Validation of the actual behaviour of a real-time application

Moustapha Bikienga

University of Koudougou,
BP 376, Koudougou, Burkina-Faso, France
Email: bmoustaph@yahoo.fr

Annie Choquet-Geniet* and Dominique Geniet

University of Poitiers and ENSMA,
BP 40109, 86961 Futuroscope Chasseneuil Cedex, France
Email: annie.geniet@univ-poitiers.fr
Email: dominique.geniet@univ-poitiers.fr
*Corresponding author

Abstract: Classically, the temporal validation of a hard real-time application is performed using the WCET's, for instance by means of simulation. But since the actual behaviour of the application uses the ACET's which may be shorter than the WCET's, the simulated and the effective behaviours may be different. Verifying whether an application behaves in accordance to a given scheduling strategy requires to precisely specify how the application is expected to behave in the case of ACET's shorter than the WCET's. For that aim, we define two notions of compliance. The non-flexible compliance imposes the strict respect of the start times given by the simulated schedule, whereas the flexible one tolerates a higher level of conservatism. We then discuss the advantages of each, and show that using any of them preserves the validity of the behaviour of the application.

Keywords: real-time applications; scheduling; actual behaviour; validation; compliance; worst case execution time; WCET; actual execution time; ACET.

Reference to this paper should be made as follows: Bikienga, M., Choquet-Geniet, A., Geniet, D. (2017) 'Validation of the actual behaviour of a real-time application', *Int. J. Critical Computer-Based Systems*, Vol. 7, No. 2, pp.190–208.

Biographical notes: Moustapha Bikienga completed his PhD in 2013 at ENSMA, in Poitiers, France, on real time scheduling issues. He worked under the supervision of Annie Choquet-Geniet and Dominique Geniet. He is now Assistant Professor at the University of Koudougou in Burkina-Faso, where he teaches computing.

Annie Choquet-Geniet is a Full Professor since 2007 at the University of Poitiers in France. She works in the Laboratory of Computer Science and Automatic Control for Systems. Her research interests concern real-time systems, real-time scheduling, real-time modelling and Petri nets.

Dominique Geniet is an Associate Professor since 2005 at the University of Poitiers in France. He works in the Laboratory of Computer Science and Automatic Control for Systems. His research interests concern real-time systems, real-time modelling and quality assessment in real-time systems.

1 Introduction

Widely used to control critical physical processes, the real-time applications are generally designed as sets of tasks, where a task implements a given function, and is subject to temporal constraints, due to the dynamic of the controlled process. The temporal constraints express e.g., the required frequency of a control activity, the validity duration of information, or the maximum delay between a given event and the resulting action. If some tasks are processed behind or ahead of time, some severe malfunctions may occur, thus a key point requirement for such applications is timeliness. An application can be considered as reliable only if it can be proved that all the temporal constraints are met. This in turn relies on the choice of an appropriate scheduling strategy, and on its implementation within the target system. Two approaches can be considered: the online method where a scheduling policy is implemented within the scheduler and the offline method: a valid schedule is computed before run-time on the basis of the worst case execution times (WCET). Then this schedule is stored, generally in a table, and the dispatcher must follow it. For applications composed of independent tasks, there exist efficient online algorithms. For instance, EDF (Liu and Layland, 1973) is an optimal strategy for independent task sets, where a scheduling algorithm (or strategy) is said to be optimal if either it produces a feasible (or valid) schedule, i.e., a schedule such that all temporal constraints are met, or there exist no feasible schedules at all. But if we assume that the tasks may be subject to precedence constraints and/or use critical resources, which must be used in mutual exclusion, it has been proved that there exist no optimal online strategies (Dertouzos and Mok, 1989; Hong and Leung, 1992). Furthermore, if critical resources are used, the scheduling problem is NP-hard (Andersson and Jonsson, 2000; Leung and Whitehead, 1982). This motivates the choice of the offline method. These approaches are more powerful than online strategies since they are clairvoyant: the scheduling decisions are made according to the global knowledge of the application while they are based on the instantaneous state of the application for online strategies. Another benefit of offline methods is that additional qualitative criteria can be considered during the computation of the schedule. There exist numerous offline scheduling methods, based on linear programming (Graham et al., 1979; Martel, 1982), branch-and-bound techniques (Xu and Parnas, 1990), or model-based approaches (Grolleau and Choquet-Geniet, 2002; Geniet and Largeteau-Skapin, 2007; Chauviere and Geniet, 2007; Choquet-Geniet and Largeteau-Skapin, 2014). Clairvoyant approaches like EDL (Chetto and Chetto, 1989) also fall into offline strategies. Then the temporal validation of the application either comes from conditions on the temporal parameters, or is deduced from the simulation of the behaviour of the application. But whatever the way it is done, the validation is performed offline before run-time. Now, generally, there exists a gap between the simulated behaviour and the actual one. This gap comes from the fact that the simulation is performed using the WCET's (upper bounds of the durations of the instances of the tasks), whereas the actual behaviour uses the actual execution time (ACET) (actual durations). To fill this gap, an important property for the scheduling algorithms is the property of predictability (Cucu-Grosjean and Goossens, 2010), which guarantees that the temporal constraints are still met, even if the ACET's are shorter than the WCET's. This is the case for instance for the algorithm EDF. But for non-independent task sets, if some duration is shorter than expected, some scheduling anomalies may occur (Buttazzo, 1997). Thus an important issue is to precisely specify how the application is expected to behave when some ACET's are different from the

WCET's. We thus have an expected theoretical behaviour, obtained by simulation using the WCET, which has been temporally validated, and an effective behaviour. Both are generally different. Our main objective is to answer the following question: 'when is the effective behaviour acceptable with regards to a given scheduling policy, and when does it highlight some errors in the behaviour of the system?'. In the case of an offline scheduling, the question is to determine what it means that the execution of the application follows the pre-computed schedule. The aim of this paper is to answer these questions, so that the behaviour of the application can be verified, according to a given online or offline scheduling policy. Besides, answering these questions for offline strategies will also provide some clues for their implementation.

By our best knowledge, this issue has not yet been considered. In fact, online strategies are generally required to be predictable, what guarantees the validity of the actual behaviour provided the simulated behaviour (based on the WCET's) was valid. But as far as we know, no results exist if the strategy cannot be proved predictable. And the problem has not been considered for offline strategies.

To solve this issue, we can adopt two different points of view. Both impose to respect the topology of the schedule, which means that task inversions are forbidden, which in turn implies the respect of the precedence constraints and of the mutual exclusions. Then in the first approach, we also impose the strict respect of the start times planned in the theoretical schedule. This approach is thus suited for offline scheduling methods, but does not fit with online scheduling, which is often conservative. In the second approach, we mix the respect of the topology with a conservative behaviour: if the next planned task is pending, it may start execution as soon as the previous task completes, whatever the planned start time. This approach can fit as well with online as with offline strategies. In both cases, the respect of the temporal constraints will be ensured, provided the theoretical schedule was temporally correct. And it will also guarantee the predictability of the behaviour. An approach closely related to ours can be found in Schranzhofer et al. (2009). But the authors consider resources in general; they do not focus on CPU, as we do. Besides, they stay at a modelling level, they do not explicitly consider effectiveness in the sense that they do not consider ACET's, they only deal with WCET's. Our approach can then be viewed as a continuation of theirs, where validity is considered regarding the ACET's.

The paper is organised as follows. In Section 2, we present our general context, and introduce our assumptions and notations. In Section 3, we introduce the notion of topology of a schedule, which expresses the succession of tasks and supports the precedence relations and the critical section rules. Then in Section 4, we propose the non-flexible compliance, first for non-pre-emptive schedules, then for pre-emptive ones. Finally, in Section 4, we present the flexible compliance, again for non-pre-emptive and then for pre-emptive schedules. In each case, we prove the preservation of the validity of the schedule. The paper ends with some concluding remarks and perspectives.

2 The context

2.1 Real-time applications

We consider a periodic real-time application, which is dedicated to the control of a physical process. It consists of a set of functions which must run periodically: $\mathcal{F} = \{f_1, f_2, \ldots, f_n\}$. To express the requirements induced by the controlled process, a temporal model is associated with the functional model.

A task τ_i is defined as a function f_i, associated with a four-tuple $< r_i, C_i, D_i, T_i >$ $(\tau_i = (f_i(r_i, C_i, D_i, T_i)))$ where r_i is the first release time of the task, C_i its WCET, which is an upper bound of the computation time of the function f_i; D_i its relative deadline, which is the maximum acceptable delay between the release and the completion of any instance of the task; and T_i its period.

The hyperperiod H of the application is defined as the least commun multiple (LCM) of the periods: $H = \text{LCM}(T_1, \ldots, T_n)$.

For any $j \geq 1$, f_i^j denotes the jth instance of the function f_i. The associated sub-task τ_i^j is characterised by $< r_i^j, C_i^j, d_i^j >$ where $r_i^j = r_i + (j-1) \times T_i$ is its release time, C_i^j its ACET, which satisfies $C_i^j \leq C_i$, and $d_i^j = r_i^j + D_i$ its absolute deadline. We denote *Instance*($\mathcal{F}$) the set of the instances of the functions in $\mathcal{F}$.

If we consider interacting tasks, the model of the application also contains the precedence relations and the mutual exclusions. In a classical way, we assume that precedence related tasks have the same period. Furthermore, we adopt the model proposed by Xu and Parnas (1990), Cottet and Babau (1994) and Grolleau and Choquet-Geniet (2002). The initial functions are decomposed into a set of sub-functions, which are related by a precedence relation $\prec$: $f_{i,j} \prec f_{k,l}$ means that the execution of any instance of the sub-function $f_{i,j}$ must precede the execution of the associated instance of the sub-function $f_{k,l}$; and by a mutual exclusion relation $\lozenge$: $f_{i,j} \lozenge f_{k,l}$ means that the sub-functions $f_{i,j}$ and $f_{k,l}$ must be processed in mutual exclusion. These relations are considered during the scheduling step.

In the rest of the paper, we assume that the decomposition of the functions has already been done. Thus $\mathcal{F}$ is a set of the resulting (sub) functions, and the relations $\prec$ and $\lozenge$ are defined on $\mathcal{F}$.

2.2 Real-time scheduling

We consider a monoprocessor system. We suppose that A is an application composed of n tasks, associated with a set of functions $\mathcal{F}$, $\prec$ is the precedence relation and $\lozenge$ the mutual exclusion relation.

Scheduling the application consists in defining a function $S: \mathbb{N} \rightarrow$ *Instance*($\mathcal{F}$) which is interpreted in the following way: $S(t) = f_p^q$ means that f_p^q is processed during the time interval $[t, t + 1)$.

The associated schedule is defined as a sequence of three-tuples called blocks: $Sch = (start(i), end(i), f_{\alpha(i)}^{\beta(i)})_{i \in \mathbb{N}^*}$, where $start(i)$ and $end(i)$ belong to $\mathbb{N}$, and verify $start(i) < end(i)$. Furthermore, $\alpha(i) \in \{1, \ldots, n\}$ and $\beta(i) \in \mathbb{N}^*$. A block ($start(i)$, $end(i)$,

$f_{\alpha(i)}^{\beta(i)}$) represents the execution of the instance $f_{\alpha(i)}^{\beta(i)}$ during the time interval [*start*(*i*), *end*(*i*)).

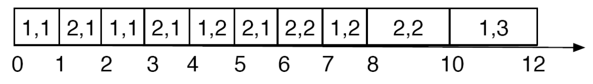
We denote $d(i)$ the length of a block: $d(i) = end(i) - start(i)$.

Let f_p^q be an instance of a function. $Be(f_p^q)$ denotes the time by which the instance starts execution, and $Cp(f_p^q)$ the time by which it completes execution.

Finally, for any integers a and b ($a < b$), $Sch(a, b)$ denotes the restriction of the schedule *Sch* on the time interval $[a, b)$.

Example. We consider an application composed of two tasks $A_1 =< f_1(0, 2, 4, 4), f_2(0, 3, 6, 6) >$. The schedule $Sch_1(0, H) = ((0, 1, f_1^1), (1, 2, f_2^1), (2, 3, f_1^1), (3, 4, f_2^1), (4, 5, f_1^2), (5, 6, f_2^1), (6, 7, f_2^2), (7, 8, f_1^2), (8, 9, f_2^2), (10, 12, f_1^3))$, is a valid (pre-emptive) schedule (see Figure 1). And for instance, we have $Be(f_1^1) = 0$, $Cp(f_1^1) = 3$, $Be(f_1^2) = 4$, $Cp(f_1^2) = 8$, etc.

Figure 1 The schedule $Sch_1(0, H)$



Next, we can formally characterise the temporal validity of a schedule. Let *Sch* be a schedule: $Sch = ((start(i), end(i), f_{\alpha(i)}^{\beta(i)}), i \in \mathbb{N}^*)$, computed using the WCET.

Property 1: the simulated schedule *Sch* is temporally valid if and only if the following properties are met:

$$
\begin{cases}
(0)\ \forall f_p^q \in Instance(\mathcal{F}), \quad \text{there exists at least one block associated with } f_p^q \\
(1)\ \forall i \geq 1, r_{\alpha(i)}^{\beta(i)} \leq start(i) \\
(2)\ \forall i \geq 1, d_{\alpha(i)}^{\beta(i)} \geq end(i) \\
(3)\ \forall i > 1, end(i-1) \leq start(i) \\
(4)\ \forall i_o \in \{1,\ldots,n\}, j_0 \geq 1, \quad \sum\limits_{i \text{ s.t. } \alpha(i)=i_0,\, \beta(i)=j_0} d(i) = C_{i_0} \\
\forall p, q \in \{1,\ldots,n\}, \\
(5)\ f_p \prec f_q, \Rightarrow \forall k, Cp\left(f_p^k\right) \leq Be\left(f_q^k\right) \\
(6)\ f_p \lozenge f_j, \forall k', \left[Be\left(f_p^k\right), Cp\left(f_p^k\right)\right) \cap \left[Be\left(f_p^{k'}\right), Cp\left(f_p^{k'}\right)\right) = \emptyset
\end{cases}
$$

Note that if we consider a non-pre-emptive schedule, for each pair (i_0, j_0), there exists only one integer i such that $\alpha(i) = i_0$ and $\beta(i) = j_0$, i.e., there is only one block per instance.

Point (0) guarantees that every instance of every task is actually scheduled. Conditions (1) and (2) ensure the respect of the release dates and of the deadlines. Condition (3) guarantees that two consecutive blocks do not overlap. (4) means that the functions are always fully processed. (5) guarantees the respect of the precedence constraints, and finally (6) ensures the respect of the mutual exclusions. In the sequel, for simplicity reasons, we will focus on synchronous task sets: we assume that the first release times are all equal, and equal to 0. In such a case, the application behaves in a cyclical way from time 0, and we just have to consider the schedule *Sch*(0, *H*). For asynchronous tasks, the application also behaves in a cyclical way, but only from a given time t_c (Choquet-Geniet and Grolleau, 2004). We would thus have to consider the schedule on the time interval $[0, t_c + H)$.

3 Topology of a schedule

Some properties only depend on the execution order of the different instances of the different tasks. To easily express these properties, we introduce the topology of a schedule which describes this execution order.

3.1 Notations

We first introduce some notions and notations which will be useful in the remainder of the paper.

Let A be a finite alphabet, $M = m_1 m_2 \ldots m_n$ a word written on the alphabet A, and a a letter.

$Ind(M, a)$ is the set of the indices of the occurrences of a in the word M: $Ind(M, a) = \{i \in \{1,\ldots,n\} \text{ s.t. } m_i = a\}$.

$|M|$ denotes the length of M, and $|M|_a$ the number of occurrences of a in M, which is also the cardinality of $Ind(M, a)$.

If M and M' are two words, then M' is a sub-word of M, what is denoted by $M' \lhd M$, if there exist $u_1, u_2,\ldots,u_p, v_0,v_1,\ldots,v_p$ in A^* such that $M' = u_1 u_2 \ldots u_p$ and $M = v_0 u_1 v_1 u_2 \ldots u_p v_p$. Note that in this case, there exists a strictly increasing mapping function ϕ from $\{1,\ldots,|M'|\}$ onto $\{1,\ldots,|M|\}$ such that for all i in $\{1,\ldots,|M'|\}$, $m'_i = m_{\phi(i)}$.

3.2 The topological sequence

The topology of a schedule *Sch*(0, *H*) describes the succession of the sub-tasks. It is given by a word written on the alphabet $Alp = \left\{ f_p^q \text{ with } 1 \le p \le n, 1 \le q \le \frac{H}{T_p} \right\}$. It is obtained by the projection of the schedule on its third component, and is called the *topological sequence*. For instance, the topological sequence associated with $M_1(0, H) = f_1^1 f_2^1 f_1^1 f_2^1 f_1^2 f_2^1 f_2^2 f_1^2 f_2^2 f_1^3$.

Thus, a schedule can be seen as a temporised topological sequence where the temporisation guarantees the respect of the temporal constraints and the topology guarantees the respect of the precedence constraints and of the mutual exclusions. These constraints can be expressed using only the topological sequence. If *Sch* is a schedule and M its associated topological sequence, the condition (5) of the property 1 can be reformulated as: $f_p \prec f_q \Rightarrow \forall k \in \left\{1,\ldots,\frac{H}{T_p}\right\}$, $\forall u \in Ind(M, f_p^k)$, $\forall v \in Ind(M, f_q^k)$, $u < v$: in the topological word, any occurrence of the letter f_p^k appears prior to any occurrence of the letter f_q^k. And the exclusion mutual condition can be reformulated as: $f_p \lozenge f_q$ if $\forall k \in \left\{1,\ldots,\frac{H}{T_p}\right\}$, $\forall k' \in \left\{1,\ldots,\frac{H}{T_p}\right\}$, $[\min(Ind(M, f_p^k)), \max(Ind(M, f_p^k))] \cap$ $[\min(Ind(M, f_q^{k'})), \max(Ind(M, f_q^{k'}))] = \emptyset$: in the topological word, the critical sections do not overlap. We thus have the next proposition:

Proposition 2: let *Sch* be a valid schedule, and let *Sch'* be a schedule such that the associated topological sequences verify $M' \lhd M$ then the precedence constraints and the mutual exclusion rules are met in *Sch'* too.

Proof: assume that *Sch'* does not meet the precedence constraints. Then there exists two functions f_p and f_q such that $f_p \lhd f_q$, but there exist $k \in \left\{1,\ldots,\frac{H}{T_p}\right\}$, $u \in Ind(M', f_p^k)$ and $v \in Ind(M', f_q^k)$ such that $v < u$. Then, since ϕ is increasing, we would have $\phi(v) < \phi(u)$ with $\phi(u) \in Ind(M', f_p^k)$ and $\phi(v) \in Ind(M', f_q^k)$. But this in not possible since *Sch* meets the precedence constraints. Thus *Sch'* also meets the precedence constraints.

Assume now that *Sch* meets the mutual exclusion rules, but not *Sch'*. There exist two functions f_p and f_q such that $f_p \lozenge f_q$ but there exist $k \in \left\{1,\ldots,\frac{H}{T_p}\right\}$, and $k' \in \left\{1,\ldots,\frac{H}{T_p}\right\}$, such that $[\min(Ind(M', f_p^k)), \max(Ind(M', f_p^k))] \cap [\min(Ind(M', f_q^{k'})), \max(Ind(M', f_q^{k'}))] \neq \emptyset$:

Let i be in $[\min(Ind(M', f_p^k)), \max(Ind(M', f_p^k))] \cap [\min(Ind(M', f_q^{k'})), \max(Ind(M', f_q^{k'}))]$.

We then have $\phi(i) \in [\phi(\min(Ind(M', f_p^k))), \phi(\max(Ind(M', f_p^k)))] \cap [(\phi(\min(Ind(M', f_q^{k'}))), \phi(\max(Ind(M', f_q^{k'})))]$ since ϕ is increasing. We also have $\min(Ind(M, f_p^k)) \le \phi(\min(Ind(M', f_p^k))) \le \phi(i) \le \phi(\max(Ind(M', f_p^k))) \le \max(Ind(M, f_p^k))$, and the same holds for $f_q^{k'}$ thus $\phi(i) \in [\min(Ind(M, f_p^k)), \max(Ind(M, f_p^k))] \cap [\min(Ind(M, f_q^{k'})), \max(Ind(M, f_q^{k'}))]$ which is empty by assumption. So the exclusion mutual rules must also be respected by *Sch'*. □

3.3 The compliance problem

In the following, we consider a synchronous task set. We assume that a valid simulated schedule has been computed (by any means) on the basis of the WCET's. Thus the topology of the simulated schedule respects the precedence relations, as well as the mutual exclusion rules, and the schedule meets all the temporal constraints. We denote $Ss = ((start_s(i), end_s(i), f_{\alpha_s}^{\beta_s}))_{i \in 1\ldots q}$ this simulated schedule on the time interval [0, *H*), and we denote $M_s = ms_1 ms_2 \ldots ms_q$ the associated topological sequence. We suppose that an effective schedule has been obtained, running effectively the application. To get this schedule, we can use an observer (Bikienga et al., 2012). Let $Se = ((start_e(i), end_e(i), f_{\alpha_e}^{\beta_e}))_{i \in 1\ldots r}$ be this schedule on the time interval [0, *H*) and

$Me = me_1 me_2 \ldots me_r$ its topological sequence. The number of blocks of the schedule *Ss* (resp. *Se*) is denoted *q* (resp. *r*). We assume that *Ss* verifies Property 1, and *Se* verifies the point (4) of this property using the ACET's instead of the WCET's: the sum of the durations of the blocks associated with a given instance equals the ACET of this instance. We will say that the schedule respects the ACET's. Our aim is to determine the required properties for *Se* to match with *Ss*. We will then say that *Se* is compliant with *Ss*, or implement *Ss*. We will consider two kinds of compliance: the non-flexible compliance and the flexible one. And we will prove that each one guarantees the respect of all constraints, i.e., guarantees that the effective schedule is still valid.

4 Non-flexible compliance

In a non-flexible approach, we consider that between $start_s(i)$ and $end_s(i)$, only $f_{\alpha_s(i)}^{\beta_s(i)}$ can be processed. If an instance is shorter than planned, the system remains idle until the next start time. The main benefit of this approach is that it requires no further verification to guarantee the respect of the temporal parameters. It is also well suited to non-conservative schedules, where the system must sometimes idle whereas some tasks are ready to run. This can be the case for instance if EDL (Chetto and Chetto, 1989) is used.

4.1 Non-pre-emptive schedules

We first consider non-pre-emptive schedules. For such schedules, only the precedence constraints have to be considered, the problem of the mutual exclusion no longer exists since the executions of two different tasks cannot interlace. Formally, we say that the effective schedule *Se* is compliant with the non-pre-emptive simulated schedule *Ss* in a non-flexible way ($Se \approx Ss$) if the Rule 3 is met:

Rule 3:

$$Se \approx Ss \text{ if } \begin{cases} (1)\ Me = Ms \\ (2)\ \forall i \in 1 \ldots r,\ start_e(i) = start_s(i) \end{cases}$$

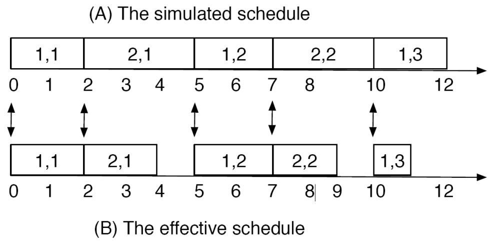
Condition (1) means that the topology is preserved. Condition (2) means that each block starts exactly at the planned start time. Then both conditions guarantee that each instance is processed within a time interval which meets the temporal constraints, as stated by the following proposition.

Proposition 4: if *Ss* is a valid non-pre-emptive schedule, computed using the WCET's, and if an effective schedule *Se* which respects the ACET's verifies Rule 3, then the schedule *Se* is valid.

Proof: to prove the validity, we must prove that *Se* meets each condition of Property 1. Condition (4) is met by assumption, so we focus on the other ones. Condition (0) comes from the equality *Ms* = *Me*. The start times are preserved in Se, because of condition (2) of the rule. Then, the lengths of the different blocks are equal to the actual durations of the different instances by assumption. Thus the end times can only have been put forward since $C_i^j \le C_i$ for all *i* and *j*. Thus conditions (1), (2) and (3) of the Property 1 are still met. And since the topological sequences of the simulated and the effective schedules are equal, the precedence relations are preserved according to Proposition 2. □

Example. We consider the application A_1. Figure 2 gives a valid non-pre-emptive simulated schedule, Ss_1 = ((0, 2, f_1^1), (2, 5, f_2^1), (5, 7, f_1^2), (7, 10, f_2^2), (10, 12, f_1^3)) and an effective schedule which implements it in the case where $C_1^3 = 1$ and $C_2^1 = C_2^2 = 2$, the other ACET's are equal to the WCET's: Se_1 = ((0, 2, f_1^1), (2, 4, f_2^1), (5, 7, f_1^2), (7, 9, f_2^2), (10, 11, f_1^3)). We have $Ms_1 = Me_1 = f_1^1 f_2^1 f_1^2 f_2^2 f_1^3$. And e.g., at time 4, the instance f_1^2 is ready, but it nevertheless waits until time 5 to start execution.

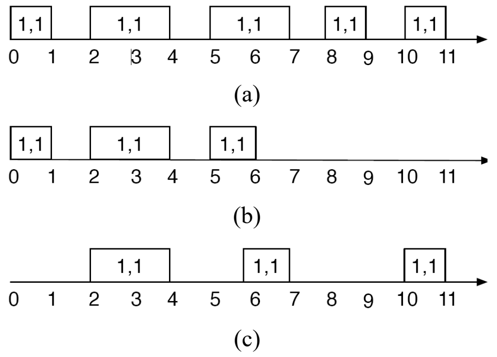
Figure 2 (a) A non-pre-emptive simulated schedule for the application A_1 (b) The non-flexible implementation of the schedule when $C_1^3 = 1$ and $C_2^1 = C_2^2 = 2$



4.2 *Pre-emptive schedules*

We now consider pre-emptive schedules which mean that some instances may appear in several blocks (see Figure 1). If an instance has an ACET shorter than its WCET, it can result in the deletion of some blocks, which must be the last ones of the instance, and the end of the last remaining block can be put ahead, as illustrated by Figure 3. Indeed, an instance cannot be suspended within a block and resume in a further one. It is processed in a conservative way within the blocks. This is required, because the ACET's are not a priori known. So the reduction of the duration of an instance cannot be anticipated.

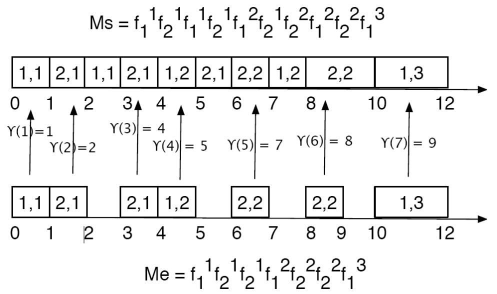
Figure 3 (a) A simulated schedule for the task $(f_1^1, (0, 7, 11, 11))$ (b) The non-flexible implementation of the schedule when $C_1^1 = 4$ (c) An effective schedule which is not an implementation of the simulated schedule



In the example of Figure 3, we have a task $\tau_1 = (f_1(0, 7, 11, 11))$. The simulated execution of the first instance of the task is given by the schedule (a), composed of five blocks. Let us assume that the ACET equals 4. The schedule (b) implements the schedule (b) in a non-flexible way: the last two blocks have been deleted, and the third one has been shortened; but the schedule (c) where the blocks 1 and 5, and the first half of the block 3 have been deleted, is not an implementation of the schedule (a).

From this example, we can see that an effective schedule can have less blocks than the simulated one ($r \leq q$).

Figure 4 A simulated schedule for the application $A1$ and the non-flexible implementation when $C_1^1 = C_1^2 = 1,\ C_2^1 = C_2^2 = 2$ – construction of the function



First, we relate the blocks of the effective schedule to the corresponding blocks of the simulated schedule. We thus define the function γ (see Figure 4):

$$\begin{cases} \gamma(1) = \inf\{i / me_1 = ms_i\} \\ \gamma(p) = \inf\{i > \gamma(p-1) / me_p = ms_i\},\ p > 1 \end{cases}$$

Thus we have $Me = me_1 me_2 \ldots me_r = ms_{\gamma(1)} ms_{\gamma(2)} \ldots ms_{\gamma(r)}$. Figure 4 illustrates the construction of the function γ. Here, the blocks 3 and 6 of the simulated schedule have been deleted.

We say that the effective schedule Se is a non-flexible implementation of the pre-emptive simulated schedule $Ss(Se \approx Ss)$ if:

Rule 5:

$$Se \approx Ss \text{ if } \begin{cases} (1)\ \forall i \in 1 \ldots q, |Me|_{ms_i} > 0 \\ (2)\ Me \lhd Ms \\ (3)\ \forall j \text{ s.t. } \exists p, \gamma(p) < j < \gamma(p+1) \\ \quad \left|me_{p+1} \ldots me_r\right|_{ms_j} = 0 \\ (4)\ \forall j \in 1 \ldots r,\ d_e(j) < d_s(\gamma(j)) \\ \quad \Rightarrow \left|me_{j+1} \ldots me_r\right|_{ms_{\gamma(j)}} = 0 \\ (5)\ \forall j \in 1 \ldots r,\ start_e(j) = start_s(\gamma(j)) \end{cases}$$

Condition (1) guarantees that no instance has been skipped. Condition (2) guarantees that the effective topology is compliant with the simulated topology (there is no instance inversion for example). Condition (3) guarantees that the deleted blocks of an instance are the last blocks of this instance: the deletion of a block number j means that the associated instance completed earlier, and thus this instance does not appear again later in the effective schedule. Condition (4) guarantees that if an effective block is shorter than the associated simulated block, then it is the last effective block of the associated instance. And finally, condition (5) expresses the non-flexibility: the start dates are strictly respected.

Consider again the schedule (C) of Figure 3. We have $\gamma(1) = 2$, $\gamma(2) = 3$, and $\gamma(3) = 5$. But condition (3) is not met: for $j = 4$, we have $\gamma(2) < j < \gamma(3)$ but $|me_3|_{ms_4} = |f_1^1|_{f_1^1} = 1$. Condition (4) neither holds: $1 = d_e(2) < d_s(\gamma(2)) = 2$, but $|me_3|_{ms_{\gamma(2)}} = 1$. And finally, condition (5) is not verified since $start_e(2) = 6 \neq start_s(\gamma(2)) = 5$. Thus the schedule (C) was not compliant with the simulated schedule.

We then have the following validity proposition:

Proposition 6: if *Ss* is a valid pre-emptive schedule, computed using the WCET's, and if an effective schedule *Se* which respects the ACET's verifies Rule 5, then the schedule *Se* is valid.

Proof: again, we must prove that the conditions 0, 1, 2, 3, 5 and 6 of Property 1 are still verified. First, point (0) is verified because *Ms* verifies it (i.e., each instance appears in *Ms*), and because of the point (1) of Rule 5 each letter which appears in *Ms* also appears in *Me*). Then, the start times are respected because of the point (5) of the rule, and the end times can only be put ahead because of the conservatism of the execution within the blocks. So the conditions (1), (2) and (3) of Property 1 are still satisfied. Then the points (5) and (6) are verified according to proposition 2. □

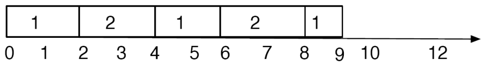
5 Flexible compliance

In a flexible policy, if an instance completes earlier than planned, and if the instance scheduled in the next block is pending, it can start execution without waiting for the date specified in the simulated schedule. Thus the simulated start date gives the latest moment by which the instance must start execution. We can note that the online strategies are flexible strategies. In the case of a mixed scheduling (with periodic and aperiodic tasks), one benefit of such an implementation is that the idle slots can be postponed if they are not yet used for an aperiodic task. This can improve the integration of the aperiodic tasks.

5.1 Non-pre-emptive schedules

Let us again consider the application A_1 and the simulated schedule of Figure 2. A flexible implementation of the schedule, if $C_1^3 = 1$ and $C_2^1 = C_2^2 = 2$, and the other ACET's equal the WCET's is $Se2 = ((0, 2, f_1^1), (2, 4, f_2^1), (4, 6, f_1^2), (6, 8, f_2^2), (8, 9, f_1^3))$ (see Figure 5). In this schedule, the three idle slots are at the end of the schedule, whereas they are separated in the non-flexible implementation. Thus if an aperiodic task of duration 3 is executed during the idle slots, is would be pre-empted twice in the non-flexible system, but it would not be pre-empted in the flexible one. The overhead will thus be smaller.

Figure 5 A flexible implementation of the schedule of Figure 2(a) when $C_1^3 = 1$ and $C_2^1 = C_2^2 = 2$



Formally, we say that the effective schedule *Se* is a flexible implementation of the non-pre-emptive simulated schedule *Ss* (*Se* ~ *St*) if:

Rule 7:

$$Se \sim St \text{ if } \begin{cases} (1)\ Me = Ms \\ (2)\ start_e(1) = start_s(1) \\ (3)\ \forall i \in \{2,\ldots,r\}, \\ Max\left(end_e(i-1),\, r_{\alpha_e(i)}^{\beta_e(i)}\right) \le start_e(i) \le start_s(i) \end{cases}$$

Here again, (1) assures the respect of the topology of the schedule and (2) and (3) assure that all the temporal parameters are met. We can notice that in such an implementation, the release dates will have to be explicitly considered at run time, what is not the case for the non-flexible implementation. The preservation of the validity is given by Proposition 8.

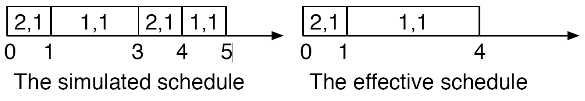
Proposition 8: if *Ss* is a valid non-pre-emptive schedule, computed using the WCET's, and if an effective schedule *Se* which respects the ACET's verifies Rule 7, then the schedule *Se* is valid.

Proof: the point (0) of Property 1 again comes from the point (1) of the rule. The conditions (1) and (3) directly result from the points (2) and (3) of the rule. The point (2) of the property comes from the points (2) and (3) of the rule, and from the fact that $C_i^j \le C_i$ for any *i* and *j*, thus again the end of an effective block can only have been put forward as to the end of the associated simulated block. And the point (5) comes from the preservation of the topological sequence, according to Proposition 2. □

5.2 Pre-emptive schedules

We now consider pre-emptive schedules. We again impose the respect of the topology of the schedule, thus the conditions (1) and (2) of rule 3 must hold. The main difficulty here comes from the following situation: if a task completes earlier than planned, the next block may also start earlier, and some blocks may be deleted (as in the non-flexible case). But this may cause the merge of some blocks, as illustrated in Figure 6. The second block of the instance f_2^1 does not exist in the effective schedule, and thus, the second block of the task τ_1 has been brought forward, and merged with the first block.

Figure 6 The simulated schedule for the task set (f_1, (0, 2, 6, 6), (f_2, (0, 3, 6, 6)) and its flexible implementation when $C_1^1 = 1$ and $C_2^1 = 3$



Let us consider a given block of the simulated schedule ($start_s(i)$, $end_s(i)$, $f_{\alpha_s(i)}^{\beta_s(i)}$). The block appears (in its exact form) in the effective schedule if there exists j such that $i = \gamma(j)$, $start_s(i) = start_e(j)$ and $end_s(i) = end_e(j)$. Else, at least one of the four following properties is verified:

1 The block does not exist: it has been deleted because the instance is shorter than planned: $\forall j \in \{1,\ldots,r\}, \gamma(j) \neq i$.

2 The block exists but starts earlier than planned because some previous instances completed earlier than planned: $start_e(j) < start_s(i)$, with $i = \gamma(j)$.

3 The block exists, but is shorter than planned because the instance is shorter: $d_e(j) < d_s(i)$ with $i = \gamma(j)$. In this case, the block must be the last one associated with the considered instance in the effective schedule, because of the conservatism within the blocks.

4 The block exists, but is longer than planned: $d_e(j) > d_s(i)$ with $i = \gamma(j)$. This comes from the merge of some blocks.

To capture these situations, we first introduce some notations, and a compliance operator.

If S is a schedule composed of k blocks ($S = B_1B_2\ldots B_k$), over the time interval $[0, H)$, then, $S[k_1\colon k_2]$ (with $1 \leq k_1 \leq k_2 \leq k$) denotes the schedule reduced to the blocks numbered from k_1 to k_2: $S[k_1\colon k_2] = B_{k1}\ldots B_{k2}$. Its associated topological sequence is then $M[k_1\colon k_2]$.

We again assume that the conditions (1) and (2) of Rule 5 are verified for the schedules Se and Ss. Let us consider the two truncated schedules $Ss[k_1\colon q]$ and $Se[k_1'\colon r]$, and a time t which is the first time by which the processor is available to process $f_{\alpha_e(k_1')}^{\beta_e(k_1')}$. Initially, we set $k_1 = k_1' = 1$ and $t = 0$. Then we say that $Se[k_1'\colon r]$ implements $Ss[k_1\colon q]$ from the time t ($Se[k_1'\colon r] \sim_t Ss[k_1\colon q]$) if we are in one of the following situations:

$$Se[k_1'\colon r] \sim_t Ss[k_1\colon q] \quad \text{if}$$

- either

  1 $Se[k_1'\colon q]$ is an empty schedule. The remaining blocks in Ss (if any) correspond to already completed instances, since no instance is skipped (condition (1) of the Rule 5).

- or

  $$2 \quad \begin{cases} (1)\ ms_{k_1} \neq me_{k_1'} \quad (\text{thus } \gamma(k_1') \neq k_1) \\ (2)\ |\,Me[k_1'\colon r]\,|_{m_{k_1}} = 0 \\ (3)\ Se[k_1'\colon r] \sim_t Ss[k_1 + 1\colon q] \end{cases}$$

  The block number k_1 of the simulated schedule has been deleted. Thus the corresponding instance cannot be scheduled later (2). And the effective schedule implements the remainder of the simulated schedule, from the time t (3).

- or

  $$3 \quad \begin{cases} (1)\ ms_{k_1} = me_{k_1'}\,(\text{thus } \gamma(k_1') = k_1) \\ (2)\ Max(t, r_{\alpha_s(k_1)}^{\beta_s(k_1)}) \leq start_e(k_1') \leq start_s(k_1) \end{cases}$$

The first block of the effective schedule implements the first block of the simulated schedule, and it starts not later than specified in the simulated schedule (but possibly earlier). In this case, we next must compare the durations of the effective and of the simulated blocks $(d_e(k_1'))$ and $d_s(k_1)$). Three cases are possible:

a If they are equal $(d_e(k_1') = d_s(k_1))$, we have a last condition which is

Rule 9:

$$Se[k_1'+1:r] \sim_{end_s} Ss[k_1+1:q]$$

The first blocks correspond to each other. We move to the end of the first effective block, and the rest of the effective schedule must implement the rest of the simulated schedule from that time. Furthermore, we have $end_e(k_1') = start_e(k_1') + d_e(k_1')$ $\leq start_s(k_1) + d_s(k_1) = end_s(k_1) \leq d_{\alpha_s(k_1)}^{\beta_s(k_1)}$.

b If the effective block is shorter than the simulated one $(d_e(k_1') < d_s(k_1))$, then the instance of the block completes earlier than planned, and cannot thus appear later in the schedule (condition (1) of rule 10). Then as in case a, the remainder of the effective schedule must implement the remainder of the simulated schedule, from a time equal to the end of the first effective block (condition (2) of Rule 10). Thus the next conditions must be verified:

Rule 10:

$$\begin{cases} (1)\ \left|Me[k_1'+1:r]\right|_{m_{k_1}} = 0 \\ (2)\ Se[k_1'+1:r] \sim_{end_e(k_1')} Ss[k_1+1:q] \end{cases}$$

And we also have $end_e(k_1') \leq end_s(k_1) \leq d_{\alpha_s(k_1)}^{\beta_s(k_1)}$.

c If the effective block is longer than the simulated one $(d_e(k_1') > d_s(k_1))$, which results from the merge of some blocks, after the deletion of some other ones, we must first determine which blocks have been deleted, and verify that the associated instances do not appear in the remainder of the schedule. Then we must identify the blocks that have been merged. If the last one has been shortened, then the associated instance cannot appear later. So we get the Rule 11. The blocks numbered $p_0, p_1, \ldots, p_s$ are those which are merged. $(1.\chi)$ means that these blocks are associated with the right instance $(f_{\alpha_s(k_1)}^{\beta_s(k_1)})$. $(1.\delta)$ means that the intermediate blocks have effectively been deleted. $(1.\in)$ and $(1.\eta)$ express the fact that exactly these blocks have been merged. (2) means that if the last merged block has been shortened, the associated instance is completed, and cannot thus appear again in the remainder of the effective schedule. And finally, (3) expresses that the remainder of the effective schedule must implement the simulated schedule starting at the block after the last merged block, and again from the end of the first effective block.

Rule 11:

$$\begin{cases} (1)\ \text{There exist}\ p_0 = k_1, p_1, \ldots, p_s\ (s \geq 1) \\ \quad \text{s.t.:}\ (\chi) \forall i \in \{1, \ldots, s\}, \left(\alpha_s(p_i), \beta_s(p_i)\right) = \left(\alpha_s(k_1), \beta_s(k_1)\right) \\ \quad\quad (\delta) \forall i \in 1 \ldots s-1, \forall j\ \text{s.t.}\ p_i < j < p_{i+1}, \left|Me[k_1' : r]\right|_{m_j} = 0 \\ \quad\quad (\in) \sum_{i=0}^{s-1} d_s(p_i) < d_e(k_1') \\ \quad\quad (\eta) \sum_{i=0}^{s} d_s(p_i) \geq d_e(k_1') \\ (2)\ d_e(k_1') < \sum_{i=0}^{s} ds(p_i) \Rightarrow \left|Me[k_1' + 1 : r]\right|_{m_{k_1}} = 0 \\ (3)\ Se[k_1 + 1 : r] \sim_{end_e(k_1')} Ss[p_s + 1 : q] \end{cases}$$

And we have $end_e(k_1') \leq end_s(p_i) \leq Cp_s(f_{\alpha_s(k_1)}^{\beta_s(k_1)}) \leq d_{\alpha_s(k_1)}^{\beta_s(k_1)}$.

Then we define the global operator. The effective schedule *Se* is a flexible implementation of the pre-emptive schedule *Ss* (*Se* ~ *Se*) if:

Rule 12: $Se \sim Se$ if

$$Se \sim Se\ \text{if} \begin{cases} (1)\ \forall i \in 1 \ldots q \left|Me\right|_{ms_i} > 0 \\ (2)\ Me \lhd Ms \\ (3)\ Se \sim_0 Ss \end{cases}$$

And we have the following validity proposition:

Proposition 13: if *Ss* is a valid pre-emptive schedule, computed using the WCET's, and if an effective schedule *Se* which respects the ACET's verifies Rule 12, then the schedule *Se* is valid.

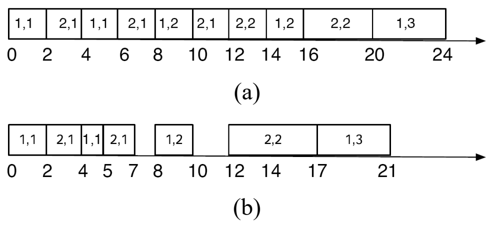
Proof: the condition (0) of property 1 results from the point (1) of Rule 12. It follows from the point (3.2) that an effective block never starts before the release time of the associated instance, so the point (1) is verified. Then, we have stated that in every case, we have $end_e(k_1') \leq d_{\alpha_s(k_1)}^{\beta_s(k_1)}$ thus the point (2) is verified too. The operator $\sim_t$ guarantees that a block starts only after the end of the previous one [Rules 9, 10-(2) and 11-(3)], so the condition (3) of the validity property is verified. And finally, the points (5) and (6) are verified according to Proposition 2. □

5.3 Illustrative example

As an illustration, we consider the task set $A_2 = < f_1(0, 4, 8, 8), f_2(0, 6, 12, 12) >$, and the simulated and effective schedules given in Figure 7. *Ss* = ((0, 2, f_1^1), (2, 4, f_2^1), (4, 6, f_1^1), (6, 8, f_2^1), (8, 10, f_1^2), (10, 12, f_2^1), (12, 14, f_2^2), (14, 16, f_1^2), (16, 20, f_2^2), (20, 24, f_1^3)).

$$Se = \begin{pmatrix} (0,2,f_1^1),(2,4,f_2^1),(4,5,f_1^1),(5,7,f_2^1), \\ (8,10,f_1^2),(12,17,f_2^2),(17,21,f_1^3) \end{pmatrix}$$

Figure 7 (a) The simulated schedule for the task set A_2 (b) The flexible implementation of the schedule when $C_1^1 = 3$, $C_1^2 = 2$, $C_1^3 = 4$, $C_2^1 = 4$ and $C_2^2 = 5$



We thus have: $Ms = f_1^1 f_2^1 f_1^1 f_2^1 f_1^2 f_2^1 f_2^2 f_1^2 f_2^2 f_1^3$ and $Me = f_1^1 f_2^1 f_1^1 f_2^1 f_1^2 f_2^2 f_1^3$. Thus we have $q = 10$ and $r = 7$.

The function γ is given by: $\gamma(1) = 1$, $\gamma(2) = 2$, $\gamma(3) = 3$, $\gamma(4) = 4$, $\gamma(5) = 5$, $\gamma(6) = 7$, $\gamma(7) = 10$.

To prove that *Se* is a flexible implementation of *Ss*, we must prove that the three points of Rule 12 are verified.

The point (1) is verified since the five letters $f_1^1, f_1^2, f_1^3, f_2^2$ and f_2^2 appear in the effective topological sequence. So no instance is skipped.

Then we have $Me \lhd Ms$, so the point (2) is verified. Finally, we must verify that the condition (3) is met, i.e., that:

Step 1 $Se[1{:}\ 7] \sim_0 Ss[1{:}\ 10]$. We are in the case (3.a). We have $Max(0, r_1^1) = 0 = start_e(1) = start_s(1)$ thus the point (3.2) is verified. Thus we apply Rule 9, i.e., we must verify that.

Step 2 $Se[2{:}\ 7] \sim_2 Ss[2{:}\ 10]$. We are again in the case (3.a). We have $Max(2, r_2^1) = 2 = start_e(2) = start_s(2)$ thus the point (3.2) is verified. Then we again apply Rule 9, i.e., we verify that.

Step 3 $Se[3{:}\ 7] \sim_4 Ss[3{:}\ 10]$. We are in the case (3.b) ($d_e(3) = 1 < d_s(3) = 2$). We have $Max(4, r_1^1) = 4 = {}_{start}e(3) = start_s(3)$ thus the point (3.2) is verified. Then we apply Rule 10.

We have $|| Me[4:7]|_{m_3} = | f_2^1 f_1^2 f_2^2 f_1^3 |_{f_1^1} = 0$ [the condition (1) is verified]. To prove the point (2) of Rule 10, we must then verify that.

Step 4 $Se[4{:}\ 7] \sim_5 Ss[4{:}\ 10]$. We are in the case (3.a). We have $Max(5, r_2^1) = 5 = start_e(4) \le start_s(4) = 6$ thus the point (3.2) is verified. Then from Rule 9, we must verify that.

Step 5 $Se[5{:}\ 7] \sim_7 Ss[5{:}\ 10]$. We are in the case (3.a). We have $Max(7, r_1^2) = 8 = start_e(5) = start_s(5)$ thus the point (3.2) is verified. Thus we apply Rule 9, and verify that.

Step 6 $Se[6:7] \sim_{10} Ss[6:10]$. We are in the case (2).

We have $|Me[6:7]|_{m_6} = |f_2^2 f_1^3|_{f_2^1} = 0$, so condition (2) is met. To prove condition (3), we must verify that.

Step 7 $Se[6:7] \sim_{10} Ss[7:10]$. We are in the case (3.c). We have $Max(10, r_2^2)$ = *12* = $start_e(6) = start_s(7)$ thus the point (3.2) is verified. We then apply Rule 11. We have $p_0 = 7$, $p_1 = 9$ and $s = 1$: $d_e(6) = 5 > d_s(7)$, and $d_e(6) < d_s(7) + d_s(9) = 6$.

For $j = 8$: $p_0 < j < p_1$, and $|Me[6:7]|_{m_j} = |f_2^2 f_1^3|_{f_1^2} = 0$, so the point (1) is verified.

Next, we have $d_e(6) < d_s(7) + d_s(9)$ and $|Me[7:7]|_{m_7} = |f_1^3|_{f_2^2} = 0$, so the point (2) is verified. Finally, to prove the point (3), we must verify that.

Step 8 $Se[7:7] \sim_{17} Ss[10:10]$. We are in the case (3.a). We have $Max(17, r_1^3) = 17 = start_e(7) < start_s(10) = 20$ thus the point (3.2) is verified. Finally we apply Rule 9, so we must verify that.

Step 9 $Se[8:7] \sim_{21} Ss[11:10]$. We are in the case (1), thus the condition is met.

So the condition (3) of Rule 12 is satisfied.

We can thus conclude that *Se* is a flexible implementation of *Ss*.

6 Conclusions and perspectives

The temporal validation of a real-time application has been widely explored in the literature. It is performed before run-time, on the basis of the WCET's. A simulated schedule is thus computed. We considered the actual behaviour of an application, at run-time. To get this behaviour, an observator (Bikienga et al., 2012) can be used. We proposed to fill the gap between the simulated and the actual behaviours, to verify whether the application behaves correctly according to the scheduling strategy, which is given by means of the simulated schedule. For that aim, we focused on the question 'what properties must an effective schedule meet to be an implementation of a given pre-computed schedule?' Besides, since the ACET's may be shorter than the WCET's, the only knowledge of the simulated schedule is not sufficient to determine how to implement the schedule. We have to precise the compliance policy, which specifies how the application must behave in the case of ACET's shorter than the WCET's. Two policies have been presented: the non-flexible policy, where the start times are strictly respected, and the flexible policy, where a task may start earlier if some previous tasks completed earlier. In the first case, no further verification must be made to guarantee the respect of the temporal constraints. In the second case, the respect of the release dates must be explicitly implemented. But in the second case, if they are not required to process some non-periodic tasks, the idle slots may be postponed, to be used later. This provides a higher flexibility to mixed applications (with periodic and aperiodic tasks).

An important issue when defining the compliance policy is to guarantee that an effective schedule compliant with a valid simulated schedule will still be valid. We have proved that our policies preserve the validity. Thus no scheduling anomalies or temporal failures will occur in the case of shorter ACET's.

If we consider an offline strategy: a valid schedule has been computed, and must be implemented to control the behaviour of the application. We then must chose if we want a flexible or a non-flexible implementation. This will give the basis of the definition of implementation tools, i.e., of code generators which will produce the code of an application whose execution will obey the pre computed schedule according to the chosen policy. The definition of such generators will be our next concern.

References

Andersson, B. and Jonsson, J. (2000) 'Fixed-priority preemptive multiprocessor scheduling: to partition or not to partition', *Proceedings of the Conference on Real-Time Computing Systems and Applications*, December, pp.337–346.

Bikienga, M., Geniet, D. and Choquet-Geniet, A. (2012) 'Observation tools for effective schedules in a rtos', *SIGBED Rev.*, June, Vol. 9, No. 2, pp.17–22.

Buttazzo, G.C. (1997) *Hard Real-Time Computing Systems*, Kluwer Academic Publishers, Boston.

Chauviere, B. and Geniet, D. (2007) 'Une approche markovienne pour l'etude de syst`emes temps-r´eel `a contraintes strictes', *Techniques et Sciences Informatiques*, October, Vol. 26, No. 10, pp.1269–1303.

Chetto, H. and Chetto, M. (1989) 'Some results of the earliest deadline scheduling algorithm', *IEEE Transactions on Software Engineering*, Vol. 15, No. 10, pp.1261–1269.

Choquet-Geniet, A. and Grolleau, E. (2004) 'Minimal schedulability interval for real time systems of periodic tasks with offsets', *Theoretical of Computer Science*, Vol. 310, Nos. 1–3, pp.117–134.

Choquet-Geniet, A. and Largeteau-Skapin, G. (2014) 'Size analysis in multiprocessor real-time scheduling', *Int. J. Critical Computer-Based Systems*, Vol. 5, Nos. 3/4, pp.197–217.

Cottet, F. and Babau, J.P. (1994) 'Off-line temporal analysis of hard real-time applications', *2nd IEEE Workshop on Real-Time Applications*.

Cucu-Grosjean, L. and Goossens, J. (2010) 'Predictability of fixed-job priority schedulers on heterogeneous multiprocessor real-time systems', *Information Processing Letters*, Vol. 110, No. 10, pp.399–402.

Dertouzos, M.L. and Mok, A.K.L. (1989) 'Multiprocessor scheduling in hard real-time environment', *IEEE Transactions on Software Engineering*, Vol. 15, No. 12, pp.1497–1506.

Geniet, D. and Largeteau-Skapin, G. (2007) 'WCET free time analysis of hard real-time systems on multi-processors: a regular language-based model', *Thoretical Computer Science*, Vol. 388, Vol. 388, Nos. 1–3, pp.26–52.

Graham, R.L., Lawler, E.L., Lenstra, J.K. and Rinnooy, A.H.G. (1979) 'Optimization and approximation in deterministic sequencing and scheduling: a survey', *Annals of Discrete Mathematics*, Vol. 5, pp.287–326.

Grolleau, E. and Choquet-Geniet, A. (2002) 'Off line computation of real time schedules by means of petri nets', *Journal of Discrete Event Dynamic Systems*, Vol. 12, pp.311–333.

Hong, K.S. and Leung, J.Y. (1992) 'On-line scheduling of real-time tasks', *IEEE Transactions on Computers*, Vol. 41, No. 10, pp.1326–1331.

Leung, J. and Whitehead, J. (1982) 'On the complexity of fixed-priority scheduling of periodic real-time tasks', *Performance Evaluation*, Vol. 2, No. 4, pp.237–250.

Liu, C.L. and Layland, J.W. (1973) ‘Scheduling algorithms for multiprogramming in a hard real-time environment’, *Journal of the ACM*, Vol. 20, No. 1, pp.46–61.

Martel, C. (1982) ‘Preemptive scheduling with release times, deadlines, and due times’, *Journal of the ACM*, Vol. 29, No. 3, pp.812–829.

Schranzhofer, A., Chen, J.J. and Thiele, L. (2009) ‘Timing predictability on multi-processor systems with shared resources’, *Workshop on Reconciliating Predictability and Efficiency at EMSOFT*, October.

Xu, J. and Parnas, D.L. (1990) ‘Scheduling processes with release times, deadlines, precedence and exclusion relations’, *IEEE Transactions on Software Engineering*, Vol. 16, No. 3, pp.360–369.