Dynamic Priority Scheduling of Periodic Tasks with Extended Precedences

Julien Forget, Emmanuel Grolleau, Claire Pagetti, Pascal Richard

Dynamic Priority Scheduling of Periodic Tasks with Extended Precedences

Julien Forget*, Emmanuel Grolleau †, Claire Pagetti‡§, and Pascal Richard §

*LIFL, France
†LISI/ENSMA, France
‡ONERA, France
§LISI, Univ. Poitiers, France

Abstract

The software architecture of a critical embedded control system generally consists of a set of multi-periodic communicating tasks. In order to be able to describe such a system, we define the notion of semaphore precedence constraint*, which supports multi-rate communications that follow regular repetitive patterns. We propose a feasibility test for EDF and we study three implementations, for periodic task sets related by such extended precedences on monoprocessor architectures.*

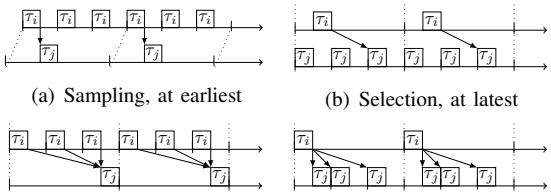
I. Introduction

A. Motivation: critical embedded control systems

This work was 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. Spacecraft and aircraft flight control systems are good examples of embedded control systems.

Such a system consists of a set of communicating tasks, usually with different periods since the devices it controls have different physical characteristics. In addition to classic real-time constraints (periods and deadlines), functional requirements have an important impact on the expected execution order of the tasks. First, a partial ordering is imposed on the tasks due to their functional dependencies, e.g. data acquisition must be performed first, then computations, then commands towards actuators. Second, a correct implementation must be functionally deterministic, meaning that the outputs of the system must always be the same for a given sequence of inputs, which requires task communications to be fully deterministic.

a) Multi-rate communications: In such a system, task of different periods communicate. Some communication examples are given in Figure 1 (notice that communicating tasks may also have different initial release times).



(a) Sampling, at earliest (b) Selection, at latest

(c) Array gathering (d) Array scattering

Fig. 1. Multi-rate communication patterns

The programmer can use the patterns of Figures 1(a), 1(b), when he wants to leave great flexibility for the execution of the slow task: the slow task consumes values produced early, i.e. samples only the first out of 3 successive jobs of the producer, and produces values late, i.e. communicates with the last out of 3 successive jobs of the consumer. The patterns of Figure 1(c) and Figure 1(d) correspond to classic signal processing, where repetitive array computations are distributed between several repetitions of the same task: on one hand the slow task scatters the content of a big array between successive jobs of its consumer and on the other hand it gathers array fragments from its producer to construct a big array. These two last patterns behave in a fashion similar to the MPI_gather and MPI_scatter primitives of the popular Message Passing Interface API [1]. This list of examples is not exhaustive and we wish to provide support for a large choice of patterns, instead of imposing a fixed set of patterns.

b) Extended precedences.: The functional determinism constraint implies that, during the execution of the system, the correct job of the producer must communicate with the correct job of the consumer. This requires, for each pair of producer job and consumer job, to ensure that: (1) the producer completes before the consumer starts (2) data produced remains available until the completion of the consumer. As far as scheduling is concerned, requirement (1) is modeled by adding a precedence constraint from the producer to the consumer. When the two tasks have the same period, we can simply impose that each job of the producer executes before one job of the consumer. This corresponds to usual *simple precedences*. Multi-rate communication patterns such as

those of Figure 1 correspond to more complex *extended precedences*, which only relate a subset of the jobs of the communicating tasks. Requirement (2) can be fulfilled using a specific communication protocol (for instance [2], [3]). In this paper we focus on respecting requirement (1).

B. Notations

The software architecture of an embedded control system can be defined as a set of strictly periodic (time driven) tasks $\{\tau_i\}_{1\leq i\leq n}$. Each task is not reentrant and has a set of real-time attributes (O_i, C_i, D_i, T_i). O_i is the first release date of the task, also called offset in the literature. T_i is the (strict) period of the task and defines the exact duration between two successive releases of the task. We denote $\tau_{i.k}$ the k^{th} instance of τ_i (starting with instance 0), which we will call a *job* (or task job). The job $\tau_{i.k}$ is released at date $o_{i.k} = O_i + kT_i$. D_i is the relative deadline of the task, every job $\tau_{i.k}$ must be completed before its absolute deadline $d_{i.k} = o_{i.k} + D_i$. Finally, C_i is the worst case execution time (WCET) of the task and represents the longest processor time used by a job of τ_i. These definitions are illustrated in Figure 2. Additionally, we define the *hyperperiod* of a task set as the least common multiple (*lcm*) of the tasks periods.

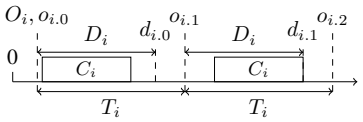


Fig. 2. Real-time attributes of a task τ_i

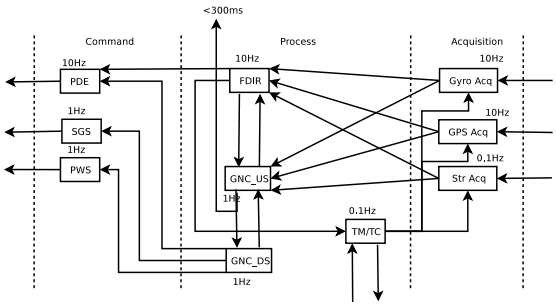
Let $\mathcal{J}$ denote the infinite set of jobs $\mathcal{J} = \{\tau_{i.k}, 1 \leq i \leq n, k \in \mathbb{N}\}$. Given a schedule, we define two functions $s, e : \mathcal{J} \to \mathbb{N}$ where $s(\tau_{i.k})$ is the start time and $e(\tau_{i.k})$ is the completion time of $\tau_{i.k}$ in the considered schedule. We say that a dependent task set is *schedulable* under a given scheduling policy if the schedule produced by this policy respects all the constraints of the task set and all its job precedence constraints. More formally:

Definition 1. *Let $\mathcal{S} = \{\tau_i\}_{1\leq i\leq n}$ be a dependent task set. $\mathcal{S}$ is* schedulable *under a given scheduling policy if and only if,* $\begin{cases} \forall \tau_{i.k},\ e(\tau_{i.k}) \leq d_{i.k} \wedge s(\tau_{i.k}) \geq o_{i.k} \\ \forall \tau_{i.k} \to \tau_{j.k'},\ e(\tau_{i.k}) \leq s(\tau_{j.k'}) \end{cases}$

C. Case study

To illustrate 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). Due to its high criticality level, such a system must follow a certified development, like the DO-178B [4] for aeronautics. This process imposes very precise and time-consuming constraints and milestones in the different development steps. This entails that the simplest and most mature solutions are preferred to ease the verification of the system and the discussion with the certification authorities. In particular, the operating system must also be certified. Thus our results are based on extensions of existing results.

The Figure 3(a) provides a simplified informal description of the FAS software architecture. It acquires several data treated by dedicated sub-functions: orientation and speed (Gyro Acq), position (GPS Acq and Str Acq) and telecommands from the ground station (TM/TC). The *Guidance Navigation and Control* function (divided into GNC_US and GNC_DS) 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. Commands are sent to the control devices: thruster orders (PDE), power distribution orders (PWS), solar panel positioning orders (SGS) and telemetry towards the ground station (TM/TC).



(a) The Flight Application Software

task	offset	deadline	wcet	period
Gyro Acq	10	100	10	100
GPS Acq	0	100	10	100
FDIR	0	100	20	100
PDE	0	100	10	100
GNC_US	50	300	50	1000
GNC_DS	0	1000	100	1000
PWS	0	1000	20	1000
SGS	0	1000	20	1000
Str Acq	1000	10 000	200	10 000
TM/TC	500	10 000	500	10 000

Gyro Acq $\xrightarrow{700}$ GNC_US
GPS Acq $\xrightarrow{800}$ GNC_US
GNC_US $\xrightarrow{200}$ FDIR
GPS Acq $\xrightarrow{0}$ FDIR
GNC_US $\xrightarrow{0}$ GNC_DS
FDIR $\xrightarrow{8000}$ TM/TC

(b) Task set description

Fig. 3. FAS architecture

The task set corresponding to this system is described Figure 3(b). The communication patterns are expressed with the formalism of SPC introduced later on: both the gather/scatter patterns and sampling/selection patterns are used, depending on the kind of computations performed by the communicating tasks. Because of design requirements, the operations are triggered at different offsets.

D. Related works

In the real-time scheduling theory, handling simple precedence constraints (i.e. every pair of dependent task is executed at the same rate) is a well-understood problem [5]. When communicating tasks do not share the same rate, several precedence models have been studied in the literature, which enable to represent different subclasses of extended precedence constraints:

- Linear Precedence Constraints (LPC) are studied in the context of cyclic jobs scheduling in [6]. The

problem is, given a task precedence graph (which can contain cycles), given a WCET for each task and assuming that each job is executed on its own processor, to find the task rates in the steady state of the system, such that the jobs are executed as often as possible (the objective is thus quite different from ours). The precedences of the task graph are characterized by 4 natural numbers q, k, q', k' such that: $\forall n \in \mathbb{N}, \tau_{i.qn+k} \rightarrow \tau_{j.q'n+k'}$

- Generalized Precedence Constraints (GPC) presented in [7] are a particular case of Linear Precedence Constraints. The objective is different, the authors focus on the scheduling of periodic tasks, related by generalized precedence constraints given in an acyclic graph. The Generalized Precedence Constraint $\tau_i \rightarrow \tau_j$ requires that the number of started jobs $S_j(t)$ of τ_j at any time instant t and the number of ended jobs $E_i(t)$ of τ_j are such that: $E_i(t) \times T_i \geq S_j(t) \times T_j$. For the schedulability analysis, the authors unfold the precedence graph over the hyperperiod and use a classic scheduling test. This validation method is exponential in space and time. At run-time, synchronization of tasks related by precedence constraints is implemented using semaphores;
- In the model of Repetitive Precedence Constraints (RPC), introduced in [8], a constraint between two tasks τ_i and τ_j is specified by a predicate $Code_{ij}(n, n')$, which is true if and only if $\tau_{i.n} \rightarrow \tau_{j.n'}$ for $n \in [1..lcm(T_i, T_j)/T_i]$ and $n' \in [1..lcm(T_i, T_j)/T_j]$. The representation of the precedence relation thus uses non-polynomial space. No restrictions are imposed on the precedence predicate, which can lead to inconsistent or redundant precedence constraints. The authors propose a non-preemptive scheduling policy for this model. [9] defines a fixed priority preemptive scheduling policy for a precedence model with the same expressiveness.

E. Contributions

We first introduce a subclass of extended precedence constraints called Semaphore Precedence Constraints, or SPC in short (Section II). SPC are a simple extension of GPC, which support most multi-rate communications that follow a repetitive pattern. SPC can be represented in polynomial space (on the contrary to RPC). They are a little less general than RPC and LPC when dealing with communications between tasks with co-prime periods, however such communications are very uncommon in practice (and the objective of LPC is different as [6] is not targeted for periodic task scheduling).

We then study the problem of scheduling periodic tasks related by SPC with dynamic priority preemptive policies. Our results rely on an extension of the technique proposed by CHETTO ET AL. in [10], where precedence constraints are encoded by adjusting task release times and deadlines and the adjusted task set is then scheduled with EDF [11].

- We propose three different ways to ensure the precedence constraints (Section IV): the first relies on counting semaphores (or equivalently mailboxes) of a real-time operating system, the second performs an off-line encoding of precedence constraints and the last performs this encoding on-line. These implementations differ in terms of memory consumption, computation overhead and complexity of the Operating System;
- Since the three implementations are equivalent in term of support of precedence constraints, they rely on the same schedulability analysis (Section III). It first consists in computing the adjusted release time and deadline of each job. The sequence of release times (resp. of deadlines) of a task is represented using ultimately periodic words (a task model close to, yet simpler than the *generalized multiframe* model of [12]). The schedulability of this encoded independent task set can then be tested using existing tests for EDF. Like it is the case for GPC, this analysis is exponential in space and time.

II. Formalizing extended precedence constraints

In this paper, we focus on extended precedence constraints that correspond to repetitive patterns of precedence constraints between jobs, which we call *semaphore precedence constraints*.

A. Simplifying redundant precedence constraints

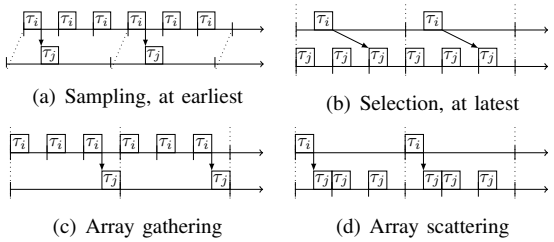
We can notice in Figure 1(c) that if a schedule respects the precedence constraint $\tau_{i.2} \rightarrow \tau_{j.0}$, then it also respects the precedence constraints $\tau_{i.1} \rightarrow \tau_{j.0}$ and $\tau_{i.0} \rightarrow \tau_{j.0}$. This suggests that we should avoid redundant precedence constraints.

Property 1. *For a feasible schedule,* $\forall k_1, k_2, k_3, k_4 \in \mathbb{N}^4$ *with* $k_1 < k_2$ *and* $k_3 < k_4$*, we have:*

1) *Non-reentrancy: the precedence constraints* $\tau_{i.k_1} \rightarrow \tau_{i.k_2}$ *and* $\tau_{j.k_3} \rightarrow \tau_{j.k_4}$ *are fulfilled;*
2) *Transitivity: if the schedule respects the precedence constraint* $\tau_{i.k_2} \rightarrow \tau_{j.k_3}$ *then it also respects* $\tau_{i.k_1} \rightarrow \tau_{j.k_3}$ *and* $\tau_{i.k_1} \rightarrow \tau_{j.k_4}$.

Proof: The first part is a direct effect of the non-reentrancy of the tasks. The second point illustrates the transitivity of the precedence constraints directly following the Def. 1. ■

Figure 4 shows the precedence constraints supporting the communication patterns of Figure 1, after deleting redundant constraints. In Figure 4(a), we have $\tau_{i.3k} \rightarrow \tau_{j.k}$. In Figure 4(b), we have $\tau_{i.k} \rightarrow \tau_{j.3k+2}$. In Figure 4(c), we have $\tau_{i.3k+2} \rightarrow \tau_{j.k}$. In Figure 4(d), we have $\tau_{i.k} \rightarrow \tau_{j.3k}$.



(a) Sampling, at earliest (b) Selection, at latest

(c) Array gathering (d) Array scattering

Fig. 4. Reduced precedence patterns

B. Semaphore precedence constraints

Like in the other precedence models presented in Section I-D, we want to define a task precedence constraint as the repetition of a job precedence pattern. For instance, in Figure 4(c), we have a pattern that consists of three successive instances of τ_i and one of τ_j, where the first instance of τ_i (in the pattern) is related to the only instance of τ_j in the pattern, and this pattern is repeated indefinitely (patterns are depicted by dotted lines in Figure 4). Generalized precedence constraints [7] can represent constraints like those in Figure 4(d) and 4(c) but not the others. A generalized precedence constraint can be represented using a counting semaphore and we propose a simple extension consisting in allowing semaphore counts to be initialized to a value different from 0. Let s be a counting semaphore, the operation $signal(s, n)$ is the usual extension of the binary semaphore operation $signal/V$: it atomically increments the count of semaphore s by value n. The operation $wait(s, n)$ is the usual extension of the binary semaphore operation $wait/P$: if the semaphore count is less than n, then the task executing this operation is blocked until the counter is incremented again, otherwise the counter is atomically decremented by value n.

Definition 2. *A semaphore precedence constraint (SPC) $\tau_i \xrightarrow{h_{i,j}} \tau_j$, with $h_{i,j} \in \mathbb{N}$, is modeled using a semaphore $sem_{i,j}$. It requires τ_i and τ_j to behave as if the following operations were performed:*

1) *Initialization: set the counter of $sem_{i,j}$ to h_{ij};*
2) *When completing job $\tau_{i.k}$: execute $signal(sem_{i,j}, T_i)$;*
3) *When releasing job $\tau_{j.k'}$: execute $wait(sem_{i,j}, T_j)$.*

Let us stress that here semaphores are only used to represent the required behavior of the tasks, it does not necessarily mean that precedence constraints are implemented using semaphores. With this definition, we can represent the extended precedence constraints of Figure 4 simply by giving $h_{i,j}$. To determine $h_{i,j}$, we need to solve a set of inequalities. For instance, the precedence constraint of Figure 4(a) is $\tau_i \xrightarrow{2T_i} \tau_j$. Indeed, we want to express the following constraints:

$$\begin{cases} h_{i,j} < T_j \\ h_{i,j} + T_i \geq T_j \end{cases}$$

Since $T_j = 3T_i$, we obtain the inequalities $2T_i \leq h_{i,j} < 3T_i$. As a consequence, $h_{i,j} = 2T_i$ is a solution. The behavior of the semaphore will be the following: initially the counter of $sem_{i,j}$ has value $2T_i$ and $\tau_{j.0}$ cannot start until the value reaches $T_j = 3T_i$ ($wait(sem_{i,j}, T_j)$). After the completion of $\tau_{i.0}$, the counter reaches $3T_i$ ($signal(sem_{i,j}, T_i)$), $\tau_{j.0}$ can execute and the counter is decreased to 0 (due to the $wait(sem_{i,j}, T_j)$ still pending). The job $\tau_{j.1}$ is then blocked until 3 more jobs of τ_i complete their execution, at which point the counter reaches $T_j = 3T_i$, then $\tau_{j.1}$ executes, the counter is decreased back to 0, and so on. Similarly, Figure 4(b) corresponds to $\tau_i \xrightarrow{2T_j} \tau_j$, Figures 4(c) and 4(d) to $\tau_i \xrightarrow{0} \tau_j$.

Property 2. *Let $\tau_i \xrightarrow{h_{i,j}} \tau_j$ be a semaphore precedence constraint (SPC). We have the following properties:*

1) *A schedule respects the SPC $h_{i,j}$ if and only if:*

$$\forall t \geq 0, h_{i,j} + E_i(t) \times T_i \geq S_j(t) \times T_j$$

where $E_i(t)$ is the number of completed jobs of τ_i at time t and $S_j(t)$ is the number of jobs of τ_j that could start despite the operation $wait$;

2) *The SPC $h_{i,j}$ generates the job precedence $\tau_{i.k} \rightarrow \tau_{j.k'}$ if and only if:*

$$\begin{cases} h_{i,j} + (k+1)T_i - (k'+1)T_j \geq 0 \\ h_{i,j} + kT_i - (k'+1)T_j < 0 \end{cases}$$

3) *The SPC $h_{i,j}$ imposes that a job $\tau_{j,k'}$ cannot be started before the end of any job $\tau_{i,k}$ with $k \leq Pred_{i,j}(k')$ where:*

$$Pred_{i,j}(k') = \left\lceil \frac{(k'+1)T_j - h_{i,j}}{T_i} \right\rceil - 1$$

4) *The SPC $h_{i,j}$ imposes to a job $\tau_{i,k}$ to be executed before the start of any job $\tau_{j,k'}$ with $k \geq Succ_{i,j}(k)$ where:*

$$Succ_{i,j}(k) = \left\lfloor \frac{kT_i + h_{i,j}}{T_j} \right\rfloor$$

Note that the function $Pred_{i,j}(k')$ can have a negative value, in this case $\tau_{j,k'}$ is a free job because of the value of $h_{i,j}$.

Proof: 1. The semaphore count cannot be less than zero, and is given by its initial value plus a count of T_i every time a job of τ_i is completed, minus a count of T_j every time a job of τ_j can execute the $wait$ instruction. Therefore the semaphore count is given, at any time by: $h_{i,j} + E_i(t) \times T_i - S_j(t) \times T_j \geq 0$.

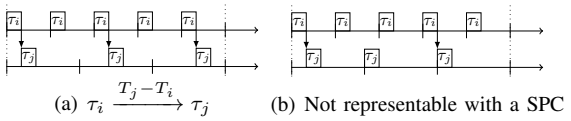
2. directly derived from the SPC definition.

3. For a given natural integer k', $Pred_{i,j}(k')$ is the greatest integer verifying the inequalities given in part 2.

4. For a given natural integer k, $Succ_{i,j}(k)$ is the smallest integer verifying the inequalities in part 2. ■

We can note that a GPC is a particular case of a SPC where the initial count $h_{i,j} = 0$ for any precedence relation.

c) Limitations: We can see in Figure 4 that in the case of harmonic periods, after deleting redundant constraints there remains only one job precedence by hyperperiod. Using a SPC, the value $h_{i,j}$ enables us to choose any single job precedence constraint in the hyperperiod and thus we can support any kind of repetitive communication pattern between tasks with harmonic periods.



(a) $\tau_i \xrightarrow{T_j - T_i} \tau_j$ (b) Not representable with a SPC

Fig. 5. Non-harmonic periods ($T_i = \frac{3}{5}T_j$)

If we consider tasks with non-harmonic periods, we may want to specify several (non-redundant) job precedence constraints inside the hyperperiod. In this case, as shown in Figure 5(b), some precedence patterns (and thus some communication patterns) are not supported by SPC. Indeed, let for instance $T_i = 3$ and $T_j = 5$, the constraint $\tau_{i.0} \rightarrow \tau_{j.0}$ requires that $h_{i,j} < T_j$ thus $h_{i,j}$ is at most equal to 4. As a result $\tau_{j.1}$ is blocked until $\tau_{i.1}$ completes, i.e. there is a precedence constraint $\tau_{i.1} \rightarrow \tau_{j.1}$ (which we do not want). The problem does not occur with Figure 5(a) because there is already a stronger constraint $\tau_{i.2} \rightarrow \tau_{j.1}$ (adding $\tau_{i.1} \rightarrow \tau_{j.1}$ does not change the scheduling problem).

III. Schedulability analysis

In this section we adapt to Semaphore Precedence Constraints the technique introduced in [10], which consists in adjusting task real-time attributes in a way that mimics the precedence constraints. In the case of semaphore precedence constraints, this requires to adjust separately the real-time attributes of different jobs of the same task.

A. Reminder: simple precedence constraints

We first recall the original result of [10] which can be applied to tasks with simple precedence constraints (same period). Precedence constraints can be encoded by adjusting the release date and deadline of every task as follows:

$$O_i^* = \max(O_i, \max_{\tau_j \in preds(\tau_i)}(O_j^*)) \quad (1)$$

$$d_i^* = \min(d_i, \min_{\tau_j \in succs(\tau_i)}(d_j^* - C_j)) \quad (2)$$

For each precedence constraint $\tau_i \rightarrow \tau_j$, Equation 1 ensures that τ_i is released before τ_j, while Equation 2 ensures that τ_i will have a higher priority than τ_j (according to EDF). Thus, τ_i will be scheduled before τ_j.

Property 3. *This can be rewritten in relative dates as:*

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

Proof: Indeed, $d_i^* = O_i^* + D_i^* = \min(D_i + O_i, \min_{\tau_j \in succs(\tau_i)}(D_j^* + O_j^* - C_j))$. Thus $D_i^* = \min(D_i - O_i^* + O_i, \min_{\tau_j \in succs(\tau_i)}(D_j^* + O_j^* - C_j - O_i^*))$. ■

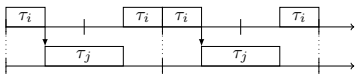
Theorem 1 ([10])**.** *Let $\mathcal{S} = \{\tau_i(O_i, D_i, D_i, T_i)\}$ be a task set with simple precedence constraints. Let $\mathcal{S}^* = \{\tau_i'(O_i^*, C_i, D_i^*, T_i)\}$ be a set of independent tasks such that O_i^* and D_i^* are given by the previous formulas:*

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

As a result a dependent task set with simple precedences can be scheduled optimally by performing this encoding and scheduling the encoded task set with EDF (as EDF is optimal for independent tasks).

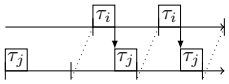
B. Tasks with Semaphore Precedence Constraints

With SPC, we need to set different adjusted attributes for jobs of the same task. Let us consider the tasks set $\mathcal{S}_1 = \{\tau_i, \tau_j\}$, with $(O_i = 0, C_i = 2, D_i = T_i, T_i = 4)$, $(O_j = 0, C_j = 4, D_j = 6, T_j = 8)$, $\tau_i \xrightarrow{T_i} \tau_j$.



We set $D_{i.n}^* = 6$ for $n \in \mathbb{N}$, $D_{i.n}^* = 2$ for $n \in 2\mathbb{N}$ and $D_{i.n}^* = 4$ for $n \in 2\mathbb{N} + 1$. $\mathcal{S}^*$ is then schedulable with EDF. If we set the same adjusted relative deadline for all jobs of τ_i, that is $D_{i.n}^* = 2$ for $n \in \mathbb{N}$, then $\tau_{j.0}$ will miss its deadline (date 6).

The sequence of adjusted release times of a task may also have a non-regular prefix before becoming periodic. Let us consider the task set $\mathcal{S}_1 = \{\tau_i, \tau_j\}$, with $(O_i = 4, C_i = 1, D_i = T_i, T_i = 3)$, $(O_j = 0, C_j = 1, D_j, T_j = 3)$, $\tau_i \xrightarrow{T_i} \tau_j$. There are $\lfloor h_{i,j}/T_j \rfloor$ (in the current case $3/3 = 1$) jobs of τ_j that can execute freely before the precedence constraint starts following a periodic pattern. We have $O_{j.0}^* = O_j = 0$ and for all $k > 0$, $O_{j.k}^* = 1$.



We can see with these examples that the sequence of deadlines (resp. the sequence of release times) of a task obtained after encoding a SPC is *periodic* (resp. *ultimately periodic*). We propose to represent these sequences as *ultimately periodic words*, where each value of the word corresponds to the release time, or the deadline, of a job. The task model, obtained after precedence encoding, is close to the *generalized multiframe* (GMF) model of [12], though there are two differences: in GMF, tasks may be sporadic and several jobs of the same task may have different WCETs.

Let the ultimately periodic word $v(u)^\omega$ denote the infinite sequence consisting of the finite prefix v and the infinite repetition of the finite sequence u. The sequence of offsets and of deadlines τ_i in $\mathcal{S}_1$ can be defined as $(0)^\omega$ and $(2.4)^\omega$, while that of τ_j in $\mathcal{S}_2$ can be defined as $0(1)^\omega$ and $9(8)^\omega$. A periodic word is a special case of

ultimately periodic word with a prefix of length 0. We introduce some additional notations:

- for a finite word v, $|v|$ denotes the length of the word.
- $w[n]$ denotes the n^{th} value of the (ultimately) periodic word w. If $w = v(u)^\omega$ then $w[n] = v[n]$ if $n < |v|$ and $w[n] = u[(n - |v|) \bmod |u|]$ otherwise.
- The minimum of two (ultimately) periodic words $w_m = min(w_i, w_j)$ is such that for all n, we have $w_m[n] = min(w_i[n], w_j[n])$. If $w_i = v_i(u_i)^\omega$ and $w_j = v_j(u_j)^\omega$, we can compute w_m by unfolding w_i into $w'_i = v'_i(u'_i)^\omega$ and w_j into $w'_j = v'_j(u'_j)^\omega$, such that $|v'_i| = |v'_j| = \max(|v_i|, |v_j|)$ and $|u'_i| = |u'_j| = lcm(|u_i|, |u_j|)$. We then simply compute the minimum of the two unfolded words point by point.
- The maximum of two (ultimately) periodic words is defined similarly.

C. Real-time attributes adjustment

Using such words, we can now adapt the encoding technique of [10] to support SPC.

Release times Formula (1) becomes $o^*_{j.n'} = \max(o_{j.n'}, \max_{\tau_{i.n} \in preds(\tau_{j.n'})}(o^*_{i.n}))$ where $preds(\tau_{j.n'}) = \{\tau_{i.n} | \tau_{i.n} \rightarrow \tau_{j.n'}\}$. The *release date word* ow_j is defined as:

$$\forall n' \in \mathbb{N}, ow_j[n'] = o^*_{j.n'} - n'T_j \tag{3}$$

Deadlines Formula (2) becomes $d^*_{i.n} = \min(d_{i.n}, \min_{\tau_{j.n'} \in succs(\tau_{i.n})}(d^*_{j.n'} - C_j))$ where $succs(\tau_{i.n}) = \{\tau_{j.n'} | \tau_{i.n} \rightarrow \tau_{j.n'}\}$. The *deadline word* dw_i is defined as:

$$\forall n \in \mathbb{N}, dw_i[n] = d^*_{i.n} - o^*_{i.n} \tag{4}$$

d) Words computation: Let $ow_{i,j}$ and $dw_{i,j}$ be defined such that $ow_j = \max_{\tau_i \in preds(\tau_j)}(ow_{i,j})$ and $dw_i = min_{\tau_j \in succs(\tau_i)}(dw_{i,j})$, i.e. each of these words encodes a single SPC.

Property 4. *For all* $\tau_i \xrightarrow{h_{i,j}} \tau_j$*:*

$$\forall n' \in \mathbb{N}, ow_{i,j}[n'] = \max(O_j, ow_i[n] + nT_i - n'T_j)$$
$$\forall k \in \mathbb{N}, dw_{i,j}[k] = \min(D_i - (ow_i[k] - O_i), ow_j[k'] + k'T_j + dw_j[k'] - C_j - ow_i[k] - kT_i)$$

where $n = Pred_{i,j}(n')$ *and* $k' = Succ_{i,j}(n)$.

Proof: Rewritings and Property 2. ∎

The complete encoding algorithm works as follows. We first compute the release date word of each task, following a topological sort, which starts with tasks without predecessors and then proceeds with tasks whose predecessors have already been processed. At each step of the sort, we adjust a single SPC using Property 4. Similarly, we then compute the deadline word of each task, following a reverse topological sort and starting with tasks without successors.

e) Size of the periodic words: We need to prove that release date words are indeed ultimately periodic and that deadline words are periodic. We give a bound on their prefix size and on their periods:

Property 5. *For a task set $\mathcal{S}$ with a finite set of SPC, for all task τ_i, the word ow_i is ultimately periodic and dw_i is periodic. Both have a periodic pattern of period a divisor of $lcm(\{T_j\})$, for every τ_j in the connected component of τ_i. The prefix of ow_i has a length equal to* $\max \lfloor h_{i,j}/T_j \rfloor$.

Proof: Let us prove the property by induction on the number of SPC. Let us assume first that there is a unique SPC $\tau_i \xrightarrow{h_{i,j}} \tau_j$. The first job $\tau_{j.j_0}$ with a precedence constraint is the one with $j_0 = \lfloor h_{i,j}/T_j \rfloor$. Thus, $ow_j[l] = O_j$ for all $l < j_0$ and j_0 is the length of the prefix. We must prove that ow_j (after the prefix) and dw_i are periodic of period $p = lcm(T_i, T_j)$. Since there is a unique precedence, $ow_j = ow_{i,j}$ and $dw_i = dw_{i,j}$. We need to prove that for all $n, n' \in \mathbb{N}^2$, $n' \geq j_0$, $ow_j[n'] = ow_j[n' + p/T_j]$ and $dw_i[n] = dw_i[n + p/T_i]$, which is equivalent to:

$$\tau_{i.n} \in preds(\tau_{j.n'}) \Leftrightarrow \tau_{i.n+p/T_i} \in preds(\tau_{j.n'+p/T_j}) \tag{5}$$

$$\tau_{i.n} \notin preds(\tau_j) \Leftrightarrow \tau_{i.n+p/T_i} \notin preds(\tau_j) \tag{6}$$

$$o_{i.n} - o_{j.n'} = o_{i.n+p/T_i} - o_{j.n'+p/T_j} \tag{7}$$

$$d_{j.n'} - d_{i.n} = d_{j.n'+p/T_j} - d_{i.n+p/T_i} \tag{8}$$

For (5), we have $\tau_{i.n} \in preds(\tau_{j.n'})$ is equivalent to $h_{i,j} + (n+1)T_i - (n'+1)T_j \geq 0$ and $h_{i,j} + nT_i - (n'+1)T_j < 0$. Since $h_{i,j} + (n + p/T_i + 1)T_i - (n' + p/T_j + 1)T_j = h_{i,j} + (n+1)T_i - (n'+1)T_j$, therefore $\tau_{i.n+p/T_i} \in preds(\tau_{j.n'+p/T_j})$.

For (6), we have $\tau_{i.n} \notin preds(\tau_j)$ is equivalent to $\nexists n' \geq j_0$, $\tau_{i.n} \in preds(\tau_{j.n'})$. By absurd, let us assume that $\exists n' \geq j_0$, $\tau_{i.n+p/T_i} \in preds(\tau_{j.n'})$. Thus, we have $h_{i,j} + (n + 1 + p/T_i)T_i - (n'+1)T_j \geq 0$ and $h_{i,j} + (n + p/T_i)T_i - (n'+1)T_j < 0$. Which is equivalent to $h_{i,j} + (n+1)T_i - (n' + 1 - p/T_j)T_j \geq 0$ and $h_{i,j} + nT_i - (n' + 1 - p/T_j)T_j < 0$. Which is equivalent to $\tau_{i.n} \in preds(\tau_{j.n'-p/T_j})$.

For (7), we have $o_{i.n+p/T_i} - o_{j.n'+p/T_j} = o_{i.n} + (p/T_i)T_i - o_{j.n'} + (p/T_j)T_j = o_{i.n} - o_{j.n'}$.

For (8), we have $d_{j.n'+p/T_j} - d_{i.n+p/T_i} = d_{j.n'} + (p/T_j)T_j - d_{i.n} + (p/T_i)T_i = d_{j.n'} - d_{i.n}$.

This concludes the case for a unique SPC.

Now, let us assume that there are l SPC in the task set and that the release date words and deadline words are all ultimately periodic. We unfold all the words on the maximal prefix and unfold the pattern on the length *HP* to simplify the computations. Let us add a new precedence constraint $\tau_i \xrightarrow{h_{i,j}} \tau_j$ (induction step, we now have $l + 1$ precedence constraints). ow_j and ow_i are respectively ultimately periodic and periodic. As a consequence so is $ow_{i,j}$ and therefore $ow_j = \max(ow_j, ow_{i,j})$ is also ultimately periodic, with a prefix of length the maximal length of the prefixes and with a period dividing *HP*. In the same way, $dw_{i,j}$ is periodic, with a period dividing *HP* and so is dw_i.

∎

D. Analysis algorithm

We can now generalize the main theorem given in [10]:

Theorem 2. *Let $\mathcal{S} = \{\tau_i(O_i, C_i, D_i, T_i)\}$ and $\mathcal{P} = \{h_{i_1,j_1}, \ldots, h_{i_l,j_l}\}$ denote a set of semaphore precedence constraints. Let $\mathcal{S}^* = \{\tau_i^*(ow_i, C_i, dw_i, T_i)\}$ be a set of independent tasks such that ow_i and dw_i are given by formulas 3 and 4. Then:*

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

Proof: Directly deduced from the construction of formulas 3 and 4 and from [10]. ■

The schedulability analysis is made as follows: once the words have been computed, precedences are encoded in task real-time attributes and the adjusted task set can be scheduled as an independent task set. This adjusted task set can thus be scheduled with EDF. As EDF is optimal for independent periodic tasks with deadline constraints (a task set such as that we obtain after precedence encoding), we can simply reuse an existing schedulability test for EDF [13], [14], [15] and apply it on the encoded task set. The only difference is that we compute job release times and deadlines based on release date words and deadline words.

E. Example

We encode the FAS simplified version depicted in Introduction and which task set is described in Figure 3(b). Let us consider release times adjustment. The operation GNC_US has two predecessors: for Gyro Acq$\xrightarrow{700}$GNC_US we have $ow_{Gyro\_Acq,GNC\_US}[n] = \max\{O_{GNS\_US}, O_{Gyro\_Acq} + 2T_{Gyro\_Acq}\} = \max\{50, 10 + 200\} = 210$. For GPS Acq$\xrightarrow{800}$GNC_US, we have $ow_{GPS\_Acq,GNC\_US}[n] = \max\{O_{GNS\_US}, O_{GPS\_Acq} + T_{GPS\_Acq}\} = \max\{50, 100\} = 100$. Thus $ow_{GNC\_US}[n] = \max\{ow_{Gyro\_Acq,GNC\_US}[n], ow_{GPS\_Acq,GNC\_US}[n]\} = 210$.

The operation FDIR has two predecessors: for GPS Acq$\xrightarrow{0}$FDIR, we have $ow_{Gyro\_Acq,FDIR}[n] = 0$. For GNC_US$\xrightarrow{200}$FDIR, we have a prefix of length 2 with $ow_{GNC\_US,FDIR}[0] = ow_{GNC\_US,FDIR}[1] = O_{FDIR} = 0$, and a pattern of length 10 with $ow_{GNC\_US,FDIR}[10n + 2] = \max\{O_{FDIR}, O_{GNC\_US} - 2T_{FDIR}\} = 10$ and $ow_{GNC\_US,FDIR}[10n + k] = 0$ if $k < 10$ and $k \neq 2$. Thus, $ow_{GNC\_US,FDIR}[n] = 0^2(10.0^9)$ and $ow_{FDIR}[n] = \max\{ow_{Gyro\_Acq,FDIR}, ow_{GNC\_US,FDIR}\} = 0^2(10.0^9)$. We proceed similarly for remaining release times and for deadlines. We finally obtain the following adjusted real-time attributes (unchanged tasks are omitted):

task	Gyro Acq	GPS Acq	FDIR	GNC_US	GNC_DS	TM/TC
offset	(10)	(0)	$0^2(10.0^9)$	(210)	(210)	(1900)
deadline	$(100^2.20.100^8)$	(80)	$(100^2.90.100^9)$	(70)	(790)	(8600)

IV. Implementations

In this section we propose three different implementations of SPC to be used with the EDF policy. The choice between these implementations depends on application related factors such as criticality level or hardware constraints.

A. Implementation with semaphore synchronizations

The first implementation is straightforward: a counting semaphore is allocated for each SPC. Tasks are then scheduled with an EDF scheduler, modified so that it updates semaphore counts at system startup and then at each task release and task completion as specified in Definition 2.

B. Implementation with off-line attributes adjustment

The second implementation consists in directly scheduling the encoded task set $\mathcal{S}^*$ with EDF. The precedence encoding is thus performed completely off-line. This approach only requires minor modifications to be made to the implementation of the EDF scheduler: at each task release, the deadline of the new job is computed based on the deadline word of the task and the release time of the next job is computed based on the release date word of the task. These operations can be implemented with constant complexity (by implementing periodic words as arrays).

C. Implementation with on-line attributes adjustment

The usual way to implement tasks periodicity in numeral real-time operating systems (like Ada or POSIX), is to compute the release date of $\tau_{i.k}$ at the end of $\tau_{i.k-1}$. Thus, in the third implementation we propose to adjust the attributes of the next job at the end of the previous job of the considered task. The attributes of the first job of every task can be computed either at the initialization of the system or off-line.

Without loss of generality, we consider that the tasks are sorted in topological order of the SPC (i.e. if $\tau_i \xrightarrow{h_{i,j}} \tau_j$ then $i < j$). The SPC can thus be represented by a constant triangular matrix SPC where $SPC[i, j] = h_{i,j}$ if $\tau_i \xrightarrow{h_{i,j}} \tau_j$ and $SPC[i, j] = -1$ otherwise.

We first remark that the release date adjustment can be rewritten as :

$$O^*_{i,k} = \max(O_{i,k}, \max_{\forall \tau_{p,q} \xrightarrow{+} \tau_{i,k}} (O_{p,q}))$$

where $\xrightarrow{+}$ is the transitive closure of the job precedence relation. Since in our model, the deadlines are constrained ($D_i \leq T_i$), a subsequent job of a task cannot have an earlier release date than any preceding job of the same task. Therefore, the release date adjustment can be rewritten as:

$$O^*_{i,k} = \max(O_{i,k}, \max_{\forall \tau_p \xrightarrow{+} \tau_n} (O_{p,q_p})) \quad (9)$$

with $q_p = \max_{\tau_{p,q} \xrightarrow{+} \tau_{i,k}} (q)$.

Given a job $\tau_{i,k}$, the Algorithm 1 first computes q_p for every task τ_q, storing it in an array LJ. Then it directly computes $O^*_{i,k}$ using equation 9.

Algorithm 1 Computation of $O^*_{i,k}$

Require: SPC represented as a topologically ordered triangular matrix, constant visible by every task

LJ: array$[1..i] \leftarrow [-1, \forall element]$ // Latest Job index
$LJ[i] \leftarrow k$
for p in reverse $i..2$ **do**
  if $LJ[p] \neq -1$ **then**
    for r in $1..p-1$ **do**
      if $SPC[r,p] \geq 0$ **then** // $\tau_r \xrightarrow{SPC[r,p]} \tau_p$
        $LJ[r] \leftarrow \max\left(LJ[r], Prec_{r,p}(LJ[p])\right)$
      end if
    end for
  end if
end for
$O^*_{i,k} \leftarrow \max\left(O_{i,k}, \max_{j=1..i}\left(O_{j,LJ[j]}\right)\right)$

The computation of the adjusted deadlines is symmetric and is not detailed due to space limitation.

D. Complexities

The complexities are summarized in Figure 6 (n is the number of tasks of the task set). First, the schedulability analysis is exponential in space and time, as we need to unfold the task set on its hyperperiod.

The implementation with semaphores has low overhead in terms of space: there are at most n^2 SPC to store, each SPC needs constant space (each task also needs constant space). The time complexity of the implementations is given per job (number of operations required to enforce real-time and precedence constraints). $\mathcal{O}(n)$ corresponds to the complexity of the reordering of tasks by increasing deadlines. This is the most efficient implementation of the three. However, as mentioned in the Introduction, highly critical systems must go through very constraining certification processes, such as the DO-178B [4] for airborne systems for instance. Therefore, relying on implementations without semaphore synchronizations leads to a more compact Operating System, which is easier to certify.

The implementation with off-line encoding has low time complexity and requires only a very simple Operating System (no need for semaphores). It is thus well-suited for highly critical systems. This approach does however have a high space complexity due to the requirement to store release date words and deadline words.

The on-line encoding uses a space in $\mathcal{O}(n^2)$ for the unique SPC matrix and additionally a linear space for every task, used to compute LJ. The space overhead is thus well-suited for embedded systems with reduced memory space compared to the off-line encoding method. Nevertheless, the time complexity is $\mathcal{O}(n^2)$ every time a job is completed.

	Analysis	Semaphores	Off-line enc.	On-line enc.
Time	exp	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$
Space	exp	$\mathcal{O}(n^2)$	exp	$\mathcal{O}(n^2)$

Fig. 6. Complexities

V. Conclusion

We defined a schedulability analysis for periodic tasks related by a subclass of extended precedences called Semaphore Precedence Constraints. The SPC support multi-rate communications that follow regular repetitive patterns. The analysis first consists in adjusting job real-time attributes to encode precedence constraints, representing the sequence of adjusted real-time attributes of the successive jobs of a task as periodic words. Then we apply a classic schedulability test for EDF on the encoded task set. We also proposed three implementations of SPC to be used for EDF. These techniques could easily be adapted to more general precedence constraints that follow repetitive patterns (for instance precedence constraints defined on several hyperperiods).

References

[1] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI, 2nd Edition*. The MIT Press, 1999.

[2] J. Forget, "A synchronous language for critical embedded systems with multiple real-time constraints," Ph.D. dissertation, Université de Toulouse, 2009.

[3] C. Sofronis, S. Tripakis, and P. Caspi, "A memory-optimal buffering protocol for preservation of synchronous semantics under preemptive scheduling," in *6th International Conference on Embedded Software (EMSOFT'06)*, 2006.

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

[5] M. Spuri and J. Stankovic, "How to integrate precedence constraints and shared resources in real-time scheduling," *IEEE Transactions on Computers*, vol. 43, no. 12, pp. 1407–1412, 1994.

[6] A. Munier, "The basic cyclic scheduling problem with linear precedence constraints," *Discrete applied mathematics*, vol. 64, no. 3, pp. 219–238, 1996.

[7] P. Richard, F. Cottet, and M. Richard, "On-line scheduling of real-time distributed computers with complex communication constraints," in *ICECCS'2001*, Skvde (Sweden), 2001.

[8] L. Cucu, R. Kocik, and Y. Sorel, "Real-time scheduling for systems with precedence, periodicity and latency constraints," in *10th International Conference on Real-Time Systems (RTS'02)*, Paris, Apr. 2002.

[9] J. Forget, F. Boniol, E. Grolleau, D. Lesens, and C. Pagetti, "Scheduling dependent periodic tasks without synchronization mechanisms," in *16th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'10)*, 2010.

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

[11] 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.

[12] S. Baruah, D. Chen, S. Gorinsky, and A. Mok, "Generalized multiframe tasks," *Real-Time Systems*, vol. 17, pp. 5–22, 1999.

[13] J. Y.-T. Leung and M. L. Merrill, "A note on preemptive scheduling of periodic, real-time tasks," *Information Process. Lett.*, vol. 11, no. 3, pp. 115–118, 1980.

[14] J. Goossens and R. Devillers, "Feasibility intervals for the deadline driven scheduler with arbitrary deadlines," in *6th International Conference on Real-Time Computing Systems and Applications (RTCSA '99)*, 1999.

[15] A. Choquet-Geniet and E. Grolleau, "Minimal schedulability interval for real-time systems of periodic tasks with offsets," *Theor. Comput. Sci.*, vol. 310, pp. 117–134, January 2004.