Downloaded from orbit.dtu.dk on: May 23, 2019

DTU Library

Optimization of Hierarchically Scheduled Heterogeneous Embedded Systems

Pop, Traian; Pop, Paul; Eles, Petru; Peng, Zebo

Published in:
International Conference on Embedded and Real-Time Computing Systems and Applications

Link to article, DOI:
10.1109/RTCSA.2005.67

Publication date:
2005

Document Version
Publisher's PDF, also known as Version of record

Link back to DTU Orbit

Citation (APA):
Pop, T., Pop, P., Eles, P., & Peng, Z. (2005). Optimization of Hierarchically Scheduled Heterogeneous Embedded Systems. In International Conference on Embedded and Real-Time Computing Systems and Applications (pp. 67-71) https://doi.org/10.1109/RTCSA.2005.67

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Optimization of Hierarchically Scheduled Heterogeneous Embedded Systems

Traian Pop, Paul Pop, Petru Eles, Zebo Peng
Dept. of Computer and Information Science
Linköping University, SE-581 83, Linköping, Sweden
E-mail: {trapo, paupo, petel, zebpe}@ida.liu.se

Abstract

We present an approach to the analysis and optimization of heterogeneous distributed embedded systems for hard real-time applications. The systems are heterogeneous not only in terms of hardware components, but also in terms of communication protocols and scheduling policies. When several scheduling policies share a resource, they are organized in a hierarchy. In this paper, we address design problems that are characteristic to such hierarchically scheduled systems: assignment of scheduling policies to tasks, mapping of tasks to hardware components, and the scheduling of the activities. We present algorithms for solving these problems. Our heuristics are able to find schedulable implementations under limited resources, achieving an efficient utilization of the system.

1. Introduction

There has been a long debate in the real-time and embedded systems communities concerning the advantages and disadvantages of different scheduling approaches [1, 7, 9, 21]. *Static cyclic scheduling* (SCS) has the advantage of *predictability* and testability [9]. However, such static approaches lack the *flexibility* offered by event-driven approaches such as *fixed priority scheduling* (FPS) and *earliest deadline first* (EDF). EDF is optimal on single processor systems, and in general leads to a high, and thus efficient, resource utilization [7]. In addition, advances in the area of priority-based preemptive scheduling show that predictable applications with hard real-time guarantees can also be handled with FPS and EDF strategies [1, 18].

An interesting comparison of scheduling approaches, from a more industrial, in particular automotive perspective, can be found in [12]. Their conclusion is that one has to choose the right scheduling approach depending on the particularities of the scheduled processes. This means not only that there is no single "best" approach to be used, but also that inside a certain application *several scheduling approaches should be used together*. When several scheduling policies share a resource, they can be organized in a *hierarchy*, where a higher level scheduler controls the activation of a lower level scheduler [10].

The same is true for the communication infrastructure. A survey and comparison of communication protocols for safety-critical embedded systems is available in [17]. On one hand, there are protocols that schedule the messages statically, at predetermined moments in time, for example, the Time-Triggered Protocol (TTP) [9]. On the other hand, there are communication protocols where message scheduling is performed dynamically, in response to an event, such as Controller Area Network (CAN) [3]. However, there is also a hybrid type of communication protocols, such as the one suggested in [14]. A mixed protocol (FlexRay) has been proposed by a consortium, to be used in automotive applications [6], allowing the sharing of the bus by event-driven and time-driven messages. In [5], the authors describe the "Universal Communication Model" (UCM), a framework for modelling at a high level of abstraction the communication infrastructure in automotive applications.

There is a large quantity of research [2, 18] related to scheduling and schedulability analysis, but few approaches have been proposed that can systematically handle heterogeneous scheduling policies on distributed systems. Some researchers have shown how two distinct scheduling policies can be combined in a system. Tindell provides an analysis for FPS processes communicating over a time-division multiple access (TDMA) bus [19]. When several scheduling policies are used together, they can also share the same resource. Our previous research has shown how SCS and FPS can be analyzed together in a system [16]. Gonzaléz Harbour has recently shown how FPS and EDF can be analyzed together when sharing the same processor [13].

Although analysis approaches exist for many protocols [17], few of them can handle hybrid protocols. Almeida has analyzed event- and time-driven traffic in FTT-CAN [14]. Our previous research has provided an analysis and optimization for the UCM [16].

In this paper, we are interested in the analysis and optimization of distributed embedded systems where several scheduling policies can be used for tasks and messages. Hence, in this paper we support combinations of SCS, FPS and EDF scheduling policies on the same node or on different nodes. Our focus in this paper is on design optimization problems characteristic to such systems. Our algorithms derive optimized implementations such that the timing constraints of the final implementation are guaranteed.

- First, we have extended our previous analysis approach [16] to handle hierarchically scheduled systems. We have proposed a holistic scheduling algorithm that builds the SCS tables and determines the FPS priorities, and provides a global analysis of the system.
- Once we can evaluate the schedulability of an implementation, we have developed an optimization approach to the assignment of scheduling policies to tasks and the mapping of tasks to the hardware nodes of the architecture.

This paper is organized in several sections. The next two sections present the hardware and software architectures considered, and the abstract model of the application. Section 4 presents the design optimization problems addressed, while Section 5 proposes optimization and scheduling algorithms for solving these problems. The optimization techniques are evaluated in Section 6, where we also discuss a real-life example. The last section presents our conclusions.

2. System Architecture

We consider architectures consisting of nodes connected by a unique broadcast communication channel (see Figure 1.a).

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. The real-time kernel contains three schedulers, for SCS, FPS, and EDF, organized *hierarchically* (Figure 1.c).

1. The top-level scheduler is a SCS scheduler, which is responsible for the activation of SCS tasks and transmission of SCS messages based on a schedule table, and for the activation of the FPS scheduler. Thus, SCS tasks and messages are time-triggered (TT), i.e., activated at predetermined points in time, and non preemptable.
2. The FPS scheduler activates FPS tasks and transmits FPS messages based on their priorities, and activates the EDF scheduler. Tasks and messages scheduled using FPS are event-triggered (ET), i.e., initiated whenever a particular event is noted, and are pre-emptable.
3. The EDF scheduler activates EDF tasks and sends EDF messages based on their deadlines. EDF tasks and messages are ET and pre-emptable.

When several tasks are ready on a node, the task with the highest priority is activated, and pre-empts the other tasks. Let us consider the example in Figure 1.d, where we have six tasks sharing the same node. Tasks τ_1 and τ_6 are scheduled using SCS, τ_2 and τ_5 are scheduled using FPS, while tasks τ_3 and τ_4 are scheduled with EDF. The priorities of the FPS and EDF tasks are indicated in the figure. The arrival time of these tasks is depicted with an upwards pointing arrow. Under these assumptions, Figure 1.d presents the worst-case response times of each task. The SCS tasks, τ_1 and τ_6, will never compete for a resource because their synchronization is performed based on the schedule table. Moreover, since SCS tasks are non preemptable and their start time is offline fixed in the schedule table, they also have the highest priority (denoted with priority level "0" in the figure). FPS and EDF tasks can only be executed in the *slack* of the SCS schedule table.

FPS and EDF tasks are scheduled based on their priorities. Thus, a higher priority task such as τ_2 will interrupt a lower priority task such as τ_3. In order to integrate EDF tasks with FPS, we use the approach in [13], by assuming that FPS priorities are not unique, and that a group of tasks having the same FPS priority on a processor are to be scheduled with EDF. Thus, whenever the FPS scheduler notices ready tasks that share the same priority level, it will invoke the EDF scheduler which will schedule those tasks based on their deadlines. Such a situation is present in Figure 1.d for tasks τ_3 and τ_4. There can be several such

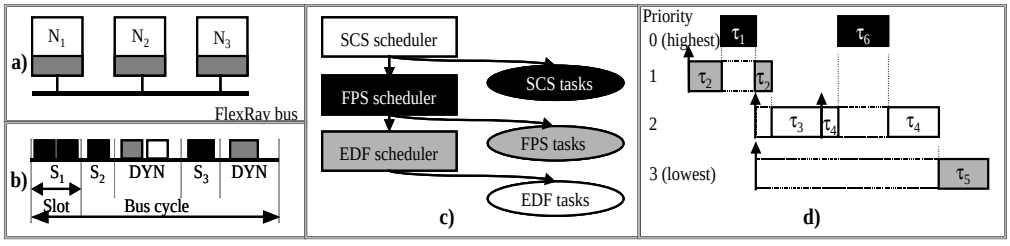

Figure 1. System Architecture

EDF priority levels within a task set on a processor. Higher priority EDF tasks can interrupt lower priority FPS tasks (as is the case with τ_3 and τ_4 which preempt τ_5) and EDF tasks. Lower priority EDF tasks will be interrupted by both higher priority FPS and EDF tasks, and SCS tasks.

We model the bus access scheme using the Universal Communication Model [5]. The bus access is organized as consecutive cycles, each with the duration T_{bus}. We consider that the communication cycle is partitioned into static (ST) and dynamic (DYN) phases (Figure 1.b).

- ST phases consist of time slots, and during a slot only the node associated to that particular slot is allowed to transmit SCS messages. The transmission times of SCS messages are stored in a schedule table.
- During a DYN phase, all nodes are allowed to send messages and the conflicts between nodes trying to send simultaneously are solved by an arbitration mechanism which allows the transmission of the message with the highest priority. Hence, the ET messages are organized in a prioritized ready queue. The integration of EDF messages within such a priority-based arbitration mechanism has been detailed in [11].

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.

3. Application Model

We model an application A as a set of directed, acyclic graphs $G_i(V_i, E_i) \in A$. A node $\tau_{ij} \in V_i$ represents the j-th task in G_i. An edge $e_{ijk} \in E_i$ from τ_{ij} to τ_{ik} indicates that the output of τ_{ij} is the input of τ_{ik}. A task becomes ready after all its inputs have arrived and it issues its outputs when it terminates. The communication time between tasks mapped on the same processor is considered to be part of the task worst-case execution time and is not modeled explicitly. Communication between tasks mapped to different processors is performed by message passing over the bus. Such message passing is modeled as a communication tasks inserted on the arc connecting the sender and the receiver task.

Let P be the set of tasks in A. The scheduling policy to be applied to each task is given by a function $S: P \rightarrow \{SCS, FPS, EDF\}$. The mapping of a task $\tau_{ij} \in P$ is given by a function $M: P \rightarrow N$, where N is the set of nodes in the architecture. For a task $\tau_{ij} \in P$, $M(\tau_{ij})$ is the node to which τ_{ij} is assigned for execution. Each task τ_{ij} can potentially be mapped on several nodes. Let $N_{\tau_{ij}} \subseteq N$ be the set of nodes to which τ_{ij} can potentially be mapped. We consider that for each $N_k \in N_{\tau_{ij}}$, we know the worst-case execution time $C_{\tau_{ij}}^{N_k}$ of task τ_{ij}, when executed on N_k. We also consider that the size of the messages is given (which can be directly converted into communication time on the particular bus). Tasks and messages activated based on events also have a priority, $Prio_{ij}$[1].

All tasks and messages in a task graph G_i have the same period $T_{ij} = T_{G_i}$ which is the period of the task graph. We consider that a deadline D_{G_i} is given for each task graph G_i. In addition, tasks can have associated individual release times and deadlines. If communicating tasks are of different periods, they are combined into a merged graph capturing all tasks activations for the hyper-period (LCM of all periods).

4. Design Optimization Problems

Considering the type of applications and systems described in the previous section, several design optimization problems can be addressed. In this paper, we address problems which are characteristic to hierarchically scheduled distributed applications. In particular, we are interested in the following issues:

- assignment of scheduling policies to tasks;
- mapping of tasks to the nodes of the architecture;
- optimization of the access to the communication infrastructure;
- scheduling of tasks and messages.

The goal is to produce an implementation which meets all the timing constraints of the application.

In this paper, by scheduling policy assignment (SPA) we denote the decision whether a certain task should be scheduled with SCS, FPS or EDF. Mapping a task means assigning it to a particular hardware node.

4.1 Scheduling Policy Assignment

Very often, the SPA and mapping decisions are 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. Moreover, due to legacy constraints, the mapping and scheduling policy of certain processes might be fixed.

Thus, we denote with $P_{SCS} \subseteq P$ the subset of tasks for which the designer has assigned SCS, $P_{FPS} \subseteq P$ contains tasks to which FPS is assigned, while $P_{EDF} \subseteq P$ contains those tasks for which the designer has decided to use the EDF scheduling policy. There are tasks, however, which do not exhibit certain particular features or requirements which obviously lead to their scheduling as SCS, FPS or EDF activities. The subset $P^+ = P \setminus (P_{SCS} \cup P_{FPS} \cup P_{EDF})$ of tasks could be assigned any scheduling policy. Decisions concerning the SPA to this set of activities can lead to various trade-offs concerning, for example, the schedulability properties of the system, the size of the schedule tables, the utilization of resources, etc.

1.As noted previously, ET tasks sharing the same priority are scheduled with EDF.

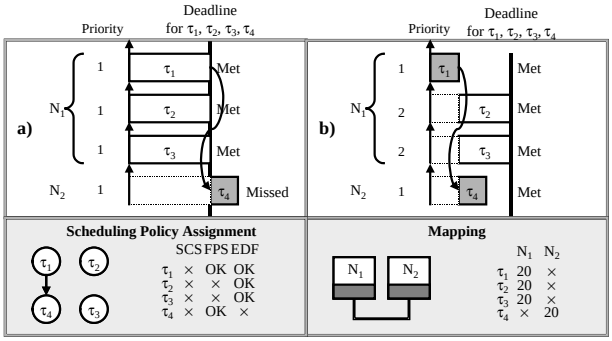

Figure 2. Scheduling Policy Assignment Example

Figure 2 shows an optimization example for the assignment of FPS and EDF policies. The application is composed of four tasks running on two nodes. Tasks τ_1, τ_2 and τ_3 are mapped on node N_1, and have the same priority "1", while task τ_4 runs on N_2. Task τ_4 is data dependent of task τ_1. All tasks in the system have the same worst case-execution times (20 ms), deadlines (60 ms) and periods (80 ms). Tasks τ_2 and τ_3 are scheduled with EDF, τ_4 with FPS, and we have to decide the scheduling policy for τ_1, between EDF and FPS.

If τ_1 is scheduled according to EDF, thus sharing the same priority level "1" with the tasks on node N_1, then task τ_4 misses its deadline (Figure 2.a). Note that in the time line for node N_1 in Figure 2 we depict several worst-case scenarios: each EDF task on node N_1 is depicted considering the worst-case interference from the other EDF tasks on N_1. However, the situation changes if on node N_1 we use FPS for τ_1 (i.e., changing the priority levels of τ_2 and τ_3 from "1" to "2"). Figure 2.b shows the response times when task τ_1 has the highest priority on N_1 (τ_1 retains priority "1") and the other tasks are running under EDF at a lower priority level (τ_2 and τ_3 share lower priority "2"). Because in this situation there is no interference from tasks τ_2 and τ_3, the worst-case response time for task τ_1 decreases considerably, allowing task τ_4 to finish before its deadline, so that the system becomes schedulable.

4.2 Mapping and Bus Access Optimization

The designer might have already decided the mapping for a part of the tasks. For example, certain tasks, due to constraints such as having to be close to sensors/actuators, have to be physically located in a particular hardware unit. They represent the set $P^M \subseteq P$ of already mapped tasks. Consequently, we denote with $P^* = P \setminus P^M$ the tasks for which the mapping has not yet been decided.

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 optimized such that they fit the particular application and the timing requirements. Parameters to be optimized are the number of static (ST) and dynamic (DYN) 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. Let us denote such a bus configuration with B.

4.3 Exact Problem Formulation

As an input we have an application A given as a set of task graphs (Section 3) and a system architecture consisting of a set N of nodes (Section 2). As introduced previously, P_{SCS}, P_{FPS} and P_{EDF} are the sets of tasks for which the designer has already assigned SCS, FPS or EDF scheduling policy, respectively. Also, P^M is the set of already mapped tasks.

As part of our problem, we are interested to:

- find a SPA S for tasks in $P^+ = P \setminus (P_{SCS} \cup P_{FPS} \cup P_{EDF})$;
- decide a mapping for tasks in $P^* = P \setminus P^M$;
- determine a bus configuration B;
- find a schedule table for the SCS tasks and priorities of FPS and EDF tasks;

such that imposed deadlines are guaranteed to be satisfied.

In this paper, we will consider the assignment of scheduling policies at the same time with the mapping of tasks to processors. Moreover, to simplify the presentation we will not discuss the optimization of the communication channel. Such an optimization can be performed with the techniques we have proposed in [16].

5. Design Optimization Strategy

The design problem formulated in the previous section is NP-complete (the scheduling subproblem, in a simpler context, is already NP-complete [20]). Therefore, our strategy, outlined in Figure 3, is to divide the problem into several, more manageable, subproblems. Our OptimizationStrategy has three steps:

1.In the first step (lines 1–3) we decide on an initial bus access configuration B^0 (function InitialBusAccess), and an initial policy assignment S^0

```
OptimizationStrategy(A)
1   Step 1: B^0 = InitialBusAccess(A)
2           (M^0, S^0) = InitialMSPA(A, B^0)
3           if HolisticScheduling(A, M^0, B^0, S^0) == schedulable then stop end if
4   Step 2: (M, S, B) = MSPAHeuristic(A, M^0, B^0)
5           if HolisticScheduling(A, M, S, B) == schedulable then stop end if
6   Step 3: B = BusAccessOptimization(A, M, S, B)
7           HolisticScheduling(A, M, B, S)
end OptimizationStrategy
```

Figure 3. The General Strategy

and mapping M^0 (function InitialMSPA). The initial bus access configuration (line 1) is determined, for the ST slots, by assigning in order nodes to the slots ($S_i = N_i$) and fixing the slot length to the minimal allowed value, which is equal to the length of the largest message in the application. Then, we add at the end of the TT slots an equal length single ET phase. The initial scheduling policy assignment and mapping algorithm (line 2 in Figure 3) maps tasks so that the amount of communication is minimized. The initial scheduling policy of tasks in P^+ is set to FPS. Once an initial mapping, scheduling policy assignment and bus configuration are obtained, the application is scheduled using the HolisticScheduling algorithm (line 3) outlined in Section 5.1.

2. If the application is schedulable the optimization strategy stops. Otherwise, it continues with the second step by using an iterative improvement mapping and policy assignment heuristic, MSPAHeuristic (line 4), presented in Section 5.2, to improve the partitioning and mapping obtained in the first step.
3. If the application is still not schedulable, we use, in the third step, the algorithm BusAccessOptimization presented in [16], which optimizes the access to the communication infrastructure (line 6). If the application is still unschedulable, we conclude that no satisfactory implementation could be found with the available amount of resources.

5.1 Holistic Scheduling

Once a partitioning and a mapping is decided, and a communication configuration is fixed, the tasks and messages have to be scheduled. For the SCS tasks this means building the schedule tables, while for the FPS tasks and EDF task groups, the priorities have to be determined and their schedulability has to be analyzed.

The basic idea is that SCS tasks are schedulable if it is possible to build a schedule table such that the timing requirements are satisfied. For FPS and EDF tasks, the answer whether or not they are schedulable is given by a *schedulability analysis*. In this paper, we use a *response time analysis*, where the schedulability test consists of the comparison between the worst-case response time R_{ij} of a task τ_{ij} and its deadline D_{ij}. In order to drive the process of finding a schedulable system, which is presented in the next sections, 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 application. For this purpose we use a cost function similar to the one described in [16].

The problem of finding a schedulable system has to consider two aspects:

1. When performing the schedulability analysis for the FPS and EDF tasks and messages, one has to take into consideration the interference from the statically scheduled activities in the system.
2. Among the possible correct schedules for SCS activities, it is important to build one which favours as much as possible the schedulability degree of FPS and EDF activities.

In Section 5.1.1 we present the schedulability analysis for a set of FPS and EDF tasks and messages, considering a fixed given static schedule for SCS activities. In Section 5.1.2 we present the actual holistic scheduling algorithm which constructs the static schedule and is driven by the objective of achieving a global schedulability of the system.

5.1.1. Response Time Analysis

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 SCS tasks are communicating only through TT phases, while the communication among FPS and EDF tasks is only through ET phases. This is not an inherent limitation of our approach and the analysis we have developed and implemented supports the general model .

A FPS or EDF task is activated by an associated event. Each task τ_{ij} 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). Offsets and jitters are the means by which dependencies among tasks are modelled for the schedulability analysis. The response time R_{ij} of a task τ_{ij} is the time measured from the occurrence of the associated event until the completion of τ_{ij}. Each task τ_{ij} has a best case response time $R_{b,ij}$.

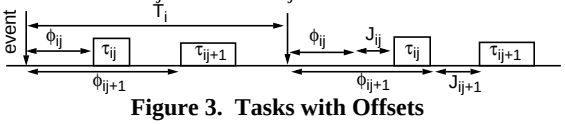

Figure 3. Tasks with Offsets

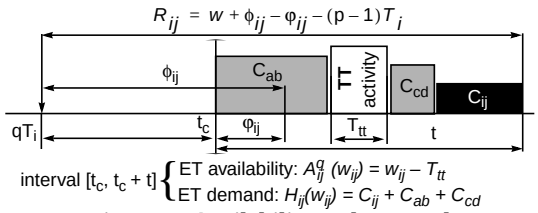

Figure 4. Availability and Demand

In order to determine if a hierarchically scheduled system is schedulable, we used as a starting point the schedulability analysis algorithm for EDF-within-FPS systems, developed in [13]. In this section, we present our extension to this algorithm, which allows us to compute the worst case response times for the FPS and EDF activities when they are interfered by the SCS activities.

In [13], the authors have developed a schedulability analysis algorithm for ET tasks running under a hierarchical FPS/EDF scheduling policy. Response times for the tasks are obtained using workload equations:

- For FPS tasks, the worst case response times are influenced only by higher priority tasks, so the completion time of an activation p of task τ_{ab} is given by the following recursion:

$$w_{ab}^{k+1}(p) = B_{ab} + p \cdot C_{ab} + \sum_{\forall \tau_{ij} \mid Prio_{ij} \geq Prio_{ab}} \left\lceil \frac{w_{ab}^{k}(p) + J_{ij}}{T_{ij}} \right\rceil \cdot C_{ij} \quad (1)$$

where p is the number of activations of τ_{ab} in the busy period, B_{ab} is the blocking time for τ_{ab} (see [13] for a discussion regarding the blocking time), and J_{ij} and T_{ij} are the maximum release jitter and the period of τ_{ij}, respectively. The worst case response time R_{ab} is then computed as the maximum for all possible values of

$$R_{ab}(p) = w_{ab}(p) - (p-1)T_{ab} + J_{ab}. \quad (2)$$

- For EDF tasks, the worst case response times are influenced by higher priority tasks and by EDF tasks running at the same priority level as the task under analysis:

$$w_{ab}(p) = B_{ab} + p \cdot C_{ab} + \sum_{Prio_{ab} = Prio_{ij}} W_{ij}(w_{ab}^{A}(p), D^{A}(p)) + \sum_{Prio_{ij} > Prio_{ab}} W_{ij}(w_{ab}^{A}(p)) \quad (3)$$

In the previous equation, the third term represents the interference from EDF tasks with the same priority, while the last term captures the interference from higher priority FPS and EDF tasks. Furthermore,

$$W_{ij}(t) = \left\lceil \frac{t + J_{ij}}{T_{ij}} \right\rceil \cdot C_{ij}$$

and $D^A(p)$ is the deadline of activation number p, when the first activation of τ_{ab} occurs at time A:

$$D_{ab}^{A}(p) = A - J_{ab} + (p-1)T_{ab} + D_{ab}. \quad (4)$$

The analyzed instants A are extracted from situations in which the task under analysis τ_{ab} has the deadline larger or equal than the deadline of the other tasks in the system. The worst case response time R_{ab} for a task running under EDF-within-FPS is then computed as the maximum for all possible values of

$$R_{ab}(p) = w^{A}{}_{ab}(p) - (p-1)T_{ab} + J_{ab} - A \quad (5)$$

A similar technique is used in the more complex case of offset-based analysis. However, regardless of the analysis used, the technique has to be enhanced to take into consideration an existing static schedule, allowing us to analyze hierarchically scheduled systems that use a combination of SCS, FPS and EDF scheduling policies.

Our extension takes into consideration the interference produced by an existing static schedule when computing the worst-case response times of FPS and EDF activities scheduled using EDF-within-FPS.

First we introduce the notion of *ET demand* associated with an FPS or EDF activity τ_{ij} on a time interval t as the maximum amount of CPU time or bus time which can be *demanded* by higher or equal priority ET activities and by τ_{ij} during the time interval t. In Figure 4, the ET demand of the task τ_{ij} during the busy window t is denoted with $H_{ij}(t)$, 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 interval t, we define the *availability* as the processing time which is not used by statically scheduled activities. In Figure 4, the CPU availability for the analyzed interval is obtained by substracting from t the amount of processing time needed for the SCS activities.

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

$$H_{ij}(t) = w_{ij}(t). \quad (6)$$

During a time interval t, the availability A_{ij} associated with task τ_{ij} is:

$$A_{ij}(t) = min[A_{ij}^{ab}(t)], \forall \tau_{ab} \in TT \mid M(\tau_{ab}) = M(\tau_{ij}), \quad (7)$$

where $A_{ij}^{ab}(t)$ is the total available CPU-time on processor $M(\tau_{ij})$ in the interval $[\phi_{ab}, \phi_{ab} + t]$, and ϕ_{ab} is the start time of task τ_{ab} as recorded in the static schedule table.

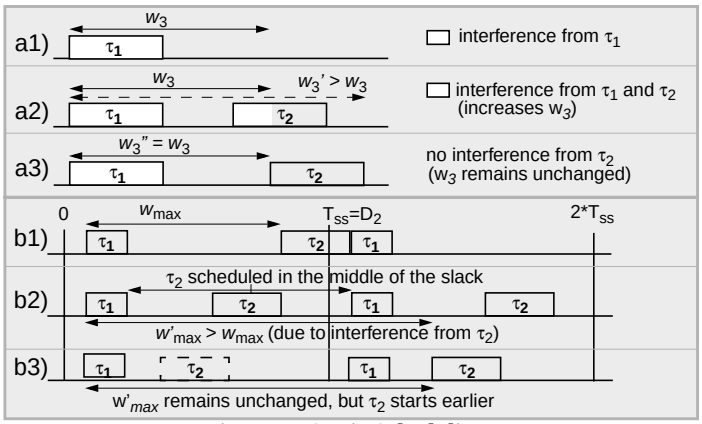

Figure 5. Static Scheduling

The discussion above is, in principle, valid for both types of ET tasks (i.e., FPS and EDF tasks) and messages. However, there exist two important differences. First, messages do not preempt each other, therefore, in the demand equation the blocking term will be non-zero, equal with the largest transmission time of any ET message. Second, the availability for a message is computed by substracting from t the length of the ST slots which appear during the considered interval; moreover, because an ET 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 ET message).

Our schedulability analysis algorithm determines the length of a busy window w_{ij} for FPS and EDF tasks and messages 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 in Section 5.1.2. It is important to notice that this process includes both tasks and messages and, thus, the resulted response times of the FPS and EDF tasks are computed by taking into consideration the delay induced by the bus communication.

5.1.2. Static Scheduling

As mentioned in the beginning of Section 5.1, building the static cyclic schedule for the SCS activities in the system has to be performed in such a way that the interference imposed on the FPS and EDF activities is minimum. The holistic scheduling algorithm is presented in Figure 6. For the construction of the schedule table with start times for SCS tasks and messages, we adopted a list scheduling-based algorithm [4] which iteratively selects tasks and schedules them appropriately

A ready list contains all SCS tasks and 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 (Figure 6, line 2) 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. 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} are finished and processor $M(\tau_{ij})$ is free. The moment $ALAP_{ij}$ is the latest time when τ_{ij} can be scheduled. With only the SCS 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 FPS and EDF 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 maximized.

In order to consider only a limited number of possible positions for the start time of a SCS task τ_{ij}, we take into account the information obtained from the schedulability analysis described in Section 5.1.1, which allows us to compute the response times of ET (i.e., FPS and EDF) tasks. We started from the observation that statically scheduling a SCS task τ_{ij} so that the length of busy-period of an ET activity is not modified will consequently lead to unchanged worst-case response time for that ET task. This can be achieved by providing for enough available processing time between statically scheduled tasks so that the busy period of the ET task does not increase. For example, in Figure 5.a we can see how statically scheduling two SCS tasks τ_1 and τ_2 influences the busy period w_3 of a FPS (or EDF) task τ_3. Figure 5.a1, presents the system with only τ_1 scheduled, situation for which the busy-period w_3 is computed. Figure 5.a2 shows how scheduling another SCS task τ_2 too early decreases the availability during the interval $[\phi_1, \phi_1 + w_3]$, and consequently leads to an increase of w_3 and R_3, respectively. Such a situation is avoided if the two SCS tasks are scheduled like in Figure 5.a3, where no extra interference is introduced in the busy period w_3. However, during the static scheduling, we have to consider two aspects:

```
HolisticScheduling(A, M, B, S)
1    while TT_ready_list is not empty
2        select τij from TT_ready_list
3        if τij is a task then
4            schedule_task(τij, M(τij))
5        else // τij is a message
6            ASAP schedule τij in slot(M(τij))
7        end if
8    end while
9    procedure schedule_task(τij, M(τij))
10       schedule τij in the middle of the slack on M(τij)
11       compute ET respose times and w'max
12       move τij earlier without increasing w'max
end HolisticScheduling
```

Figure 6. Holistic Scheduling Algorithm

1.The interference with the FPS and EDF activities should be minimized;
2.The deadlines of TT activities should be satisfied.

The technique presented in Figure 5.a takes care only of the first aspect, while ignoring the second. One may notice that scheduling a SCS task later increases the probability that we will not be able to find feasible start times for that particular task or for the SCS tasks which depend on τ_2 and are not scheduled yet (for example, in Figure 5.b1, task τ_2 misses its deadline and the resulted schedule is not valid). We reduce such a risk by employing the technique presented in Figure 5.b2-b3, where we first schedule the second task so that we maximize the continuous slack between the jobs of tasks τ_1 and τ_2; for this reason, we place τ_2 in the middle of the slack between the last SCS task in the first period of the static schedule (the first job of task τ_1), and the first task scheduled in the second period (the second job of task τ_1). In such a situation, the maximum busy period w_{max} of the ET tasks may increase due to interference from task τ_2 (Figure 5.b2). However, considering that such an increase is acceptable (in the sense that no ET tasks miss their deadlines), then we can now improve the probability of finding a valid static schedule by scheduling the task τ_2 earlier in time, as long as the maximum ET busy period w_{max} does not increase (Figure 5.b3)

The scheduling algorithm is presented in Figure 6. If the selected SCS activity extracted from the *ready_list* is a task τ_{ij}, then the task is first scheduled in the middle of the slack at the end of the period T_{ss} of the static schedule (line 10). In order to determine the response times of the ET activities and the maximum busy period w_{max} in the resulted system, the scheduled application is analyzed using the technique in Section 5.1.1 (line 11). The value obtained for w_{max} is then used for determining how early the task τ_{ij} can be scheduled without increasing the response times of the ET tasks (line 12). When scheduling a 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 (line 6). If all SCS tasks and messages have been scheduled and the schedulability analysis for the ET tasks indicates that all ET activities meet their deadlines, then the global system scheduling has succeeded.

5.2 Mapping & Scheduling Policy Assignment Heuristic

In Step 2 of our optimization strategy (Figure 3), the following design transformations are performed with the goal to produce a schedulable system implementation:

- change the scheduling policy of a task;
- change the mapping of a task;
- change the priority level of a FPS of EDF task.

Our optimization algorithm is presented in Figure 7 and it implements a greedy approach in which every task in the system is iteratively mapped on each node (line 4) and assigned to each scheduling policy (line 8), under the constraints imposed by the designer. The next step involves adjustments to the bus access cycle (line 10), which are needed for the case when the bus cycle configuration cannot handle the minimum requirements of the current internode

```
MSPAHeuristic(A, M, B, S)
1   for each activity τij in the system do
2       for each processor Ni ∈ N in the system do
3           if τij in P+ then -- can be remapped
4               M(τij) = Ni
5           end if
6           for policy = SCS, FPS do
7               if τij in P+ then -- the scheduling policy can be changed
8                   S(τij) = policy
9               end if
10              adjust bus cycle(A, M, B, S)
11              recompute FPS priority levels
12              for all FPS tasks τab sharing identical priority levels do
13                  S(τab) = EDF
14              end for
15              HolisticScheduling(A, M, B, S)
16              if δA < best_δA then
17                  best_policyij = S(τij); best_processorij = M(τij)
18                  best_δA = δA
19              end if
20              if δA < 0 then
21                  return best (M, B, S)
22              end if
23          end for
24      end for
25  end for
end MSPAHeuristic
```

Figure 7. Policy Assignment and Mapping Heuristic

communication. Such adjustments are mainly based on enlargement of the static slots or dynamic phases in the bus cycle, and are required in the case the bus has to support larger messages than before. New messages may appear on the bus due to, for example, remapping of tasks. For more details on the subject of bus access optimization and adjustment, the reader is referred to [16].

Before the system is analyzed for its timing properties, our heuristic also tries to optimize the priority assignment of tasks running under FPS (line 11). The state of the art approach for such a task is the HOPA algorithm for assigning priority levels to tasks in multiprocessor systems [8]. However, due to the fact that HOPA is computationally expensive to be run inside such a design optimization loop, we use a scaled down greedy algorithm, in which we drastically reduce the number of iterations needed for determining an optimized priority assignment.

Finally, the resulted system configuration is analyzed (line 15) using the scheduling and schedulability analysis algorithm presented in Section 5.1. The resulted cost function will decide whether the current configuration is better than the current best one (lines 16–19). Moreover, if all activities meet their deadlines ($\delta_A < 0$), the optimization heuristic stops the exploration process and returns the current best-so-far configuration (lines 20-22).

6. Experimental Results

For the evaluation of our design optimization heuristic we have used synthetic applications as well as a real-life example consisting of a vehicle cruise controller. Thus, we have randomly generated applications of 40, 60, 80 and 100 tasks on systems with 4 processors. 56 applications were generated for each dimension, thus a total of 224 applications were used for experimental evaluation. An equal number of applications with processor utilizations of 20%, 40%, 60% and 80% were generated for each application dimension. All experiments were run on an AMD AthlonXP 2400+ processor, with 512 MB RAM.

We were first interested to determine the quality of our design optimization approach for hierarchically scheduled systems, the MSPAHeuristic (MSPA); see Figure 7. We have compared the percentage of schedulable implementations found by MSPA with the number of schedulable solutions obtained by the InitialMSPA algorithm described in Figure 5 (line 2), which derives a straightforward system implementation, denoted with SF. The results are depicted in Figure 8.a. We can see that our MSPA heuristic (the black bars) performs very well, and finds a number of schedulable systems that is considerably and consistently higher than the number of schedulable systems obtained with the SF approach (the white bars). On average, MSPA finds 44.5% more schedulable solutions than SF.

Second, we were interested to determine the impact of the scheduling policy assignment (SPA) decisions on the number of schedulable applications obtained. Thus, for the same applications, we considered that the task mapping is fixed by the SF approach, and only the SPA is optimized. Figure 8.a presents this approach, labelled "MSPA/No mapping", corresponding to the gray bars. We can see that most of the improvement over the SF approach is obtained by carefully optimizing the SPA in our MSPA heuristic.

We were also interested to find out what is the impact of the processor utilization of an application on the quality of the implementations produced our optimization heuristic. Figure 8.b presents the percentage of schedulable solutions found by MSPA and SF as we ranged the utilization from 20% to 80%.

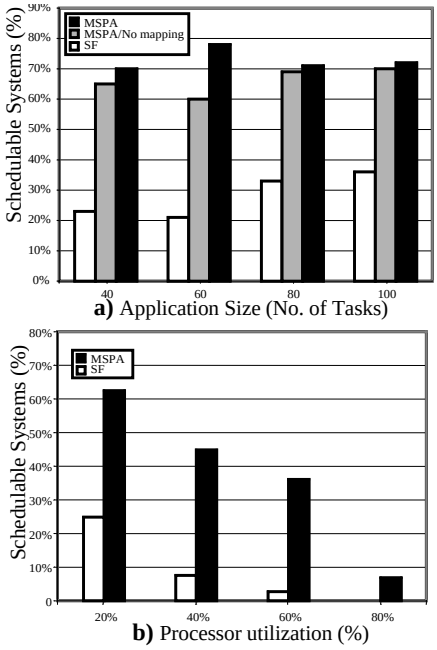

Figure 8. Performance of the Design Optimisation

We can see that SF degrades very quickly with the increased utilization, with under 10% schedulable solutions for applications with 40% utilization and without finding any schedulable solution for applications with 80% utilization, while MSPA is able to find a significant number of schedulable solutions even for high processor utilizations.

Considering the complex optimization steps performed, our design optimization heuristic produces good quality results in a reasonable amount of time. For example, the heuristic will finish on average in less than 500 seconds for applications with 80 tasks that were found schedulable.

Finally, we considered a real-life example implementing a vehicle cruise controller (CC). The process graph that models the CC has 32 processes, and is described in [15]. The CC was mapped on an architecture consisting of three nodes: Electronic Throttle Module (ETM), Anti-lock Breaking System (ABS) and Transmission Control Module (TCM). We have considered a deadline of 250 ms. In this setting, SF failed to produce a schedulable implementation. Our design optimization heuristic MSPA was considered first such that the mapping is fixed by SF, and we only allowed reassigning of scheduling policies. After 29.5 seconds, the best scheduling policy allocation which was found still resulted in an unschedulable system, but with a "degree of schedulability" three times higher than obtained by SF. When mapping was allowed, and a schedulable system was found after 28.49 seconds.

7. Conclusions

In this paper we have addressed the analysis and optimization of hierarchically scheduled heterogeneous real-time systems. Several scheduling policies are used for tasks, such as static cyclic scheduling, fixed-priority preemptive scheduling and earliest deadline first, organized as a hierarchy. Messages are transmitted using the Universal Communication Model that combines both time-triggered and event-triggered slots.

We have proposed a holistic scheduling analysis that is able to handle the hierarchical scheduling policies. As our main contribution we have proposed a design optimization heuristic for the assignment of scheduling policies to tasks, the mapping of tasks to hardware components, and the scheduling of the activities such that the timing constraints of the application are guaranteed.

As our experiments have shown, our heuristic is able to find schedulable implementations under limited resources, achieving an efficient utilization of the system.

References

[1] N. Audsley, K. Tindell, A. Burns, "The End of Line for Static Cyclic Scheduling?", *Proceedings of the 5th Euromicro Workshop on Real-Time Systems*, 36–41, 1993.

[2] N. C. Audsley, A. Burns, R. I. Davis, K. W. Tindell and A. J. Wellings, "Fixed Priority Preemptive Scheduling: An Historical Perspective", *Real-Time Systems*, 8(2/3), 129–154, 1995.

[3] R. Bosch GmbH, *CAN Specification Version 2.0*, 1991.

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

[5] T. Demmeler, P. Giusto, "A Universal Communication Model for an Automotive System Integration Platform", *Proceedings of the Design, Automation and Test in Europe Conference*, 47–54, 2001.

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

[7] G. Buttazzo, "Rate Monotonic vs. EDF: Judgment Day", *Real-Time Systems*, 29 (1), 5–26, January 2005.

[8] J. J. Gutiérrez García, M. González Harbour, "Optimized priority assignment for tasks and messages in distributed hard real-time systems", *Proc. of the 3rd Workshop on Parallel and Distributed Real-Time Systems*, 124–132, 1995.

[9] H. Kopetz, *Real-Time Systems: Design Principles for Distributed Embedded Applications*, Kluwer Academic Publishers, 1997.

[10] G. Lipari, E. Bini, "Resource Partitioning among Real-Time Applications", *Proceedings of the 15th Euromicro Conference on Real-Time Systems*, 151–160, 2003.

[11] M. A. Livani, J. Kaiser, "EDF Consensus on CAN Bus Access for Dynamic Real-Time Applications", *Proceedings of the IPPS/SPDP Workshops*, 1088–1097, 1998.

[12] H. Lönn, J. Axelsson, "A Comparison of Fixed-Priority and Static Cyclic Scheduling for Distributed Automotive Control Applications", *Proceedings of the Euromicro Conf. on Real-Time Systems*, 142–149, 1999.

[13] M. Gonzaléz Harbour, J. C. Palencia, "Response Time Analysis for Tasks Scheduled under EDF within Fixed Priorities", *Proceedings of Real-Time Systems Symposium*, 200–209, 2003.

[14] P. Pedreiras, L. Almeida, "Combining Event-Triggered and Time-Triggered Traffic in FTT-CAN: Analysis of the Asynchronous Messaging System", Proceedings of the Workshop on Factory Communication Systems, 67–75, 2000.

[15] P. Pop, P. Eles, Z. Peng, *Analysis and Synthesis of Distributed Real-Time Embedded Systems*, Kluwer Academic Publishers, 2004.

[16] T. Pop, P. Eles, Z. Peng, "Schedulability Analysis for Distributed Heterogeneous Time/Event-Triggered Real-Time Systems", Proceedings of the15th Euromicro Conference on Real-Time Systems, 257–266, 2003.

[17] J. Rushby, "Bus Architectures for Safety-Critical Embedded Systems," *Springer–Verlag Lecture Notes in Computer Science*, vol. 2211, pages 306–323, 2001.

[18] L. Sha et al., "Real Time Scheduling Theory: A Historical Perspective", *Real-Time Systems*, 28, 101–155, 2004.

[19] K. Tindell, J. Clark, "Holistic Schedulability Analysis for Distributed Hard Real-Time Systems", *Microprocessing & Microprogramming*, 40(2–3), 117–134, 1994.

[20] D. Ullman, "NP-Complete Scheduling Problems," *Journal of Computer Systems Science*, 10, 384–393, 1975.

[21] J. Xu, D.L. Parnas, "On satisfying timing constraints in hard-real-time systems", *IEEE Transactions on Software Engineering*, 19(1), 132–146, 1993.