggering event has occurred and they are ready for activation, or when they have been pre-empted.

The real-time kernel will always activate a TT task at the particular time fixed for that task in the schedule table. If at that moment, an ET task is running on that node, that task will be pre-empted and placed into the ready queue according to its priority. If no tasks are active, ET tasks are extracted from the ready queue and are (re)activated. ET tasks can pre-empt each other based on their priority.

The transmission of messages is handled in a similar way: for each node, the sending and receiving times of ST messages are stored in the schedule table; the DYN messages are organized in a prioritized ready queue. ST messages will be placed at predetermined time moments into a bus slot assigned to the sending node. DYN messages can be potentially sent during any dynamic phase. Conflicts due to simultaneous transmission of messages from different nodes are avoided, based on message priorities, by the communication controllers. In order to prevent the delay of an ST message by a DYN frame or the retransmission of a pre-empted DYN message, the DYN messages will be sent only if there is enough time available for that message before the dynamic phase ends.

TT activities are triggered based on a local clock available in each processing node. The synchronization of local clocks throughout the system is provided by the communication protocol.

2.3 Application Model

We model an application as a set of task graphs. Nodes represent tasks and arcs represent communication (and implicitly dependency) between the connected tasks. Each task is mapped on a certain node of the distributed application.

- A task belongs either to the TT or to the ET domain.
- Communication between tasks mapped to different nodes is performed by message passing over the bus. Such a message passing is modelled as a communication task inserted on the arc connecting the sender and the receiver tasks. The communication time between tasks mapped on the same node is considered to be part of the task execution time. Thus, such a communication activity is not modelled explicitly. For the rest of the paper, when referring to messages, we consider only the communication activity over the bus.
- A message belongs either to the static (ST) or to the dynamic (DYN) domain.
- All tasks in a certain task graph belong to the same domain, either ET, or TT, which is called the domain of the task graph. However, the messages belonging to a certain task graph can belong to any domain (ST or DYN). Thus, in the most general case, tasks

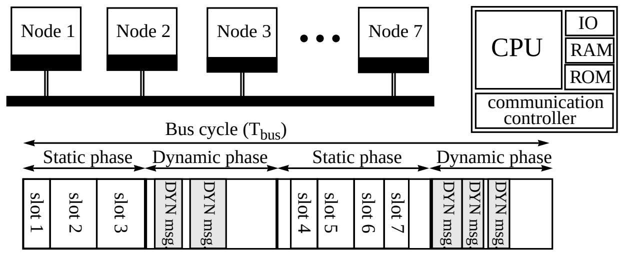


Figure 1. System Architecture

belonging to a TT graph, for example, can communicate through both ST and DYN messages.

- Each task τ_{ij} (belonging to the task graph Γ_i) is mapped on processor *Processor*(τ_{ij}), has a worst case execution time C_{ij}, a period T_{ij}, and a deadline D_{ij} (which, in the case of ET tasks, can be longer than the period). Each ET task also has an uniquely assigned priority $Prio_{ij}$. Individual release times or deadlines of tasks can be constrained by introducing dummy tasks with an appropriate execution time (such dummy tasks are not mapped on any of the nodes).
- All tasks τ_{ij} belonging to a task graph Γ_i have the same period T_i which is the period of the task graph. For sporadic ET tasks, T_i represents the minimum inter-arrival time.
- For each message we know its size (which can be directly converted into communication time on the particular communication bus). The period of a message is identical with that of the sender task. DYN messages also have an uniquely assigned priority.

Figure 2 shows an application modelled as two task graphs mapped on two nodes (processors).

In order to keep the separation between the TT and ET domains, which are based on fundamentally different triggering policies, communication between tasks in the two domains is not included in the model. Technically, such a communication is implemented by the kernel, based on asynchronous non-blocking send and receive primitives (using proxy tasks if the sender and receiver are on different nodes). The transmission and reception of such a message are not considered as communication tasks or respectively events in the context described by our model, therefore they are outside the scope of our holistic analysis. Such messages are typically non-critical and are not affected by hard real-time constraints.

3. Holistic Scheduling

Given an application and a system architecture as presented in Section 2, the following problem has to be solved: construct a correct static schedule for the TT tasks and ST messages (a schedule which meets all time constraints related to these activities) and conduct a schedulability analysis in order to check that all ET tasks meet their deadlines. Two important aspects should be noticed:

1. When performing the schedulability analysis for the ET tasks and DYN messages, one has to take into consideration the interference from the statically scheduled TT tasks and ST messages.
2. Among the possible correct schedules for TT tasks and ST messages, it is important to construct one which favours, as much as possible, the schedulability of ET tasks and DYN messages.

In Section 3.1 we present the schedulability analysis for a set of ET tasks and DYN messages, considering a fixed given static schedule of TT tasks and ST messages. In Section 3.2 we discuss the construction of the static schedule which is driven by the objective of achieving global schedulability of the system. Three alternative scheduling heuristics are presented and they will be evaluated and compared in Section 6.

In order to keep the presentation reasonably simple and given the space limitations, we present here the analysis for a restricted model, in the sense that TT tasks are communicating only through ST messages, while the communication between ET tasks is only through DYN messages. This is not an inherent limitation of our approach and the analysis we have developed and implemented supports the general model (in [18] and [19], for example, we have presented an approach to schedulability analysis of ET tasks communicating through ST messages).

3.1 Schedulability Analysis of the ET Sub-System Considering the Influence of a Given Static Schedule

An ET task graph Γ_i is activated by an associated event which occurs with a period T_i. Each activity τ_{ij} (task or message) in an ET task graph has an offset ϕ_{ij} which specifies the earliest activation time of τ_{ij} relative to the occurrence of the triggering event. The delay between the earliest possible activation time of τ_{ij} and its actual activation time is modelled as a jitter J_{ij} (Figure 3.a). Offsets and jitters are the means by which dependencies among tasks are modelled for the schedulability analysis. The response time of an activity τ_{ij} is the time measured from the occurrence of the associated event until the completion of τ_{ij}. Each ET activity τ_{ij} has a best case response time $R_{b,ij}$. The worst case response time R_{ij} of an activity τ_{ij} is determined by creating first a critical instant t_c, which represents the starting point of the worst-case busy window w_{ij}, a time interval which ends when τ_{ij} finishes execution (Figure 3.b).

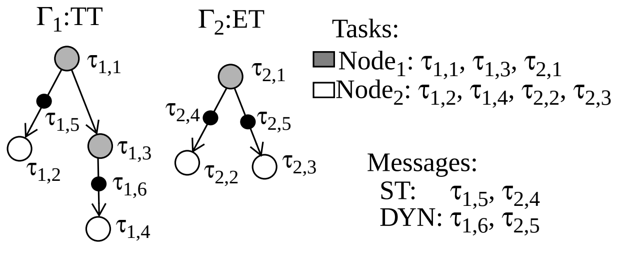


Figure 2. Application Model Example

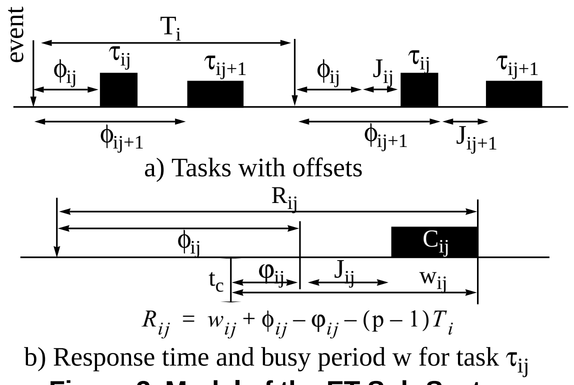
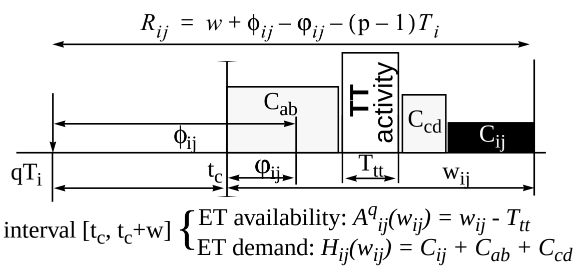


Figure 3. Model of the ET Sub-System



$R_{ij} = w + \phi_{ij} - \varphi_{ij} - (p-1)T_i$
C_{ab}
TT activity
C_{cd}
C_{ij}
ϕ_{ij}
qT_i
t_c
φ_{ij}
T_{tt}
w_{ij}
interval $[t_c, t_c+w]$
ET availability: $A^q_{ij}(w_{ij}) = w_{ij} - T_{tt}$
ET demand: $H_{ij}(w_{ij}) = C_{ij} + C_{ab} + C_{cd}$

Figure 4. Availability and Demand

During the busy window w_{ij}, *Processor*(τ_{ij}) executes only task τ_{ij} or higher priority tasks. φ_{ij} is the time interval between the critical instant and the earliest time for the first activation of the task after this instant.

Considering a set of data dependent ET tasks mapped on a single processor, the analysis in [15] computes the worst case response time R_{ij} of a task τ_{ij}, based on the length of its busy period, considering all the critical instants initiated by higher priority activities in Γ_i and by τ_{ij} itself, and all job instances p of τ_{ij} which can appear in the busy window w_{ij}:

$$R_{ij} = max([max(w_{ijk}(p) - \varphi_{ijk} - (p-1)T_i + \phi_{ij})]), \quad \forall k | Prio_{ik} > Prio_{ij}, \forall p$$

where $w_{ijk}(p)$ is the worst-case busy window of the p-th job of τ_{ij}, numbered from the critical instant t_c initiated by τ_{ik}; φ_{ijk} is the time interval between the critical instant initiated by τ_{ik}, and the earliest time for the first activation of τ_{ij} after this instant.

The value of $w_{ijk}(p)$ is determined as follows:

$$w_{ijk}(p) = B_{ij} + (p - p_{0,ijk} + 1) \cdot C_{ij} + W_{ik}(\tau_{ij}, w_{ijk}(p)) + \sum_{\forall (a \neq i)} W^*_a(\tau_{ij}, w_{ijk}(p)))$$

where B_{ij} represents the maximum interval during which τ_{ij} can be blocked by lower priority activities[1], $W_{ik}(\tau_{ij},t)$ is the interference from higher priority activities in the same task graph Γ_i at time t, and $W^*_a(\tau_{ij},t)$ is the maximum interference of activities from other task graphs Γ_a on τ_{ij}. One problem that arises during the computation of response times is that the length of the busy window depends on the values of task jitters, which, in turn, are computed as the difference between the response times of two successive tasks (for example, if t_{ij} precedes t_{ik} in Γ_i, then $J_{ik} = R_{ij} - R_{b,ij}$). Because of this cyclic dependency, the process of computing R_{ij} is an iterative one: it starts by assigning $R_{b,ij}$ to R_{ij} and then computes the values for J_{ij}, $w_{ijk}(p)$ and then again R_{ij}, until the response times converge to their final value.

Starting from the analysis in [15], we had to consider the following additional aspects:

- The interference from the set of statically scheduled tasks.
- The computation of worst case delays for the messages communicated on the bus and the global schedulability analysis of the distributed task set.

1. Such blocking can occur at access to a shared critical resource.

We solve both aspects using an analysis similar to the one developed in [16]. First we introduce the notion of *ET demand* associated with an ET activity τ_{ij} as the amount of CPU time or bus time which is *demanded* only by higher priority ET activities and by τ_{ij} during the busy window w_{ij}. In Figure 4, the ET demand of the task τ_{ij} during the busy window w_{ij} is represented with $H_{ij}(w_{ij})$, and it is the sum of worst case execution times for task τ_{ij} and two other higher priority tasks τ_{ab} and τ_{cd}. During the same busy period w_{ij}, we define the *availability* as the processing time which is not used by statically scheduled activities. In Figure 4, the CPU availability for the interval of length w_{ij} is obtained by substracting from w_{ij} the amount of processing time needed for the TT activities.

During a busy window w_{ij}, the *ET demand* H_{ij} of a task τ_{ij} is equal with the length of the busy window which would result when considering only ET activity on the system:

$$H_{ij}(w_{ij}) = B_{ij} + (p - p_{0,ijk} + 1) \cdot C_{ij} + W_{ik}(\tau_{ij}, w_{ij}) + \sum_{\forall (a \neq i)} W^*_a(\tau_{ab}, w_{ij})$$

During the same busy window w_{ij}, the availability A_{ij} associated with task τ_{ij} is:

$$A_{ij}(w_{ij}) = min[A^q_{ij}(w_{ij})], \quad q = 0, \frac{LCM(T_i, T_{SS})}{T_i}$$

where $A^q_{ij}(w)$ is the total available CPU-time on *Processor*(τ_{ij}) in the interval $[q\,T_i + \phi_{ij} - \varphi_{ijk}, q\,T_i + \phi_{ij} - \varphi_{ijk} + w_{ij}]$, T_i is the period of Γ_i and T_{SS} is the period of the static schedule (see Section 3.2). Figure 4 presents how $A^q_{ij}(w)$ and the demand are computed for a task τ_{ij}: the busy window of τ_{ij} starts at the critical instant $q\,T_i + t_c$ initiated by task τ_{ab} and ends at moment $qT_i + t_c + w_{ij}$, when both higher priority tasks (τ_{ab}, τ_{cd}), all TT tasks scheduled for execution in the analysed interval, and τ_{ij} have finished execution.

The discussion above is, in principle, valid for both ET tasks and DYN messages. However, there exist two important differences. First, messages do not pre-empt each other, therefore, the demand equation is modified so that it will not consider the time needed for the transmission of the message under analysis (once the message has gained the bus it will be sent without any interference [16]). Second, the availability for a message is computed by substracting from w_{ij} the length of the ST slots which appear during the considered interval; moreover, because a DYN message will not be sent unless there is enough time before the current dynamic phase ends, the availability is further decreased with C_A for each dynamic phase in the busy window (where C_A is the transmission time of the longest DYN message).

Our schedulability analysis algorithm determines the length of a busy window w_{ij} for an ET task or DYN message by identifying the appropriate size of w_{ij} for which the ET demand is satisfied by the availability: $H_{ij}(w_{ij}) \le A_{ij}(w_{ij})$.

This procedure for the calculation of the busy window is included in the iterative process for calculation of response times, presented earlier in this subsection. It is important to notice that this process includes both tasks and messages and, thus, the resulted response times of the ET tasks are computed by taking into consideration the delay induced by the bus communication.

After performing the schedulability analysis, we can check if $R_{ij} \le D_{ij}$ for all the ET tasks. If this is the case, the set of ET activities is schedulable. In order to drive the global scheduling process, as it will be explained in the next section, it is not sufficient to test if the task set is schedulable or not, but we need a metric that captures the "degree of schedulability" of the task set. For this purpose we use the function *DSch*, similar with the one described in [18]:

$$\text{DSch} = \begin{cases} f_1 = \sum_{i=1}^{N} \sum_{j=1}^{N_i} max(0, R_{ij} - D_{ij}), \text{ if } f_1 > 0 \\ f_2 = \sum_{i=1}^{N} \sum_{j=1}^{N_i} (R_{ij} - D_{ij}), \text{ if } f_1 = 0 \end{cases}$$

where N is the number of ET task graphs and N_i is the number of activities in the ET task graph Γ_i.

If the task set is not schedulable, there exists at least one task for which $R_{ij} > D_{ij}$. In this case, $f_1 > 0$ and the function is a metric of how far we are from achieving schedulability. If the set of ET tasks is schedulable, $f_2 \le 0$ is used as a metric. A value $f_2 = 0$ means that the task set is "just" schedulable. A smaller value for f_2 means that the ET tasks are schedulable and a certain amount of processing capacity is still available.

Now, that we are able to perform the schedulability analysis for the ET tasks considering the influence from a given static schedule of TT tasks, we can go on to perform the global scheduling and analysis of the whole application.

3.2 Static Schedule Construction and Holistic Analysis

For the construction of the cyclic static schedule for TT tasks and ST messages, we use a list-scheduling based algorithm [5]. Assuming that in our application we have N time-triggered task graphs $\Gamma_1, \Gamma_2, ..., \Gamma_N$, the static schedule will be computed over a period $T_{SS} = \text{LCM}(T_1, T_2, ..., T_N)$. The input to the list scheduling algorithm is a graph consisting of n_i instances of each Γ_i, where $n_i = T_{SS}/T_i$. A ready list contains all TT tasks and ST messages which are ready to be scheduled (they have no predecessors or all their predecessors have been scheduled). From the ready list, tasks and messages are extracted one by one to be scheduled on the processor they are mapped to, or into a static bus-slot associated to that processor on which the sender of the message is executed, respectively. The priority function which is used to select among ready tasks and messages is a critical path metric, modified for the particular goal of scheduling tasks mapped on distributed systems [17]. Let us consider a particular task τ_{ij} selected from the ready list to be scheduled. We consider that $ASAP_{ij}$ is the earliest time moment which satisfies the condition that all preceding activities (tasks or messages) of τ_{ij} in graph Γ_i are finished and *Processor*(τ_{ij}) is free. The moment $ALAP_{ij}$ is the latest time when τ_{ij} can be scheduled. With only the TT tasks in the system, the straightforward solution would be to schedule τ_{ij} at $ASAP_{ij}$. In our case, however, such a solution could have negative effects on the schedulability of ET tasks. What we have to do is to place task τ_{ij} in such a position inside the interval [$ASAP_{ij}$, $ALAP_{ij}$] so that the chance to finally get a globally schedulable system is maximised.

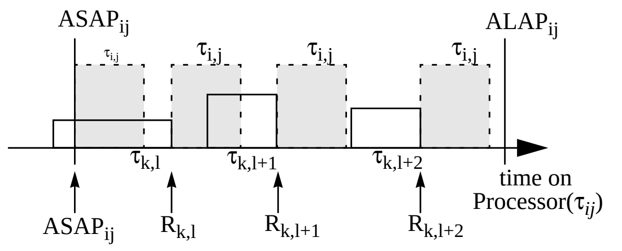


Figure 5. Selection of Alternative Start Times

In order to consider only a limited number of possible positions for the start time of a TT task τ_{ij}, we take into account the information obtained from the schedulability analysis described in Section 3.1, which allows us to compute the response times of ET tasks. We started from the obvious observation that statically scheduling τ_{ij} after an ET task τ_{kl} has finished its execution will guarantee that task τ_{ij} will not interfere with τ_{kl}. Thus, we consider as alternative start times for τ_{ij} the response times of all ET tasks which finish their execution inside the [$ASAP_{ij}$, $ALAP_{ij}$] interval:

$$\text{alternative\_start\_times}(\tau_{ij}) = ASAP_{ij} \cup (\{R_{kl} | \tau_{kl} \in ETdomain, Processor(\tau_{kl}) = Processor(\tau_{ij})\} \cap [ASAP_{ij}, ALAP_{ij}])$$

The moment referred by $ASAP_{ij}$ was added to *alternative_start_times* so that the set of alternative start times of a TT task will not be empty even if no ET tasks finish their execution during the interval [$ASAP_{ij}$, $ALAP_{ij}$].

We illustrate the choice of possible start times of a TT task τ_{ij} in Figure 5 where three ET tasks $\tau_{k,l}$, $\tau_{k,l+1}$, $\tau_{k,l+2}$ finish their execution inside [$ASAP_{ij}$, $ALAP_{ij}$] leading to *alternative_start_times*(τ_{ij}) = {$ASAP_{ij}$, $R_{k,l}$, $R_{k,l+1}$, $R_{k,l+2}$}. Statically scheduling τ_{ij} at time $R_{k,l}$ avoids the interferences from τ_{ij} to τ_{kl}., while scheduling τ_{ij} even later, at $R_{k,l+1}$, will guarantee that τ_{ij} does not interfere with either τ_{kl} or $\tau_{k,l+1}$.

After identifying the set of candidate start times of a task, we have to select one of them as the static schedule for that task. Two aspects have to be considered in this context:

1. The interference with the ET activities should be minimised;
2. The deadlines of TT activities should be satisfied.

In order to evaluate the first goal, the value of the function *DSch* (see Section 3.1) is computed for each alternative start time *t* after performing the schedulability analysis of the ET task set considering the influence from the TT

```
00 compute [ASAP,ALAP] for each TT activity
01 while ready_list is not empty
02     select TT activity $\tau_{ij}$
03     if $\tau_{ij}$ is a task
04         Schedulability Analysis -> Compute
                   response times of ET activities
05         compute alternative_start_times($\tau_{ij}$)
06         for t in alternative_start_times($\tau_{ij}$)
07             schedule $\tau_{ij}$ at t
08             Schedulability Analysis ->
                   Compute DSch ->
                   Compute CostFunction
09         endfor
10         schedule $\tau_{ij}$ at t for which
             the CostFunction is minimum
11     else // $\tau_{ij}$ is a message
12         ASAP schedule $\tau_{ij}$ in slot$_i$
13     endif
14     update ready_list
15 endwhile
```

Figure 6. Static Scheduling Algorithm

tasks, with τ_{ij} scheduled at *t*. As will be shown in the following section, a global cost function is computed, which combines both goals defined above, and, based on a greedy approach, the start time of the task will be selected.

The scheduling algorithm is presented in Figure 6. If the selected TT activity extracted from the *ready_list* is a task τ_{ij}, then the *alternative_start_times* are evaluated and the algorithm selects the one which generates the smallest value of the cost function. When scheduling an ST message extracted from the ready list, we place it into the first bus-slot associated with the sender node in which there is sufficient space available. If all TT tasks and ST messages have been scheduled and the schedulability analysis for the ET tasks indicates $DSch \leq 0$, then the global system scheduling has succeeded.

For the case that no correct schedule has been produced, we have implemented a backtracking mechanism in the list scheduling algorithm, which allows to turn back to previous scheduling steps and to try alternative solutions. In order to avoid excessive scheduling times, the maximum number of backtracking steps can be limited.

In the following subsections we present three alternative ways to compute the cost function which drives the heuristic in Figure 6. The three alternatives are identified as MxS1, MxS2 and MxS3 (from mixed scheduling).

3.2.1 MxS1

Scheduling a TT task τ_{ij} inside its [$ASAP_{ij}$, $ALAP_{ij}$] interval will, of course, guarantee that deadlines related to this particular task are satisfied, and that there exists the possibility that a valid static schedule can be constructed for the system. However, due to the data dependencies, scheduling τ_{ij} later inside [$ASAP_{ij}$, $ALAP_{ij}$] decreases the probability of finding a feasible static schedule for the tasks further down. This is why, for the evaluation of the alternative start times of a TT task τ_{ij} (line 08 in Figure 6), we introduced a cost function which combines the degree of schedulability of the ET activities (*DSch* in Section 3.1) with a second metric which captures the "risks" taken by scheduling τ_{ij} at later times:

$$f(t, \tau_{ij}) = A \cdot e^{-slack(t, \tau_{ij})} \cdot t + B \cdot DSch$$

where *t* is one of the alternative start times, *A* and *B* are normalization constants, and *slack(t*, τ_{ij}*)* represents the available cpu-time on *Processor(*τ_{ij}) (the processing time inside the interval [$t + C_{ij}$, T_{SS}] which is not used by any of the ET or TT tasks). The value of *slack* is computed as follows:

$$slack(t, \tau_{ij}) = [T_{SS} - (t + C_{ij})] - H_{ET}(t + C_{ij}, T_{SS}, Proc_{ij}) - UnschH_{TT}(Proc_{ij})$$

where $UnschH_{TT}$ represents the sum of execution times of all yet unscheduled TT tasks mapped on *Processor(*τ_{ij}*)*. The term H_{ET} represents the time demanded by ET tasks in the interval [$t + C_{ij}$, T_{SS}] on *Processor(*τ_{ij}*)* and is computed in the following steps:

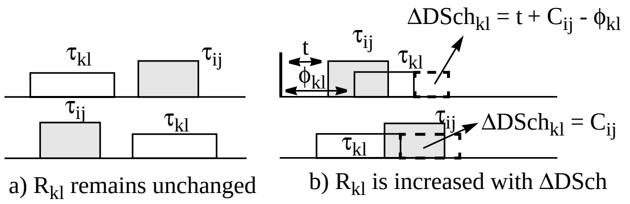
1. For each ET task τ_{ab} mapped on *Processor(*τ_{ij}*)* consider its worst case response interval $I_{ab} = [\phi_{ab}, R_{ab}]$ using the response times computed in line 08 of the algorithm in Figure 6.
2. For each scheduled TT task τ_{ab} mapped on *Processor(*τ_{ij}*)*, we know the start time t_{ab} and consider the associated execution interval $I_{ab} = [t_{ab}, t_{ab} + C_{ab}]$.
3. Compute the unions of intervals in which ET and scheduled TT activities take place: $I_{ET} = \bigcup_{\forall \tau_{ab} \in ET} I_{ab}$ and $I_{schedTT} = \bigcup_{\forall \tau_{ab} \in schedTT} I_{ab}$.
4. Compute H_{ET} as the sum of lengths of each of the intervals in $(I_{ET} \cup I_{TT}) \cap [t_{ij} + C_{ij}, T_{SS}]$.

It is easy to notice that if the *slack* has a very small value (even negative), then the first term in function *f* (the one depending on time *t*) has a much greater weight on the value of *f*. Consequently, earlier start times for τ_{ab} will be preferred. On the other hand, if there is more available processor time than needed (in other words, *slack* has a high value), the function *f* will depend mainly on the value of the second term, thus the main aspect taken into consideration will be the influence of TT activities on the ET ones, which is captured by *DSch*.

The static scheduling algorithm will select, among the alternative start times, that time *t* for which the value of the cost function *f* is minimum.

3.2.2 MxS2

The schedulability analysis algorithm described in Section 3.1 is applied very often during the static scheduling procedure presented in Figure 6, both in order to compute the values of the possible start times (line 04) and the *Cost* associated with each such start time (line 08). In order to reduce the amount of time needed for scheduling, we experimented with an algorithm which uses the schedulability analysis only for determining the set of *start_times*(line 04), while the evaluation process in line 08 is based on a simpler version of function *f*. In MxS1, when the alternative start times of a TT task are evaluated, running the schedulability analysis returns the value *DSch* which reflects the amount of new interference that has been introduced in the ET subsystem at a global level. The simpler function *f*, which we use in this second algorithm, avoids calling the global schedulability analysis for each



τ_{kl}
τ_{ij}
τ_{ij}
τ_{kl}
$\Delta DSch_{kl} = t + C_{ij} - \phi_{kl}$
t
τ_{ij}
τ_{kl}
ϕ_{kl}
τ_{ij}
$\Delta DSch_{kl} = C_{ij}$
τ_{kl}
a) R_{kl} remains unchanged
b) R_{kl} is increased with ΔDSch

Figure 7. Estimation of ET Response Times in MxS2 Algorithm

possible start time of a TT task τ_{ij} and considers *only* the interferences produced by τ_{ij} on the ET tasks mapped on *Processor*(τ_{ij}):

$$f'(t, \tau_{ij}) = e^{-slack(t, \tau_{ij})} \cdot t + B \cdot (DSch + \Delta DSch)$$

where the value of *DSch* (as expressed in Section 3.1) is computed (on line 4, Figure 6) before τ_{ij} has been scheduled, and Δ*DSch* is the amount of interference introduced by τ_{ij} on the ET tasks mapped on *Processor*(τ_{ij}):

$$\Delta DSch = \sum_{\substack{\tau_{kl} \in ET \\ Processor(\tau_{kl}) = Processor(\tau_{ij})}} (R'_{kl} - R_{kl}) \quad ,$$

where R_{kl} is the response time of an ET task τ_{kl} before τ_{ij} has been scheduled and R'_{kl} is an approximation of the response time of τ_{kl} after τ_{ij} has been scheduled at time *t*. We estimate that, depending on the time *t* when a TT task τ_{ij} is scheduled, the response time of an ET task τ_{kl} mapped on the same *Processor*(τ_{ij}) either remains unchanged (is not influenced at all) or is increased with a value up to the worst case execution time C_{ij} of the TT task. Figure 7 presents which are the situations when the response time of an ET task τ_{kl} remains unchanged and when it is increased because of the influence of a TT task τ_{ij}. The cases represented in Figure 7.a) show that when a TT task τ_{ij} is scheduled at time *t* so that its associated execution interval [t, $t + C_{ij}$] does not intersect with the time interval where an ET task executes in the worst case [ϕ_{kl}, R_{kl}], then we estimate that after scheduling τ_{ij} at *t*, the response time for τ_{kl} will be the same, $R_{kl}' = R_{kl}$. However, if the intersection is not empty (like in the cases in Figure 7.b), then $R_{kl}' = R_{kl} + \Delta DSch_{kl}$. The value for the increment used in the function $f'(t,\tau_{ij})$ will be computed as $\Delta DSch = \Sigma\Delta DSch_{kl}$, for all τ_{kl} in the ET domain and *Processor*(τ_{kl}) = *Processor*(τ_{ij}).

In MxS2, the schedulability analysis of the system is called only once for each TT task (step 04), which will lead, as we will see in Section 6, to faster computation times.

3.2.3 MxS3

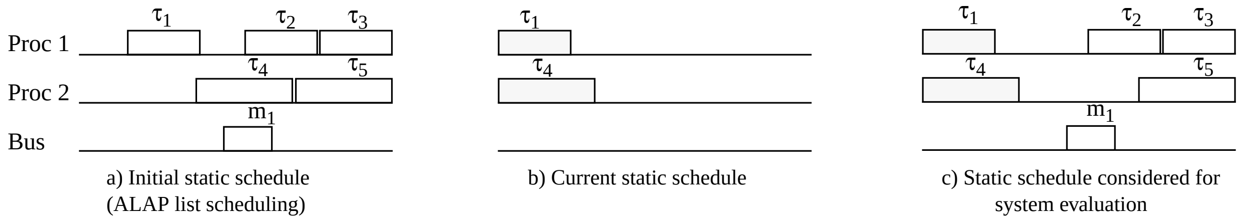
List scheduling, which is the basis for our scheduling algorithm, is a constructive method that builds the static schedule table incrementally, by adding one TT task or ST message at a time. In the previous two versions of the algorithm (Section 3.2.1 and Section 3.2.2), at each step, the effect of the static schedule, including the newly introduced task, on the ET subsystem is measured by function *DSch*. However, the problem is that the available static schedule is not complete when estimating, for a given task τ_{ij}, the global influence of TT activities on the set of ET ones. For the alternatives MxS1 and MxS2 we have chosen the following simple approach: for evaluating the influence of the decision of which alternative start time to select for τ_{ij}, we consider only that part of the static schedule which already has been built, up to that particular moment. The selection is fair, as the same conditions are applied to all alternative times; however, it is inaccurate, since a part of the final static schedule is ignored when taking the decision. For the alternative MxS3 we have considered a solution which tries to improve on this lack of accuracy by considering the whole set of TT activities when evaluating the degree of schedulability of the ET tasks and messages. This is solved by considering an approximate static schedule for the yet unscheduled TT activities. Therefore, a preliminary step is performed in preparation of the algorithm in Figure 6.

First, we build an initial static schedule by using a simpler and faster version of the algorithm in Figure 6. In this version, the response times of the ET activities are computed only once in the beginning of the algorithm and the evaluation of possible start times is performed using a simple function like in MxS2. This step allows us to rapidly obtain a static schedule which will be at the basis of the second step of our approach.

After the preparation step, we run the algorithm in Figure 6, but whenever schedulability analysis of the ET subsystem is performed, we consider that the interfering static schedule not only contains the TT activities which were scheduled so far, but all the TT tasks and ST messages in the system. We obtain such a complete static schedule by considering:

- the start times of the TT tasks/ STmessages scheduled so far in this second step;
- for the unscheduled TT tasks/ ST messages, we consider their start times as identified in the first step of the algorithm

Figure 8 illustrates the way we obtain such a complete static schedule. The static schedule considered during the schedulability analysis of the ET subsystem (Figure 8.c) contains all the TT tasks and ST messages in the system. Such a complete schedule is obtained by putting together



Proc 1
Proc 2
Bus
τ_1
τ_2
τ_3
τ_4
τ_5
m_1
a) Initial static schedule (ALAP list scheduling)
b) Current static schedule
c) Static schedule considered for system evaluation

Figure 8. Construction of a Static Schedule for Complete Evaluation

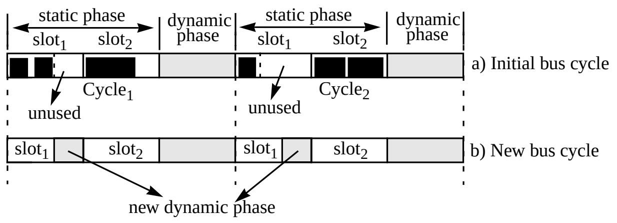


Figure 9. Transformation of Unused Static Bandwidth into Dynamic Phases

the start times for tasks τ_1 and τ_4 (which have been scheduled already, see Figure 8.b) and the start times identified in the first step for the unscheduled tasks τ_2, τ_3, τ_5 and message m_1 (Figure 8.a).

4. System Optimization

Considering a hard real-time system like the one described in Section 2, several design problems emerge. There are, of course, the classical issues as selection of an architecture (e.g. number and kind of nodes), the mapping of tasks on the processing nodes, or the assignment of priorities to ET tasks and DYN messages [1],[9],[25]. However, due to the heterogeneous ET and TT nature of the application and the mixed synchronous/dynamic bus protocol, some new, very interesting problems can be identified:

- *Partitioning of the system functionality into TT and ET activities*. During the design process, a decision should be made on which tasks and messages will be implemented as TT/ET and ST/DYN activities, respectively. Typically, this decision is taken, based on the experience and preferences of the designer, considering aspects like the functionality implemented by the task, the hardness of the constraints, sensitivity to jitter, etc. There exists, however, a subset of tasks/messages which could be assigned to any of the domains. Decisions concerning the partitioning of this set of activities can lead to various trade-offs concerning, for example, the size of the schedule table or the schedulability properties of the system.
- *Determining the optimal structure of the bus access cycle*. The configuration of the bus access cycle has a strong impact on the global performance of the system. The parameters of this cycle have to be optimised such that they fit the particular application and the timing requirements at the task level. Parameters to be optimised are the number of static and dynamic phases during a communication cycle, as well as the length and order of these phases. Considering the static phases, parameters to be fixed are the order, number, and length of slots assigned to the different nodes.

The optimization problems identified above can be approached once the holistic scheduling technique presented in Section 3 is available. In the next section we illustrate this by considering a particular problem related to bus access optimization.

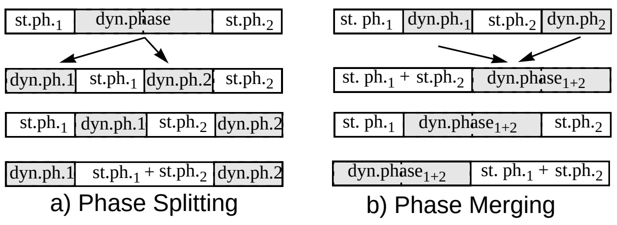


Figure 10. Operations on Dynamic Phases

5. Bus Access Optimization

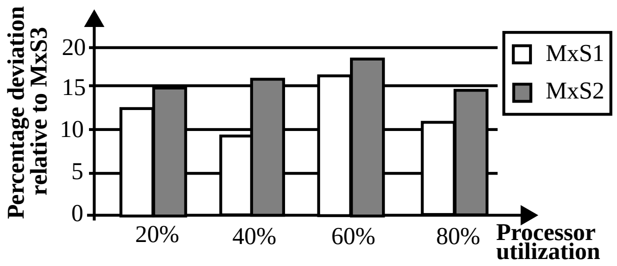
We consider an application and an architecture like the one described in Section 2. The designer has mapped the tasks on the nodes of the system and has set the bus cycle according to his best knowledge. After running the holistic scheduling presented in Section 3, it turns out that a correct static schedule for the TT tasks and ST messages has been generated, but the ET task set is not schedulable. One of the reasons for this could be that there is not sufficient bandwidth allocated for the communication of messages between ET tasks. The problem to be solved is to find a structure of the bus cycle such that more bandwidth is allocated to the dynamic phases with the goal to improve the schedulability of ET tasks while maintaining a correct static schedule.

As a first step, the optimization algorithm transforms some parts of the static phases into dynamic phases. For each static slot in the bus cycle and for each round in the static schedule we transform the periodically unused part of the slot into a dynamic phase (see Figure 9).

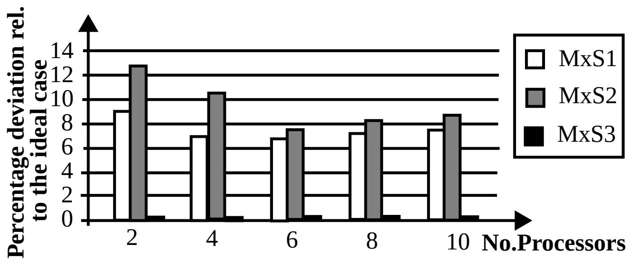
After this initial step, various bus cycle configurations are explored by splitting and merging bus phases. Figure 10 illustrates the operations on dynamic phases. Three possible outcomes are shown for both the splitting and the merging example. We have implemented a simulated annealing based algorithm which applies successive splitting and merging transformations with the goal to improve the schedulability of the ET task set and the constraint of achieving a correct static schedule for TT tasks. The objective function driving the algorithm is the function *DSch* introduced in Section 3.1

6. Experimental Results

For evaluation of our scheduling and analysis algorithm we generated a set of 3600 tests representing systems of 2 to 10 nodes. The number of tasks mapped on each node varied between 10 and 30, leading to applications with a number of 20 up to 300 tasks. The tasks were grouped in task-graphs of 5, 10 or 15 tasks. Between 20% and 80% of the total number of tasks were considered as event-triggered and the rest were set as time-triggered. The execution times of the tasks were generated in such a way that the utilization on each processor was between 20% and 80%. In a similar manner we assured that 20% and up to 60% of the total utilization on a processor is required by the ET activities. All experiments were run on an AMD Athlon 850MHz PC.



a) Deviation of Number of solutions (smaller is better)



b) Degree of interference (smaller is better)

Figure 11. Evaluation of Scheduling Heuristics MxS1, MxS2 and MxS3

The first set of experiments compares the three versions of the holistic scheduling algorithm we proposed in Section 3.2. In Figure 11.a) we illustrate the capacity of MxS1 and MxS2 to produce schedulable systems, compared to that of MxS3. For example, in the case of a 60% load, MxS2 was able to generate 18% and MxS1 16% less schedulable solutions than MxS3. In addition, for each heuristic, we computed the quality of the identified solutions, as the percentage deviation of the schedulability degree ($DSch_{MxS}$) of the ET activities in the resulted system, relative to the schedulability degree of an ideal solution ($DSch_{ref}$) in which the static schedule does not interfere at all with the execution of the ET activities:

$$Interference = \frac{DSch_{ref} - DSch_{MxS}}{DSch_{ref}} \cdot 100$$

In other words, we used the function *DSch* as a measure of the interference introduced by the TT activities on the execution of ET activities. In Figure 11.b), we present the average quality of the solutions found by the three algorithms. For this graph we used only those results where all three algorithms managed to find a schedulable solution. It is easy to observe that the solutions obtained with MxS3 are constantly at a minimal level of interference. The heuristics MxS1 and MxS2 produce solutions in which the TT interference is considerably higher, resulting in significantly larger response times of the ET activities and consequently to a decrease of the schedulability degree by 7-13%. Not surprisingly, our experiments prove that the heuristic MxS3 is the most accurate and consequently produces results of the best quality. MxS2, which uses local approximation for the evaluation of the ET schedulability, has a slightly lower quality than MxS1.

In Figure 12 we illustrate the average execution times of the three scheduling heuristics. According to expectations, MxS2 is the fastest of the three heuristics, while MxS_3 is slightly slower than MxS1. In conclusion, the heuristic MxS3 is the one which offers the best solutions at an acceptable computation time. MxS2 is very fast and can be used in certain particular cases like, for example, inside a design space exploration loop with an extremely large number of iterations. MxS3 has been used for the set of experiments presented in the rest of this section.

For the evaluation of the bus access optimization heuristic in Section 5, we generated a total of 400 applications, each of them consisting of 100 tasks mapped on 10 processor nodes. The percentage of ET tasks was 20%, 40%, 60%, or 80% of the total number of tasks. Processor utilisation was 60% or 80%. The bus bandwidth was equally divided between the dynamic and the static phases and the static phase was equally divided in a number of slots identical with the number of nodes. This set of experiments concerns the potential of the bus access optimization discussed in Section 5. For this purpose we selected that part of the generated applications for which the ET component resulted unschedulable. Table 1 shows the results after running our optimization heuristic for this application set. As can be observed, the average improvement of the schedulability obtained by bus access optimization is between 22% and 29% for the tests with balanced numbers of ET and TT activities (the two central columns), with an average optimization time below 6 minutes. For unbalanced distributions, the improvement can be even much larger. As discussed in Section 5, these improvements have been obtained considering only a very limited optimization issue, namely the distribution of bandwidth between the static and the dynamic phases. This demonstrates the huge optimization potential of the different design problems discussed in Section 4.

Finally, we considered a real-life example implementing a vehicle cruise controller and a control application related to the Anti Blocking System. The cruise controller consists of 42 TT tasks mapped over 5 nodes. The second control system consists of 30 ET tasks which are mapped on 3 of the same 5 nodes. Initially, the bandwidth on the communication bus is equally divided between the static and dynamic phases. The scheduling of the system took 4 seconds and resulted in a correct static schedule and an

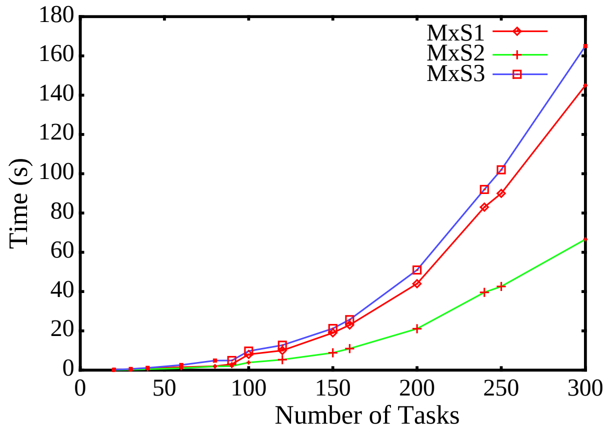


Figure 12. Average Computation Times

Processor utilisation	80% ET tasks		60% ET tasks		40% ET tasks		20% ET tasks	
	schedulability improvement	optimization time	schedulability improvement	optimization time	schedulability improvement	optimization time	schedulability improvement	optimization time
60%	15%	301 s	23%	316 s	29%	275 s	82%	139 s
80%	26%	479 s	25%	294 s	22%	320 s	102%	145 s

Table 1: Bus Optimization Results

unschedulable ET domain. After running the bus access optimization, the schedulability (expressed in terms of the function *DSch*) has improved by more than one order of magnitude, resulting in a completely schedulable system. The optimization was solved in aproximatively 4 minutes.

7. Conclusions

Distributed embedded systems based on mixed static/dynamic communication protocols are becoming a new standard for automotive applications. Such systems typically run applications consisting of both ET and TT tasks. We have presented a holistic scheduling and timing analysis approach for this class of systems. A static cyclic schedule is constructed for TT tasks and ST messages and the schedulability of ET tasks and DYN messages is verified. The static schedule is constructed in such a way that it fits the schedulability requirements of the ET domain. We have identified a new class of system optimization issues typical for the heterogeneous systems considered in the paper. In particular, we have considered a bus access optimization problem and have shown that the system performance can be improved by carefully adapting the bus cycle to the particular requirements of the application.

8. References

[1] N. Audsley, K. Tindell, A. et. al., “The End of Line for Static Cyclic Scheduling?”, 5th Euromicro Works. on Real-Time Systems, 1993.

[2] N. Audsley, A. Burns, et. al., “Fixed Priority Preemptive Scheduling: An Historical Perspective”, Real-Time Systems, 8(2/3), 1995.

[3] F. Balarin, L. Lavagno, et. al., “Scheduling for Embedded Real-Time Systems”, IEEE Design and Test of Computers, January-March,1998.

[4] R. Bosch GmbH, “CAN Specification Version 2.0”, 1991.

[5] E.G. Coffman Jr., R.L. Graham, “Optimal Scheduling for two Processor Systems”, *Acta Informatica*, 1, 1972.

[6] T. Demmeler, P. Giusto, “A Universal Communication Model for an Automotive System Integration Platform”, DATE, 2001.

[7] R. Dobrin, G. Fohler, “Implementing Off-Line Message Scheduling on Controller Area Network (CAN)”, Proceedings of the 8th IEEE International Conference on Emerging Technologies and Factory Automation, 1, 2001.

[8] H. Ermedahl, H. Hansson, M. Sjödin, “Response Time Guarantees in ATM Networks”, Proceedings of Real-Time Systems Symposium, 1997.

[9] R. Ernst, “Codesign of Embedded Systems: Status and Trends”, IEEE Design&Test of Comp., April-June, 1998.

[10] FlexRay homepage: http://www.flexray-group.com/.

[11] H. Hansson, M. Sjödin, K. Tindell, “Guaranteeing Real-Time Traffic Through an ATM Network”, Proceedings of the 30th Hawaii International Conference on System Sciences, 5, 1997.

[12] H. Kopetz, G. Fohler, et. al., “The Programmer’s View of MARS”, Proceedings of Real-Time Systems Symposium, 1992.

[13] H. Kopetz, “Real-Time Systems - Design Principles for Distributed Embedded Applications”, Kluwer Academic Publisher, 1997.

[14] H. Lönn, J. Axelsson, “A Comparison of Fixed-Priority and Static Cyclic Scheduling for Distributed Automotive Control Applications”, Euromicro Conf. on RTS, 1999.

[15] J. C. Palencia, M. Gonzaléz Harbour, “Schedulability Analysis for Tasks with Static and Dynamic Offsets”, Proceedings of the 19th IEEE Real-Time Systems Symposium, 1998.

[16] L. Almeida, P. Pedreiras, J. A. G. Fonseca, “The FTT-CAN Protocol: Why and How”, IEEE Transactions on Industrial Electronics, 49(6), 2002.

[17] P. Pop, P. Eles, Z. Peng, A. Doboli, “Scheduling with Bus Access Optimization for Distributed Embedded Systems“, IEEE Transactions on VLSI Systems, 8(5), 2000.

[18] P. Pop, P. Eles, Z. Peng, “Bus Access Optimization for Distributed Embedded Systems Based on Schedulability Analysis“, DATE, 2000.

[19] P. Pop, P. Eles, Z. Peng, “Schedulability-Driven Communication Synthesis for Time-Triggered Embedded Systems”, Real-Time Systems Journal (accepted for publication)

[20] P. Pleinevaux, “An Improved Hard Real-Time Scheduling for the IEEE 802.5”, Journal of Real-Time Systems, 4(2), 1992.

[21] P. Raja, G. Noubir, “Static and Dynamic Polling Mechanisms for Fieldbus Networks”, ACM Operating Systems Review, 27(3), 1993.

[22] J.K. Strosneider, T.E. Marchok, “Responsive, deterministic IEEE 802.5 Token Ring Scheduling”, Journal of Real-Time Systems, 1(2), 1989.

[23] K. Tindell, J. Clark, “Holistic Schedulability Analysis for Distributed Hard Real-Time Systems”, Microprocessing & Microprogramming, Vol. 50, Nos. 2-3, 1994.

[24] K. Tindell, H. Hansson, A.J. Wellings, “Analysing Real-Time Communications: Controller Area Network (CAN)”, Proceedings of the Real-Time Systems Symposium, 1994

[25] W. Wolf, “Hardware-Software Co-Design of Embedded Systems”, *Proceedings of the IEEE*, V82, N7, 1994.

[26] J. Xu, D.L. Parnas, “On satisfying timing constraints in hard-real-time systems”, IEEE Transactions on Software Engineering, 19(1), 1993.