Analysis and Optimisation of Hierarchically Scheduled Multiprocessor Embedded Systems

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

Published in:
International Journal of Parallel Programming

Link to article, DOI:
10.1007/s10766-007-0059-9

Publication date:
2008

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

Int J Parallel Prog (2008) 36:37–67
DOI 10.1007/s10766-007-0059-9

Analysis and Optimisation of Hierarchically Scheduled Multiprocessor Embedded Systems

Traian Pop · Paul Pop · Petru Eles · Zebo Peng

Received: 16 November 2006 / Accepted: 5 March 2007 / Published online: 20 November 2007

Abstract We present an approach to the analysis and optimisation of heterogeneous multiprocessor embedded systems. 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 organised in a hierarchy. In this paper, we first develop a holistic scheduling and schedulability analysis that determines the timing properties of a hierarchically scheduled system. Second, 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 also present several algorithms for solving these problems. Our heuristics are able to find schedulable implementations under limited resources, achieving an efficient utilisation of the system. The developed algorithms are evaluated using extensive experiments and a real-life example.

Keywords Hierarchical schedulers · Multiprocessor embedded systems · Static/dynamic communication protocols

T. Pop (✉) · P. Eles · Z. Peng
Department of Computer and Information Science, Linköping University, Linkoping 581 83, Sweden
e-mail: trapo@ida.liu.se

P. Eles
e-mail: petel@ida.liu.se

Z. Peng
e-mail: zebpe@ida.liu.se

P. Pop
Informatics and Mathematical Modelling, Technical University of Denmark, Building 322, office 228, Kongens Lyngby 2800, Denmark
e-mail: paul.pop@imm.dtu.dk

1 Introduction

There has been a lot of debate in the literature on the suitability of the event-triggered (ET) paradigm as opposed to the time-triggered (TT) one, for implementation of real-time systems [5,25,63]. Several arguments have been brought concerning composability, flexibility, fault tolerance, jitter control or efficiency in processor utilisation. The same discussion has also been extended to the communication infrastructure which can also be implemented according to the time-triggered or event-triggered paradigm.

An interesting comparison of the TT and ET approaches, from a more industrial, in particular automotive, perspective, can be found in [32]. 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.

Hierarchically scheduled systems are systems that have their functionality divided into several sets that are each handled by a different scheduler. These schedulers also control each other in a hierarchical manner.

The growing amount and diversity of functions to be implemented by the current and future embedded applications (like for example, in automotive electronics [23]) has shown that, in many cases, time-triggered and event-triggered functions have to coexist on the computing nodes and to interact over the communication infrastructure. When time-triggered and event-triggered activities have to share the same processing node, a natural way for the execution support can be provided through a hierarchical scheduler. Similarly, when such heterogeneous applications are mapped over a multiprocessor architecture, the communication infrastructure should allow for message exchange in both time-triggered and event-triggered manner in order to ensure a straightforward interconnection of heterogeneous functional components.

Safety-critical hard real-time distributed applications running on such hierarchically scheduled multiprocessor architectures are difficult to analyse. Due to the hierarchical nature of the schedulers, various execution interferences have to be carefully accounted for during the timing analysis that determines the worst-case response times of the system activities. Moreover, due to the distributed nature of the architecture, message delays have to be taken into consideration during the analysis. Such an analysis is further complicated by the particular characteristics of the communication protocol that mixes both static and dynamic transmission of messages.

Once the timing analysis has been provided, the entire system can be optimised by adjusting its configuration parameters. Such an optimisation process can be directed by the results from the timing analysis, so that in the end the timing constraints of the application are satisfied.

In order to cope with the complexity of designing such heterogeneous embedded systems, only an adequate design environment can effectively support decisions leading in an acceptable time to cost-efficient, reliable and high performance solutions. Developing flexible and powerful tools for the design and analysis of such kind of heterogeneous systems represents the motivation behind the work presented in this paper.

1.1 Related Work

This section presents an overview of the previous research in the area of analysis and system level design for distributed embedded systems. We concentrate in particular on scheduling and communication synthesis, with focus on the time-triggered and event-triggered aspects.

1.1.1 Scheduling and Schedulability Analysis of Real-Time Systems

Task scheduling and schedulability analysis have been intensively studied for the past decades, one of the reasons being the high complexity of the targeted problems [54,60]. The reader is referred to [6] and [7] for surveys on this topic.

A comparison of the two main approaches for scheduling hard real-time systems (i.e., *static cyclic scheduling* and *fixed priority scheduling*) can be found in [33].

The *static cyclic* (*non-preemptive*) *scheduling* approach has been long considered as the only way to solve a certain class of problems [63]. This was one of the main reasons why it received considerable attention. Solutions for generating static schedules are often based on *list scheduling* in which the order of selection for tasks plays the most important role [9,22] (see also Sect. 3.5). However, list scheduling is not the only alternative, and branch-and-bound algorithms [1,21], mixed integer linear programming [49], constraint logic programming [13,26], or evolutionary [52] approaches have also been proposed (Fig. 1).

For event-triggered tasks, in this paper we are interested both in static and dynamic priority based scheduling policies. In our work we will focus our attention on *fixed priority scheduling* (FPS) and *earliest-deadline-first scheduling* (EDF). For both policies, determining whether a set of tasks is schedulable involves two aspects:

1. *The assignment of priorities* to system activities, i.e., what priority should be associated with each task and message in the system so that the task set is schedulable.

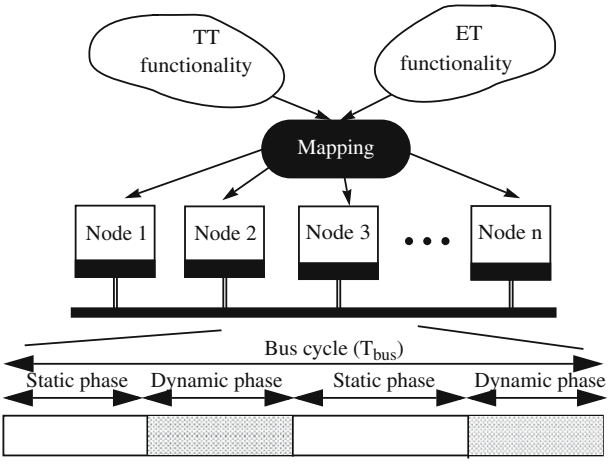

Fig. 1 Heterogeneous TT/ET distributed system

2. *The schedulability test*, which determines whether all activities in the system will meet their deadlines under the current policy.

In the case of EDF scheduling, the priorities are assigned dynamically, at run-time, according to the criticality of each ready task, i.e., tasks that are closer to their deadline will receive higher priorities.

In the case of *fixed priority scheduling*, the priorities are associated to tasks off-line, before the system is deployed. In order to solve the problem of assigning priorities to system activities so that the system is schedulable, two main policies have been developed; they both work under restricted assumptions, i.e., the task set to be scheduled is composed of periodic and independent tasks mapped on a single processor:

a. Rate-monotonic (RM) [30] which assigns higher priorities to tasks with shorter periods; it works under the constraint that task deadlines are identical with task periods.
b. Deadline-monotonic (DM) [29] which assigns higher priorities to tasks with shorter relative deadlines; this policy assumes that task deadlines are shorter than task periods.

Under a particular set of restrictions regarding the system specification, such policies are optimal. However, if, for example, tasks are not independent, then the optimality does not hold anymore for RM and DM policies. Therefore, in [5], the authors proposed a priority assignment in the case of tasks with arbitrary release times. Their algorithm is of polynomial complexity in the number of tasks. However, for the case of multiprocessor/distributed hard real-time systems, obtaining an optimal solution for priority assignment is often infeasible, due to complexity reasons. A solution based on simulated annealing has been proposed in [56], where the authors present an algorithm which simultaneously maps the tasks on processors and assigns priorities to system activities so that the resulted system is schedulable. In order to avoid the large amount of computation time required by such a general-purpose approach, an optimised priority assignment heuristic called HOPA has been suggested in [19], where the authors iteratively compute deadlines for individual tasks and messages in the system, while relying on the DM policy to assign priorities to the tasks. Their algorithm has shown a better efficiency than the one proposed in [56], both in quality and especially in speed, making it appropriate for being used inside a design optimisation loop which requires many iterations. As an example, HOPA has been adapted for the design optimisation of multi-cluster distributed embedded systems [43].

For the second aspect of fixed priority scheduling, there are two main approaches for performing schedulability tests:

a. *Utilisation based tests*, in which the schedulability criterion is represented by inequations involving processor utilisation and utilisation bounds. However, such approaches are valid only under restricted assumptions [8,29,30].
b. Response time analysis, in which determining whether the system is schedulable or not requires first the computation of the worst-case response time of a task or message. The worst case response time of an activity is represented by the longest possible time interval between the instant when that activity is initiated in the system and the moment when the same activity is finished. If the worst case

response time resulted for each task/message is lower or equal than the associated deadline for that activity, then the system is schedulable.

Response time analysis is usually more complex but also more powerful than the utilisation based tests. The main reason for this is because response time analysis can take into consideration more factors that influence the timing properties of tasks and messages in a system.

The response time analysis in [27] offers a necessary and sufficient condition for scheduling tasks running on a mono-processor system, under fixed priority scheduling and restricted assumptions (independent periodic tasks with deadlines equal with periods). In order to increase the range of target applications, relaxing assumptions is necessary. Moreover, considering the effects of more and more factors that influence the timing properties of the tasks decreases the pessimism of the analysis by determining tighter worst case response times and leading to a smaller number of false negatives (which can appear when a system which is practically schedulable cannot be proven so by the analysis). Over the time, extensions have been offered to response time analysis for fixed priority scheduling by taking into account task synchronisation [53], arbitrary deadlines [28], precedence constraints between tasks [35] and tasks with varying execution priorities [17], arbitrary release times [5,59], tasks which suspend themselves [34], tasks running on multiprocessor systems [34,57], etc. In [50] and [51], the authors model the multiprocessor heterogeneous systems as components that communicate through event streams and propose a technique for integrating different local scheduling policies based on such event-model interfaces. Another compositional approach is presented in [61], where the authors propose real-time interfaces and a component model that support incremental design of real-time systems.

1.1.2 Communication in Real-Time Systems

The aspects related to communication in real-time systems are receiving a continuously increasing attention in the literature. Building safety critical real-time systems requires consideration for all the factors that influence the timing properties of a system. For the case of distributed systems, in order to guarantee the timing requirements of the activities in the system, one should consider the effects of communication aspects like the communication protocol, bus arbitration, clock synchronisation, packaging of messages, characteristics of the physical layer, etc. Due to the variety of communication protocols, scheduling and schedulability analysis involving particular communication protocols has become a prolific area of research. Following a similar model for determining task response time under rate monotonic analysis, message transmission times have been analysed for protocols like TTP bus [24], Token Ring [37,55], FDDI [2], ATM [15,20] and CAN bus [58].

Usually, communication protocols allow either static (TT) or dynamic (ET) services, influencing several levels in the design flow and giving more weight in the design output to either flexibility or time-determinism of the system. As a result, a lot of work has been concentrated on coping with the disadvantages of the TT/ET approaches and on trying to combine their advantages. For example, in [40] and [41], the authors present a method for dealing with flexibility in TTP based systems by

considering consecutive design stages in a so called *incremental design flow*. In order to combine the advantages of rigid off-line static scheduling with flexible online fixed priority scheduling, in [11] and [12] fixed priority scheduling is adapted in such a way that it emulates static cyclic schedules which are generated offline.

In the case of bus-based distributed embedded systems, one of the main directions of evolution for communication protocols is towards mixed protocols, which support both ET and TT traffic. The proponents of the Time-Triggered Architecture showed that TTP can be enhanced in order to transmit ET traffic, while still maintaining time composability and determinism of the system, properties which are normally lost in event-triggered systems [24]. A modified version of CAN, called Flexible Time-Triggered CAN [3,4], is based on communication cycles which are divided into asynchronous and synchronous windows. Several other mixed communication protocols can be found in [16,62].

1.2 Contributions

Our approach considers 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.

We have considered that the time-triggered activities are executed according to a static cyclic schedule, while the event-triggered activities follow a EDF-within-priorities policy[1] [18], which is preemptive for the execution of tasks and non-preemptive for the transmission of messages. We have modelled the heterogeneous communication protocol using UCM.

The main contributions of this paper are two-fold. The first part consists in a holistic schedulability analysis for heterogeneous TT/ET task sets which communicate through mixed ST/DYN communication protocols [45,46]. Such an analysis presents two aspects:

a. It computes the response times of the ET activities while considering the influence of a static schedule;
b. It builds a static cyclic schedule for the TT activities while trying to minimise the response times of the ET activities.

Second, we show how the scheduling and schedulability analysis can be used inside a design optimisation loop in order to improve the timing properties of the system.

1.3 Paper Overview

The next section presents the system model we used. In Sect. 3, we present our analysis method for deriving response times of tasks and of messages in a heterogeneous TT/ET system. In Sect. 4, we first discuss some optimisation aspects which are particular to the studied systems, and then we define and solve the design optimisation

[1] EDF-within priorities allows tasks to be executed according to their fixed priorities, with the exception that in the case of tasks running at the same level of priority, the scheduling conflicts are solved according to EDF.

problem that aims at improving the overall system schedulability. Finally, in Sect. 5 we draw some conclusions and discuss possible research directions for the future.

2 System Model

In this section we present the system model that we use during scheduling and design optimisation. First, we briefly describe the hardware architecture and the structure of the bus access cycle. Then, we present the minimal requirements regarding the software architecture for a system which is able to run both event-triggered and time-triggered activities. The last part of this section presents the abstract representation which we use for modelling the applications that are assumed to implement the functionality of the system.

2.1 Hardware Architecture

We consider architectures consisting of nodes connected by a unique broadcast communication channel. Each node consists of:

- A *communication controller* which controls the transmission and reception of both ST and DYN messages;
- A *CPU* for running the processes mapped on that particular node;
- *Local memories* for storing the code of the kernel (ROM), the code of the processes and the local data (RAM); and
- *I/O interfaces* to sensors and actuators.

Such hardware architectures are common in applications such as automotive electronics, robotics, etc. In Fig. 2, we illustrate a heterogeneous distributed architecture interconnected by a bus based infrastructure.

2.2 Bus Access

We model the bus access scheme using the Universal Communication Model (see Sect. 2). The bus access is organised as consecutive cycles, each with the duration T_{bus}. We consider that the communication cycle is partitioned into static and dynamic phases (Fig. 2). Static phases consist of time slots, and during a slot only one node is

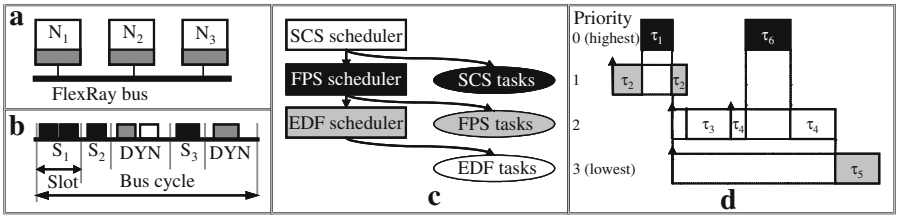

Fig. 2 System architecture

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.

2.3 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. The real-time kernel contains three schedulers, for SCS, FPS, and EDF, organized *hierarchically* (Fig. 2c).

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 Fig. 2d, 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, Fig. 2d 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 synchronisation is performed based on the schedule table. Moreover, since SCS tasks are non preemptable and their start time is off-line 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 [18], 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.[2] Such a situation is present in Fig. 2d for tasks τ_3 and τ_4. There can be several such 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.

Every node in the architecture has a communication controller that implements the protocol services. The controller runs independently of the node's CPU. We model the bus access scheme using the Universal Communication Model [10]. The bus access is organised as consecutive cycles, each with the duration T_{bus}. We consider that the communication cycle is partitioned into static (ST) and dynamic (DYN) phases (Fig. 2b).

- 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 [31].

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

2.4 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.

- A task can belong either to the TT or to the ET domain. We consider that TT tasks are scheduled using SCS, while ET tasks are scheduled under FPS and EDF.
- 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 can belong either to the static (ST) or to the dynamic (DYN) domain. We consider that static messages are those sent during the ST phases of the bus cycle, while dynamic messages are those transmitted during the DYN phases.

[2] In the discussion of this paper, tasks having the same priority on a processor are considered EDF tasks. However, [34] also shows how it is possible to handle same-priority FPS tasks by scheduling them using a FIFO policy.

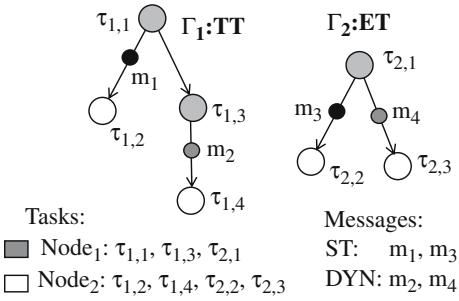

Fig. 3 Application model example

- 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. 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. In this paper we restrict our discussion to the situation when TT tasks communicate through ST messages and ET tasks communicate through DYN messages.
- Each task τ_{ij} (belonging to the task graph Γ_i) has a period T_{ij}, and a deadline D_{ij} and, when mapped on node $Proc_k$, it has a worst case execution time C_{ij} $(Proc_k)$. The node on which τ_{ij} is mapped is denoted as $\mathcal{M}(\tau_{ij})$. Each ET task also has a given priority $Prio_{ij}$. Individual release times or deadlines of tasks can be modelled by introducing dummy tasks in the task graphs; such dummy tasks have an appropriate execution time and are not mapped on any of the nodes [14].
- 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. Also, DYN messages have given priorities.

Figure 3 shows an application modelled as two task-graphs Γ_1 and Γ_2 mapped on two nodes, $Node_1$ and $Node_2$. Task-graph Γ_1 is time-triggered and task-graph Γ_2 is event-triggered. Data-dependent tasks mapped on different nodes communicate through messages transmitted over the bus, which can be either statically scheduled, like m_1 and m_3, or dynamic, like the messages m_2 and m_4.

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). Such messages are typically non-critical and are not affected by hard real-time constraints.

3 Scheduling and Schedulability Analysis of Heterogeneous TT/ET Systems

In this section we present an analytic approach for computing task response times and message transmission delays for heterogeneous TT/ET systems.

3.1 Problem Formulation

Given an application and a system architecture as presented in 2., the following problem has to be solved: construct a correct static cyclic 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 and DYN messages 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 the next sections, we will present the schedulability analysis algorithm proposed in [34] for distributed real-time systems and we will show how we extended this analysis in order to consider the interferences induced by an existing static schedule. Section 3.2 presents a general view over our approach for the global scheduling and schedulability analysis of heterogeneous TT/ET distributed embedded systems. Section 3.3 describes the regular schedulability analysis for FPS and EDF tasks sharing the same resources, as developed in [18]. Section 3.4 extends the schedulability analysis so that SCS tasks are taken into consideration when computing the response times of FPS and EDF activities. In Sect. 3.5 we present our complete scheduling algorithm, which statically schedules the TT activities while trying to minimise the influence of TT activities onto ET ones [48]. The performance of our approach is evaluated in Sect. 3.6, where we present the experimental results.

It has to be mentioned that our analysis is restricted, for the moment, to the model in which TT tasks communicate only through ST messages, while communication between ET tasks is performed by DYN messages. This is not an inherent limitation of our approach. For example, schedulability analysis of ET tasks communicating through ST messages has been presented in [38] and [42].

3.2 Holistic Scheduling

Figure 4 illustrates our strategy for scheduling and schedulability analysis of heterogeneous TT/ET distributed embedded systems: the activities to be scheduled are the TT and ET task graphs, consisting of TT tasks/ST messages and ET tasks/DYN messages respectively. The TT activities are statically scheduled and, as an output, a static cyclic schedule will be produced. Similarly, the worst case response times of the ET activities are determined using the schedulability analysis presented in the previous section. As a result, the system is considered schedulable if the static schedule is valid and if the ET activities are guaranteed to meet their deadlines. For the case of a mixed TT/ET system, building a static cyclic schedule for the TT activities has to take into consideration both the characteristics of the mixed ST/DYN communication protocol and our assumption that execution of TT tasks is non-preemptible, while the execution of an ET task can be interrupted either by a TT task or by another ET task which has

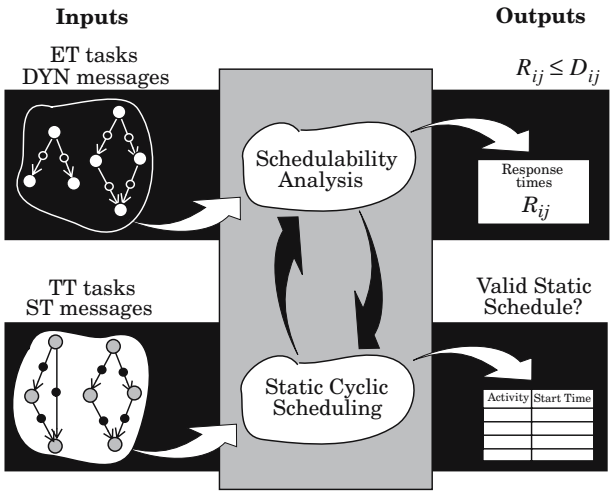

Fig. 4 Scheduling and schedulability analysis for mixed TT/ET distributed embedded systems

a higher priority. This means that the static schedule will have not only to guarantee that TT activities meet their deadlines, but also that the interference introduced from such a schedule will not increase in an unacceptable way the response times of ET activities. In conclusion, an efficient scheduling algorithm requires a close interaction between the static scheduling of TT activities and the schedulability analysis of the ET activities.

3.3 Schedulability Analysis of Event-Triggered Task Sets

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 [18]. 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.

An ET task graph Γ_i is activated by an associated event which occurs with a period T_i. ET tasks (FPS or EDF) and DYN messages are modelled similarly, by considering the bus as a processing node and accounting for the non-preemptability of the messages during the analysis. 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} (Fig. 5). Offsets are the means by which dependencies among tasks are modelled for the schedulability analysis. For example, if in Fig. 5, task τ_{ij+1} is data dependant on task τ_{ij}, then such a relation can be enforced by associating to τ_{ij+1} an offset ϕ_{ij+1} which is equal or greater than the worst case response time R_{ij} of its predecessor, τ_{ij}. In this way, it is guaranteed that task τ_{ij+1} starts only after its predecessor has finished execution. The response time

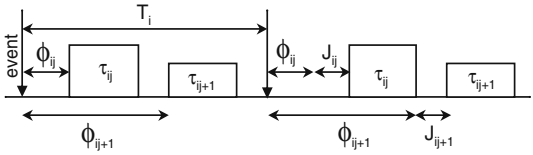

Fig. 5 Tasks with offsets

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 $R_{b,ij}$.

In [18], 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:

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

  where p is the number of activations of τ_{ab} in the busy period, B_{ab} is the blocking time for τ_{ab} (see [18] 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:

$$w_{ab}(p) = B_{ab} + p \cdot C_{ab} + \sum_{\forall \tau_{ab} \neq \tau_{ij} | Prio_{ab} = Prio_{ij}} W_{ij}\left(w_{ab}^{A}(p), D^{A}(p)\right) + \sum_{\forall \tau_{ij} | Prio_{ij} > Prio_{ab}} W_{ij}\left(w_{ab}^{A}(p)\right). \quad (3)$$

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:

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

The analysed 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

```
1 do
2   Done = true
3   for each transaction Γi do
4     for each task τij in Γi do
5       for each task τik in Γi do
6         if Prioik ≥ Prioij and M(τik) = M(τij) then
7           for each job p of τij do
8             Consider that τik initiates tc
9             Compute Rij^p
10            if Rij^p > Rij^max then
11              Rij^max = Rij^p
12            endif
13          endfor
14        endif
15      endfor
16      if Rij^max > Rij^max then -- larger Rij found
17        Rij = Rij^max
18        Done = false
19        for each successor τik of τij do
20          Jij = Rij - Rij^b -- update jitters
21        endfor
22      endif
23    endfor
24  endfor
25  while (Done!= true)
```

Fig. 6 Schedulability analysis Algorithm

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

$$R_{ab}(p) = w_{ab}^{A}(p) - (p-1)T_{ab} + J_{ab} - A \tag{5}$$

A similar technique is used in the more complex case of offset-based analysis (for such extensions, the reader is referred to [34] and [36]). However, regardless of the analysis used, the technique has to be enhanced to take into consideration an existing static schedule, allowing us to analyse hierarchically scheduled systems that use a combination of SCS, FPS and EDF scheduling policies.

Figure 6 represents the pseudocode for the schedulability analysis proposed in [34]. According to this algorithm, the worst case response time R_{ij} of each task τ_{ij} is computed by considering all critical instants initiated by each task τ_{ik} mapped on the same node $\mathcal{M}(\tau_{ij})$ and with a higher priority than $Prio_{ij}$. According to the same schedulability analysis, jitters are taken into consideration when the algorithm computes the length of the busy windows and, implicitly, the response times of the tasks [34]. This means 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 τ_{ij} precedes τ_{ik} in Γ_i, then $J_{ik} = R_{ij} - R_{b,ij}$, like in lines 20–21 in Fig. 6). Due to this cyclic dependency (response times depend on jitters and jitters depend on response times), 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 jitters J_{ij}, busy windows $w_{ijk}(p)$ and then again the worst-case response times R_{ij}, until the response times converge to their final value.

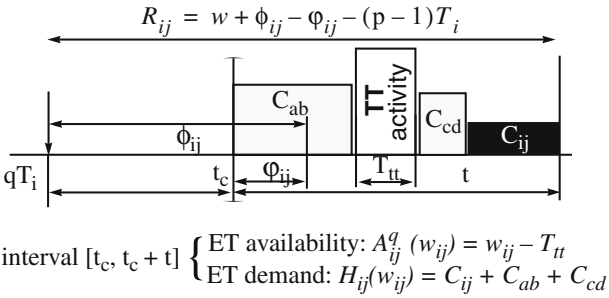

Fig. 7 Availability and demand

3.4 Schedulability Analysis of Event-Triggered Activities under the Influence of a Static Cyclic Schedule

Considering the algorithm presented in the previous section as a starting point, we have to solve the following problem: compute the worst case response time of a set of ET tasks and DYN messages by taking into consideration:

- The interference from the set of statically scheduled tasks.
- The characteristics of the communication protocol, which influence the worst case delays induced by the messages communicated on the bus.

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 Fig. 7, 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 Fig. 7, the CPU availability for the analysed interval is obtained by substracting from t the amount of processing time needed for the SCS activities. Figure 7 presents how the *availability* $A^q_{ij}(w)$ and the *demand* $H_{ij}(w)$ are computed for a task τ_{ij}: the busy window of τ_{ij} starts at the critical instant $qT_i + t_c$ 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.

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:

$$H_{ij}(t) = W_{ij}(t). \tag{6}$$

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

$$A_{ij}(t) = min\left[A^{ab}_{ij}(t)\right], \quad \forall \tau_{ab} \in TT | \mathcal{M}(\tau_{ab}) = \mathcal{M}(\tau_{ij}), \tag{7}$$

Fig. 8 Determining the length of the busy window

```
1 w_ij = p • C_ij + B_ij
2 do
3   Compute demand H_ij(w_ij)
4   Compute availability A_ij(w_ij)
5   if H_ij(w_ij) > A_ij(w_ij) then
6       w_ij = H_ij(w_ij) - A_ij(w_ij)
7   endif
8 while H_ij(w_ij) ≥ A_ij(w_ij)
9 return w_ij
```

where $A_{ij}^{ab}(t)$ is the total available CPU-time on processor $\mathcal{M}(\tau_{ij})$ 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 (Fig. 8).

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: $H_{ij}(w_{ij}) \leq A_{ij}(w_{ij})$. This procedure for the calculation of the busy window is included in the iterative process for calculation of response times presented in Sect. 3.3. 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.

After performing the schedulability analysis, we can check if $R_{ij} \leq D_{ij}$ 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 [38].

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

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, $f_2 \leq 0$ 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.

3.5 Static Cyclic Scheduling of Time-Triggered Activities in a Heterogeneous TT/ET Environment

As mentioned in the beginning of Sect. 3.1, 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 Fig. 9. For the construction of the schedule table with start times for SCS tasks and messages, we adopted a list scheduling-based algorithm [9] 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 (Fig. 9, 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 [39]. 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 $\mathcal{M}(\tau_{ij})$ 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}

Fig. 9 Holistic scheduling algorithm

```
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
```

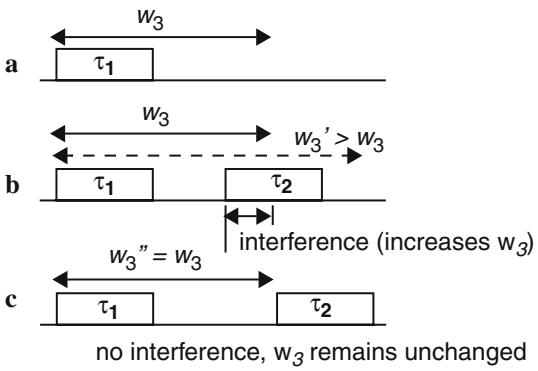

Fig. 10 Static scheduling

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 Sect. 1.1.1, 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 Fig. 10 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 10a, presents the system with only τ_1 scheduled, situation for which the busy-period w_3 is computed. Figure 10b 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 Fig. 10c, where no extra interference is introduced in the busy period w_3. However, during the static scheduling, we have to consider two aspects:

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 Fig. 10 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 Fig. 11a, task τ_2 misses its deadline and the resulted static schedule is not valid). We reduce such a risk by employing the technique presented in Fig. 11b and c, 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

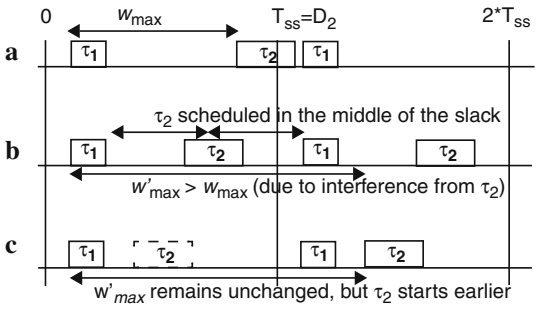

Fig. 11 Optimised static scheduling

maximum busy period $w_{\max}$ of the ET tasks may increase due to interference from task τ_2 (Fig. 11b). 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 (Fig. 11c).

The scheduling algorithm is presented in Fig. 9 . 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 analysed using the technique in Sect. 3.4. 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.

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.

3.6 Experimental Results

For the evaluation of our scheduling and analysis algorithm we generated a set of 2,970 tests representing systems of 2–10 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 utilisation on each processor was between 20 and 80%. In a similar manner we assured that 20% and up to 60% of the total utilisation on a processor is required by the ET activities. All experiments were run on an AMD Athlon 850 MHz PC.

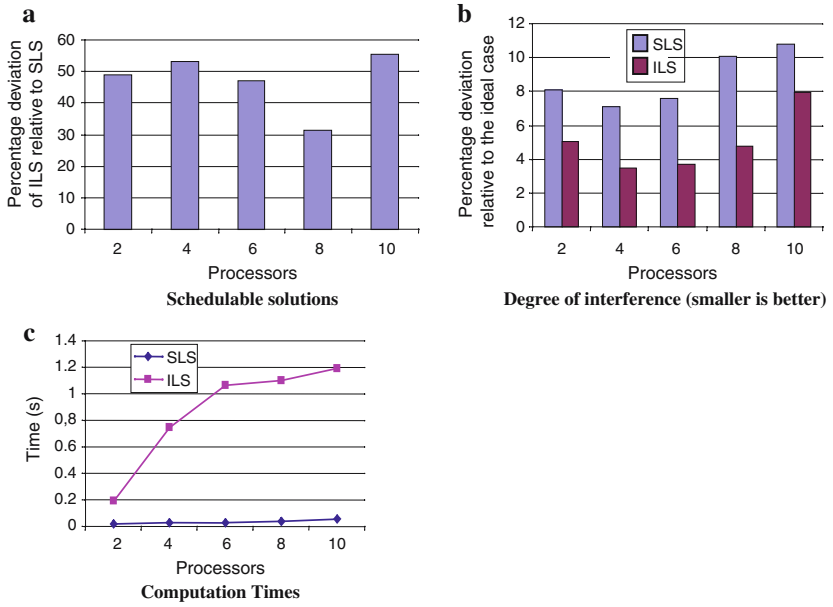

Fig. 12 Performance of the scheduling and schedulability analysis algorithm

The first set of experiments compares our holistic scheduling algorithm that we proposed above with a similar algorithm that uses simple list scheduling. In Fig. 12a we illustrate the capacity of our proposed algorithm (based on an improved list scheduling ILS) to produce schedulable systems, compared to that of simple list scheduling (SLS). The figure shows that ILS was able to generate between 31 and 55% more schedulable solutions than when a simple list scheduling was used. In addition, we computed the quality of the identified solutions, as the percentage deviation of the schedulability degree ($DSch_{xLS}$) 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 Fig. 12b, we present the average quality of the solutions found by the two algorithms. For this diagram, we used only those results where both algorithms managed to find a schedulable solution. It is easy to observe that the solutions obtained with ILS are constantly at a minimal level of interference, while the SLS heuristic produces solutions in which the TT interference is considerably higher, resulting in significantly larger response times of the ET activities.

In Fig. 12c we present the average execution times of the our scheduling heuristic. According to expectations, the execution time for our scheduling and schedulability

analysis algorithm increases with the size of the application. However, even for large applications, the algorithm is still fast enough so that it could be used in certain particular cases like, for example, inside a design space exploration loop with an extremely large number of iterations.

4 Design Optimisation Problems

Considering the type of applications and systems described in Sect. 2, several design optimisation 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;
- Optimisation 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 $\mathcal{P}_{SCS} \subseteq \mathcal{P}$ the subset of tasks for which the designer has assigned SCS, $\mathcal{P}_{FPS} \subseteq \mathcal{P}$ contains tasks to which FPS is assigned, while $\mathcal{P}_{EDF} \subseteq \mathcal{P}$ 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 $\mathcal{P}^+ = \mathcal{P} \backslash (\mathcal{P}_{SCS} \cup \mathcal{P}_{FPS} \cup \mathcal{P}_{EDF})$ 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 utilisation of resources, etc.

Let us illustrate some of the issues related to SPA in such a context. In the example presented in Fig. 13 we have an application[3] with six tasks, τ_1 to τ_6, and three nodes, N_1, N_2 and N_3. The worst-case execution times on each node are given in the table labeled "Mapping". Note that an "x" in the table means that the task is not allowed to be

[3] Communications are ignored for the examples in this subsection.

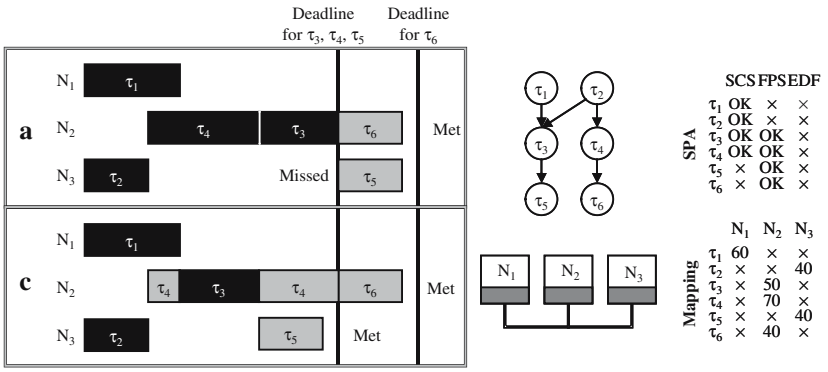

Fig. 13 Scheduling policy assignment example #1

mapped on that node (the mapping of tasks is thus fixed for this example). The scheduling policy assignment is captured by the table labelled "SPA". Thus, tasks τ_1 and τ_2 are scheduled using SCS, while tasks τ_5 and τ_6 are scheduled with FPS. Similarly, an "x" in the table means that the task cannot be scheduled with the corresponding scheduling policy. We have to decide which scheduling policy to use for tasks τ_3 and τ_4, which can be scheduled with any of the SCS or FPS scheduling policies.

We can observe that the scheduling of τ_3 and τ_4 have a strong impact on their successors, τ_5 and τ_6, respectively. Thus, we would like to schedule τ_4 such that not only τ_3 can start on time, but τ_4 also starts soon enough to allow τ_6 to meet its deadline. As we can see from Fig. 13a, this is impossible to achieve by scheduling τ_3 and τ_4 with SCS. Although τ_3 meets its deadline, it finishes too late for τ_5 to finish on deadline. However, if we schedule τ_4 with FPS, for example, as in Fig. 13b, both deadlines are met. In this case, τ_3 finishes on time to allow τ_5 to meet its deadline. Moreover, although τ_4 is pre-empted by τ_3, it still finishes on time, meets its deadline, and allows τ_6 to meet its deadline, as well. Note that using EDF for τ_4 (if it would share the same priority level with τ_6, for example) will also meet the deadline. The idea in this example is to allow preemption for τ_4.

For a given set of preemptable tasks, the example in Fig. 14 shows the optimisation of the assignment of FPS and EDF policies. In Fig. 14 we have an application 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 (Fig. 14a). Note that in the time line for node N_1 in Fig. 14 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 14b shows the response times when task τ_1 has the highest priority on N_1 (τ_1 retains priority "1") and the other tasks

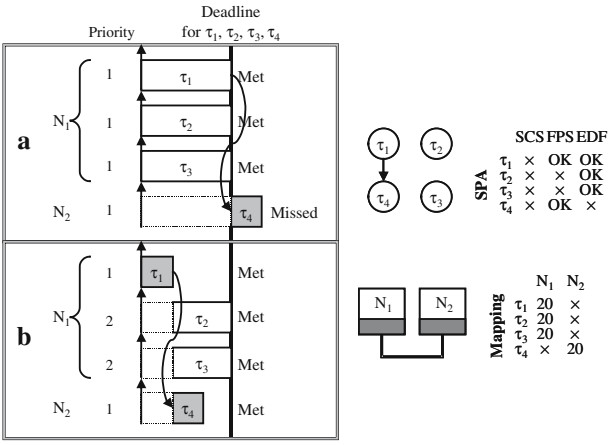

Fig. 14 Scheduling policy assignment example #2

are running under EDF at a lower priority level (τ_2 and τ_3 share lower priority "2"). Since 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

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 $\mathcal{P}^M \subseteq \mathcal{P}$ of already mapped tasks. Consequently, we denote with $\mathcal{P}^* = \mathcal{P} \backslash \mathcal{P}^M$ the tasks for which the mapping has not yet been decided.

For a distributed heterogeneous system, the communication infrastructure has an important impact on the design and, in particular, on the mapping decisions. Let us consider the example in Fig. 15 where we have an application consisting of four tasks, τ_1 to τ_4, and an architecture with three nodes, N_1 to N_3. Thus, the bus will have three static slots, S_1 to S_3 for each node, respectively. The sequence of slots on the bus is S_2 followed by S_1 and then S_3. We have decided to place a single dynamic phase within a bus cycle, labelled "DYN" and depicted in grey, preceding the three static slots (see Sect. 2.2 for the details about the bus protocol). We assume that τ_1, τ_3 and τ_4 are mapped on node N_1, and we are interested to map task τ_2. Task τ_2 is allowed to be mapped on node N_2 or on node N_3, and its execution times are depicted in the table labelled "mapping". Moreover, the scheduling policy is fixed for each task, such that all tasks are scheduled with SCS.

In order to meet the deadline, one would map τ_2 on the node it executes fastest, i.e., node N_2 see Fig. 15a. However, this will lead to a deadline miss due to the bus slot configuration which introduces communication delays. The application will meet the deadline only if τ_2 is counter intuitively mapped on the slower node, i.e., node N_3, as depicted in Fig. 15b.

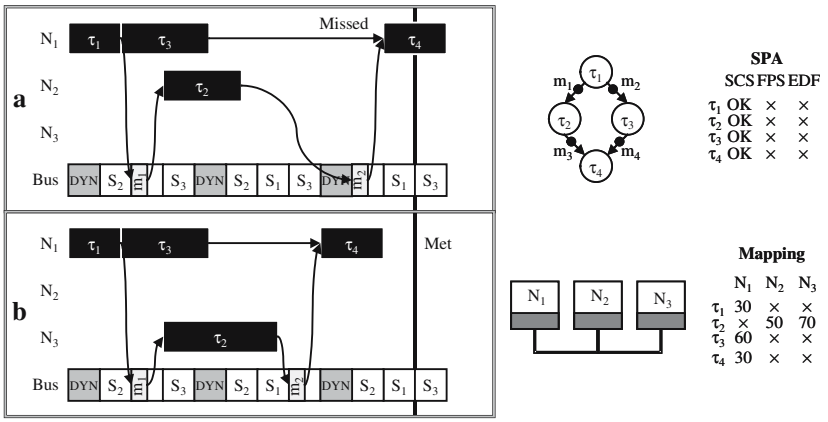

Fig. 15 Mapping example

4.3 Bus Access Optimisation

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. Parameters to be optimised 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 $\mathcal{B}$.

4.4 Exact Problem Formulation

As an input we have an application $\mathcal{A}$ given as a set of task graphs (Sect. 2.4) and a system architecture consisting of a set $\mathcal{N}$ of nodes (Sect. 2.1). As introduced previously, $\mathcal{P}_{SCS}$, $\mathcal{P}_{FPS}$ and $\mathcal{P}_{EDF}$ are the sets of tasks for which the designer has already assigned SCS, FPS or EDF scheduling policy, respectively. Also, $\mathcal{P}^M$ is the set of already mapped tasks.

As part of our problem, we are interested to:

- Find a scheduling policy assignment $\mathcal{S}$ for tasks in $\mathcal{P}^+ = \mathcal{P} \setminus (\mathcal{P}_{SCS} \cup \mathcal{P}_{FPS} \cup \mathcal{P}_{EDF})$;
- Decide a mapping for tasks in $\mathcal{P}^* = \mathcal{P} \setminus \mathcal{P}^M$;
- Determine a bus configuration $\mathcal{B}$;
- Determine the 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 optimisation of the communication channel. Such an optimisation can be performed with the techniques we have proposed in [47].

```
OptimisationStrategy(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) returns schedulable then stop end if
4  Step 2:(M, S, B) = MSPAHeuristic(A, M^0, B^0)
5          if HolisticScheduling(A, M, S, B) returns schedulable then stop end if
6  Step 3:B = BusAccessOptimisation(A, M, S, B)
7          HolisticScheduling(A, M, B, S)
end OptimisationStrategy
```

Fig. 16 The general strategy

5 Design Optimisation Strategy

The design problem formulated in the previous section is NP-complete (the scheduling subproblem, in a simpler context, is already NP-complete [60]). Therefore, our strategy, outlined in Fig. 16, is to divide the problem into several, more manageable, subproblems. Our OptimisationStrategy has three steps:

1. In the first step (lines 1–3) we decide on an initial bus access configuration $\mathcal{B}^0$ (function InitialBusAccess), and an initial policy assignment $\mathcal{S}^0$ and mapping $\mathcal{M}^0$ (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 Fig. 16) maps tasks so that the amount of communication is minimized. The initial scheduling policy of tasks in $\mathcal{P}^+$ 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 Sect. 3.2.
2. If the application is schedulable the optimisation strategy stops. Otherwise, it continues with the second step by using an iterative improvement mapping and policy assignment heuristic, MSPAHeuristic (line 4), presented in Sect. 5.1, 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 BusAccessOptimisation presented in [47], which optimises 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 Mapping and Scheduling Policy Assignment Heuristic

In Step 2 of our optimisation strategy (Fig. 16), 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.

```
MSPAHeuristic(A, M, B, S)
13 for each activity τij in the system do
14    for each processor Ni ∈ N in the system do
15       if τij in P* then -- can be remapped
16          M(τij) = Ni
17       end if
18       for policy = SCS, FPS do
19          if τij in P+ then -- the scheduling policy can be changed
20             S(τij) = policy
21          end if
22          adjust bus cycle(A, M, B, S)
23          recompute FPS priority levels
24          for all FPS tasks τab sharing identical priority levels do S(τab) = EDF end for
25          HolisticScheduling(A, M, B, S)
26          if δA < best_δA then
27             best_policyij = S(τij); best_processorij = M(τij)
28             best_δA = δA
29          end if
30          if δA < 0 then
31             return best (M, B, S)
32          end if
33       end for
34    end for
35 end for
end MSPAHeuristic
```

Fig. 17 Policy assignment and mapping heuristic

Our optimisation algorithm is presented in Fig. 17 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 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; consequently, there may be new ST messages that are larger than the current static slot for the sender node (or similarly the bus will face the situation where new DYN messages are larger than the largest DYN phase in the bus cycle). For more details on the subject of bus access optimisation and adjustment, the reader is referred to [47].

Before the system is analysed for its timing properties, our heuristic also tries to optimise 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 [19]. However, due to the fact that HOPA is computationally expensive to be run inside such a design optimisation loop, we use a scaled down greedy algorithm, in which we drastically reduce the number of iterations needed for determining an optimised priority assignment.

Finally, the resulted system configuration is analysed (line 13) using the scheduling and schedulability analysis algorithm presented in Sect. 3.2. The resulted cost function will decide whether the current configuration is better than the current best one (lines 14–17). Moreover, if all activities meet their deadlines ($\delta_{\mathcal{A}} < 0$), the optimisation heuristic stops the exploration process and returns the current best-so-far configuration (lines 18–20).

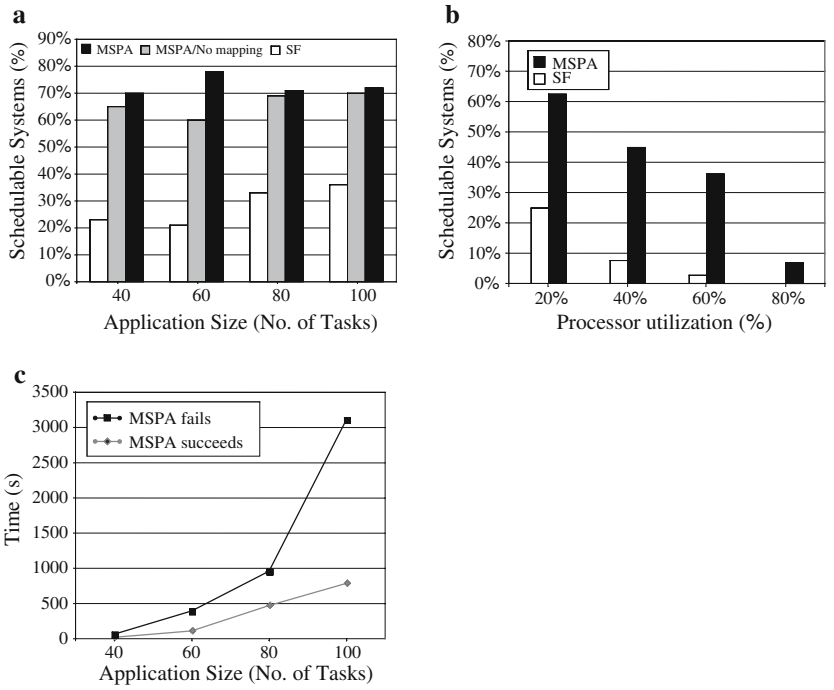

Fig. 18 Performance of the design optimisation heuristic

6 Experimental Results

For the evaluation of our design optimisation 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 four processors. Fifty-six applications were generated for each dimension, thus a total of 224 applications were used for experimental evaluation. An equal number of applications with processor utilisations 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 optimisation approach for hierarchically scheduled systems, the MSPAHeuristic (MSPA); see Fig. 17. We have compared the percentage of schedulable implementations found by MSPA with the number of schedulable solutions obtained by the InitialMSPA algorithm described in Fig. 5 (line 2), which derives a straight-forward system implementation, denoted with SF. The results are depicted in Fig. 18a. 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 optimised. Figure 18a presents this approach, labeled "MSPA/No mapping", corresponding to the grey bars. We can see that most of the improvement over the SF approach is obtained by carefully optimising the SPA in our MSPA heuristic.

We were also interested to find out what is the impact of the processor utilisation of an application on the quality of the implementations produced our optimisation heuristic. Figure 18b presents the percentage of schedulable solutions found by MSPA and SF as we ranged the utilisation from 20 to 80%. We can see that SF degrades very quickly with the increased utilisation, with under 10% schedulable solutions for applications with 40% utilisation and without finding any schedulable solution for applications with 80% utilisation, while MSPA is able to find a significant number of schedulable solutions even for high processor utilisations.

In Fig. 18c, we show the average runtimes obtained by applying our MSPA heuristic on the examples presented in Fig. 18a. The upper curve illustrates the average execution times for those applications which were not found schedulable by our heuristic. This curve can be considered as an upper bound for the computation time of our algorithm. For the examples that were found schedulable, our heuristic stops the exploration process earlier, thus leading to smaller computation times, as shown in the lower curve in Fig. 18c. We can see that, considering the complex optimisation steps performed, our design optimisation heuristic produces good quality results in a reasonable amount of time (for example, the heuristic will finish on average in less than 500 s 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 [44]. 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 optimisation heuristic MSPA was considered first such that the mapping is fixed by SF, and we only allowed reassigning of scheduling policies. After 29.5 s, 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 s.

7 Conclusions

In this paper we have defined and solved the problem of scheduling heterogeneous ET/TT distributed embedded systems. We have proposed several alternative solutions for the scheduling algorithm and we have run extensive experiments in order to compare the efficiency of the alternatives. We have also proposed a design optimisation 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 utilisation of the system.

References

1. Abdelhazer, T.F., Shin, K.G.: Combined task and message scheduling in distributed real-time systems. IEEE T. Parall. Distr. **10**(11), 1179–1191 (1999)
2. Agrawal, G., Chen, B., Zhao, W., Davari, S.: Guaranteeing synchronous message deadlines with the timed token medium access control protocol. IEEE T. Comput. **43**(3), 327–339 (1994)
3. Almeida, L.: Flexibility and timeliness in fieldbus-based real-time systems. Ph. D. Thesis, University of Aveiro, Portugal (1999)
4. Almeida, L., Pedreiras, P., Fonseca, J.A.G.: The FTT-CAN protocol: why and how. IEEE T. Ind. Electron. **49**(6), 1189–1201 (2002)
5. Audsley, N., Tindell, K., Burns, A.: The end of line for static cyclic scheduling? 5th Euromicro Workshop on Real-Time Systems (1993)
6. Audsley, N., Burns, A., et. al.: Fixed priority preemptive scheduling: an historical perspective. Real-Time Syst **8**(2/3) (1995)
7. Balarin, F., Lavagno, L., et. al.: Scheduling for embedded real-time systems. IEEE Des Test Comput, January–March (1998)
8. Bini, E., Butazzo, G., Butazzo, G.: A hyperbolic bound for the rate monotonic algorithm. Proceedings of the 13th Euromicro Conference on Real-Time Systems, pp. 59–66 (2001)
9. Coffman, E.G. Jr., Graham, R.L.: Optimal scheduling for two processor systems. Acta Inform **1**, (1972)
10. Demmeler, T., Giusto, P.: A Universal Communication Model for an Automotive System Integration Platform, pp. 47–54. Design, Automation and Test in Europe (DATE'01) Conference, Munich (2001)
11. Dobrin, R., Fohler, G.: 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)
12. Dobrin, R., Fohler, G., Puschner, P.: Translating off-line schedules into task attributes for fixed priority scheduling. Proceedings of Real-Time Systems Symposium (2001)
13. Ekelin, C., Jonsson, J.: Solving embedded system scheduling problems using constraint programming. Chalmers University of Technology, Sweden, Report number TR 00–12 (2000)
14. Eles, P., Doboli, A., Pop, P., Peng, Z.: Scheduling with bus access optimization for distributed embedded systems. IEEE T. VLSI Syst. **8**(5), 472–491 (2000)
15. Ermedahl, H., Hansson, H., Sjödin, M.: Response time guarantees in ATM networks. Proceedings of Real-Time Systems Symposium (1997)
16. FlexRay homepage: http://www.flexray-group.com/ (2006)
17. González Harbour, M., Klein, M.H., Lehoczky, J.P.: Fixed priority scheduling of periodic tasks with varying execution priority. Proceedings of 12th IEEE Real-Time Systems Symposium, pp. 116–128 (1991)
18. Gonzaléz Harbour, M., Palencia, J.C.: Response time analysis for tasks scheduled under EDF within fixed priorities. Proceedings of Real-Time Systems Symposium, pp. 200–209 (2003)
19. Gutiérrez García, J.J., González Harbour, M.: Optimized priority assignment for tasks and messages in distributed hard real-time systems. Proceedings of the 3rd Workshop on Parallel and Distributed Real-Time Systems, Santa Barbara, pp. 124–132 (1995)
20. Hansson, H., Sjödin, M., Tindell, K.: Guaranteeing real-time traffic through an ATM network. Proceedings of the 30th Hawaii International Conference on System Sciences, **5** (1997)
21. Jonsson, J., Shin, K.J.: A parametrized branch-and-bound strategy for scheduling precedence-constrained tasks on a multiprocessor system. Proceedings of the International Conference on Parallel Processing (ICPP), pp. 158–165 (1997)
22. Jorgensen, P.B., Madsen, J.: Critical path driven cosynthesis for heterogeneous target architectures. Proceedings of the 5th International Workshop on Hardware-Software Co-design, pp. 15–19 (1997)
23. Koopman, P.: Critical embedded automotive networks. IEEE Micro. **22**(4), 14–18 (2002)

24. Kopetz, H., Fohler, G., Grünsteidl, G., Kantz, H., Pospischil, G., Puschner, P., Reisinger, J., Schlatterbeck, R., Schütz, W., Vrchoticky, A., Zainlinger, R.: The programmer's view of MARS. Proceedings of 13th IEEE Real-Time Systems Symposium, pp. 223–226 (1992)
25. Kopetz, H.: Real-time systems—design principles for distributed embedded applications. Kluwer Academic Publisher (1997)
26. Kuchcinski, K.: Embedded system synthesis by timing constraint solving. Proceedings of the International Symposium on System Synthesis, pp. 50–57 (1997)
27. Lehoczky, J., Sha, L., Ding, Y.: The rate monotonic scheduling algorithm: exact characterization and average case behaviour. Proceedings of 11th Real-Time Systems Symposium, pp. 166–171 (1989)
28. Lehoczky, J.P.: Fixed priority scheduling of periodic task sets with arbitrary deadlines. Proceedings of 11th IEEE Real-Time Systems Symposium, pp. 201–209 (1990)
29. Leung, J.Y.T., Whitehead, J.: On the complexity of fixed priority scheduling of periodic, real-time tasks. Perform. Evaluation **2**(4), 237–250 (1989)
30. Liu, C.L., Layland, J.W.: Scheduling algorithms for multiprogramming in a hard real-time environment. J. ACM **20**(1), 46–61 (1973)
31. Livani, M.A., Kaiser, J.: EDF consensus on CAN bus access for dynamic real-time applications. Proceedings of the IPPS/SPDP Workshops, pp. 1088–1097 (1998)
32. Lönn, H., Axelsson, J.: A comparison of fixed-priority and static cyclic scheduling for distributed automotive control applications. Proceedings of the 11th Euromicro Conference on Real-Time Systems, pp. 142–149 (1999)
33. Douglas Locke, C.: Software architecture for hard-real time applications: cyclic executives vs. fixed priority executives. J. Real-Time Syst. **4**, 37–53 (1992)
34. Palencia, J.C., González Harbour, M.: Schedulability analysis for tasks with static and dynamic offsets. Proceedings of the 19th IEEE Real-Time Systems Symposium, pp. 26–37 (1998)
35. Palencia, J.C., González Harbour, M.: Exploiting precedence relations in the schedulability analysis of distributed real-time systems. Proceedings of the 20th IEEE Real-Time Systems Symposium (1999)
36. Palencia, J.C., Gonzaléz Harbour, M.: Offset-based response time analysis of distributed systems scheduled under EDF. Proceedings of the Euromicro Conference on Real-Time Systems, pp. 3–12 (2003)
37. Pleinevaux, P.: An improved hard real-time scheduling for IEEE 802.5. J. Real-Time Syst. **4**(2) (1992)
38. Pop, P., Eles, P., Peng, Z.: Bus access optimization for distributed embedded systems based on schedulability analysis, pp. 567–574. Design, Automation and Test in Europe (DATE'00) (2000)
39. Pop, P., Eles, P., Peng, Z., Doboli, A.: Scheduling with bus access optimization for distributed embedded systems. IEEE T. VLSI Syst. **8**(5), 472–491 (2000)
40. Pop, P., Eles, P., Pop, T., Peng, Z.: An approach to incremental design of distributed embedded systems. Proceedings of the 38th Design Automation Conference (DAC), pp. 450–455. Las Vegas, USA (2001)
41. Pop, P., Eles, P., Pop, T., Peng, Z.: Minimizing system modification in an incremental design approach. Proceedings of the 9th International Workshop on Hardware/Software Codesign (CODES 2001), pp. 183–188. Copenhagen, Denmark (2001)
42. Pop, P., Eles, P., Peng, Z.: Schedulability-driven communication synthesis for time-triggered embedded systems. Real-Time Syst. J. **24**, 297–325 (2004)
43. Pop, P., Eles, P., Peng, Z.: Schedulability analysis and optimization for the synthesis of multi-cluster distributed embedded systems. Design, Automation and Test in Europe (DATE'03) Conference, pp. 184–189. Munich, Germany (2003)
44. Pop, P., Eles, P., Peng, Z.: Analysis and synthesis of Distributed Real-Time Embedded Systems. Kluwer Academic Publishers (2004)
45. Pop, T., Eles, P., Peng, Z.: Holistic scheduling and analysis of mixed time/event-triggered distributed embedded systems. Proceedings of the 10th International Symposium on Hardware/Software Codesign (CODES 2002), pp. 187–192. Estes Park, Colorado, USA (2002)
46. Pop, T., Eles, P., Peng, Z.: Schedulability analysis for distributed heterogeneous time/event-triggered real-time systems, 15th Euromicro Conference on Real-Time Systems (ECRTS 2003), pp. 257–266 (2003)
47. Pop, T., Eles, P., Peng, Z.: Design optimization of mixed time/event triggered distributed embedded systems. CODES+ISSS'03 (first merged conference) (2003)

48. Pop, T., Pop, P., Eles, P., Peng, Z.: Optimization of hierarchically scheduled heterogeneous embedded systems. 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'05), pp. 67–71 (2005)
49. Prakash, S., Parker, A.: SOS: synthesis of application specific heterogeneous multiprocessor systems. J. Parallel Distr. Com. **16**, 338–351 (1992)
50. Richter, K., Ernst, R.: Event model interfaces for heterogeneous system analysis. Proceedings of Design, Automation and Test in Europe Conference (DATE'02), Paris, France (2002)
51. Richter, K., Jersak, M., Ernst, R.: A formal approach to MpSoC performance verification. IEEE Comput. **36**(4), 60–67 (2003)
52. Schwehm, M., Walter, T.: Mapping and scheduling by genetic algorithms. Conference on Algorithms and Hardware for Parallel Processing, pp. 832–841 (1994)
53. Sha, L., Rajkumar, R., Lehoczky, J.P.: Priority inheritance protocols: an approach to real time synchronization. IEEE T. Comput. **39**(9), 1175–1185 (1990)
54. Stankovic, J.A., Spuri, M., di Natale, M., Butazzo, G.C.: Implications of classical scheduling results for real-time systems. Technical Report UM-CS-94–089, Computer Science Department, University of Massachusetts (1994)
55. Strosneider, J.K., Marchok, T.E.: Responsive, deterministic IEEE 802.5 Token Ring Scheduling. J. Real-Time Syst. **1**(2), (1989)
56. Tindell, K., Burns, A., Wellings, A.J.: Allocating hard real-time tasks (An NP-hard problem made easy). J. Real-Time Syst. **4**(2), 145–165 (1992)
57. Tindell, K., Clark, J.: Holistic schedulability analysis for distributed hard real-time systems. Microproc. Microprog. **50**(2–3) (1994)
58. Tindell, K., Hansson, H., Wellings, A.J.: Analysing real-time communications: controller area network (CAN). Proceedings of Real-Time Systems Symposium (1994)
59. Tindell, K.: Adding time-offsets to schedulability analysis. Department of Computer Science, University of York, Report Number YCS-94–221 (1994)
60. Ullman, D.: NP-complete scheduling problems. J. Comput. Syst. Sci. **10**(3), 384–393 (1975)
61. Wandeler, E., Thiele, L.: Real-time interfaces for interface-based design of real-time systems with fixed priority scheduling. ACM Conference on Embedded Software (EMSOFT), pp. 80–89 (2005)
62. World FIP fieldbus, http://www.worldfip.org/, April 2003
63. Xu, J., Parnas, D.L.: On satisfying timing constraints in hard-real-time systems. IEEE T. Software Eng. **19**(1) (1993)