Holistic Scheduling and Analysis of Mixed Time/Event-Triggered Distributed Embedded 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 the new standard for automotive applications. We have developed a holistic timing analysis and scheduling approach for this category of systems. 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 behavior 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 [16], Tindell provided a framework for holistic analysis of event-triggered task sets interconnected through an infrastructure based on either the CAN protocol or a generic TDMA protocol. In [13] and [14] we have developed a holistic analysis allowing for either time-triggered or event-triggered task sets communicating over a particular TDMA protocol, the TTP. 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 [9]. In the event-triggered (ET) approach, task activities are initiated whenever a particular event is noted. In the time-triggered (TT) approach, task activities are initiated at predetermined points in time. 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], [9], [18]. 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 (like with the CAN bus [4]), or statically, at predetermined moments in time (as in the case of TDMA protocols and, in particular, the TTP [9]).

An interesting comparison of the TT and ET approaches, from a more industrial, in particular automotive, perspective, can be found in [10]. 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 [12] and [15]. Recently, the first mixed protocol has been proposed by a consortium, to be used in automotive applications [8]. 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. Their approach is targeted towards simulation and refinement without considering the aspect of timing analysis with hard real-time constraints.

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.

This paper is the first one, to our knowledge, to handle the holistic analysis and the design optimization of heterogeneous TT&ET systems which are of great importance for future automotive applications.

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

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.

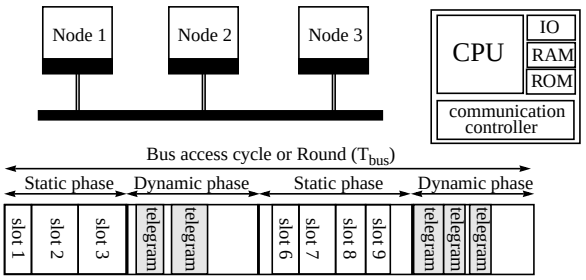


Figure 1. System Architecture

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 ST 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 TT 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 preempted 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 and conflicts are solved by the communication controllers based on message priorities. In order to prevent the delay of an ST message by a DYN frame or the retransmission of a preempted 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 preformed 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 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 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 $Proc_{ij}$, 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 a uniquely assigned priority $Prio_{ij}$.
- 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 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.

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

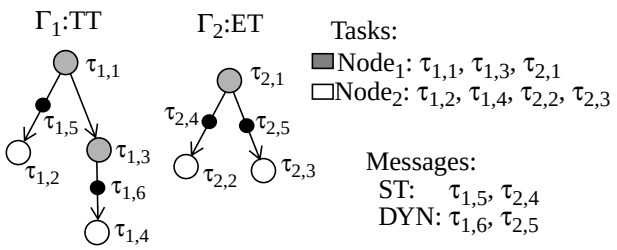


Figure 2. Application Model Example

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

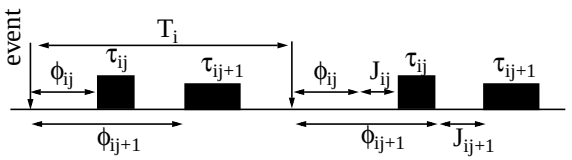
Given an application and a system architecture as presented in Section 2, the following problem has to be solved: construct a correct static schedule for the TT tasks and ST messages (a schedule which meets all time constraints related to these activities) and conduct a schedulability analysis in order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

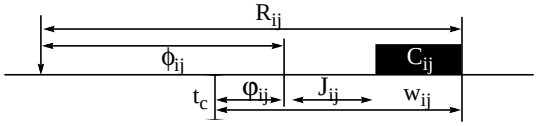
In Section 3.1 we present the schedulability analysis for a set of ET tasks and DYN messages, considering a fixed given static schedule of TT tasks and ST messages. In Section 3.2 we discuss the construction of the static schedule which is driven by the objective of achieving global schedulability of the system. 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 [14], 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 R_{ij} of an activity τ_{ij} is the time measured from the occurrence of the associated event until the completion of τ_{ij}. Each ET



a) Tasks with offsets



$$R_{ij} = w_{ij} + \phi_{ij} - \varphi_{ij} - (p-1)T_i$$

b) Response time and busy period w for task τ_{ij}

Figure 3. Model of the event-triggered sub-system

activity τ_{ij} has a best case response time $R_{b,ij}$. The worst case response time R_{ij} of an activity τ_{ij} occurs when τ_{ij} is released at the same time moment t_c together with all possible higher priority activities on $Proc_{ij}$ [11]. The moment t_c is called critical instant and it represents the starting point of the busy window w_{ij}, a time interval which ends when τ_{ij} finishes execution (Figure 3.b). During the busy window w_{ij}, processor $Proc_{ij}$ 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 [11] 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 τ_{ik} in Γ_i and all job instances p of τ_{ij} which can appear in the busy window w_{ij}:

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

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}.

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 [11], 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.

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:

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

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

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

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

where $A^q_{ij}(w)$ is the total available CPU-time on $Proc_{ij}$ 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 ST 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 [12]). 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.

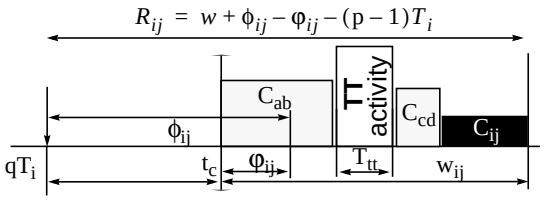
After performing the schedulability analysis, we can check if $R_{ij} \le D_{ij}$ for all the ET tasks. If this is the case, the set of ET activities is schedulable. In order to drive the global scheduling process, as it will be explained in the next section, it is not sufficient to test if the task 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 a cost function similar with the one described in [14]:

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

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 cost 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.



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

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 $T_{SS} = LCM(T_1, T_2, ..., T_N)$. 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, respectively into a static bus-slot associated to that processor on which the sender of the message is executed. 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 [13]. Let us consider a particular task τ_{ij} selected from the ready list to be scheduled. θ_1 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 the processor $Proc_{ij}$ is free. $\theta_2 = ALAP(\tau_{ij})$ is the latest time when τ_{ij} can be scheduled. With only the TT tasks in the system, the straight forward solution would be to schedule τ_{ij} at θ_1. 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 τ_{ij} in such a position inside the interval $[\theta_1, \theta_2]$ that the chance to finally get a globally schedulable system is maximised.

In order to find the right position for τ_{ij}, we try k different alternatives:

$$\text{start\_time}(\tau_{ij}) = \theta_1 + \frac{\theta_2 - \theta_1}{k-1} \times x, \quad x = 0, .., k-1$$

For each alternative we perform the schedulability analysis of the ET task set considering the influence from those TT tasks which are already scheduled. We will select that start time for τ_{ij} which produces the minimum value for *Cost* (see Section 3.1).

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 $Cost \le 0$, the global system scheduling has succeeded.

There are two aspects to be mentioned:

1. How large should be the number k of alternatives to be tried for the placement of a task τ_{ij}? If k is large, we will increase the chance to generate a schedulable system, however the execution time for the scheduling algorithm could become

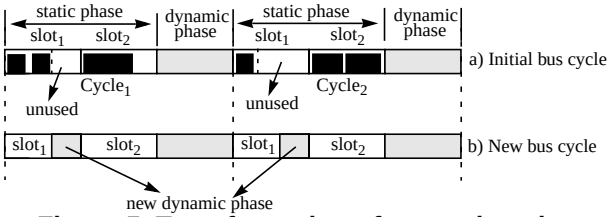


Figure 5. Transformation of unused static bandwidth into dynamic phases

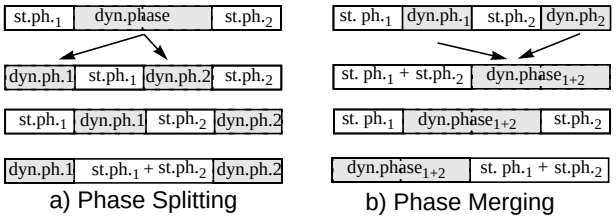


Figure 6. Operations on dynamic phases

unacceptably large. At the same time, for relatively large intervals $[\theta_1, \theta_2]$ it is reasonable to try more alternatives than for tight intervals. In our current implementation we set the number k as follows:

$$k = max(\frac{(\theta_2 - \theta_1)}{\Delta} \times N, 1)$$

The value Δ is determined at the beginning of the scheduling process after an initial ASAP and ALAP schedule has been constructed for the TT tasks. Δ is the average of (ALAP(τ_{ij})-ASAP(τ_{ij})) over all TT tasks τ_{ij}. Thus, the value of k will oscillate around the value N, getting larger values for long intervals $[\theta_1, \theta_2]$ and small values for short intervals. The value N is set by the designer. In Section 6 we present some experimental results showing the influence of N on the scheduling time and on the quality of the generated schedules.

2. 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.

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],[7],[17]. 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.

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 in a dynamic phase (see Figure 5).

After this initial step, various bus cycle configurations are explored by splitting and merging bus phases. Figure 6 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 *Cost* introduced in Section 3.1

6. Experimental Results

For evaluation of our scheduling and analysis algorithm and of the bus access optimization heuristic, we generated a total of 80 applications. Each application consisted of 80 tasks mapped on 10 processor nodes. The percentage of ET tasks was 40% of the total number of tasks for half of the application set and 60% for the other half. Processor utilisation was 60% and 80%. The bus bandwidth was equally divided between the dynamic and the static phases. All experiments were run on an AMD Athlon 850MHz PC.

The first set of experiments concerns the holistic scheduling algorithm and, in particular, the trade-off between speed and quality. In Section 3 we have shown that the number of alternatives considered for the placement of a TT task depends on the coefficient N. A larger number of such alternatives improves the quality of the schedule but increases the schedule time. Figure 7

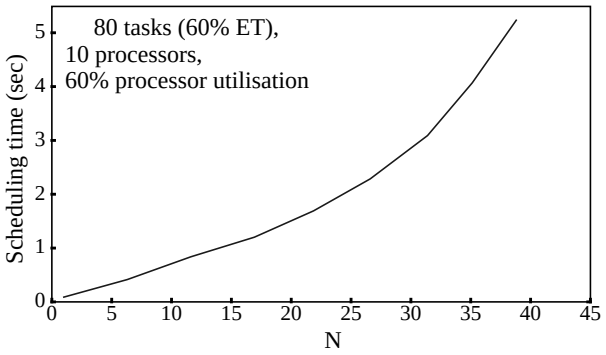


Figure 7. Optimization time

shows how the scheduling time grows with *N*. When following Figure 8, however, we can observe that the quality of the schedule (expressed through the function *Cost*) at the beginning very quickly improves with growing *N*, and then practically keeps at a constant level. For all experiments a value of *N* around 5 already provided for the best quality schedule.

The next 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 is between 24% and 34%, with an average optimization time just above 1 minute. 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 32 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 0.57 seconds and resulted in a correct static schedule and an unschedulable ET domain. After running the bus access optimization, the schedulability (expressed in terms of the function *Cost*) has improved by more than one order of magnitude, resulting in a completely schedulable system. The optimization was solved in less than 2 minutes.

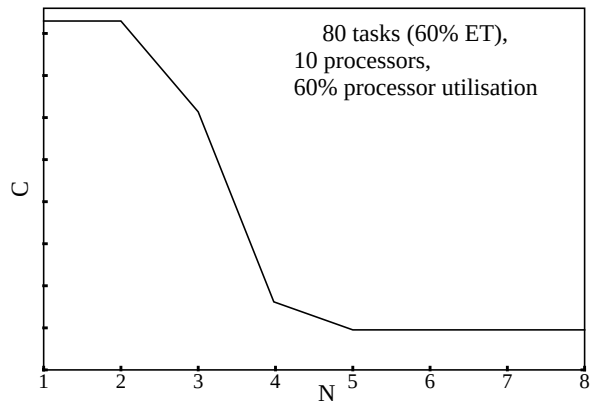


Figure 8. Schedulability improvement with *N*

Processor utilisation	60% ET tasks		40% ET tasks	
	schedulability improvement	optimization time (sec)	schedulability improvement	optimization time (sec)
60%	34%	67.4	25%	109.8
80%	29%	64.7	24%	71.5

Table 1: Bus Optimization Results

7. Conclusions

Distributed embedded systems based on mixed static/dynamic communication protocols are becoming the 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. Ernst, "Codesign of Embedded Systems: Status and Trends", IEEE Design&Test of Comp., April-June, 1998.

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

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

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

[11] J. C. Palencia, M. G. Harbour, “Schedulability Analysis for Tasks with Static and Dynamic Offsets”, Proceedings of the 9th IEEE Real-Time Systems Symposium, 1998.

[12] P. Pedreiras, L. Almeida, “Combining Event-Triggered and Time-Triggered Traffic in FTT-CAN: Analysis of the Asynchronous Messaging System”, WFCS, 2000.

[13] 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.

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

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

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

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

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