Scheduling Dependent Periodic Tasks Without Synchronization Mechanisms

Julien Forget, Frédéric Boniol, Emmanuel Grolleau, David Lesens, Claire Pagetti

Scheduling Dependent Periodic Tasks Without Synchronization Mechanisms

Julien Forget*, Frédéric Boniol*, Emmanuel Grolleau [‡], David Lesens[†] and Claire Pagetti*[§]
*ONERA, Toulouse, France, Email: firstname.lastname@onera.fr
[†]EADS Astrium Space Transportation, Les Mureaux, France
[‡]LISI/ENSMA, Université de Poitiers, France, Email: grolleau@ensma.fr
[§]ENSEEIHT, Toulouse, France

***Abstract*—This article studies the scheduling of critical embedded systems, which consist of a set of communicating periodic tasks with constrained deadlines. Currently, tasks are usually sequenced manually, partly because available scheduling policies do not ensure the determinism of task communications. Ensuring this determinism requires scheduling policies supporting task precedence constraints (which we call dependent tasks), which are used to force the order in which communicating tasks execute. We propose fixed priority scheduling policies for different classes of dependent tasks: with simultaneous or arbitrary release times, with simple precedences (between tasks of the same period) or extended precedences (between tasks of different periods). We only consider policies that do not require synchronization mechanisms (like semaphores). This completely prevents deadlocks or scheduling anomalies without requiring further proofs.**

I. Introduction

This work was originally motivated by the programming of *highly critical embedded control systems*, which consist of control loops including sensors, control algorithms and actuators that regulate the state of a system in its environment. The implementation of this kind of systems usually involves several persons and teams working in parallel to develop the different parts of the system. An integrator then assembles these elements, let us say operations or *tasks*, to provide the complete implementation of the system. This assembly process currently often amounts to sequencing manually the set of tasks offline to ensure that the execution of the implementation is predictable and deterministic. Reliable embedded real-time operating systems have emerged such as OSEK [1] for automotive, RTEMS [2] for aerospace or systems respecting the ARINC 653 standard [3] for aeronautics ([4] for instance). They enable to replace the manual task sequencing by a multi-task implementation, where tasks are scheduled concurrently online (at system execution) by the operating system. However, they do not directly provide scheduling policies for *dependent tasks* (tasks related by data-dependencies), meaning that the determinism of task communications is usually ensured manually by the programmers. For instance, the dependent tasks can first be ordered manually and the operating system then schedules this partially "pre-scheduled" task set. This is unfortunately tedious and time consuming. The purpose of this paper is to investigate scheduling policies for critical embedded control systems, which directly support dependent tasks.

A. General Characteristics

Spacecraft and aircraft flight control systems are good examples of embedded control systems. Their objective is to supervise the position, speed and attitude of the vehicle thanks to physical devices, such as control surfaces for an aircraft or thrusters for a spacecraft. Such a system must respect a series of hard real-time constraints. First, it is often multi-periodic since the devices have different physical characteristics and must therefore be controlled at different rates. Second, the system must respect deadline constraints, which may correspond for instance to a maximum end-to-end latency requirement between observations (inputs) and the corresponding reactions (outputs).

A correct implementation must respect all the real-time constraints and must also be functionally deterministic, meaning that the outputs of the system are always the same for a given sequence of inputs. Scheduling theory already provides scheduling policies to respect the temporal requirements. Ensuring the functional determinism requires to control the order in which communicating tasks execute. Indeed, when data produced by an operation is consumed by other operations, the relative order of the producer and of the consumers has a direct impact on the values produced by the operations. From the scheduling point of view, controlling this order amounts to adding precedence constraints between operations, which ensure that the producer of the data will be executed before the consumer of the data. Fully ensuring functional determinism also requires a deterministic communication protocol, but such a protocol is out of the scope of this paper (see for instance [5], [6], [7]).

B. Case Study: The Flight Application Software

To motivate our work, we consider an adapted version of the Flight Application Software (FAS) of the Automated Transfer Vehicle (ATV) designed by EADS Astrium Space Transportation for resupplying the International Space Station (ISS).

1) General Presentation: The FAS handles all the software functionalities of the system as long as no fault is detected. Fig. 1 provides a simplified informal description of its software architecture. Each operation is represented by a box and arrows between boxes represent data-dependencies. Arrows without sources represent system inputs and arrows without destinations represent system outputs.

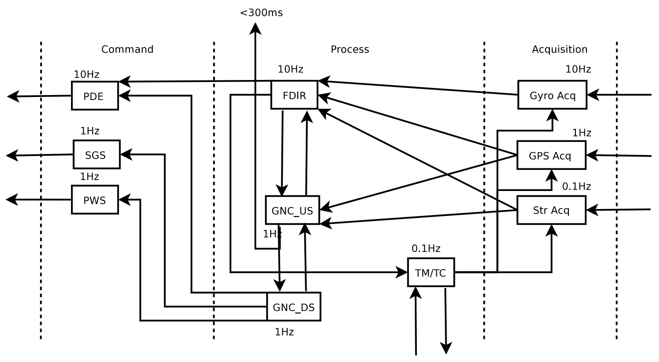


Figure 1. The Flight Application Software

The FAS first acquires data: the orientation and speed from gyroscopic sensors (Gyro Acq), the position from the GPS (GPS Acq) and from the star tracker (Str Acq) and telecommands from the ground station (TM/TC). The *Guidance Navigation and Control* function (divided into an upstream part, GNC_US, and a downstream part, GNC_DS) then computes the commands to apply while the *Failure Detection Isolation and Recovery* function (FDIR) verifies the state of the FAS and checks for possible failures. Finally, commands are computed and sent to the control devices: thruster orders for the *Propulsion Drive Electronics* (PDE), power distribution orders for the *Power System* (PWS), solar panel positioning orders for the *Solar Generation System* (SGS) and telemetry towards the ground station (TM/TC). Each operation has its own rate, ranging from 0.1Hz to 10Hz. An intermediate deadline constraint is imposed on data produced by the GNC_US (300ms while the period is 1s).

2) Deterministic Communications: The system contains two different kinds of communications, which correspond to two different kinds of precedences. When two tasks of the same period are related by a data-dependency, we can simply impose that the producer always executes before the consumer. This corresponds to usual *simple precedences*. When the producer and the consumer have different periods, there are several possible communication patterns. For instance, if the producer is 10 times faster than the consumer, the specification can impose that the consumer takes data produced by the second instance out of 10 successive instances of the producer. Such communication patterns correspond to more complex *extended precedences*, which only relate a subset of the instances of the communicating tasks. In this paper, we will give two specifications of the FAS with simple and extended precedences. We study the scheduling problem of periodic tasks with these two kinds of precedence constraints.

C. General Reminder and Notations

A system $\mathcal{S}$ consists of a set of tasks, where each task τ_i has a set of real-time attributes (T_i, C_i, O_i, D_i). τ_i is instantiated periodically with period T_i, $\tau_i[p]$ denotes the p^{th} iteration of task τ_i. C_i is the worst case execution time (wcet) of the task. O_i is the release time of the first instance of the task (its offset with respect to the start time of the system). Let $o_i[p]$ denote the release time of the instance p of τ_i, we have $o_i[p] = O_i + pT_i$. D_i is the relative deadline of the task. Let $d_i[p]$ denote the absolute deadline of the instance p of τ_i, we have $d_i[p] = o_i[p] + D_i$. These definitions are illustrated in Fig. 2.

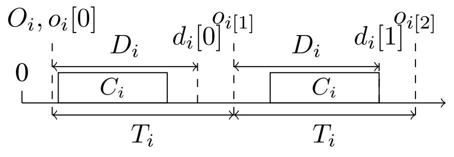


Figure 2. Real-time characteristics of task τ_i

Tasks contain no internal synchronization points, meaning that a task cannot execute a blocking instruction (at least not for longer than its wcet). A task can however be preempted by the scheduler during its execution in order to execute another task (it is temporally suspended and resumed later).

Periodic task scheduling can be solved efficiently by using a priority-based scheduling approach. Task priorities can either be static - the priority of a task remains the same for the complete program execution - or dynamic - the priority of a task can change during execution. Let $k_i[p]$ be the starting time of $\tau_i[p]$ and $f_i[p]$ be the completion time of $\tau_i[p]$ in the schedule produced by a given priority assignment. We say that a task set is *feasible* under a given priority assignment if the schedule produced by this assignment respects all the constraints of the task set. More formally:

Definition 1: Let $\mathcal{S}$ be a set of independent tasks. $\mathcal{S}$ is *feasible* under a given priority assignment if and only if, $\forall\tau_i, \forall p$:

$$f_i[p] \leq d_i[p] \wedge k_i[p] \geq o_i[p]$$

When we simply say that a task set is feasible (i.e. not with respect to a given priority assignment), this means that there exists a priority assignment under which the task set is feasible.

For independent periodic tasks, that is to say without precedence constraints, several optimal[1] policies are available for different families of systems. Among these policies, the fundamental work of [8] proposed the *rate-monotonic*

[1]If there exists a priority assignment that schedules the task set correctly, then the task set is feasible under the optimal assignment.

(RM) static priority policy, where tasks with a shorter period are assigned a higher priority and the *earliest-deadline-first* (EDF) dynamic priority policy, where task instances with a shorter absolute deadline are assigned a higher priority.

D. Related Works

We are interested in dependent periodic tasks. We need optimal priority-based preemptive scheduling policies, and associated schedulability tests, for tasks with simple or extended precedences. There are mainly two different approaches available. The first approach relies on the use of (binary) semaphore synchronizations: a semaphore is allocated for each precedence and the destination task of the precedence must wait for the source task of the precedence to release the semaphore before it can start its execution. In the second approach, the respect of precedence constraints is simply ensured by the way priorities and release times are assigned to tasks.

1) Dependent Periodic Tasks, simple precedences: Using the first approach, [9] describes a schedulability test for periodic tasks with predefined static priorities, simultaneous release times (the same release time for every task), deadline constraints and precedence constraints, but only considers a special case of simple precedences: each periodic task consists of a set of subtasks totally ordered by precedences (i.e. each task consists of a *sequence* of subtasks). This restriction is clearly not adapted to our context because with this model the precedence relation between tasks can only form a set of disjoint precedence chains, while we want to enable the precedence relation to form an arbitrary complex Directed Acyclic Graph. [10] gives a sufficient schedulability test for simple precedences in general, tasks can then be scheduled using the *deadline-monotonic* (DM) policy and the respect of precedence constraints is ensured by semaphores.

Using the second approach, [11] encodes precedences in task real-time attributes by adjusting the release times and the deadlines of the tasks and the adjusted task set is then scheduled with the EDF policy. Tasks can have arbitrary release times. This technique was originally only proposed for aperiodic tasks with deadline and precedence constraints. It is optimal in the sense that a valid schedule can be found for the original task set if and only if a valid schedule can be found for the adjusted task set. As a result, schedulability can be tested by applying an EDF schedulability test on the encoded task set. The encoding technique directly applies to the case of periodic tasks with constrained deadlines and simple precedences and remains optimal. However, [11] neither considers static priority scheduling policies nor extended precedences. We propose extensions for both points.

2) Dependent Periodic Tasks, extended precedences: [12] extended the technique of [11] to particular periodic extended precedences. When the consumer has a period greater than or equal to the one of the producer, data consumed is the last value produced in the period of the consumer. This extension consists in unfolding the extended precedence graph on the hyperperiod of the tasks (the hyperperiod HP is equal to the least common multiple of the periods of the tasks), replacing each task τ_i by HP/T_i duplicates of period HP, and using [11] to encode the simple precedences of the unfolded graph. The encoded task set can then be validated with respect to EDF. If semaphore synchronizations are allowed, the graph unfolding is only conceptual: real-time attributes are modified only for the validation of the system, not for its execution and at run-time precedences are ensured by semaphore synchronizations. However, if semaphore synchronizations are not allowed, the task graph must be unfolded for the execution. Indeed, duplicates of the same task cannot be merged into a single task, as they may have different real-time attributes after precedence encoding. This can lead to important computation overhead as the scheduler needs to make its decisions according to a large task set. This also implies high memory consumption as many tasks are allocated. Our work avoids such costly duplications and supports more general extended precedences.

E. Scheduling with/without Synchronization Mechanisms

On the one hand, allowing semaphore synchronizations can lead to more powerful scheduling policies. Let us for instance consider a system with simultaneous release times and deadlines equal to periods, made up of three tasks, $\tau_1(T = 8, C = 3)$, $\tau_2(T = 12, C = 5)$ and $\tau_3(T = 12, C = 2)$, and with a precedence from τ_2 to τ_3. If we do not allow semaphore synchronizations, this system is not schedulable with a static priority policy (it is easy to check that each possible priority assignment causes at least one task to miss its deadline). However, as illustrated in Fig. 3, this problem is schedulable if we allow semaphore synchronizations: we assign task priorities such that $prio(\tau_3) > prio(\tau_1) > prio(\tau_2)$ (ticks on the time axis represent task periods). With dynamic priorities, allowing semaphore synchronizations does not seem to produce more powerful scheduling policies, though this remains to be proved.

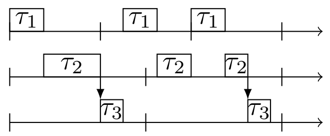


Figure 3. Static priority scheduling with semaphore synchronizations

On the other hand, policies without semaphore synchronizations have significant advantages, which make them better suited for highly critical systems. First, semaphore synchronizations can lead to scheduling anomalies, where a system is proved schedulable using schedulability tests, but during the execution the system becomes unschedulable due to a task that does not take its complete wcet to execute [13], [10]. Second, known schedulability tests for scheduling priorities with synchronization mechanisms ([9], [10]) only provide sufficient schedulability conditions, but not necessary schedulability conditions. The complexity of the schedulability problem actually remains an opened question. Finally, highly critical systems must go through certification processes, such as the DO-178B [14] for airborne systems or the IEC 61508 for more general embedded systems. These certification processes apply to any part of the embedded system, including the Operating System. Therefore, relying on policies without semaphore synchronizations leads to a more compact Operating System, which is easier to certify. For these reasons, this paper focuses on policies without semaphore synchronizations.

F. Contribution

We propose fixed priority scheduling policies, without semaphore synchronizations, and associated schedulability tests for a set of periodic tasks with deadline and precedence constraints. We successively consider periodic task sets with simple precedences and simultaneous release times in Sec. II, periodic task sets with simple precedences and arbitrary release times in Sec. III and finally periodic task sets with extended precedences and arbitrary release times in Sec. IV. Some of the results presented here can quite easily be derived from existing results, but to the best of our knowledge these new results have neither been established formally nor proved optimal before.

II. Simple Precedences and Simultaneous Release Dates

We first consider the problem of scheduling a set of periodic tasks with simultaneous release times, constrained deadlines and simple precedences. The policy proposed below is derived from DM [15], where tasks are assigned priorities according to their deadlines: the task with the shortest deadline being assigned the highest priority. DM is optimal in the class of static priority policies for scheduling the considered family without precedences.

A set of simple precedences is formalized by a relation $\rightarrow$ which is a subset of $\mathcal{S} \times \mathcal{S}$: $\tau_i \rightarrow \tau_j$ states that τ_i must execute before τ_j. We assume that the graph of precedence constraints is acyclic, otherwise the system is not causal (we cannot find an execution order that respects all the precedence constraints). Let $preds(\tau_i) = \{\tau_j | (\tau_j, \tau_i) \in \rightarrow\}$ denote the predecessors of τ_i and $succs(\tau_i) = \{\tau_j | (\tau_i, \tau_j) \in \rightarrow\}$ denote its successors.

Definition 2: Let $\mathcal{S}$ be a set of dependent tasks where $\rightarrow \subseteq \mathcal{S} \times \mathcal{S}$ is the precedence relation on $\mathcal{S}$. $\mathcal{S}$ is *feasible* under a given priority assignment if and only if, $\forall \tau_i, \forall p$:

- $f_i[p] \leq d_i[p] \wedge k_i[p] \geq o_i[p]$;
- $\forall \tau_j \in preds(\tau_i), f_j[p] \leq k_i[p]$

A. Scheduling Policy

The idea behind the precedence encoding technique of [11] applies to the present scheduling problem: "the relative urgency of a task depends both on its deadline and on the deadlines of its successors". Thus, the deadline of a task can be adjusted as follows to *encode* precedence constraints:

$$D_i^* = min(D_i, \min_{\tau_j \in succs(\tau_i)} (D_j^* - C_j)) \qquad (1)$$

The scheduling policy then consists in adjusting task deadlines and in applying the DM policy on the adjusted task set. As explained in [11], adjusting the deadlines of a task set can be performed using a topological sort: we start by adjusting the deadlines of tasks without successors and then progressively we adjust the deadlines of the tasks the successors of which have already been adjusted.

B. Optimality and Complexity

We show that the theorem proved in [11], which assumes the use of a dynamic priority scheduling policy, also holds for static priority (for tasks with simultaneous release times in our case).

Theorem 1: Let $\mathcal{S} = \{\tau_i(T_i, C_i, 0, D_i)\}$ be a set of dependent tasks and $\rightarrow \subseteq \mathcal{S} \times \mathcal{S}$. Let $\mathcal{S}^* = \{\tau_i^*(T_i, C_i, 0, D_i^*)\}$ be a set of independent tasks such that D_i^* is given by the formula (1). Considering only fixed priority scheduling policies, we have:

$\mathcal{S}$ is feasible if and only if $\mathcal{S}^*$ is feasible.

Proof: Easily adapted from [11].

(If part) Suppose $\mathcal{S}^*$ is feasible. Then for all $\tau_i[p]$, $f_i[p] \leq d_i^*[p]$. As $d_i^*[p] \leq d_i[p]$, then $f_i[p] \leq d_i[p]$, so a valid schedule for $\mathcal{S}^*$ respects the deadline constraints of $\mathcal{S}$. Now for precedence constraints, for all task τ_i, τ_j, if $\tau_i \rightarrow \tau_j$ then for all p, $d_i^*[p] < d_j^*[p]$. As DM is optimal for scheduling independent tasks, it can be used to schedule $\mathcal{S}^*$. Then, τ_i will have a higher priority than τ_j, as it has a shorter deadline. As $o_i[p] = O_j[p]$ (due to simultaneous release times), we have $k_j[p] > f_i[p]$ and thus the precedence constraint is met.

(Only if part) Suppose $\mathcal{S}$ is feasible. Then for all $\tau_i[p]$, $f_i[p] < d_i^*[p]$, otherwise the successors of τ_i cannot respect their deadlines. Thus $\mathcal{S}^*$ is feasible. ■

As DM is optimal for scheduling $\mathcal{S}^*$, the theorem implies that the proposed scheduling policy is optimal for simple precedences with simultaneous release times.

This policy has a complexity of $\mathcal{O}(n^2)$, where n denotes the number of tasks to schedule. Indeed, the encoding requires to perform a topological sort, which can be achieved with a complexity of $\mathcal{O}(n^2)$. We can then use standard array sorting algorithms to sort tasks by increasing deadlines, with a complexity of $\mathcal{O}(nlog_2(n))$.

C. Feasibility Analysis

We can simply reuse a feasibility analysis defined for DM, for instance [15] or [16]. We compute the schedule obtained for our priority assignment until the longest first deadline of the tasks and verify that deadlines are met (precedence constraints are met by construction).

D. Example

We provide a specification of the FAS with simple precedences, denoted V_1. For now, we accept a certain degree of non determinism in the communications between tasks with different periods, we only impose an order between operations of the same period. It is up to the designer to choose this order. The system contains communication loops (cycles), for instance between GNC_US and GNC_DS, thus not all communications can imply precedence constraints. We can for instance impose that GNC_US is always executed before GNC_DS, thus there is a precedence from GNC_US to GNC_DS, but not from GNC_DS to GNC_US. In this case, the communication from GNC_DS to GNC_US is called a *delayed communication* and each instance of GNC_US consumes the value produced by the previous instance of GNC_US. The operations order is specified by the following precedences (implicitly, communications without precedences are all delayed communications):

10Hz	Gyro Acq → FDIR, FDIR →PDE
1Hz	GNC_US → GNC_DS, GPS Acq → GNC_US, GNC_DS → SGS, GNC_DS → PWS

If we implement the FAS according to the specification V_1 with this first scheduling policy, we obtain the following system which is feasible. This is verified easily using the CHEDDAR tool [17] for instance.

Name	T in ms	C	D	D*	Priority
PDE	100	5	100	100	3
SGS	1000	15	1000	1000	7
PWS	1000	20	1000	1000	8
FDIR	100	10	100	95	2
GNC_US	1000	20	300	300	5
GNC_DS	1000	20	1000	980	6
TM/TC	10000	200	10000	10000	10
Gyro Acq	100	15	100	85	1
GPS Acq	1000	10	1000	280	4
Str Acq	10000	100	10000	10000	9

As FDIR precedes PDE, its adjusted deadline is: $D^{\star}_{FDIR} = D_{PDE} - C_{PDE} = 100 - 5 = 95$. Then, as Gyro Acq precedes FDIR, we have: $D^{\star}_{GyroAcq} = D^{\star}_{FDIR} - C_{FDIR} = 95 - 10 = 85$. Notice that we must compute $D^{\star}_{FDIR}$ before computing $D^{\star}_{GyroAcq}$: this is ensured by the fact that we adjust the deadlines of the task set following a topological sort starting from the tasks without successors (for instance starting from PDE here). We proceed similarly for the other precedences of the task set. Once this encoding is complete, priorities are simply assigned according to the DM policy.

III. Simple Precedences and Arbitrary Release Times

We now consider the problem of scheduling a set of periodic tasks with arbitrary release times, constrained deadlines and simple precedences. The policy proposed below is derived from the policy described by Audsley in [18], which is optimal for this class of systems (in the class of fixed priority policies) when there are no precedences.

A. Scheduling Policy

1) Adjusting Release Times: Let $\mathcal{S}$ be a task set made up of two tasks, τ_i and τ_j, such that $\tau_i \to \tau_j$ and $O_i > O_j$. Obviously, if we do not use semaphore synchronizations and if we do not adjust task real-time attributes, then we cannot respect the precedence constraint as τ_j will start before τ_i, whatever priorities are assigned to the two tasks. This suggests that we should adjust task release times as follows:

$$O_i^* = max(O_i, \max_{\tau_j \in preds(\tau_i)}(O_j^*)) \tag{2}$$

Since we manipulate deadlines relative to release times, we have to modify deadlines as follows:

$$D_i^* = D_i + O_i - O_i^* \tag{3}$$

Notice that if for any τ_i we have $D_i^* < O_i^*$, then $\mathcal{S}$ is trivially not feasible (but this condition is not a sufficient test).

Then, we have:

Lemma 1: Let $\mathcal{S} = \{\tau_i(T_i, C_i, O_i, D_i)\}$ and $\to \subseteq \mathcal{S} \times \mathcal{S}$. Let $\mathcal{S}^* = \{\tau_i'(T_i, C_i, O_i^*, D_i^*)\}$ be a set of tasks such that O_i^* and D_i^* are given by the formulas (2) and (3), and let $\to' \subseteq \mathcal{S}^* \times \mathcal{S}^*$ be the transposition of $\to$ to $\mathcal{S}^*$:

$\mathcal{S}$ is feasible if and only if $\mathcal{S}^*$ is feasible.

Proof: Again, easily deduced from [11].

(If part) Suppose $\mathcal{S}^*$ is feasible. Then for all instances $\tau_i[p]$, $k_i[p] \geq O_i^*[p] = O_i^* + lT_i$. As $O_i^* \geq O_i$, then $k_i[p] \geq O_i + pT_i$. We have also $f_i[p] \leq d_i^*[p] = O_i^*[p] + D_i^* = O_i^* + pT_i + D_i^* = O_i^* + pT_i + D_i + O_i - O_i^* = O_i + D_i + pT_i = d_i[p]$. Thus $\mathcal{S}$ is feasible.

(Only if part) Suppose $\mathcal{S}$ is feasible. Then for all τ_i, $k_i[p] \geq O_i[p]$. For any predecessor τ_j, we have $k_i[p] \geq O_j[p] = O_j + pT_j$. Thus, $k_i[p] \geq max(O_i + pT_i, O_j + pT_j) = max(O_i, O_j) + pT_i$. By applying this reasoning on all the predecessors, we obtain that $k_i[p] \geq O_i^*[p]$. Moreover, $f_i[p] \leq d_i[p] = O_i + pT_i + D_i = D_i^* + O_i^* + pT_i$. Thus $\mathcal{S}^*$ is feasible. ■

As for deadlines adjustment, release times adjustment can be performed using a topological sort, this time starting from tasks without predecessors. Notice that we did not encode the precedence relation, we just adjusted release times to fix the problem emphasized previously. Unfortunately, the policy proposed in [18] does not assign task priorities based on their deadlines, so adjusting task deadlines to encode precedences, as we did for DM, will not work with this policy.

2) Relative Priorities of Tasks Related by Precedence Constraints: Let $\Phi : \mathcal{S} \rightarrow \mathbb{N}$ denote an injective priority assignment mapping priorities to tasks. $\Phi(\tau_i)$ denotes the priority of τ_i in assignment Φ and 1 denotes the highest priority in the task set. Let $\tau_i \overset{*}{\rightarrow} \tau_j$ denote a transitive precedence from τ_i to τ_j (i.e. the relation $\overset{*}{\rightarrow}$ is the transitive closure of the relation $\rightarrow$). We make the following observation:

Lemma 2: For any task set S, for any tasks τ_i and τ_j in S such that $O_i = O_j$, $T_i = T_j$ and $\tau_i \overset{*}{\rightarrow} \tau_j$:

A priority assignment Φ respects the precedence $\tau_i \overset{*}{\rightarrow} \tau_j$ if and only if $\Phi(\tau_i) < \Phi(\tau_j)$.

Proof: For the precedence to be respected, we must prove that $k_j \geq f_i$.

(If part) Suppose $\Phi(\tau_i) < \Phi(\tau_j)$. As $O_i = O_j$, then obviously $k_j \geq f_i$.

(Only if part) Suppose $\Phi(\tau_i) > \Phi(\tau_j)$ (Φ is injective so we do not consider the case $\Phi(\tau_i) = \Phi(\tau_j)$). As $O_i = O_j$, then $k_j \leq f_i$ and thus the assignment does not respect the precedence constraint. So the assignment respects the precedence constraint only if $\Phi(\tau_i) < \Phi(\tau_j)$. ■

3) Ordered Task Priorities Assignment: The key principle of the scheduling policy proposed in [18] is that task priorities can be assigned to tasks in order, starting from the lowest priority and up to the highest priority. This can directly be transposed to our scheduling problem.

Definition 3: Let $\mathcal{S}$ be a task set and let $\tau_i \in \mathcal{S}$. We say that τ_i is feasible under priority k if and only if there exists a feasible priority assignment Φ for $\mathcal{S}$ such that $\Phi(\tau_i) = k$.

Let Φ_i denote a *partial* priority assignment, where only priorities $[i, n]$, have been mapped to tasks in $\mathcal{S}$. We have:

Theorem 2: Let $\mathcal{S}$ be a task set. Let Φ_i be a partial priority assignment for $\mathcal{S}$, such that tasks assigned priorities $[i, n]$, are feasible under those priorities. If there exists a feasible priority assignment for $\mathcal{S}$, then there exists a feasible priority assignment that assigns priorities $[i, n]$, as Φ_i.

Proof: The proof provided in [18] does not depend on the way the feasibility test is actually performed. Thus modifying this test to support precedences does not change the proof of the theorem and the theorem holds in our context.

To summarize, we prove that any feasible priority assignment Φ can be modified so that priorities $[i, n]$ are assigned as in Φ_i. This is proved by induction. We prove that we can successively move, in assignment Φ, tasks $\Phi_i^{-1}(n)$, $\Phi_i^{-1}(n-1)$, ..., $\Phi_i^{-1}(i)$, to priorities n, $n-1$, ..., i, and that at each step the assignment remains feasible. ■

4) The policy: We adapt the policy of [18] based on the previous observations to support precedence constraints. The policy is described in Alg. 1. The first difference with the algorithm of [18] is that we start by adjusting release times and deadlines according to formulas (2) and (3) (see Lemma 1). Priorities are then assigned starting from the lowest priority and up to the highest priority. At each step, the algorithm tries to find a task that is feasible if we assign it this priority. The second difference with the algorithm of [18] is that we consider that the priority lvl can be assigned to a task only if priorities lower than lvl have previously been assigned to all its successors (see Lemma 2). If this test succeeds, we then test whether the task will respect its deadline when affected the priority lvl. If so, the task is assigned priority lvl, otherwise we look for another task that can be assigned this priority. If no task can be assigned this priority, then the task set is not feasible.

Algorithm 1 Scheduling policy for a task set $\mathcal{S}$ with simple precedences and arbitrary release times

```
1:  $\mathcal{S}^* \leftarrow adjust(\mathcal{S})$
2:  for $lvl = |\mathcal{S}|$ to 1 do
3:      $assigned \leftarrow false$
4:      for $\tau_i \in \mathcal{S}^*$ do
5:          if $\forall \tau_j \in succs(\tau_i),\ \Phi(\tau_j) > lvl$ then
6:              if $respects\_deadline(\tau_i, lvl)$ then
7:                  $\Phi(\tau_i) \leftarrow lvl$
8:                  $\mathcal{S}^* \leftarrow \mathcal{S}^* - \tau_i$; $assigned \leftarrow true$
9:                  break the current loop
10:             end if
11:         end if
12:     end for
13:     if assigned=false then the system is not feasible
14:     end if
15: end for
```

B. Feasibility Analysis

This scheduling policy directly includes a feasibility analysis, performed by the successive steps of the main loop, as the algorithm assigns a priority to a task only if this task is feasible with this priority. Thus, if the algorithm succeeds, it provides a priority assignment for the task set and it also ensures that the task set is feasible. Otherwise, the task set is not feasible. Testing whether a task is feasible with a given priority is done in two steps: first, verifying that precedence constraints are met and second, verifying that real-time constraints (periodicity, deadline) are met.

According to Lemma 2, a precedence constraint is met if and only if the preceding task has a higher priority than the preceded task. Thus, if a task is assigned priority lvl in assignment Φ, it will meet its precedence constraints if and only if $\forall \tau_j \in succs(\tau_i),\ \Phi(\tau_j) > lvl$. This is checked in our algorithm by the test: **if** $\forall \tau_j \in succs(\tau_i),\ \Phi(\tau_j) > lvl$.

Then, to check that the real-time constraints of a task are met when it is assigned a given priority, we can simply reuse the feasibility test proposed in [18].

C. Optimality and complexity

The optimality of the policy directly derives from the optimality of the policy of [18] and from Lemma 1.

Our policy requires to compute the transitive closure of the precedence relation, which can be achieved with a complexity of $\mathcal{O}(n^3)$ (where n denotes the number of tasks). This does not increase the complexity of the policy of [18], which is exponential.

D. Example

We now refine the specification V_1 of the FAS by specifying task release times. The external inputs of the FAS arrive on a MIL-1553 bus. This bus is time triggered and data arrives at some predefined time steps. This means that acquisition tasks Gyro Acq, GPS Acq and Str Acq do not start simultaneously. If we implement the FAS with the specification V_1 and taking release dates into account with this second scheduling policy, we obtain the following system which is feasible. This is again verified using CHEDDAR.

Name	T	C	D	D*	O	O*	P
PDE	100	5	100	100	0	0	3
SGS	1000	20	1000	990	0	10	7
PWS	1000	20	1000	990	0	10	8
FDIR	100	10	100	100	0	0	2
GNC_US	1000	20	300	290	0	10	5
GNC_DS	1000	20	1000	990	0	10	6
TM/TC	10000	200	10000	10000	30	30	10
Gyro Acq	100	15	100	100	0	0	1
GPS Acq	1000	10	1000	1000	10	10	4
Str Acq	10000	100	10000	10000	20	20	9

As GPS Acq precedes GNC_US, we must adjust the real time attributes of GNC_US as follows: $O^{\star}_{GNC\_US} = O_{GPSAcq} = 10$ and $D^{\star}_{GNC\_US} = D_{GNC\_US} + O_{GNC\_US} - O^{\star}_{GNC\_US} = 300 + 0 - 10 = 290$. Then we have: $O^{\star}_{GNC\_DS} = O^{\star}_{GNC\_US} = 10$ and $D^{\star}_{GNC\_DS} = D_{GNC\_DS} + O_{GNC\_DS} - O^{\star}_{GNC\_DS} = 1000 + 0 - 10 = 990$. We proceed similarly for the other precedences of the task set and we adjust the release times of the task set following a topological sort, starting from the tasks without predecessors.

Once this adjustment is complete, we can assign task priorities. There are 10 tasks in the task set, so we first try to find a task which is feasible when assigned priority 10. This priority can only be assigned to tasks without successors, as a task can be assigned priority 10 only if its successors have already been assigned a priority higher than 10. So there are only five candidates: PDE, SGS, PWS, TM/TC and Gyro Acq (the outgoing dependencies of TM/TC and Gyro Acq are all delayed). We try to assign priority 10 to each one. According to the feasability test of [18], task TM/TC is feasible when assigned this priority, thus it is assigned priority 10. Then we try to assign priority 9. It can only be assigned to tasks the successors of which have already been assigned a priority, thus it can only be assigned to either PDE, SGS, PWS or Gyro Acq. Task Gyro Acq is feasible with this priority and is assigned priority 9. Likewise, we assign priority 8 to PWS and priority 7 to SGS. Then, for priority 6, the only candidate is GNC_DS, as it is the only remaining task the successors of which have been assigned a priority. It is feasible with this priority and is assigned priority 6. Then, GNC_US becomes the only candidate for priority 5. We proceed similarly until every task of the task set has been assigned a priority. If at some point no task among the "unassigned" tasks can be assigned the current priority, then the task set is not feasible (which is not the case in this example).

IV. EXTENDED PRECEDENCES

We now consider the problem of scheduling a set of periodic tasks with arbitrary release times, constrained deadlines and periodic extended precedences.

A. Multi-rate Communications

We now consider communications between operations with different periods. Describing such communications precisely requires to detail which instances of the two tasks communicate. For instance, let us consider the communication loop between FDIR and GNC_US. Since FDIR is 10 times faster than GNC_US, there are several possible communication scenarios:

- Each instance of GNC_US executes before 10 successive instances of FDIR and the 10 instances all consume data produced by the same instance of GNC_US. Each instance of GNC_US consumes data produced by the last previous value produced by FDIR (delayed communication);
- Each instance of GNC_US consumes data produced by the first out of 10 successive instances of FDIR (i.e. executes after this first instance of FDIR). All the 10 instances of FDIR consume the previous values produced by GNC_US (delayed communication);
- The first instance of GNC_US consumes data produced by the second instance of FDIR, the second instance consumes data produced by the twelfth instance of FDIR, ...

There is actually an infinite number of possible deterministic communication schemes between these tasks. Since the tasks are periodic, we only consider *periodic extended precedences*, that is to say precedences between operations of different rates that can be described as repetitive patterns. This is defined more formally in the next section.

B. Definition

An extended precedence between two tasks τ_i and τ_j corresponds to a set of precedences between the instances of the tasks. Let $\tau_i[n] \rightarrow \tau_j[n']$ denote a precedence from the instance n of τ_i to the instance n' of τ_j.

Definition 4 (Extended Precedence): Let τ_i, τ_j be two tasks, let $M_{i,j} \subseteq \mathbb{N}^2$, we define the extended precedence $\tau_i \overset{M_{i,j}}{\rightarrow} \tau_j$ as the following set of task instance precedences:

$$\forall(n,n') \in M_{i,j}, \tau_i[n] \rightarrow \tau_i[n']$$

A simple precedence $\tau_i \to \tau_j$ is actually a particular case of extended precedences where $M_{i,j} = \{(n,n) | n \in \mathbb{N}\}$ and $T_i = T_j$. Indeed, we have: $\forall n \in \mathbb{N}, \tau_i[n] \to \tau_j[n]$.

This general definition is not very practical as task precedences are represented as infinite sets of task instance precedences. Therefore, we only consider task precedences which can be represented as repetitive patterns of task instance precedences, in other words we want the set $M_{i,j}$ to be finite (and to represent repetitive constraints). For any $n \in \mathbb{N}$, let $\mathcal{I}_n$ denote the set of integers of the interval $[0, n[$. Let $lcm(n, n')$ denote the least common multiple of n and n'.

Definition 5 (Periodic Extended Precedence): Let τ_i and τ_j be two tasks, $p = lcm(T_i, T_j)$ and $M_{i,j} \subseteq \mathcal{I}_{p/T_i} \times \mathcal{I}_{p/T_j}$ (notice that $M_{i,j}$ is a finite set). The periodic extended precedence $\tau_i \overset{M_{i,j}}{\to} \tau_j$ is defined as the extended precedence $\tau_i \overset{M'_{i,j}}{\to} \tau_j$ such that:

$$M'_{i,j} = \{(n,n') | \begin{array}{l} \exists k \in \mathbb{N}, (m,m') \in M_{i,j}, \\ (n,n') = (m,m') + (k\frac{p}{T_i}, k\frac{p}{T_j}) \end{array} \}$$

A simple precedence $\tau_i \to \tau_j$ is actually a particular case of periodic extended precedences where $M_{i,j} = \{(0,0)\}$. This definition is illustrated in Fig. 4, where we give a possible schedule respecting different periodic extended precedences. For instance, in Fig.4(a) we have $\tau_i[0] \to \tau_j[0]$, $\tau_i[1] \to \tau_j[3]$, $\tau_i[2] \to \tau_j[6]$, ... In Fig.4(d), we have $\tau_i[2] \to \tau_j[0]$, $\tau_i[5] \to \tau_j[1]$, $\tau_i[8] \to \tau_j[2]$, ... In Fig.4(e), we have $\tau_i[0] \to \tau_j[0]$, $\tau_i[2] \to \tau_j[1]$, $\tau_i[3] \to \tau_j[2]$, $\tau_i[5] \to \tau_j[3]$, $\tau_i[7] \to \tau_j[4]$, $\tau_i[8] \to \tau_j[5]$, ...

Definition 6 (Periodic Extended Precedence Relation): For a set of tasks $\mathcal{S} = \{\tau_i(T_i, C_i, O_i, D_i)\}$, we define the periodic extended precedence relation as a finite set $M = \{M_{i_1,j_1}, \dots, M_{i_l,j_l}\}$ of subsets $M_{i_k,j_k} \subseteq \mathcal{I}_{lcm(T_{i_k},T_{j_k})/T_{i_k}} \times \mathcal{I}_{lcm(T_{i_k},T_{j_k})/T_{j_k}}$, where each M_{i_k,j_k} defines the periodic extended precedence $\tau_{i_k} \overset{M_{i_k,j_k}}{\to} \tau_{j_k}$ ($\forall k = 1, \dots, l$).

Though periodic extended precedences are only a special case of extended precedences, they seem to cover a large class of applications. Furthermore, this definition can easily be extended to patterns longer than the tasks hyperperiod and the results of the following sections still hold.

We are now able to give a specification V_2 of the FAS with extended precedences. For the tasks with same period, we keep the specification V_1. Implicitly, the communications for which we do not specify precedences are delayed communications. We have the following extended precedences:

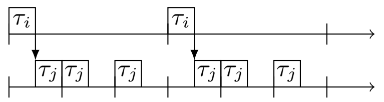
Tasks	$M_{i,j}$
FDIR $\to$ TM/TC	$\{(2,0)\}$
FDIR $\to$ GNC US	$\{(0,0)\}$
GNC_DS $\to$ PDE	$\{(0,9)\}$

C. On Semaphore Synchronizations

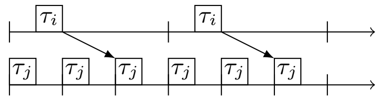
Before presenting our scheduling policy, we emphasize that handling extended precedences through explicit synchronization mechanisms requires more complex primitives than simple binary semaphores. As an example, we consider the system of Fig. 4(a). Synchronizations can be implemented in two different ways:

- Using binary semaphores: each instance of τ_i *signals* the semaphore release, but only the first out of three successive instances of τ_j *waits* for the semaphore release.
- Using $n-$ary semaphores: each instance of τ_i *signals* 3 semaphore releases and each instance of τ_j *waits* for a semaphore release.

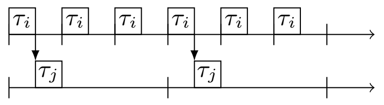
While such modifications allow to execute tasks correctly (i.e. tasks respect precedence constraints), they do not solve the schedulability test problem.



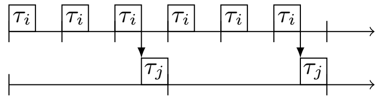
(a) $T_i = 3T_j$, $M_{i,j} = \{(0,0)\}$



(b) $T_i = 3T_j$, $M_{i,j} = \{(0,2)\}$



(c) $T_i = T_j/3$, $M_{i,j} = \{(0,0)\}$



(d) $T_i = T_j/3$, $M_{i,j} = \{(2,0)\}$

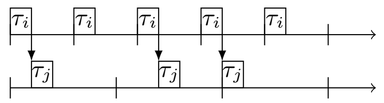

τ_i τ_i τ_i τ_i τ_i
τ_j τ_j τ_j
(e) $T_i = T_j * \frac{3}{5}$, $M_{i,j} =$ $\{(0,0),(2,1),(3,2)\}$

Figure 4. Periodic extended precedence $\tau_i \overset{M_{i,j}}{\to} \tau_j$

D. Scheduling Policy

In this section we extend the results of Sect. III to periodic extended precedences. As for simple precedences, we need to adjust the release times of tasks related by precedence constraints but this time we need to consider in detail which instances of the tasks are related: for each precedence $\tau_i[n] \to \tau_j[n']$ we must adjust release times such that $o_j[n'] \geq o_i[n]$. However, as we consider static priority policies, we cannot adjust the release times of different instances of the same task separately, we can only adjust the initial release times of the tasks. Therefore, we must adjust the initial release time of the tasks in a way that respects the previous property for all task instances.

Property 1: Let τ_i, τ_j be two tasks such that $\tau_i \overset{M_{i,j}}{\rightarrow} \tau_j$. Let $O_j^* = \max(O_j, \max_{(n,n')\in M_{i,j}}(O_j+(O_i^*+nT_i)-(O_j+n'T_j)))$. Let $o_j^*[n] = O_j^* + nT_j$, that is to say, $o_j^*[n]$ is the adjusted release time of $\tau_j[n]$. Then:

$$\forall(n,n') \in M'_{i,j},\ o_i^*[n] \le o_j^*[n']$$

Proof: First we have by definition $M'_{i,j} = \{(n,n')|\exists k \in \mathbb{N}, (m,m') \in M_{i,j}, (n,n') = (m,m') + (k\frac{p}{T_i}, k\frac{p}{T_j})\}$.

If $O_j^* = O_j$, it means that for all $(m,m') \in M_{i,j}$, $o_i^*[m] \le o_j^*[m']$. Thus, for all $(n,n') \in M'_{i,j}$, with $n = m + k\frac{p}{T_i}$ and $n' = m' + k\frac{p}{T_j}$, we have: $o_j^*[n'] = O_j + n'T_j = O_j + m'T_j + kp \ge O_i^* + nT_i + kp = o_i^*[n]$.

Otherwise, $O_j^* = \max_{(n,n')\in M_{i,j}}(O_j+(O_i^*+nT_i)-(O_j+n'T_j))$. We have $\forall n'$,

$$\begin{aligned} o_j^*[n'] &= O_j^* + n'T_j \\ &= O_j + \max_{(l,l')\in M_{i,j}} \\ &\quad ((O_i^* + l'T_i) - (O_j + lT_j)) + n'T_j \end{aligned}$$

Then, let $n = m+k\frac{p}{T_i}$ and $n' = m'+k\frac{p}{T_j}$ with $(m,m') \in M_{i,j}$, we have:

$$\begin{aligned} o_j^*[n'] &\ge O_j + (O_i^* + mT_j) - (O_j + m'T_j) + n'T_j \\ &\ge O_i^* + mT_i + k\frac{p}{T_j}T_j \\ &\ge O_i^* + mT_i + kp = O_i^* + (m' + k\frac{p}{T_i})T_i \\ &\ge o_i^*[n] \end{aligned}$$

■

This property enables us to compute the adjusted release time of a task for a single precedence. Taking all the precedences into account, we have:

$$O_i^* = O_i + max(0, \max_{M_{i,j}}(\max_{(n,n')\in M_{i,j}}((O_j^*+nT_j)-(O_i+n'T_i))))$$

Since we manipulate deadlines relative to the release times, we have to modify deadlines as follows:

$$D_i^* = D_i + O_i - O_i^* \qquad (4)$$

Then, as a straightforward generalization of Lemma 1, we have:

Lemma 3: Let $\mathcal{S} = \{\tau_i(T_i, C_i, O_i, D_i)\}$ and $M = \{M_{i_1,j_1}, \ldots, M_{i_l,j_l}\}$ be a set of periodic extended precedences. Let $\mathcal{S}^* = \{\tau'_i(T_i, C_i, O_i^*, D_i^*)\}$ be a set of tasks such that O_i^* and D_i^* are modified as explained above and let $\rightarrow = \{(\tau_{i_k}, \tau_{j_k})|k = 1, \ldots, l\}$, we have:

$\mathcal{S}$ is feasible if and only if $\mathcal{S}^*$ is feasible.

As a result, the scheduling policy and feasibility test proposed for simple precedences in Sect. III-A4 can also be used for periodic extended precedences, using the formula for release times adjustment provided above. The optimality of the method also holds for periodic extended precedences.

E. Example

Using this scheduling policy, we obtain the following system which is feasible. This is again verified using CHEDDAR.

Name	T	C	D	D⋆	O	O⋆	P
PDE	100	5	100	100	0	0	6
SGS	1000	20	1000	990	0	10	7
PWS	1000	20	1000	990	0	10	8
FDIR	100	10	100	100	0	0	2
GNC_US	1000	20	300	290	0	10	4
GNC_DS	1000	20	1000	990	0	10	5
TM/TC	10000	200	10000	9860	30	170	10
Gyro Acq	100	15	100	100	0	0	1
GPS Acq	1000	10	1000	1000	10	10	3
Str Acq	10000	100	10000	10000	20	20	9

We have $FDIR \overset{\{(2,0)\}}{\rightarrow} TM/TC$ so we must adjust the release time of TM/TC as follows:

$$\begin{aligned} O^\star_{TM/TC} &= \max(O_{TM/TC}, \max_{(n,n')\in\{(2,0)\}}(O_{TM/TC}+ \\ &\quad (O^\star_{FDIR} + nT_{FDIR}) - (O_{TM/TC} + n'T_{TM/TC}))) \\ &= \max(30, (0 + (0 + 2*100) - (30 + 0*10000))) \\ &= 170 \end{aligned}$$

And then: $D^\star_{TM/TC} = D_{TM/TC} + O_{TM/TC} - O^*_{TM/TC} = 10000 + 30 - 170 = 9860$.

The adjustment due to the other extended precedences has no impact on the real-time attributes of the task set. The priority assignment then works exactly as described in the example of Sect. III-D.

V. Conclusion

We studied static priority based scheduling policies without synchronization mechanisms for dependent periodic tasks. We chose policies without synchronization mechanisms because they are well suited for critical systems. We gave scheduling policies and sufficient and necessary schedulability tests for different classes of systems. In the future, we will study dynamic priority policies for extended precedences and the optimality of the encoding approach compared to the use of synchronization mechanisms for dynamic priorities. We will also propose a deterministic communication protocol which will extend existing ones.

References

[1] OSEK, *OSEX/VDX Operating System Specification 2.2.1*, OSEK Group, 2003, www.osek-vdx.org.

[2] RTEMS, *RTEMS C user's guide, Edition 4.9.2, for RTEMS 4.9.2*, OAR Corporation, 2009.

[3] ARINC, *ARINC Specification 653: Avionics Application Software Standard Interface*, Aeronautical Radio INC, 2005.

[4] VxWorks, *VxWorks 653 - DO-178B Certified ARINC 653 Real-Time Operating System*, Wind River, 2006.

[5] C. Sofronis, S. Tripakis, and P. Caspi, "A memory-optimal buffering protocol for preservation of synchronous semantics under preemptive scheduling," in *Sixth International Conference on Embedded Software (EMSOFT'06)*, Seoul, South Korea, Oct. 2006.

[6] F. Boniol, M. Cordovilla, J. Forget, and C. Pagetti, "Implantation multitâche de programmes synchrones multipériodiques," in *7iéme colloque francophone sur la Modelisation des Systemes Réactifs (MSR'09)*, Nov. 2009.

[7] J. Forget, "A synchronous language for critical embedded systems with multiple real-time constraints," Ph.D. dissertation, Université de Toulouse - ISAE/ONERA, Toulouse, France, Nov. 2009.

[8] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM*, vol. 20, no. 1, 1973.

[9] M. G. Harbour, M. H. Klein, and J. P. Lehoczky, "Fixed priority scheduling of periodic tasks with varying execution priority," in *Real-Time Systems Symposium (RTSS'91)*, Dec. 1991.

[10] M. Richard, P. Richard, E. Grolleau, and F. Cottet, "Contraintes de précédences et ordonnancement mono-processeur," in *Real-time and embedded systems (RTS'02)*, 2002.

[11] H. Chetto, M. Silly, and T. Bouchentouf, "Dynamic scheduling of real-time tasks under precedence constraints," *Real-Time Systems*, vol. 2, 1990.

[12] P. Richard, F. Cottet, and C. Kaiser, "Validation temporelle d'un logiciel temps réel : application un laminoir industriel," *Journal Européen des Systèmes Automatisés*, vol. 35, no. 9, 2001.

[13] R. L. Graham, "Bounds on multiprocessing timing anomalies," *SIAM Journal on Applied Mathematics*, vol. 17, no. 2, Mar. 1969.

[14] *Software Considerations in Airborne systems and Equipment Certification*, RTCA, 1992.

[15] J. Y. T. Leung and J. Whitehead, "On the complexity of fixed-priority scheduling of periodic, real-time tasks," *Performance Evaluation*, vol. 2, no. 4, 1982.

[16] M. Joseph and P. Pandya, "Finding response times in real-time system," *The Computer Journal*, vol. 29(5), pp. 390–395, 1986.

[17] F. Singhoff, J. Legrand, L. Nana, and L. Marcé, "Cheddar: a flexible real time scheduling framework," *Ada Lett.*, vol. XXIV, no. 4, 2004.

[18] N. C. Audsley, "Optimal priority assignment and feasibility of static priority tasks with arbitrary start times," Dept. Computer Science, University of York, Tech. Rep. YCS 164, Dec. 1991.