ELSEVIER

WCET free time analysis of hard real-time systems on multiprocessors: A regular language-based model

Dominique Geniet[a,*], Gaëlle Largeteau[b]

[a] *Laboratoire d'Informatique Scientifique et Industrielle, Université de Poitiers, Téléport 2, Site du Futuroscope, F-86960 Futuroscope Chasseneuil Cedex, France*
[b] *Signal, Image, Communication, Université de Poitiers, Téléport 2, Site du Futuroscope, F-86960 Futuroscope Chasseneuil Cedex, France*

Communicated by M. Mavronicolas

Abstract

This paper presents the initial step of an aid design method earmarked for operational validation of hard real-time systems. We consider systems that are composed of sequential hard real-time tasks, which are embedded on centralized multiprocessor architectures. We introduce a model based upon untimed finite automata and meant to collect the operational behaviors of the system compatible with its time specifications, and we go on to provide a feasibility decision result for systems composed of tasks presenting CPU loads which are exact values: execution times are not WCET values. This is why we call this approach *WCET-free analysis*. The results we have achieved likewise involve hardware specifications such as multiprocessors and speeds of processors.
© 2007 Elsevier B.V. All rights reserved.

Keywords: Finite automata; Real-time systems; Operational validation

1. Introduction

Real-Time scheduling has been implemented in real-time kernels by means of fixed priority policies (mainly RM and DM). Dynamic priority policies (EDF, LLF) also exist [29], but they have not been implemented in operating systems. The scheduling power of all these policies has been studied for uniprocessor and multiprocessor targets [30, 18]. Since they have not been designed to deal with precedence or synchronization constraints, additional specific techniques have been designed [34,8].

Real-time validation has been studied: it consists in deciding whether specific software can be scheduled on a given target (feasibility) or if a specific on-line policy can be used to schedule the software (validation). The knowledge of all valid scheduling sequences concerning a task system designed to run on a specific hardware target is useful (in particular) to choose the best scheduling technique which addresses the case study.

Actual *system* approaches are software-engineering oriented: for instance, [2] is designed to build specific policies or controllers, others are based on a categorization and the analysis of system constraints (global vs local in [2]), [35] integrates the real-time validation step into the software life cycle (see Fig. 1), [14] proposes ACSR algebra, whose
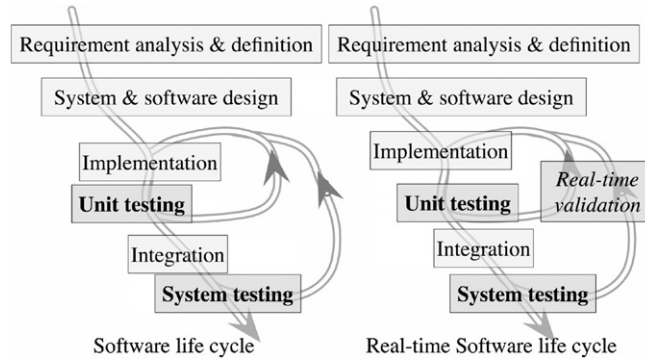

Fig. 1. The operational validation in the software life cycle.

purpose is to facilitate conception and analysis into the specification step of the life cycle, etc. A more controller-oriented technique is proposed by the TIMES project [5]: producing a scheduler from the time specifications and calculating the worst response times of software tasks.

The problem of feasibility for multiprocessor context has been studied by few authors: [9] deals with independent task systems with fixed CPU loads, and studies multiprocessor scheduling analytic criteria; [11] also deals with fixed CPU loads, and gives sufficient scheduling analytic conditions; in [10], the authors define the concept of *sustainability*, which characterizes the persistence of the validity when relaxing time characteristics of tasks. All these works deal with feasibility analysis and characterization, not with collecting scheduling sequences. As far as we know, this problem of collecting all valid scheduling sequences has not been studied for multiprocessor targets. Solving this problem will be useful to help the user to choose a good scheduling technique. It is the aim of this paper.

Previously existing feasibility studies (they address uniprocessor targets) stand on modeling scheduling sequences, like timed automata [4,3], Petri nets [24], etc. [37,32]. Feasibility addresses the existence of a valid scheduling sequence: in terms of languages, this problem corresponds to the emptiness of a language. [19] shows that solving the emptiness problem for untimed automata is linear in the number of states, whereas [3] shows that this problem is PSPACE complete for timed automata. This comparative analysis leads us to use untimed finite automata: this model offers the highest level of decidability for the properties we deal with, and the computing complexities are compatible with an effective computing (see [3] and [19] for a presentation of timed and untimed automata).

Each task of the system is translated into a finite automaton whose accepted words correspond to all possible behaviors of the task. The automaton is computed on the basis of the time characteristics of the task, in order to accept only the task behaviors compatible with its time constraints. Concurrency is modeled by products of automata, and synchronization (processor and resource sharing, communication) thanks to the Arnold–Nivat model [6].

Since we address the problem of collecting all possible behaviors of the systems we study, we do not deal with the *WCET* assumption [17], this is why we call our approach *WCET-free analysis*. In [26], Krčál and Yi show the decidability of the scheduling problem for systems specified by the [Min, Max] CPU loads. Amnell et al. cite this problem to be addressed [5]. In this paper, we are presenting a hopefully quite effective solution: we deal with systems specified by all possible CPU loads for all tasks of the system, and we are proposing an algorithmic technique based on untimed automata to reach this decision. Our technique revolves around solving the reachability problem, whose complexity is known to be lower for untimed automata. All these factors have got to express our choice on untimed finite automata (i.e. regular languages) to model real-time systems.

We have organised this paper as follows.

- Since scheduling decisions depend on both the analysed task system and the hardware and software context (see [25] for a complete analysis which points this fact out), the first part of the paper (Section 2) addresses the task systems we deal with and a taxonomy of possible contexts, in order to define a precise canvas for following studies.
- The second step consists in the technical aspects of the work: we present the basic elements of our language-based technique: Section 3 deals with the language-based model and the validation process; Section 4 shows how to extend this technique to address synchronization; Section 5 shows how the *WCET* assumption can be avoided.
- The final section (Section 7) deals with the internals of the computing techniques and the results of experimentation to point out the improvement level brought by our optimization technique.

2. Validation of real-time systems

In this section, we describe both the structure of real-time systems and a model meant to represent execution contexts.

2.1. Real-time systems

A real-time system is composed of a finite set $(\tau_i)_{i \in [1,n]}$ of tasks. Each task τ_i is a sequence of tasks (or instances) τ_{ij} ($j \in \mathbb{N}$). Each task τ_{ij} is activated by the occurrence of an incoming event. When the flow of incoming events associated with τ_i is periodic (the time interval between two successive incoming events is a constant $T \in \mathbb{N}^*$), τ_i is called periodic, and T is one of its characteristics. If not, τ_i is called sporadic.

When all tasks of a real-time system are periodic, the system is likewise called periodic.

Each job τ_i which belongs to a real-time system is time-specified with 4 time characteristics (r_i, T_i, C_i, D_i). If τ_i is periodic, the semantics of these characteristics are as follows:

T_i is the period of τ_i: it is the constant delay that separates the activation dates of two consecutive tasks τ_{ij} and τ_{ij+1}.

r_i is the first activation date: it is the occurrence date for the first event of the flow associated with τ_i. The activation date of a task τ_{ij} is $r_i + jT_i$.

C_i is the CPU load of tasks $(\tau_{ij})_{j \in \mathbb{N}}$: it is the CPU time that must be allocated to τ_{ij} to complete its execution. Here, we consider the context of invariable CPU times for jobs: all tasks τ_{ij} share the same CPU load C_i. In Section 5, we extend this model to address tasks whose successive instances do not share the same CPU load.

D_i is the critical delay of τ_i: it determines the deadlines of all tasks $(\tau_{ij})_{j \in \mathbb{N}}$, that are the dates when τ_{ij} must be completed. The deadline for task τ_{ij} is $r_i + jT_i + D_i$. In this paper, we assume that $D_i \leq T_i$.

It is impossible to specify at what exact time the initial activation event of an *alarm job* will occur: if τ_i is sporadic, r_i may be undefined, and T_i indicates the minimum delay that separates two successive activations of τ_i.

2.2. Execution contexts

Scheduling a real-time system consists not only in attributing CPU allocation time to jobs, but also in doing so according to the deadline constraints. The feasibility of scheduling a real-time system most notably depends on the hardware framework (number of processors, distribution, etc.).

A scheduling context provides an accurate description of the hardware and software configuration the scheduling algorithm is supposed to work with. A syntax meant to describe these contexts has been proposed in [22], and has been extended in [12] and [23]. Here, we use a syntax adapted from that of [23].

An execution context is specified by a table

Hardware			Software		
Architecture	Clock	Communication	Structure	Synchronisation	Preemption
(n, p)	*Mode*	(Cpx, Snc)	(Jb, St, Ld)	*Constraints*	*Prmpt*

that can be understood with the following semantics:

- **Hardware specifications**
  - . **Architecture** $= (n, p) \in (\mathbb{N}^* \cup \{n\})^2$
    n specifies the number of nodes, and p the maximum number of processors on each one.
  - . **Clock** $= Mode \in \{common, harmonic, synch, independent\}$
    specifies the time dynamics of the processors of each node. Its specification is based on *unit* (the common duration of all atomic statements) and *start* (starting date of the processor driving clock). Using processors $(p_i)_{i \in [1,p]}$ on a node, the semantics to be associated with the possible choices are the following:
    common all processors follow the same clock:
    $$(i, j) \in [1, p]^2 \Rightarrow \begin{cases} speed(p_i) = speed(p_j) \\ start(p_i) = start(p_j) \end{cases}$$

harmonic speeds of processors follow the same clock, in a harmonic way:
$$(i, j) \in [1, p]^2 \Rightarrow \begin{cases} speed\,(p_i) \in speed\,(p_j)\,\mathbb{N} \vee speed\,(p_j) \in speed\,(p_i)\,\mathbb{N} \\ start\,(p_i) = start\,(p_j) \end{cases}$$

synch all processors follow clocks which start simultaneously:
$$(i, j) \in [1, p]^2 \Rightarrow start\,(p_i) = start\,(p_j)$$

independent processors follow different clocks, and then there is no relation between speeds and starts

. **Communication**= $(Cpx, Snc) \in \{H^0, H^-, H^+\} \times \{s, b, a\}$

Cpx specifies communication complexities: H^0 for *centralized* (and then a null complexity), H^+ for *non-null same complexity communications*, H^- for *heterogeneous communication*.

Snc specifies the synchronism level of communication channels. Channels between two tasks are implemented on buffers of size $k \in \bar{\mathbb{N}}$, that are specified in the following way: **s** for *synchronous* (i.e. $k = 0$), **b** for *asynchronous with limited buffer* (i.e. $k \in \mathbb{N}^*$), **a** for asynchronous ($k = +\infty$).

- **Software specifications**
  . **Structure**= $(Jb, St, Ld) \in \{s, p, a\} \times \{0, r_i, \perp\} \times \{C_f, C_v\}$
  with the following semantics:
  s All tasks are sporadic
  p All tasks are periodic
  a There are periodic and sporadic tasks
  0 The first instances of all tasks are activated synchronously at time 0
  r_i The first instances of all tasks are not activated synchronously at time 0
  $\perp$ No specification is given on first activation dates
  C_f All instances of a job require the same CPU allocation time
  C_v Two different instances of a same job may require different CPU allocation times
  . **Synchronization** = $Constraints \in \mathfrak{P}\,(\{r, c, p, x\})$
  with the following semantics:
  r There is resource sharing between jobs
  c There is communication between jobs
  p There are precedence constraints between jobs
  x There are exclusion constraints between jobs
  . **Preemption** = $Prmpt \in \{notp, parp, totp\}$
  with the following semantics:
  notp Tasks cannot be preempted
  parp Tasks can be preempted, excluding some specific contexts (e.g. critical sections)
  totp Tasks can be preempted whatever the context

3. Time validation process for periodic systems

In this section, we consider the context

Hardware			Software		
Architecture	Clock	Communication	Structure	Synchronisation	Preemption
$p \geq 1$	*Common*	(H^0, s)	(p, r_i, C_f)		*totp*

This context is called $ICCT\,(p)$ in the following (*I* for *independent tasks*, the first *C* for *common clock*, the second *C* for *centralized system*, *T* for *periodic real-time system* and (*p*) for *p processors*). In the following, the task system $(\tau_i)_{i \in [1,n]}$ is noted Γ: *n* is the number of tasks.

3.1. Basic notions on languages

The reader is presumed to be familiar with the basic notions about words, languages and automata [19]. However, we wish to recall two basic notions and some notations and results, of which we have made use in this paper.

In the following, Reg(Σ) is the class of regular languages on the alphabet Σ.

Definition 1. Let Σ_1 and Σ_2 be two alphabets. The *Shuffle* ($\amalg$) is a binary operation on words or languages, defined in the following way:

(1) $\forall a \in \Sigma_1 \cup \Sigma_2, a \amalg \epsilon = \epsilon \amalg a = \{a\}$
(2) $\forall (a, b, \omega, \xi) \in \Sigma_1 \times \Sigma_2 \times \Sigma_1^* \times \Sigma_2^*, a\omega \amalg b\xi = a\,(\omega \amalg b\xi) \cup b\,(a\omega \amalg \xi)$
(3) $\forall L_1 \subset \Sigma_1^*, \forall L_2 \subset \Sigma_2^*, L_1 \amalg L_2 = \underset{(\alpha,\beta) \in L_1 \times L_2}{\cup} (\alpha \amalg \beta)$

Proposition 1. *Let Σ_1 and Σ_2 be two alphabets, and $L_1 \subset \Sigma_1^*$ and $L_2 \subset \Sigma_2^*$. We get $L_1 \in \mathrm{Reg}(\Sigma_1) \wedge L_2 \in \mathrm{Reg}(\Sigma_1) \Rightarrow L_1 \amalg L_2 \in \mathrm{Reg}(\Sigma_1 \cup \Sigma_2)$.*

Definition 2. Let $L \subset \Sigma^*$, and let $\omega \in L$. We call the prefix of ω every $\alpha \in \Sigma^*$ such that $\exists \beta \in \Sigma^*$ such that $\omega = \alpha\beta$.

The set of prefixes of a word ω is denoted $\mathrm{Pref}(\omega)$. This definition is extended to languages by way of $\mathrm{Pref}(L) = \underset{\alpha \in L}{\cup} \mathrm{Pref}(\alpha)$.

Definition 3. Let $L \subset \Sigma^*$, and $\omega \in L$. We call infinitely extendable prefix of L every $\alpha \in \Sigma^*$ such that $\forall n \in \mathbb{N}, \exists \beta \in \Sigma^*$ such that $|\beta| > n$ and $\alpha\beta \in L$. The set of infinitely extendable prefixes of L is called $\mathrm{Center}(L)$.

Proposition 2. *We obtain*

(1) $\mathrm{Center}(L) = L^*\mathrm{Pref}(L)$
(2) $L \in \mathrm{Reg}(\Sigma) \Rightarrow \mathrm{Center}(L) \in \mathrm{Reg}(\Sigma)$.

3.2. The language of job-valid behaviors

We define the notion of time unit in the following way: it is the execution time of an atomic statement (i.e. an assembler statement) on the target machine. In this work, we consider this value to be shared by all atomic statements.

Let us consider the job τ_i, whose time parameters are $r_i \in \mathbb{N}$, $C_i \in \mathbb{N}^*$, $D_i \in \mathbb{N} \cap [C_i, +\infty[$ and $T_i \in \mathbb{N} \cap [D_i, +\infty[$. From its activation date $r_i + k \times T_i$, the kth instance of τ_i must own a CPU resource for C_i time units on the time interval $[r_i + k \times T_i, r_i + k \times T_i + D_i[$. Let us note a_i the state τ_i *owns a CPU for one time unit*, and $\bullet$ the state τ_i *does not own a CPU for one time unit*. Every word of $a_i^{C_i} \amalg \bullet^{D_i - C_i}$ corresponds, on any time interval of the form $[r_i + k \times T_i, r_i + k \times T_i + D_i[$, to a processor time allocation compatible with the time constraints of τ_i. This set is regular. If the scheduling configuration is valid, τ_i is inactive on every time interval of the form $]r_i + k \times T_i + D_i, r_i + (k+1) \times T_i[$. This inactivity is modeled by the word $\bullet^{T_i - D_i}$. Then, every word of $(a_i^{C_i} \amalg \bullet^{D_i - C_i}) \bullet^{T_i - D_i}$ is a correct CPU time allocation for τ_i on any time interval of the form $[r_i + k \times T_i, r_i + (k+1) \times T_i[$.

The task τ_i is defined as the sequence $(\tau_{i\,j})_{j \in \mathbb{N}}$ of its instances. A processor allocation compatible with τ_i's time constraints is a sequence of processor allocations compatible with τ_i's successive instance time constraints. Let $(\omega_j)_{j \in \mathbb{N}} \in ((a_i^{C_i} \amalg \bullet^{D_i - C_i}) \bullet^{T_i - D_i})^{\mathbb{N}}$. For each $n \in \mathbb{N}$, the word $\omega_0 \omega_1 \dots \omega_n$ models a time valid processor time allocation for any sequence of $n + 1$ successive instances of τ_i. In a general way, any word ω of $((a_i^{C_i} \amalg \bullet^{D_i - C_i}) \bullet^{T_i - D_i})^*$ models a valid processor time allocation for τ_i on any time interval of the form $[r_i + k \, T_i, r_i + k \times T_i + |\omega|[$, and then, in particular, on the time interval $[r_i, r_i + |\omega|[$. Insofar as τ_i is inactive on interval $[0, r_i[$, word $\bullet^{r_i} \omega$ models a valid processor allocation on the time interval $[0, r_i + |\omega|[$. ω can be as long as we wish. Then, the language $\bullet^{r_i} ((a_i^{C_i} \amalg \bullet^{D_i - C_i}) \bullet^{T_i - D_i})^*$ collects all the valid processor allocation for τ_i.

The scheduling validation problem consists, at time t, in deciding on the evolution possibilities of τ_i in both a given hardware and a given software context. Of course, the past of τ_i is known. Here, this past is the history of τ_i's CPU allocations, that is to say a finite word ω of $\{a_i, \bullet\}^*$. During this past, some instances of τ_i were completed, and the current instance is on (we can say so even at the outset). Then, by construction, ω is of the form $\omega_1.\mu$, where $\omega_1 \in \bullet^{r_i} ((a_i^{C_i} \amalg \bullet^{D_i - C_i}) \bullet^{T_i - D_i})^*$ (the completed past instances), and $\exists \nu \in \{a_i, \bullet\}^*$ such that $\mu\nu \in ((a_i^{C_i} \amalg \bullet^{D_i - C_i}) \bullet^{T_i - D_i})$ (the current instance can be completed according to the time constraints). So, ω is a prefix of a word of $\bullet^{r_i} ((a_i^{C_i} \amalg \bullet^{D_i - C_i}) \bullet^{T_i - D_i})^*$. Moreover, by construction as well, any instant f belonging to $]t, +\infty[$ (the future), there exists $\eta \in ((a_i^{C_i} \amalg \bullet^{D_i - C_i}) \bullet^{T_i - D_i})^*$ such that $|\omega\nu\eta| > f$. Following which, at each time t, the past

ω of τ_i is a word of the center of $\bullet^{r_i}((a_i^{C_i} \amalg \bullet^{D_i-C_i})\bullet^{T_i-D_i})^*$. Reciprocally, by definition, every word of this center language is the past of a valid processor allocation configuration.

Definition 4. We call Time-Valid Behavior of τ_i every word of

$$\text{Center}\left(\bullet^{r_i}\left(\left(a_i^{C_i} \amalg \bullet^{D_i-C_i}\right)\bullet^{T_i-D_i}\right)^*\right).$$

A time-valid behavior of τ_i models a processor time allocation (for τ_i) such that we can guarantee that, in the future, the past of τ_i can effectively be extended according to τ_i time constraints. In the following, we note $\mathcal{L}(\tau_i)$ in this language.

Remark 1. By following a similar line of reasoning, we can show that the time-valid behaviors of a sporadic job τ_i, of time parameters[1] C_i, D_i and T_i belong to

$$\text{Center}\left(\bullet^*\left(\left(a_i^{C_i} \amalg \bullet^{D_i-C_i}\right)\bullet^{T_i-D_i}\bullet^*\right)^*\right).$$

In the following, all the techniques we use and the properties we obtain arise from the property $\mathcal{L}(\tau_i)$ *collects all valid behaviors*. Then, since this property also applies to sporadic tasks, all these techniques and properties may be applied to periodic as well as sporadic jobs. For simplicity's sake, in the following, we deal only with periodic jobs, but all the obtained results are applicable for the general case.

3.3. Model for concurrency: The language of system-valid behaviors

In order to model concurrency, we use the homogeneous product of regular languages, which is defined as follows:

Definition 5. Let Σ_1 and Σ_2 be finite alphabets, and $L_1 \subset \Sigma_1^*$ and $L_2 \subset \Sigma_2^*$.

- Let $\alpha = \alpha_1\alpha_2\ldots\alpha_n \in L_1$ and $\beta = \beta_1\beta_2\ldots\beta_n \in L_2$ two words of the same length n. The homogeneous product of α and β is the word $\alpha\Omega\beta = \binom{\alpha_1}{\beta_1}\binom{\alpha_2}{\beta_2}\ldots\binom{\alpha_{|\alpha|}}{\beta_{|\beta|}} \in (\Sigma_1 \times \Sigma_2)^n$.

- The homogeneous product of languages L_1 and L_2 is the language $L_1 \Omega L_2 = \bigcup_{n\in\mathbb{N}}\left(\bigcup_{\substack{\alpha\in L_1\cap\Sigma_1^n \\ \beta\in L_2\cap\Sigma_2^n}}\{\alpha\Omega\beta\}\right)$

Ω is a binary operator. Then, the expression $(a\,\Omega\,b)\,\Omega\,c$ is equal to $\binom{\binom{a}{b}}{c}$, and the expression $a\,\Omega\,(b\,\Omega\,c)$ is equal to $\binom{a}{\binom{b}{c}}$. In this case, the semantics associated with a vector is *simultaneity*. Then, $\binom{\binom{a}{b}}{c}$ and $\binom{a}{\binom{b}{c}}$ share the same semantics, which is *simultaneous execution of a, b and c*. Of course, such semantics can also be associated with the vector $\begin{pmatrix} a \\ b \\ c \end{pmatrix}$. We consider that $\binom{\binom{a}{b}}{c} \equiv \binom{a}{\binom{b}{c}} \equiv \begin{pmatrix} a \\ b \\ c \end{pmatrix}$. It follows that the operator Ω is associative, and can naturally be generalized to n-uples of languages.

A behavior of Γ corresponds to a n-uple of behaviors of the τ_i's, the semantics associated with $\binom{a}{b}$ being simultaneity of a and b. That is why the set of behaviors of Γ is $\Omega_{i=1}^{i=n}(\mathcal{L}(\tau_i))$. The homogeneous product does not reduce the set of behaviors for jobs. And since all $\mathcal{L}(\tau_i)$'s are centers of regular languages, $\Omega_{i=1}^{i=n}(\mathcal{L}(\tau_i))$ is also a center of language.

If there are fewer processors than jobs (a frequent case!), owning a processor is a resource-sharing problem. It is integrated within the model thanks to Arnold–Nivat's technique [6]: for a p processor architecture, we take into account the language

$$S_p = \left\{\omega \in \prod_{i=1}^{i=n}\{\bullet, a_i\} \text{ such that } |\omega|_\bullet \geq n - p\right\}$$

---

[1] Recall that for sporadic jobs, T_i specifies the minimum time interval between two successive instances of the alarm signal associated with τ_i.
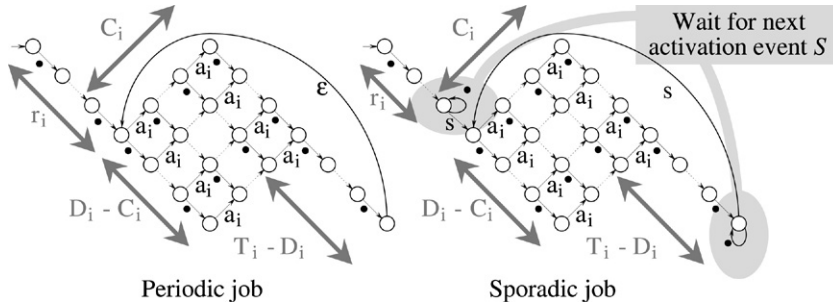
Fig. 2. Generic automata for periodic and sporadic tasks.

that collects the instantaneous configurations corresponding to valid executions on p processors. The language $\Omega_{i=1}^{i=n} (\mathcal{L}(\tau_i)) \cap S_p^*$ collects all time behaviors that are compatible with both τ_i's time specifications and p-processor hardware architectures.

The class of regular language centers is not closed by intersection. Here, this property means that processor-sharing can lead a real-time job system to miss at least one of the imposed deadlines. Since we are interested in the set of time-valid behaviors, we only consider the subset of $\Omega_{i=1}^{i=n} (\mathcal{L}(\tau_i)) \cap S_p^*$ that collects the time-valid behaviors (i.e. the scheduling sequences that can be indefinitely extended according to the system time constraints) for the whole real-time system.

The language $\Omega_{i=1}^{i=n} (\mathcal{L}(\tau_i)) \cap S_p^*$ is partitioned into two languages

$$L_{\text{inf}} = \text{Center} \left(\underset{i=1}{\overset{i=n}{\Omega}} (\mathcal{L}(\tau_i)) \cap S_p^* \right)$$

and

$$L_{\text{fin}} = \left(\underset{i=1}{\overset{i=n}{\Omega}} (\mathcal{L}(\tau_i)) \cap S_p^* \right) \setminus \text{Center} \left(\underset{i=1}{\overset{i=n}{\Omega}} (\mathcal{L}(\tau_i)) \cap S_p^* \right).$$

L_{inf} collects all behaviors compatible with the p-processor constraint, and which can be infinitely extended according to τ_i's time specifications: they are *valid* behaviors. On the contrary, L_{fin} collects other behaviors: since they do not belong to the center of the language, they model CPU allocation sequences ensuring that, in the future, at least one of the τ_i's will miss its deadline. Then, L_{inf} indicates the exact set of time-valid behaviors.

This is why we define the set of time-valid behaviors of a job system in the following way:

Definition 6. We call time-valid behavior for the system Γ within the context $ICCT(p)$ every element of the language

$$\mathcal{L} \left(\frac{\Gamma}{ICCT(p)} \right) = \text{Center} \left(\underset{i=1}{\overset{i=n}{\Omega}} (\mathcal{L}(\tau_i)) \cap S_p^* \right).$$

3.4. Feasibility decision

The language $\mathcal{L}(\tau_i)$ is implemented through its canonical associated finite automata $A(\tau_i)$ (they follow generic structures: see Fig. 2). Thus, the language $\mathcal{L} \left(\frac{\Gamma}{ICCT(p)} \right)$ is also implemented thanks to the computing of its associated finite automaton $A \left(\frac{\Gamma}{ICCT(p)} \right)$, which is calculated from the $A(\tau_i)$'s through algebraic operations on automata. The decisions pertaining to these languages arise from the structural properties of the corresponding automata. Thus, the existence of these automata is necessary to reach the decision. That is why, in the following, we always show that synchronized products are regular.

The language $\mathcal{L} \left(\frac{\Gamma}{ICCT(p)} \right)$ is designed to contain all valid behaviors of the system Γ. Thus, deciding on the feasibility of Γ under the context $ICCT(p)$ comes about in an obvious way through the construction of this language:

Theorem 1. *The system Γ is valid within the context $ICCT(p)$ if and only if $\mathcal{L} \left(\frac{\Gamma}{ICCT(p)} \right) \neq \emptyset$.*
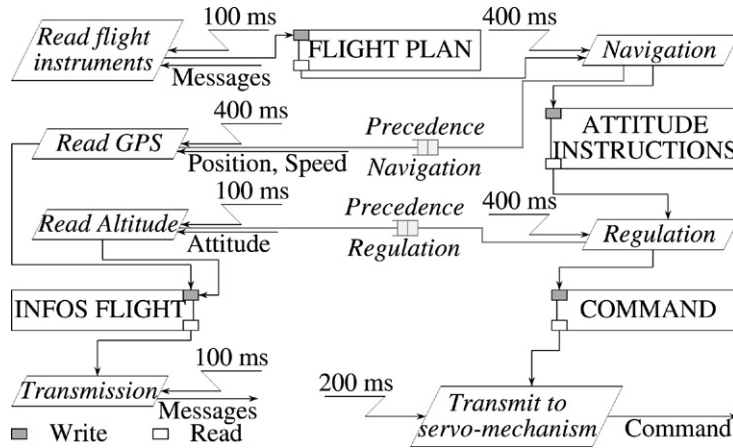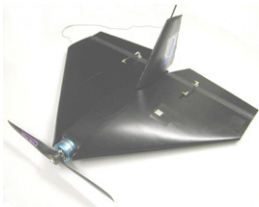
Fig. 3. Preliminary DARTS diagram for the UAV controller.

3.5. A case study: The AMADO project

From 2002 through July 2005, the French National Research Agency for Space and Aeronautics (ONERA) organized an international competition for the design of miniature unmanned air vehicles (UAV): the AMADO[2] project (see [1] for details). This competition was open to all universities and engineering schools in the world, and was sponsored by the *Délégation Générale de l'Armement* (DGA), of the French Defense Ministry.

One goal of this competition was to demonstrate the technical feasibility of miniature UAV as an infantry aid. Even though DGA is an organization devoted to weapon conception, UAVs were designed for the sake of observation, not destruction; they may be viewed as flying binoculars.

Three Poitiers-based labs[3] took part in this competition, and are designing a UAV prototype.

In this paper, we are only interested in controller program design, and more specifically in its autonomous running mode. In this context, the embedded software must drive the UAV in autonomous mode. This part of the software is composed of 7 periodic jobs with invariable CPU load. These jobs share 6 critical resources, which are not considered in this section. Fig. 3 presents its DARTS preliminary diagram. Time specifications for the different jobs that compose this controller are the following:

Job	Role	r_i	D_i	T_i	C_i
τ_1	Read Attitude	0	5	5	4
τ_2	Read Flight Instruments	0	5	5	4
τ_3	Read GPS	0	12	25	10
τ_4	Transmit to servo-mechanism	0	6	10	2
τ_5	Transmission	0	5	5	2
τ_6	Navigation	0	15	25	10
τ_7	Regulation	0	16	25	10

---

[2] Automated Miniature Aircraft for Detection and Observation.

[3] - LISI (Research Ministry Team (EA) nr 1232): Lab of Applied Computer Science
- SIC (CNRS Emerging Team (FRE) nr 2731): Signal, Image, Communication
- LEA (CNRS Research Unit (UMR) nr 6609): Lab of Aerodynamics.

Computation of the validation automaton for this system is carried out following the sequence presented below, which shows both the order of the computing and the sizes (edge counting) of homogeneous (Ω) as well as synchronized (Π) product automata. Recall that homogeneous products model concurrency, and synchronized products model processor sharing. The reader may note that in the following table, both $|\Omega|$ and $|\Pi|$ share the same values for subsystems limited to the first four jobs: since we have four processors at our disposal, there is no processor sharing, so synchronized products are not computed. On the contrary, there is processor sharing as soon as the 5th job is integrated in the product, and then there exist differences between $|\Omega|$ and $|\Pi|$.

job	$|A_i|$	$\dfrac{|\Omega|}{|\Pi|}$	$\dfrac{|\Omega|}{|\Pi|}$	$\dfrac{|\Omega|}{|\Pi|}$	$\dfrac{|\Omega|}{|\Pi|}$	$\dfrac{|\Omega|}{|\Pi|}$	$\dfrac{|\Omega|}{|\Pi|}$
τ_1	13	$\dfrac{35}{35}$					
τ_2	13		$\dfrac{591}{591}$				
τ_3	84			$\dfrac{1513}{1513}$			
τ_4	26				$\dfrac{5552}{5544}$		
τ_5	17					$\dfrac{33\,417}{33\,041}$	
τ_6	120						$\dfrac{57\,618}{48\,975}$
τ_7	140						

The final product is not empty: this system is feasible for a 4-processor target, without considering critical resource sharing.

4. Integrating job constraints

In this section, we integrate job interdependence to the model, and we show that the validation results always stand. The constraints we wish to consider are critical resource sharing and message-based communications. Thus, we consider the context

Hardware			Software		
Architecture	Clock	Communication	Structure	Synchronisation	Preemption
$p \geq 1$	*Common*	(H^0, s)	(p, r_i, C_f)	*rc*	*totp*

Let Δ be the set of items (resources, messages, etc.) concerning the job system synchronization. The corresponding context is called $D(\Delta)\,CCT(p)$ (with $D(\Delta)$ for *dependent jobs under constraint set Δ*) in the following.

4.1. Using Arnold–Nivat synchronized products

To integrate job interdependence (communication, resource sharing) between the τ_i's, we also use Arnold–Nivat's technique [6]: each resource R (shared resource or communication message) is modeled by a virtual job ξ_R, designed to trace its states (busy/idle, for instance, for a shared resource using a basic protocol, but more highly elaborated protocols can likewise be modeled [7]).

The principle of the Arnold/Nivat model is presented in Fig. 4:

- We consider a set of tasks sharing a critical resource: each task is modeled by a finite automaton. The first step consists in building the product automaton, which models the concurrent system composed of all the tasks.
- The protocol used to manage the constraint (shared resource or any other constraint) is a process: it is modeled by the use of a specific automaton (see [7], for instance). In the case of Fig. 4, the constraint consists in executing b before executing ψ (precedence constraint). Step 2 consists in producing the product automaton that models the concurrent running of both the task system and the protocol. Step 3 consists in erasing all states (and corresponding edges) of the automaton which correspond to a violation of the protocol.
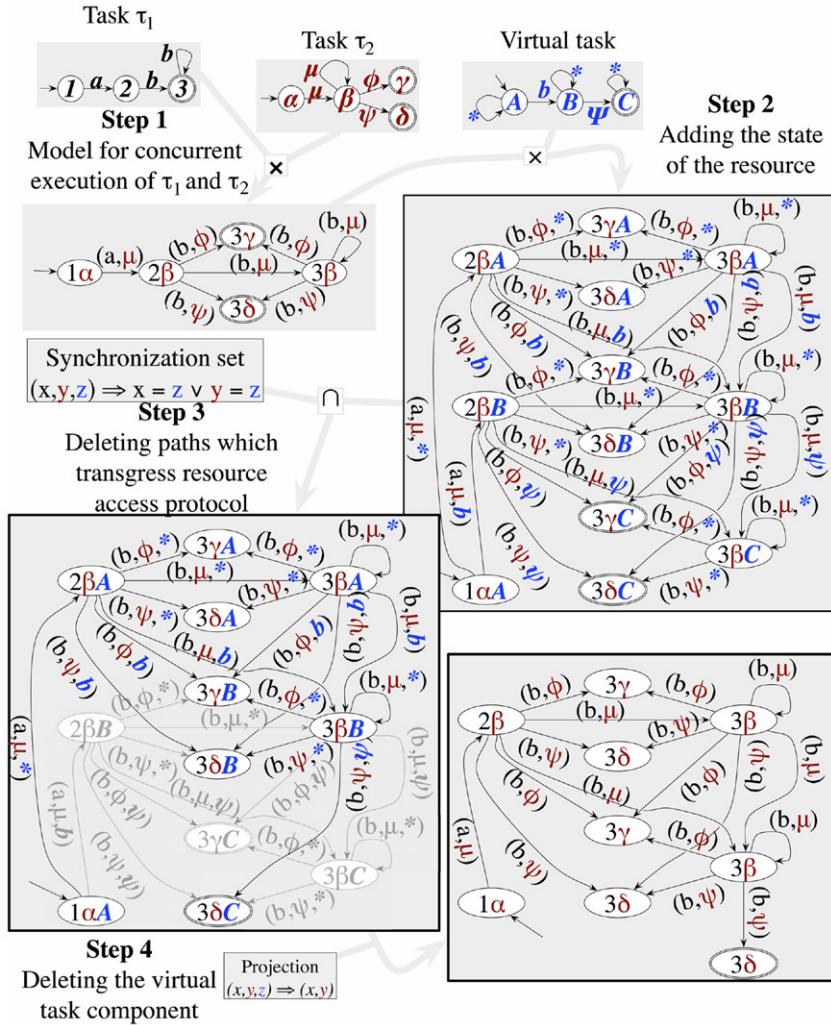
Fig. 4. Arnold–Nivat model for resource sharing.

- The product automaton is composed of the states which satisfy the constraint: the protocol component is now useless (it is not part of the task system), it can be erased through use of a projection (step 4).

The general process can be viewed as follows.

$(\Omega_{i=1}^{i=n}\mathcal{L}(\tau_i))\Omega\mathcal{L}(\xi_R)$ collects the behaviors of the system composed of both the τ_i's and the resource R. Let us now consider the S_R set of the instantaneous configurations compatible with the resource protocol management. $((\Omega_{i=1}^{i=n}\mathcal{L}(\tau_i))\Omega\mathcal{L}(\xi_R)) \cap S_R^*$ collects only the behaviors of Γ which are compatible with the resource protocol management.

Now, we use the same property as for processor sharing (see Section 3.4): the class of regular language centers is not closed by intersection. Here, this property means that sharing a critical resource can lead a real-time system to miss at least one of the imposed deadlines. Since we are still interested in the set of time-valid behaviors, we again consider the subset of $((\Omega_{i=1}^{i=n}\mathcal{L}(\tau_i))\Omega\mathcal{L}(\xi_R)) \cap S_R^*$ that collects the time-valid behaviors (i.e. the scheduling sequences that can be indefinitely extended according to the system time constraints) for the whole system. Following the same reasoning as for context $ICCT(p)$, we collect all infinitely extendable behaviors of this set, i.e. we consider

$$\mathcal{L}\left(\frac{\Gamma}{D\left(R\right)CCT\left(\infty\right)}\right) = \text{Center}\left(\left(\left(\overset{i=n}{\underset{i=1}{\Omega}}\mathcal{L}(\tau_i)\right)\Omega\mathcal{L}\left(\xi_R\right)\right) \cap S_R^*\right).$$

This language is associated with the context $D\left(\Delta\right)CCT\left(\infty\right)$, because it collects all time-valid behaviors according to both time and resource protocol constraints, but without considering processor sharing constraints. These can be integrated in the model through renewed use of Arnold–Nivat's technique, in the same way as for $ICCT\left(p\right)$ context, i.e. computing $\mathcal{L}\left(\frac{\Gamma}{D(R)CCT(p)}\right) = \text{Center}\left(\mathcal{L}\left(\frac{\Gamma}{D(R)CCT(\infty)}\right) \cap S_p^*\right)$.

Remark 2. Let ρ_j be the projection $(x_i)_{i\in[1,n]} \to x_j$, and let us extend this notation to intervals, by defining $\rho_{[a,b]}$ as the projection

$$(x_i)_{i\in[1,n]} \to (x_i)_{i\in[\text{Max}(1,a),\text{Min}(b,n)]}$$

In the language $\mathcal{L}\left(\frac{\Gamma}{D(R)CCT(p)}\right)$, the components corresponding to virtual jobs modeling resources are now useless: we erase them from the model by using $\rho_{[1,n]}$. Thus, we finally consider the language $\rho_{[1,n]}\left(\mathcal{L}\left(\frac{\Gamma}{D(R)CCT(p)}\right)\right)$. For the sake of simplicity, we shall denote this language by $\mathcal{L}\left(\frac{\Gamma}{D(R)CCT(p)}\right)$: then, in the following, we omit writing the ρ_i's.

Since a synchronization model is based upon intersections of languages, we obtain

$$\mathcal{L}\left(\frac{\Gamma}{D\left(\{R_1, R_2\}\right)CCT\left(p\right)}\right) = \text{Center}\left(\mathcal{L}\left(\frac{\Gamma}{D\left(R_1\right)CCT\left(p\right)}\right) \cap S_{R_2}^*\right).$$

Applying this property by induction, we obtain the following definition:

Definition 7. Let Γ be a real-time system and $\Delta = (R_j)_{j\in[1,r]}$ be a set of synchronization constraints (resource or messages). We call time-valid behavior of Γ under the context $D\left(\Delta\right)CCT\left(p\right)$ every element of the language

$$\mathcal{L}\left(\frac{\Gamma}{D\left(\Delta\right)CCT\left(p\right)}\right) = \text{Center}\left(\mathcal{L}\left(\frac{\Gamma}{ICCT\left(p\right)}\right) \cap \left(\bigcap_{i=1}^{i=n}\left(S_{R_i}^*\right)\right)\right)$$

where S_{R_i} is the set of valid instantaneous configurations, according to resource R_i management protocol.

In [6], the author shows how this technique can be used in order to model resource sharing, precedence and communication by message.

4.2. Feasibility decision

Like $\mathcal{L}\left(\frac{\Gamma}{ICCT(p)}\right)$, $\mathcal{L}\left(\frac{\Gamma}{D(\Delta)CCT(p)}\right)$ is a language that collects time-valid behaviors of the system. Then, Theorem 1 remains valid. Thus, we obtain: Γ is valid under context $D\left(\Delta\right)CCT\left(p\right) \Leftrightarrow \mathcal{L}\left(\frac{\Gamma}{D(\Delta)CCT(p)}\right) \neq \emptyset$.

4.3. The AMADO project: Feasibility with critical resources

Let us now complete the feasibility study we began in Section 3.5, by including critical resource sharing in the synchronized product computing. First, we complete the table that presents time characteristics of jobs along with the resources used (resources are identified as R_1, R_2, and so on):

Job	Role	r_i	D_i	T_i	C_i	Resources used
τ_1	Read Attitude	0	5	5	4	Infos Flight (R_1) Precedence Regulation (R_2)
τ_2	Read Flight Instruments	0	5	5	4	Flight plan (R_3) Command (R_4)
τ_3	Read GPS	0	12	25	10	Infos Flight (R_1) Precedence Navigation (R_5)
τ_4	Transmit to servo-mechanism	0	6	10	2	Command (R_4)
τ_5	Transmission	0	5	5	2	Infos Flight (R_1)
τ_6	Navigation	0	15	25	10	Precedence Navigation (R_5) Flight plan (R_3) Attitude Instructions (R_6)
τ_7	Regulation	0	16	25	10	Precedence Regulation (R_2) Command (R_4) Attitude Instructions (R_6)

Computing the validation automaton for this system is carried out following the sequence presented in the table drawn below, in the same way as for an independent job case: Ω and $R_j \Pi$ denote homogeneous and resource R_j synchronized product automata, and C denotes the center language automaton, which is computed after each synchronized product. Each synchronization addresses both resource and processor sharing. The reader can note that in the following computation, tasks are integrated within the product following a different order than in Section 3.5: we begin by integrating the more constrained tasks, in order to limit product automata sizes as much as possible.

	$|A_i|$	$\dfrac{|\Omega|}{R_j\binom{|\Pi|}{|C|}}$					
τ_7	140	$\dfrac{1055}{R_6\binom{884}{786}}$					
τ_6	120		$\dfrac{8233}{R_5\binom{8179}{8179}}$				
τ_3	84			$\dfrac{20\,868}{R_3\binom{18\,429}{17\,358}}\ R_4\binom{14\,083}{10\,210}$	$\dfrac{23\,098}{R_2\binom{17\,710}{15\,701}}\ R_1\binom{11\,506}{9946}$	$\dfrac{15\,701}{R_1\binom{3939}{0}}$	$\dfrac{0}{R_4\binom{0}{0}}$
τ_2	13						
τ_1	13						
τ_5	17						
τ_4	26						

The final product is empty: this system is not feasible for a 4-processor target, according to critical resource sharing. Since this system is feasible without considering resource sharing (see Section 3.5), the resources are responsible for infeasibility.

Let us now observe the synchronization constraint levels involved by each resource, for instance by evaluating the ratios $\frac{|C|}{|\Pi|}$. With this objective, for each resource R_i, we study the subsystem composed of all jobs that share R_i, without processor sharing (i.e. using as many processors as jobs). Resource R_1 is the most constraining: it leads, for a 3-job system, to a level of $\frac{94}{654}$, that is to say about 16%. Overloading R_1-sharing with processor sharing, this level drops to 9% for a 2-processor target, and to 1% for a 1-processor. That is why the whole system is not feasible. The feasibility system diagnosis advises the conceptor to observe R_1 sharing to render the system feasible.

5. WCET free analysis

In general, real-time jobs execute programs whose CPU load is variable: they contain both *if...then...else* or *for...do* statements. In this case, the CPU load of each instance of the job depends on the values of some of its variables, i.e. its functional semantics. In previous sections, the task τ_i is associated with the time parameter C_i, which specifies its CPU allocation time. This value must be viewed as the WCET: it is the minimal CPU allocation time needed to schedule τ_i in the worst case (see Fig. 6).
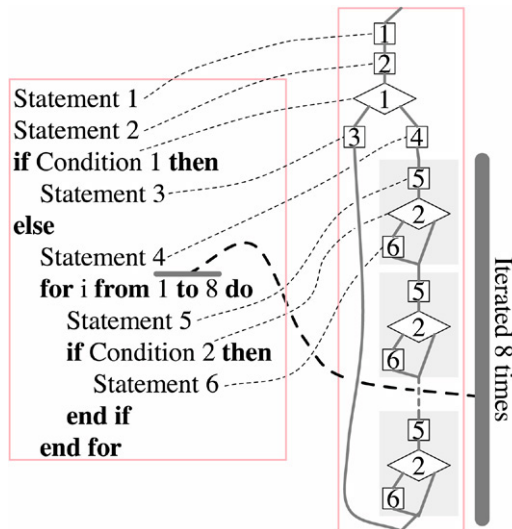
Fig. 5. From the program to the graph.

In this section, we are extending our model in order to analyse systems in accordance with the real values of C_i, as opposed to WCET values. We have termed this manner of proceeding WCET-free analysis.

In this study, we consider programs composed of both atomic (:=), choices (if... statements) and static loop statements (Ada like for...). We consider neither dynamic loops nor recursive subroutine calls: in the context of hard real-time software, this restriction is realistic.

Thus, the context considered here is

Hardware			Software		
Architecture	Clock	Communication	Structure	Synchronisation	Preemption
$p \geq 1$	*Common*	$\left(H^0, s\right)$	(p, r_i, C_v)	rc	*totp*

This context is called $D\left(\Delta\right) CCVT(p)$ (with V for *variable CPU load tasks*) in the following.

5.1. The language of job-valid behaviors

To integrate these kinds of jobs, one must model many CPU loads for each τ_i. However, not all CPU loads are allowed: the only ones allowed are those which effectively correspond to at least one of the functional behaviors of the job. Let us consider, for instance, the program presented on Fig. 5. It is associated with its canonical finite automaton representation by applying the morphism *Statement* $\to a_i$ to all values that label edges (they are statements). Thus, every path through this automaton is labeled with a word a_i^j, where j is a CPU allocation duration compatible with at least one of the behaviors of τ_i. In this example, the set is $a^2 \{a^2, a^2 (a^2 \{a, a^2\})\}$, which can also be described by $\left\{a^4, \left(a^{28+i}\right)_{i\in[1,8]}\right\}$. We denote this set with P_{τ_i}. It is always finite, because there are no dynamic statements in the program (no dynamic loops, no recursive subroutine calls). For $\omega \in P_{\tau_i}$, Center($\bullet^{r_i} \left(\left(\omega\, \text{Ш}\, \bullet^{D_i - |\omega|}\right) \bullet^{T_i - D_i}\right)^*$) collects the set of time-valid behaviors of τ_i such that each instance of τ_i uses a CPU allocation of $|\omega|$ time units. Now, let us call $(\omega_k)_{k\in I}$ the words that belong to P_{τ_i}. Since P_{τ_i} is finite, I is finite too. In general, time-valid behaviors of successive instances of τ_i correspond to different ω_k's. Then, a time-valid behavior of τ_i belongs to $\mathcal{L}(\tau_i) = \text{Center}(\bullet^{r_i} ((\bigcup_{k\in I} (\omega_k\, \text{Ш}\, \bullet^{D_i - |\omega_k|})) \bullet^{T_i - D_i})^*)$. Since I is a finite set, this language remains regular.

The $\mathcal{L}(\tau_i)$'s being sets of τ_i's valid behaviors, computing the set of valid behaviors of the whole system Γ is based upon both homogeneous products and Arnold–Nivat's synchronization technique.
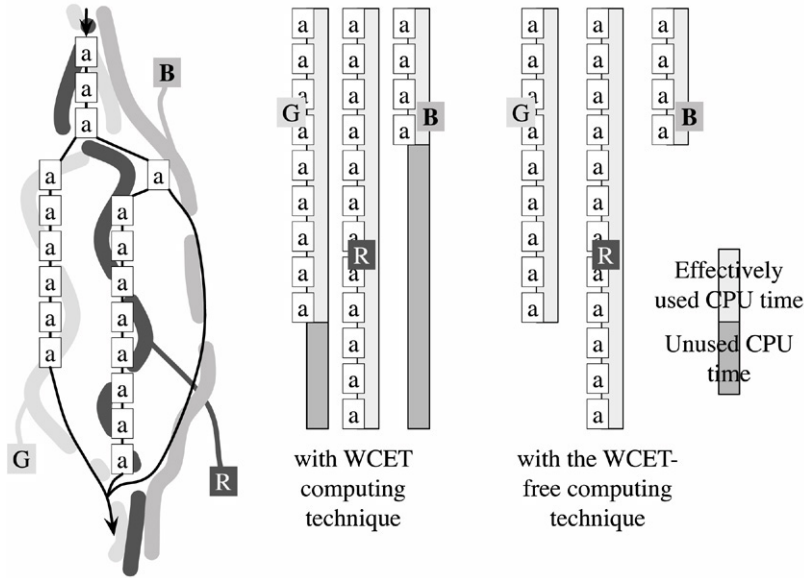
Fig. 6. From the graph to the CPU load set.

5.2. Feasibility decision

The role of Arnold–Nivat's synchronization technique is to forbid some behaviors of the τ_i's. Under the context $D(\Delta)CCT(p)$, forbidding a behavior corresponds to forbidding a specific CPU allocation sequence, but no restrictions are imposed on specific functional behaviors of the job.

Under the context $D(\Delta)CCVT(p)$, synchronizing may forbid τ_i from following some of its behaviors (the shorter ones, for instance), and allow it to follow others (longer...). Thus, the predicate $\mathcal{L}\left(\frac{\Gamma}{D(\Delta)CCVT(p)}\right) \neq \emptyset$ reaches the property *for each job, and there exists at least one valid path*. We call this property *weak feasibility*:

Notation 1. *Recall that we are noting ρ_i the projection of a vector V on its ith component. At this point, we are introducing a new projection: π.*

Let A be an alphabet, and $B \subset A$. We note respectively π_B and $\pi_{\neg B}$ the two morphisms

$$\pi_B : A \to B, \begin{cases} x \in B \to x \\ x \in A \setminus B \to \epsilon \end{cases} \text{ and } \pi_{\neg B} : A \to B, \begin{cases} x \in B \to \epsilon \\ x \in A \setminus B \to x. \end{cases}$$

When $A = \{a\}$, we note $\pi_{\{a\}}$ as well as π_a.

Definition 8. A real-time system $\Gamma = \left\{(\tau_i)_{i\in[1,n]}\right\}$ is weakly feasible under context C if $\forall i \in [1, n], \exists \omega \in P_{\tau_i}$ such that $\exists \alpha \in \mathcal{L}\left(\frac{\Gamma}{C}\right)$ such that $\pi_{a_i}(\rho_i(\alpha)) = \omega$.

Thus, Theorem 1 addresses weak feasibility.

When job CPU loads are invariable, *weak feasibility* and *feasibility* are equivalent, since the only two alternatives are the existence and the non-existence of one occurrence of ω. That is why Theorem 1 is useful when addressing feasibility in this context.

On the other hand, as soon as there exists one τ_i whose successive instances follow different CPU loads, this theorem is useless, since *weak feasibility* and *feasibility* are no longer equivalent: for such systems, weak feasibility means that there are compatible behaviors. Others can nevertheless be forbidden, because of functional incompatibilities (resource sharing, for instance).

Let us consider, for instance, a job system $(\tau_i)_{i\in[1,2]}$, where τ_1 and τ_2 share a critical resource. The body of τ_1 is presented in Fig. 7.a. The structures of their respective bodies are such that, when resource sharing is used, both of the two jobs miss their deadlines. However, for each of these two jobs, there exists a possible behavior compatible with its time constraints: for τ_1, it is the set of behaviors which avoids resource sharing (see Fig. 7.b). Thus, the system is

a Body of job τ_1 **b** Valid part of the body of job τ_1
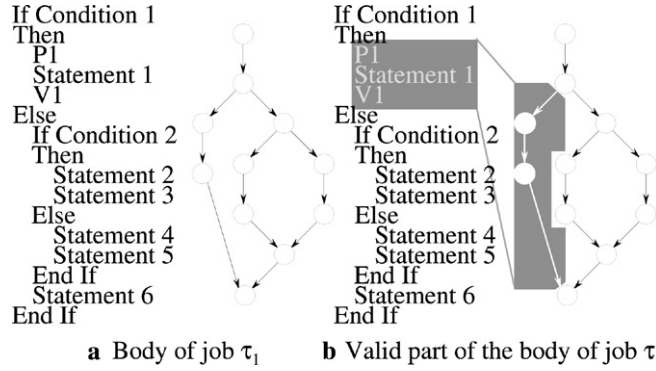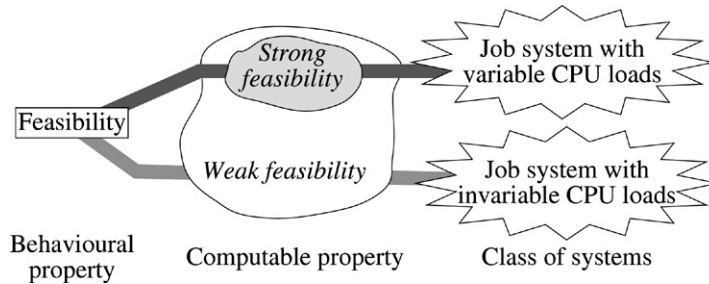
Fig. 7. Example of invalid configuration.

Fig. 8. Feasibility computation.

not feasible since some functional behaviors of jobs are excluded from systemic constraints, but it is weakly feasible: it satisfies Theorem 1!

Thus, Theorem 1 does not stand within context $D\left(\Delta\right)CCVT\left(p\right)$.

In a feasible system, every job may always be allowed to follow any of its (functional) behaviors according to both resource sharing and time constraints. This property is stronger than weak feasibility. We call it *strong feasibility*:

Definition 9. A real-time system $\Gamma = \left\{(\tau_i)_{i\in[1,n]}\right\}$ is strongly feasible under context C if and only if

$$\forall\,(\omega_i)_{i\in[1,n]} \in \prod_{i=1}^{i=n} P_{\tau_i},\, \exists\alpha \in \mathcal{L}\left(\frac{\Gamma}{C}\right) \text{ such that } \forall i \in [1,n],\, \pi_{a_i}\left(\pi_i(\alpha)\right) = \omega_i.$$

The feasibility of a system Γ is reached as soon as $\mathcal{L}\left(\frac{\Gamma}{C}\right) \neq \emptyset$ and Γ is strongly feasible.

Let us consider the following example. We consider a system of two tasks τ_1 and τ_2 whose time characteristics are the following: $r_1 = r_2 = 0$, $T = T_1 = T_2 = D_1 = D_2 = 2600$ and C_i's depend on the number $n \in [0, 255]$ of the current instance of the task in the following way: $C_1 = 10n$, $n \le 10 \Rightarrow C_2(n) = 100$ and $n > 10 \Rightarrow C_2(n) = 255 - n$. The WCET $C_{1\max} = 2550$ is reached for $n = 255$, and the WCET $C_{2\max} = 244$ for $n = 11$. For the context of this example (synchronous independent tasks with implicit deadlines), the feasibility on p processors stands on the analytic condition $\frac{C_1 + C_2}{T} \le p$. Using this criteria with a classical WCET approach for a uniprocessor feasibility decision, we get $\frac{C_{1\max} + C_{2\max}}{T} = \frac{2550 + 244}{2600} = \frac{2794}{2600} > 1$: the system is considered to be invalid. Using a WCET-free analysis consists in computing the exact values $C_1(n) + C_2(n)$ for all possible n. We can see that this value increases from 110 (for $n = 0$) to 2550 (for $n = 255$). Then we get $\forall n \in [0, 255]$, $\frac{C_1(n) + C_2(n)}{T} < 1$. The task system is valid. This example shows that examples exist that are valid, but detected as invalid by using a classical WCET analysis.

These considerations lead us to generalize Theorem 1 in the following way (see Fig. 8).

Theorem 2. *A system* $\Gamma = \{(\tau_i)\}_{i\in[1,n]}$ *is feasible under context* $D\left(\Delta\right)CC\ VT\left(p\right)$ *if and only if*
$$\begin{cases} \mathcal{L}\left(\frac{\Gamma}{D(\Delta)CCVT(p)}\right) \neq \emptyset \\ \Gamma \text{ is strongly feasible.} \end{cases}$$

Note that for invariable CPU load, both weak and strong feasibility are equivalent, and that Theorems 1 and 2 address the same property: feasibility.

If Γ satisfies Theorem 2, the set $\mathcal{L}\left(\frac{\Gamma}{D(\Delta)CCVT(p)}\right)$ of its time-valid behaviors is a regular language. For such languages, the star lemma [19] shows that every sufficiently long word contains an iterated subword.

Thus, this lemma leads to the following property: if Γ is a set of interdependent tasks, to be scheduled under the context $D(\Delta)CCVT(p)$, the time-valid behaviors of Γ are cyclic.

This consequence shows that the cyclicity problem for multiprocessor contexts [15] can be solved, hence on-line scheduling can be validated by simulation for multiprocessors.

5.3. The AMADO project: Considering variable CPU load jobs

In the UAV controller, *Navigation* is the most complex job, in terms of control statements. Its program contains some `if/then/else` constructors, and calls for some elementary computing subroutines (3 or 4 atomic statements for each one, without control statements). This program and its translation into an operational model are presented in the table presented in Fig. 9.

Translated into a set of operational behaviors, the program presented in this figure leads to

$$a^3 P(R_5) a^{13} V(R_5) \left\{a, a^2\right\} P(R_3) a^{12} V(R_3) \left\{a, a^2\right\} P(R_6) a^8 \left\{a, a^2\right\}^6 V(R_6) a.$$

Since $\left\{a, a^2\right\}^6 = \left\{a^6, a^7, a^8, a^9, a^{10}, a^{11}, a^{12}\right\}$, operational behaviors of this program correspond to 24 different CPU loads, the least loaded being a 51-time-units behavior, and the most loaded a 59-time-units.

Computing of the synchronized product automaton associated with this variable CPU time real-time system follows this sequence:

	$|A_i|$	$\dfrac{|\Omega|}{R_j\binom{|\Pi|}{|C|}}$					
τ_7	140	$\dfrac{2080}{R_6\binom{2074}{2074}}$	$\dfrac{13\,768}{R_5\binom{12\,866}{12\,678}}$	$\dfrac{32\,705}{R_3\binom{29\,382}{27\,858}}$	$\dfrac{30\,328}{R_2\binom{21\,269}{18\,770}}$	$\dfrac{21\,445}{R_1\binom{2160}{1707}}$	$\dfrac{2302}{R_4\binom{1324}{379}}$
τ_6	212						
τ_3	84			$\dfrac{}{R_4\binom{25\,327}{14\,296}}$	$\dfrac{}{R_1\binom{15\,431}{13\,904}}$		
τ_2	13						
τ_1	13						
τ_5	17						
τ_4	26						

The final product is not empty: in accordance with critical resource sharing, the system is weakly feasible for a 4-processor target, but not strongly feasible: the one and only path accepted for scheduling purposes is the shortest (51 time units).

6. Integrating hardware specifications

Let us now suppose that Γ is designed to run on a multiprocessor machine where some processors do not run at the same speed. Moreover, we suppose that each τ_i is designed to run on a fixed processor (there is no task migration, this hypothesis is realistic in the framework of real-time software). Let us consider the context

Hardware			Software		
Architecture	Clock	Communication	Structure	Synchronisation	Preemption
$p \geq 1$	*Common starts*	$\left(H^0, s\right)$	(p, r_i, C_v)	*rc*	*totp*

This context is called $D(\Delta)SCVT(p)$ (with *S* for *Common starting clocks*) in the following.

Let τ_1 and τ_2 be two tasks designed to run on processors p_1 and p_2, whose running speeds are different, i.e. whose CPU time unit are different. We note by u_i the execution time of any atomic statement on p_i (for $i \in \{1, 2\}$). Obviously, we suppose that $u_1 \neq u_2$. Then, any symbol which appears in words of $\mathcal{L}(\tau_1)$ (resp. $\mathcal{L}(\tau_2)$) is supposed to

Navigation job Program Code	Operationals
``void Navigation (Job_Traiter_GPS)
{
double Pos[3],ActDir,test1;
double NxtPos[3];
double Dir,DifDir;
GetResource(Res_InfosGPS);
Pos[Lat]=LatLngChk(GPS_Tr.Lat,GPS_Tr.LatFrac);
Pos[Lng]=LatLngChk(GPS_Tr.Lng,GPS_Tr.LngFrac);
Pos[FLlev]=GPS_Tr.AltRef/1000.0;
ActDir=GPS_Tr.COG/100;
ReleaseResource(Res_InfosGPS);
if
(
(Abs(NxtPos[Lat],Pos[Lat])<AprxLat)
&&
(Abs(NxtPos[Lng],Pos[Lng])<AprxLng)
)
PntToRchNW=(PntToRchNW+1)%NWPtTotNb;
GetResource(Res_Waypoints);
NxtPos[Lat]=LatCmp(TabWayPoint[PntToRchNW]);
NxtPos[Lng]=LngCmp(TabWayPoint[PntToRchNW]);
NxtPos[FLlev]=FLCmp(TabWayPoint[PntToRchNW]);
ReleaseResource(Res_Waypoints);
if(ActDir>180)
ActDir-=360;
GetResource(Res_ConsignesAttitude);
Dir=DirDet(Pos, NxtPos);
DifDir= Dir- ActDir;
Consigne.W0_Ref=NxtPos[AltCmp]-Pos[AltCmp];
if(Consigne.W0_Ref>VrtLmtSpeed)
Consigne.W0_Ref=VrtLmtSpeed;
if(Consigne.W0_Ref<-VrtLmtSpeed)
Consigne.W0_Ref=-VrtLmtSpeed;
if(DifDir>180) DifDir-=360;
if(DifDir<-180) DifDir+=360;
Consigne.Phi_Ref=0.2*DifDir;
if(Consigne.Phi_Ref>LmtIncl)
Consigne.Phi_Ref=LmtIncl;
if(Consigne.Phi_Ref<-LmtIncl)
Consigne.Phi_Ref=-LmtIncl;
ReleaseResource(Res_ConsignesAttitude);
TerminateTask();

}```	void Navigation ()
	 {
	 $a;$
	 $a;$
	 $a;$
	 $P(R_5);$
	 $a^4;$
	 $a^4;$
	 $a^4;$
	 $a;$
	 $V(R_5);$
	 $\{a,a^2\};$
	 $P(R_3);$
	 $a^4;$
	 $a^4;$
	 $a^4;$
	 $V(R_3);$
	 $\{a,a^2\};$
	 $P(R_6);$
	 $a^4;$
	 $a;$
	 $a^3;$
	 $\{a,a^2\};$
	 $\{a,a^2\};$
	 $\{a,a^2\};$
	 $\{a,a^2\};$
	 $\{a,a^2\};$
	 $\{a,a^2\};$
	 $V(R_6);$
	 $a;$
	 }

Fig. 9. Navigation job program vs operationals for one procedure of the AMADO project code.

be associated with the delay u_1 (resp u_2): the time semantics correspondingly associated with the language depends on the target processor. We take this into account by indexing the language with this time unit: the language which collects the behavior of τ_1 when it is running on p_1 is denoted $\mathcal{L}_{u_1}(\tau_1)$.

In previous sections, all processors were running at the same speed u, this speed was omitted in the description of the relevant languages, and the synchronized product was naturally associated with u. At present, however, we cannot compute the synchronized product of $\mathcal{L}_{u_1}(\tau_1)$ and $\mathcal{L}_{u_2}(\tau_2)$; this is due to the fact that it is impossible to associate a time unit with the edges of the product.

In order to avoid this problem, in this section we are proposing a technique allowing us to determine a unit u such that

- $\mathcal{L}_{u_1}(\tau_1)$ can be mapped into $\mathcal{L}_u(\tau_1)$,
- $\mathcal{L}_{u_2}(\tau_2)$ can be mapped into $\mathcal{L}_u(\tau_2)$.

These two languages bring us back to the previous context: the technique presented earlier in this article can be used to compute the synchronized product of $\mathcal{L}_u(\tau_1)$ and $\mathcal{L}_u(\tau_2)$, whose edges are naturally associated with the time u.

In Section 6.1, we show how u can be determined, and we give the mapping to compute $\mathcal{L}_u(\tau_1)$ from $\mathcal{L}_{u_1}(\tau_1)$.

6.1. The language of job-valid behaviors

$\mathcal{L}_{u_1}(\tau_1)$ is computed from the time characteristics of τ_1. Some of these characteristics are time-absolute values which do not depend on the characteristics of the target processor: T_i, D_i and r_i. On the other hand, the value C_i depends on the speed of the processor: we get

$$\frac{C_i}{CPU \ with \ a \ time \ unit \ u} = \frac{\frac{C_i}{k}}{CPU \ with \ a \ time \ unit \ \frac{u}{k}}.$$

Computing $\mathcal{L}_u(\tau_1)$ from $\mathcal{L}_{u_1}(\tau_1)$ presupposes our keeping the specified values of T_i, D_i and r_i, and likewise presupposes our considering for C_i the value corresponding to the time unit u.

Example 1. Let τ_i be a job of characteristics $r_i = 3$ ms, $D_i = 8$ ms, $T_i = 10$ ms and $C_i = 3u_i$. With $u_i = 1$ ms, we get

$$\mathcal{L}_{u_i}(\tau_i) = \text{Center}\left(\bullet^3\left(\left(a^3\text{III}\bullet^5\right)\bullet^2\right)^*\right).$$

With $u_i = 250$ ns, we get the (different) language

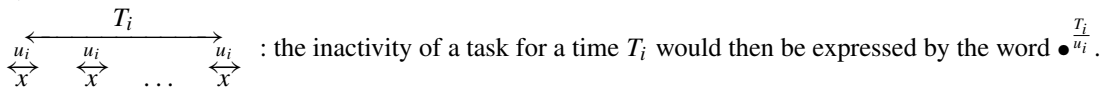$$\mathcal{L}_{u_i}(\tau) = \text{Center}\left(\bullet^{12}\left(\left(a^3\text{III}\bullet^{29}\right)\bullet^8\right)^*\right).$$

Observing Fig. 10, one may note that u_1 and u_2 must be multiples of u: the natural value for u is $gcd(u_1, u_2)$, which can always be computed as soon as u_1 and u_2 are integer values. This last property is yielded as soon as time units are expressed in terms compatible with the characteristics of the processors put to work (μs, ns, ps, etc.)).

Words belonging to $\mathcal{L}_{u_i}(\tau_i)$ integrate parameters r_i, D_i and T_i by enumerating job inactivity in terms of time units. A word which describes the behavior of a task on a time interval of duration t is composed of letters whose individual time interval is of length u_i, then t must be a multiple of u_i: $\overset{t}{\underset{a}{\overset{u_i}{\longleftrightarrow}}\ \underset{a}{\overset{u_i}{\longleftrightarrow}}\ \ldots\ \underset{a}{\overset{u_i}{\longleftrightarrow}}}$. Computing the language $\mathcal{L}_{u_i}(\tau_i)$ involves the translating of time values T_i, D_i and r_i into word lengths. For T_i, for instance, we get the figure $\overset{T_i}{\underset{x}{\overset{u_i}{\longleftrightarrow}}\ \underset{x}{\overset{u_i}{\longleftrightarrow}}\ \ldots\ \underset{x}{\overset{u_i}{\longleftrightarrow}}}$: the inactivity of a task for a time T_i would then be expressed by the word $\bullet^{\frac{T_i}{u_i}}$.

On the one hand r_i, D_i and T_i are usually greater than 20 ms; on the other hand u_i is about some μs. Since external time specifications for jobs can often be reviewed by constraining or relaxing them (these modifications must be compatible with external specifications of the captors and activators), we suppose in the following that $\frac{T_i}{u_i} \in \mathbb{N}$, $\frac{D_i}{u_i} \in \mathbb{N}$ and $\frac{r_i}{u_i} \in \mathbb{N}$.

The expression of $\mathcal{L}_{u_i}(\tau_i)$ comes from the translation into time units of the time specifications of the job (period, deadline and first activation date). Then, for the job τ_i, whose behaviors of P_{τ_i} are the $(\omega_k)_{k \in I}$ (see Section 5.1), we obtain

$$\mathcal{L}(\tau_i) = \text{Center}\left(\bullet^{\frac{r_\tau}{u_i}}\left(\left(\bigcup_{k \in I}\left(\omega_k\text{III}\bullet^{\frac{D_i - |\omega_k|}{u_i}}\right)\right)\bullet^{\frac{T_i - D_i}{u_i}}\right)^*\right).$$

We call *Validity class* of τ_i the set

$$\textcircled{L}(\tau_i) = \left\{\mathcal{L}_u(\tau_i), \text{where } u \in \mathbb{N}^*, r_i \in u\mathbb{N}, D_i \in u\mathbb{N} \text{ and } T_i \in u\mathbb{N}\right\}.$$

This set collects all languages that model operational behaviors of τ_i on all possible classes of targets, in terms of CPU speed: note that the number of languages in this class depends on the unit u as it is expressed in (μs, ns, etc.), the choice of unit being up to the user.
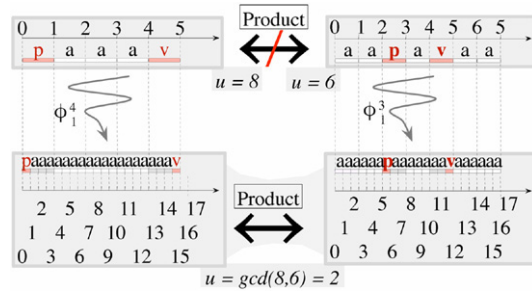
Fig. 10. CPU time units and observation time units.

To map languages $\mathcal{L}_{u_1}(\tau_1)$ and $\mathcal{L}_{u_2}(\tau_2)$ on $\mathcal{L}_u(\tau_1)$ and $\mathcal{L}_u(\tau_2)$, we use the morphism $\phi_{u_i}^u$, which is defined in the following way (see Fig. 10 for the use of this morphism):

Definition 10. Let $u \in \mathbb{N}^*$, $u_i \in u\mathbb{N}^*$, $\Sigma = \{a, \bullet\}$ be the alphabet associated with a job, and $\Sigma' = \{P, V\}$ be an alphabet associated with a resource-like constraint. The morphism $\phi_{u_i}^u$ is defined in the following way:

$$\phi_{u_i}^u : \Sigma \cup \Sigma' \to \left(\Sigma \cup \Sigma' \right)^{\frac{u_i}{u}}, \quad \begin{cases} \bullet \to \bullet^{\frac{u_i}{u}} \\ a \to a^{\frac{u_i}{u}} \\ P \to Pa^{\left(\frac{u_i}{u}-1\right)} \\ V \to a^{\left(\frac{u_i}{u}-1\right)}V. \end{cases}$$

Note that by induction, $\phi_{u_i}^u$ not only addresses task behaviors involved by a single resource, but also those involved by many resources.

We can now give the computing process for synchronized products:

(1) The input data are the system Γ (composed of the τ_i's), the execution context C, the $\mathcal{L}_{u_i}(\tau_i)$'s, and the u_i's.
(2) We compute $u = gcd_{i \in [1,n]}(u_i)$.
(3) For each τ_i, we compute the language $\mathcal{L}_u(\tau_i)$, which is equivalent to the language $\mathcal{L}_{u_i}(\tau_i)$ in the sense that both languages belong to $\mathbb{L}(\tau_i)$. Note that by construction, $\mathcal{L}_u(\tau_i)$ is unique.
(4) We compute the synchronized product language $\mathcal{L}_u\left(\frac{\Gamma}{C}\right)$, following the techniques presented in previous sections, since all languages now share the same time unit.

6.2. Feasibility decision

This synchronized product language is associated with a unique time unit, and it is a regular language. Consequently, Theorem 2 remains valid.

7. Computing techniques

The decision process is based upon both the computing of the center of the synchronized product and the evaluation of the strong feasibility (Theorem 2).

Computing synchronized products is based upon classical filter techniques [13]: the synchronized product is a subset of the homogeneous product: we construct it by starting from the initial state (it is unique), and by building accessible states, i.e. states whose incoming edges are labeled with elements of the current synchronization set (according to Arnold–Nivat's technique). To limit the size of intermediate computing automata, we use the principle *the more resources the job uses, the earlier its integration in the synchronized product*.

The *center* operation is likewise implemented through filtering techniques, which can be applied while building the automaton: one must decide, for each edge to be built, if it leads to a valid or an invalid state.

The reader can observe the details of computing using these techniques in the synchronized product trace tables for the AMADO project validation (see Sections 3.5, 4.3 and 5.3): the consequence of this technique is the size-limitation of temporary automata. Here, we are trying to lessen this limitation by avoiding the construction of useless components belonging to the product automaton.
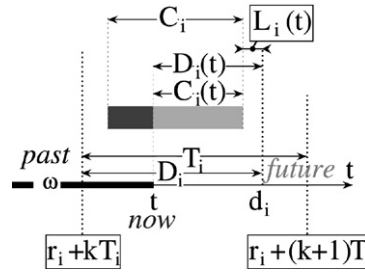
Fig. 11. Dynamic time parameters of tasks.

7.1. Improvement criterion

By studying the dynamic load of the system when in state s of the automaton, one can detect edges outgoing from s and approaching invalid components of the automaton. The goal of this section is to point out an analytic criterion to detect such edges as frequently as possible, in order to omit them in the construction of the automaton.

We deal with context $D(\Delta)CCVT(p)$. All reasonings involve the synchronized product automaton, not automata associated with individual jobs. Thus, they also stand for context $D(\Delta)SCVT(p)$, since a unique time unit is associated with the synchronized product in both frameworks.

Let us have a state s of $A_u\left(\frac{\Gamma}{D(\Delta)CCVT(p)}\right)$. We consider an edge e, labeled with x, outgoing from s. This edge is useless if for every word α which labels a path from the initial state to s, there does not exist a word β such that $\alpha x\beta \in \mathcal{L}_u\left(\frac{\Gamma}{D(\Delta)CCVT(p)}\right)$. Deciding this from minimal information is possible. Since generic automata associated with jobs only accept time-valid behaviors, time constraints need not be controlled when computing the synchronized product: the only constraints requiring consideration are resource synchronization (resp. message transmission). Then, we must define an analytical criterion, addressed when synchronizing with a resource, in order to compute the validity of each edge to build. The aim of this part is to define the criterion.

Recall that for τ_i, C_τ is computed in terms of time units. To be compared with both the D_i, T_i, and so on, it must be expressed in absolute time, by using the formula $C_i u_i$.

We note by T_Γ the value $lcm_{i\in[1,n]}(T_i)$.

We call $L_i(t)$ the dynamic laxity of task τ_i at time t: it is the number of possible idle time units for τ_i before its next deadline (see Fig. 11). The past of Γ is a behavior between times 0 and t: it is modeled by a word ω of $\mathcal{L}_u\left(\frac{\Gamma}{D(\Delta)CCVT(p)}\right)$ of length t. Thus, the laxity $L_i(t)$ can always be computed from ω. That is why in the following, we note $L_i(t)$ as well as $L_\tau(\omega)$. The same notation is also used for the dynamic CPU load $C_i(t)$ (remaining CPU load for τ_{ik} to be completed, see Fig. 11).

Let τ_{ik} be the kth instance of τ_i, presumed to be on at time t. From ω, the dynamic laxity $L_i(t)$ of τ_{ik} can be obtained in the following way. By construction, $\pi_i(\omega)$ can be broken down into $\omega_{ba}\omega_{pi_1}\ldots\omega_{pi_{k-1}}\mu$: ω_{ba} models the inactivity of τ_i before its first activation, the ω_{pi_j}'s its past completed instances, and μ the beginning of the ongoing instance τ_{ik}. We consequently obtain $\forall j \in [1, k-1]$, $\left|\omega_{pi_j}\right| = T_i$ and $|\mu| \le T_i$. There exists a set Λ of words such that $\nu \in \Lambda \Rightarrow |\mu\nu| = T_i \wedge \pi_i(\omega)\nu \in \mathcal{L}_u(\tau_i)$. $L_i(\omega)$ is obtained as $Min_{\nu\in\Lambda}(|\nu|_\bullet)$: it is the most constrained dynamic laxity of τ_{ik} induced by the scheduling sequence ω.

We note by Σ_i the alphabet associated with $\mathcal{L}_u(\tau_i)$. A word β of $\prod_{i=1}^{i=n+r}\Sigma_i$ is valid (and then must not be filtered by the criterion) if $\omega\beta \in \mathcal{L}_u\left(\frac{\Gamma}{D(\Delta)CCVT(p)}\right)$: if this property stands, the behavior $\omega\beta$ leads Γ, at time $t + |\beta|$, to reach the activation of the $k+1$th instance of τ_i according to both time and resource constraints. If not, choosing e leads to a time fault: e may be omitted in the construction of $A_u\left(\frac{\Gamma}{D(\Delta)CCT(p)}\right)$.

[16] gives such a criterion for the context $D(\Delta)CCT(1)$. It uses the order $\prec_\omega$ on Γ, which is defined in the following way: at time t (i.e. at the end of ω), $\tau_i \prec_\omega \tau_j \Leftrightarrow L_i(\omega) < L_j(\omega)$. The criterion is defined by:

$$\omega\beta \in \mathcal{L}_u\left(\frac{\Gamma}{D(\Delta)CCVT(1)}\right) \Rightarrow \forall \tau_i \in \Gamma, \sum_{\substack{\tau_j \in \Gamma \\ \tau_j \preceq_\omega \tau_i}}\left(C_j(\omega\beta)\right) \le \frac{L_i(\omega\beta)}{u}.$$
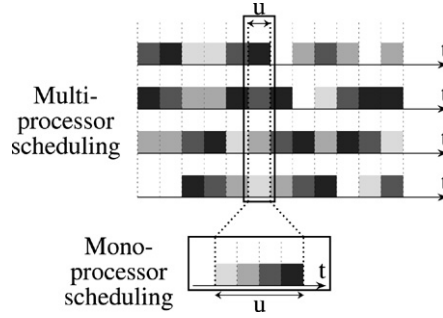
Fig. 12. Equivalent monoprocessor model.

At this point, we extend this criterion to the context $D(\Delta)\,CCVT(p)$.
We obtain:

Theorem 3.

$$\omega\beta \in \mathcal{L}_u\left(\frac{\Gamma}{D(\Delta)\,CCVT(p)}\right) \Rightarrow \forall\tau_i \in \Gamma,\ \sum_{\substack{\tau_j \in \Gamma \\ \tau_j \preceq_\omega \tau_i}} \frac{C_j(\omega\beta)}{p} \le \frac{L_i(\omega\beta)}{u}.$$

Proof. We enrich the notations we consider for context, by including time units in their descriptions: we note $D(\Delta)\,CCT(p[u])$ to indicate the (common) time unit associated with the p processors of the target system.

Let C_p be a $D(\Delta)\,CCT(p[u])$ configuration, $\omega \in \mathcal{L}_u\left(\frac{\Gamma}{D(\Delta)CCVT(p[u])}\right)$ and $\beta \in \prod_{i=1}^{i=n}\Sigma_i$, such that $\omega\beta \in \mathcal{L}_u\left(\frac{\Gamma}{D(\Delta)CCVT(p[u])}\right)$.

We consider a configuration C_1 following $D(\Delta)\,CCT\left(1\left[\frac{u}{p}\right]\right)$: its processor is p times faster than the p processors of C_p. The load capacities of C_1 and C_p are identical. There exist $\xi \in \mathcal{L}_{\frac{u}{p}}\left(\frac{\Gamma}{D(\Delta)CCVT(1)}\right)$ and $\beta' \in \prod_{i=1}^{i=n}\Sigma_i$ such that (see Fig. 12)

$$\begin{cases} |\xi| = p \times |\omega| \\ \forall i \in [1,n],\ \pi_{\neg\bullet}(\pi_i(\omega)) = \pi_{\neg\bullet}(\pi_i(\xi)) \\ |\beta'| = p \\ \forall i \in [1,n],\ \pi_{\neg\bullet}(\pi_i(\beta)) = \pi_{\neg\bullet}(\pi_i(\beta')) \\ \forall k \in [1,|\omega|]\quad \begin{cases} \pi_i(\omega_k) \neq \bullet \Leftrightarrow \pi_i(\omega_k) = \pi_i\left(\pi_{\neg\bullet}\left(\xi_{(k-1)\times p+1}\ldots\xi_{k\times p}\right)\right) \\ \pi_i(\omega_k) = \bullet \Leftrightarrow \pi_i\left(\pi_{\neg\bullet}\left(\xi_{(k-1)\times p+1}\ldots\xi_{k\times p}\right)\right) = \epsilon. \end{cases} \\ \forall i \in [1,n] \end{cases}$$

The correlation between duration and length leads $L_i(\omega\beta)$ to be equal to $L_i(\xi\beta')$, for each $\tau_i \in \Gamma$, and $\preceq_\omega$ to be equivalent to $\preceq_\xi$. Then, since $\forall\tau_i \in \Gamma,\ uC_i(\omega) = p \times \frac{u}{p}C_i(\xi)$, we obtain $\sum_{\substack{\tau_j \in \Gamma \\ \tau_j \preceq_\xi \tau_i}} \frac{u}{p}\left(C_j(\xi\beta')\right) = \sum_{\substack{\tau_j \in \Gamma \\ \tau_j \preceq_\omega \tau}}\left(\frac{uC_i(\omega\beta)}{p}\right)$.

Since $\omega\beta \in \mathcal{L}_u\left(\frac{\Gamma}{D(\Delta)CCVT(p[u])}\right)$, we have $\forall x \in \mathbb{N}, \exists\alpha \in \left(\prod_{i=1}^{i=n}\Sigma_i\right)^x\left(\prod_{i=1}^{i=n}\Sigma_i\right)^*$ such that $\omega\beta\alpha \in \mathcal{L}_u\left(\frac{\Gamma}{D(\Delta)CCVT(p[u])}\right)$. By a construction such as that of ξ and β', we can build a word α' of length x such that $\xi\beta'\alpha' \in \mathcal{L}_{\frac{u}{p}}\left(\frac{\Gamma}{D(\Delta)CCT\left(1\left[\frac{u}{p}\right]\right)}\right)$. As a result, $\xi\beta'$ belongs to the center of the monoprocessor language, and the monoprocessor criterion stands. Then, we obtain

$$\forall\tau_i \in \Gamma,\ \sum_{\substack{\tau_j \in \Gamma \\ \tau_j \preceq_\xi \tau_i}}\left(\frac{u}{p}C_j(\xi\beta')\right) \le L_i(\xi\beta')$$
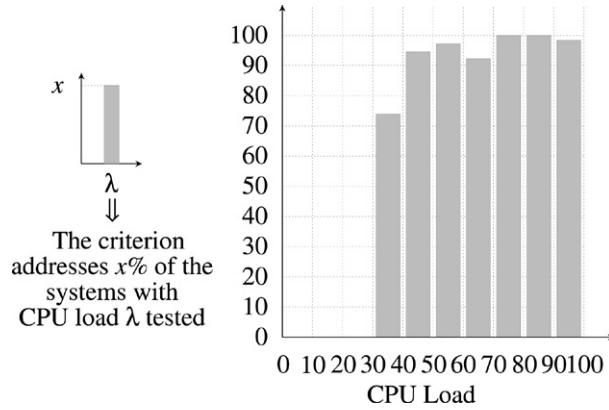
Fig. 13. Criterion base.

Since $\sum_{\substack{\tau_j \in \Gamma \\ \tau_j \preceq_\xi \tau_i}} \frac{u}{p} \left(C_j(\xi \beta') \right) = \sum_{\substack{\tau_j \in \Gamma \\ \tau_j \preceq_\omega \tau}} \left(\frac{u C_j(\omega \beta)}{p} \right)$, we obtain

$$\forall \tau_i \in \Gamma, \sum_{\substack{\tau_j \in \Gamma \\ \tau_j \preceq_\omega \tau_i}} \left(u \frac{C_j(\omega \beta)}{p} \right) \leq L_i(\omega \beta)$$

and then

$$\forall \tau_i \in \Gamma, \sum_{\substack{\tau_j \in \Gamma \\ \tau_j \preceq_\omega \tau_i}} \left(\frac{C_j(\omega \beta)}{p} \right) \leq \frac{L_i(\omega \beta)}{u}. \quad \square$$

7.2. Experimental analysis

Let us now evaluate the improvement brought about by this multiprocessor branching-bound criterion. This evaluation is performed by applying the operational validity decision process with and without the use of the criterion on a random sample ξ generated thanks to a real-time configuration random generator, which is designed to produce configurations to implement on multiprocessor targets. The generator we have used was proposed by Goossens and Macq in [21]. Since operational validation is connected with CPU loads, this sample is analysed by CPU load sections.

For a real-time system $\Gamma = (\tau_i)_{i \in [1,n]}$ designed to run under the context $D(\Delta) CCT(p)$, the CPU load is $\frac{\sum_{i=1}^{i=n} \frac{C_i}{T_i}}{p}$. For each CPU load $\lambda \in [0, 1]$, we call ξ_λ the subset of ξ composed of configurations whose CPU load is λ. We note $\xi_{[\lambda_1, \lambda_2]}$ the set $\bigcup_{\lambda \in [\lambda_1, \lambda_2]} \xi_\lambda$. We do not deal with configurations with CPU loads of less than 0.3, because they are of negligible significance in a real-time context.

The random sample ξ has been generated from a CPU load uniform random law on the interval $\left[\frac{3}{10}, 1 \right]$. Then, ξ is partitioned in $\left\{ \left(\xi_{\left[\frac{i-1}{10}, \frac{i}{10} \right]} \right)_{i \in [4,10]} \right\}$, and each $\xi_{\left[\frac{i-1}{10}, \frac{i}{10} \right]}$ contains around 200 configurations. On the bar graphs presented in Figs. 13–15, each presented value corresponds to the average value for the corresponding sample $\left\{ \left(\xi_{\left[\frac{i-1}{10}, \frac{i}{10} \right]} \right)_{i \in [4,10]} \right\}$.

We evaluate the criterion firstly on its base aspect (is it often useful?), and secondly on its improvement aspect (does it bring about a real improvement, in both space and time complexity aspects?).

We get the following results:

- Firstly, we evaluate the criterion base (see Fig. 13): for each sample ζ of $\left\{ \left(\xi_{\left[\frac{i-1}{10}, \frac{i}{10} \right]} \right)_{i \in [4,10]} \right\}$, we compute the operational validity decision for all configurations of ζ. While computing, we enumerate the p configurations
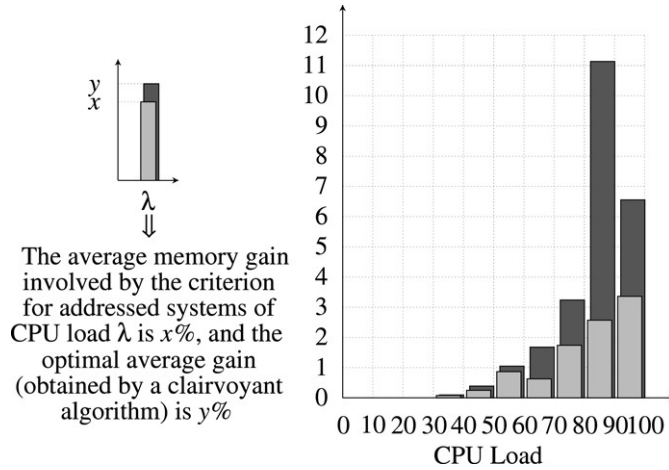
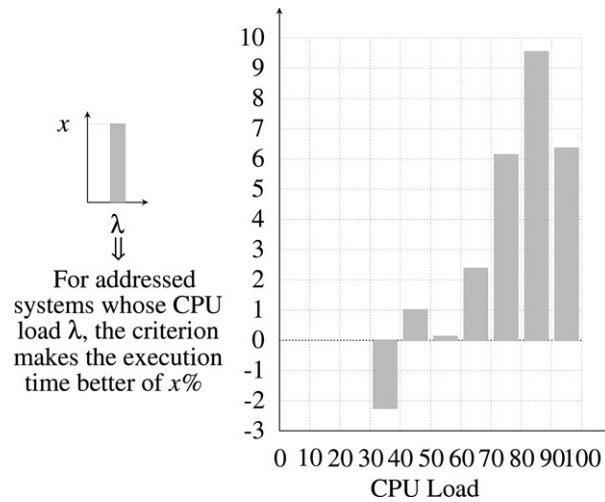Fig. 14. Memory gain involved by the criterion.

Fig. 15. Time gain involved by the criterion.

whose operational validity decision computing is improved by the use of the criterion: the value $x\%$ presented on the figure is $\frac{p}{|\zeta|}$.

Observing Fig. 13 shows that the criterion is nearly always useful for real-time addressed configurations (CPU loads over 70%).

- Secondly, the main problem of model checking techniques in such studies is space explosion. Such criteria are usually followed in order to improve both space and time complexities. In Fig. 14, we evaluate space gain.

This gain is computed from the space used by the operational validity decision algorithm: computing automata for each job, making products and intersections, computing the center language accepting automaton, evaluating the emptiness of this center language by analysing this automaton. For $C \in \zeta$, let us call $bspace(C)$ the space complexity (i.e. the larger instantaneous amount of memory needed by the decisional algorithm) for obtaining the validity decision following this technique. Now, let us call $cspace(C)$ the space complexity of the algorithm when it is upgraded with the criterion, and let us call $ospace(C)$ the space complexity of a clairvoyant algorithm, which built only the edges that remain in the final automaton.

In Fig. 14, we present the average values, for the ten samples $\xi_{\left[\frac{i-1}{10}, \frac{i}{10}\right]}$ previously described, of both $1 - \frac{cspace(C)}{space(C)}$ (they compose the bright histogram) and $1 - \frac{ospace(C)}{space(C)}$ (they compose the dark histogram). These two histograms provide a comparison between the criterion and the optimal possible performance.

We can see that the clairvoyant algorithm never deletes more than about 10% of built edges. In Section 3, we have designed our model in order to collect only valid scheduling sequences. By only improving our basic space complexity about 10% for the worst case section, and by about 5% on the others, the clairvoyant algorithm both validates our approach and yields improvements that would otherwise be hard to obtain. However, we see in Fig. 14 that our criterion improves space complexity about 3% (for addressed configurations), that is to say about 50% of the clairvoyant algorithm capacities (the optimum). However, for Section [80%, 90%], the criterion improves only about 25% with regard to the optimum. This case can be explained in the following way: such configurations are not loaded enough to often be invalid, but are loaded enough to lead to deadline missing generated by markedly earlier scheduling decisions. Such time properties lead the center automaton computing to delete a great part of the graph: this is precisely the scope of improvement of the criterion, so it is natural that having to process a larger part of the graph than in other sections, it loses efficiency against the clairvoyant algorithm.

- Thirdly, evaluating the criterion uses processor time in addition to basic algorithm processor time. Here, we cannot consider the clairvoyant algorithm as a reference, since such an algorithm does not exist (because of the complexity class of the problem). That is why, on Fig. 15, we evaluate only the time gain involved by use of the algorithm: in many cases, the additional CPU time involved by the criterion is made up for by the edge gain obtained: the only section to be in deficit is [30%, 40%]. This deficit stands because, for such systems, the basic algorithm is very efficient (see Fig. 14), and then the additional expense involved by computing the criterion for each edge is not made up for by avoiding edges. For addressed sections, the time-gain is over 6%, and about 10% for the section [80%, 90%]. Recall that space improvement of the criterion for this section is only about 2.5% (see Fig. 14), but these 2.5% are connected with a time gain about 10%: the best of all sections.

So, these experimentations show that our model achieves its objectives, since it built only a few useless edges, and that the criterion is definitely useful for configurations addressed in real-time conception processes.

8. Conclusion

In this paper, we have used algebraic compositions of regular languages to define a model to decide upon the operational validity of periodic real-time systems. This model is valid for periodic real-time systems with offsets, where jobs can communicate or share critical resources, and it takes hardware specifications into account (e.g. processors of different speeds).

Experiments have shown that this model is quite efficient: the validity decision process built very few useless edges, and the branching-bound criterion reduces decision execution times by 5%–10%. Moreover, the AMADO project study shows that this approach is useful for Real-Time Conception Aid Design. The classes of properties that can be evaluated are currently being studied.

This work yields many contributions pertaining to the real-time operational validation problem:

- Reducing job CPU loads to their worst case is known to lead to acceptance of invalid configurations. We have provided a novel methodology which avoids worst case analysis in the operational validation process.
- This methodology applies to monoprocessor target architectures, but (and this is new) also to multiprocessors.
- Since we represent the set of valid sequences with regular languages, the cyclicity theorem [28,16] remains valid for multiprocessor scheduling.

The second step of this research consists in defining and experimenting measure indicators useful in software quality assessment. We are presently working in two directions:

- we are characterizing the classes of indicators we can compute on the basis of our model: from a technical point, we are comparing the respective performances of enumeration-based indicators [36] and of probability-based indicators [33,31] for their computing time as well as for their expressivity powers;
- we are characterizing, in the automaton, the paths corresponding to RM scheduling sequences (resp. DM, EDF, LF): measure indicators will be useful in determining how a real-time system needs to be modified so as to be rendered valid for a specific on-line policy.

Moreover, our results encourage us to further the approach we have studied to more general systems: we have extended the scope of this methodology to real-time systems composed of both periodic and sporadic jobs [20]: a sporadic job

is associated with an alarm signal, which is obviously not periodic. At another location, observation of the geometric properties of the automata's graphs has prompted us to adopt a new approach toward validating real-time, and it appears to be highly efficient [27]. At some future time, we are planning to apply these results so as to provide assistance for real-time conceivers in the operational specification process.

Acknowledgements

Appendix. Notations and definitions used in this paper

Mathematical notations

$\prod_{i \in I} E_i$	Cartesian product of all sets E_i such that $i \in I$
$\bigcup_{i \in I} E_i$	Union of all sets E_i such that $i \in I$
$\bigcap_{i \in I} E_i$	Intersection of all sets E_i such that $i \in I$
$A \setminus B$	$\{x \in A \cup B \text{ such that } x \in A \wedge x \notin B\}$
$\pi_B \in B^A$	Function such that $x \in B \to \pi_B(x) = x$ and $x \notin B \to \pi_B(x) = \epsilon$
$\pi_{\neg B} \in B^A$	Function such that $x \in B \to \pi_{\neg B}(x) = \epsilon$ and $x \notin B \to \pi_{\neg B}(x) = x$

Words and languages

$|\omega|$	Length (number of characters) of word ω
$|\omega|_x$	Number of occurrences of pattern x in the word ω
$\text{Reg}(\Sigma)$	Set of regular languages on alphabet Σ
$\text{Pref}(\omega)$	Set of prefixes of ω (can be a word or a language)
$\text{Center}(L)$	Center of the language L
a^x	The pattern a repeated x times
$\Omega_{i \in I} L_i$	Homogeneous product of all languages L_i such that $i \in I$
$\amalg$	Shuffle of languages

Real-time tasks

Γ	the currently considered real-time system
τ_i	The ith task of a real-time system
τ_{ij}	The jth instance of task τ_i
r_i	First activation date of task τ_i
D_i	Critical delay of task τ_i
C_i	CPU load of task τ_i
T_i	Period of task τ_i
$L_i(t)$	Dynamic laxity of task τ_i
$C_i(t)$	Dynamic CPU load of task τ_i

Scheduling contexts

$ICCT(p)$	- Independent tasks - Common clock - Centralized system	- Periodic real-time system - p processors
$D(\Delta)CCT(p)$	- Dependent tasks with constraints Δ - Common clock - Centralized system	- p processors - Periodic real-time system
EDF	Earliest Deadline First scheduling policy	
LL	Least Laxity First scheduling policy	
RM	Rate Monotonic scheduling policy	
DM	Deadline Monotonic scheduling policy	

Model

a	The task is running for one observation time unit
$\bullet$	The task is suspended for one observation time unit
$\mathcal{L}\left(\frac{\Gamma}{C}\right)$	Model language for system Γ under context C
$A\left(\frac{\Gamma}{C}\right)$	Acceptation finite automaton for $\mathcal{L}\left(\frac{\Gamma}{C}\right)$
$\mathcal{L}_u\left(\frac{\Gamma}{C}\right)$	Model language for system Γ under context C and observation time unit u
$A_u\left(\frac{\Gamma}{C}\right)$	Accept finite automaton for $\mathcal{L}_u\left(\frac{\Gamma}{C}\right)$
S_R	Arnold–Nivat synchronization set for resource R management protocol
ϕ_n^p	Transformation function which replaces a by $a^{\frac{n}{p}}$, P by $Pa^{\left(\frac{n}{p}-1\right)}$ and V by $a^{\left(\frac{n}{p}-1\right)}V$. p must be element of $n\mathbb{N}$

References

[1] DGA (French Arms Procurement Agency), ONERA (French aeronautics, and space research centre), International universities mini uav competition. Closed in September, 2005. http://concours-drones.onera.fr/.

[2] K. Altisen, A. Clodic, F. Maraninchi, E. Rutten, Using controller-synthesis techniques to build property-enforcing layers, in: Proc. of 12th European Symposium on Programming, in: Lecture Notes in Computer Science, vol. 2618, Springer Verlag, 2003, pp. 174–188.

[3] R. Alur, D. Dill, A theory of timed automata, Theoretical Computer Science 126 (1994) 183–235.

[4] R. Alur, D.L. Dill, Automata for modeling real-time systems, in: Proceedings of the 17th International Colloquium on Automata, Languages and Programming, in: Lecture Notes in Computer Science, vol. 443, Springer-Verlag, London, UK, 1990, pp. 322–335.

[5] T. Amnell, E. Fersman, L. Mokrushin, P. Pettersson, W. Yi, Times: A tool for schedulability analysis and code generation of real-time systems, in: Proc. of 1st International Workshop on Formal Modeling and Analysis of Timed Systems, in: Lecture Notes in Computer Science, vol. 2791, 2003, pp. 60–72.

[6] A. Arnold, Finite Transition Systems, Prentice-Hall, 1994.

[7] A. Arnold, A. Griffault, G. Point, A. Rauzy, The altarica formalism for describing concurrent systems, Fundamenta Informaticæ 40 (2000) 109–124.

[8] T.P. Baker, Stack-based scheduling of real-time processes, The Journal of Real-Time Systems 3 (1991) 67–99.

[9] S.K. Baruah, Scheduling periodic tasks on uniform multiprocessors, Information Processing Letters 80 (2) (2001) 97–104.

[10] S.K. Baruah, A. Burns, Sustainable scheduling analysis, in: Proc. of the 27th International Real-Time Systems Symposium, IEEE Computer Society, Rio de Janeiro, Brazil, 2006, pp. 159–168.

[11] S.K. Baruah, N. Fisher, The feasibility analysis of multiprocessor real-time systems, in: Proc. of the 18th Euromicro Conference on Real-Time Systems, IEEE Computer Society, Dresden, Germany, 2006, pp. 85–96.

[12] J.P. Beauvais, Étude d'Algorithmes de Placement de Tâches Temps Réel Périodiques Complexes dans un Système Réparti, Ph.D. Thesis, École Centrale de Nantes, France, 1996.

[13] P. Bozanis, N. Kitsios, C. Makris, A.K. Tsakalidis, New upper bounds for generalized intersection searching problems, in: Proc. of 22nd International Colloquium on Automata, Languages and Programming, 1995, pp. 464–474.

[14] P. Brémond-Grégoire, J. Choi, I. Lee, A complete axiomatization of finite-state acsr processes, Information and Computation 138 (2) (1997) 124–159.

[15] A. Choquet-Geniet, Un premier pas vers l'étude de la cyclicité en environnement multi-processeur, in: Proc. of Real-Time Systems 2005, Teknea, 2005, pp. 289–302.

[16] A. Choquet-Geniet, E. Grolleau, Minimal schedulability interval for real-time systems of periodic tasks with offsets, Theoretical Computer Science 310 (2004) 117–134.

[17] A. Colin, I. Puaut, Worst case execution time analysis for a processor with branch prediction (Worst-case execution time analysis), Real-Time Systems 18 (2) (2000) 249–274 (special issue).

[18] M.L. Dertouzos, A.K. Mok, Multiprocessor on-line scheduling of hard-real-time tasks, IEEE Transactions on Software Engineering 15 (12) (1989) 1497–1506.

[19] S. Eilenberg, Automata Languages and Machines, vol. A, Academic Press, 1976.

[20] D. Geniet, J.P. Dubernard, Scheduling hard sporadic tasks with regular languages and generating functions, Theoretical Computer Science 313 (2004) 119–132.

[21] J. Goossens, C. Macq, Limitation of the hyper-period in real-time periodic task set generation, in: Proc. of Real-Time Systems 2001, Teknea, 2001, pp. 133–148.

[22] R.L. Graham, E.W. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, Optimization and approximation in deterministic sequencing and scheduling: A survey, Annals of Discrete Mathematics 5 (1979) 287–326.

[23] E. Grolleau, Ordonnancement Temps-Réel Hors-Ligne Optimal à l'Aide de Réseaux de Petri en Environnement Monoprocesseur et Multiprocesseur, Ph.D. Thesis, Univ. Poitiers, 1999.

[24] E. Grolleau, A. Choquet-Geniet, Off-line computation of real-time schedules by means of petri nets, Journal of Discrete Event Dynamic Systems 12 (2002) 311–333.

[25] J. Carpenter, S. Funk, P. Holman, A. Srinivasan, J. Anderson, S. Baruah, A Categorization of Real-Time Multiprocessor Scheduling Problems and Algorithms, in: Handbook of Scheduling: Algorithms, Models, and Performance Analysis, Chapman and Hall/CRC, 2004, pp. 30-1–30-19 (Chapter).

[26] P. Krčál, W. Yi, Decidable and undecidable problems in schedulability analysis using timed automata, in: K. Jensen, A. Podelski (Eds.), Proc. of the 10th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, in: Lecture Notes in Computer Science, vol. 2988, Springer-Verlag, 2004, pp. 236–250.

[27] G. Largeteau, D. Geniet, É. Andres, Discrete geometry applied in hard real-time systems validation, in: Proc. of 12th Discrete Geometry for Computer Imagery, in: Lecture Notes in Computer Science, vol. 3429, Springer-Verlag, 2005, pp. 23–33.

[28] J.Y.T. Leung, M.L. Merill, A note on preemptive scheduling of periodic real-time tasks, Information Processing Letters 11 (3) (1980) 115–118.

[29] C.L. Liu, J.W. Layland, Scheduling algorithms for multiprogramming in a hard real-time environment, Journal of the ACM 20 (1) (1973) 46–61.

[30] A.K. Mok, Fundamental design problems for the hard real-time environments, Ph.D. Thesis, MIT, 1983.

[31] A.M. Odlyzko, Enumeration of strings, in: A. Apostolico, Z. Galil (Eds.), Combinatorial Algorithms on Words, in: NATO Advance Science Institute Series. Series F: Computer and Systems Sciences, vol. 12, Springer-Verlag, 1985, pp. 205–228.

[32] S. Pailler, A. Choquet-Geniet, Off-line scheduling of real-time applications with variable duration tasks, in: Proc. of 7th Workshop on Discrete Event Systems, 2004, pp. 373–378.

[33] A. Paz, Introduction to Probabilistic Automata, Academic Press, 1971.
[34] L. Sha, R. Rajkumar, J. Lehockzy, Priority inheritance protocols: An approach to real-time synchronisation, IEEE Transaction Computers 39 (9) (1990).
[35] I. Sommerville, Software Engineering, Addison-Wesley, 2004.
[36] L. Thimonier, Generating functions and random words, Thèse d'état, Univ. Paris 11, October 1988.
[37] J. Xu, D.L. Parnas, Scheduling processes with release times, deadlines, precedence and exclusion relations, IEEE Transactions on Software Engineering 16 (3) (1990) 360–369.