Integrating semantic properties within a Petri net based scheduling tool

Christian Fotsing
LIAS, ENSMA, France
fotsingc@ensma.fr

Annie Geniet
LIAS, University of Poitiers, France
annie.geniet@univ-poitiers.fr

***Abstract*—We consider real-time applications with interacting periodic tasks which may contain conditional statements. When the behaviours of the tasks are linked to the context, they may be interdependent, and some theoretical configurations may actually be impossible. Our concern is to reduce the schedulability analysis to the only coherent behaviours. We thus propose a model-driven approach. We first model the application and the semantic constraints coming from the tests. Then we propose a Petri net based model, whose aim is to enumerate the possible behaviours, and which takes the structural, the temporal and the semantic constraints into account.**

I. Introduction

We consider real-time applications dedicated to process control, which are modelled by a set of periodic tasks. They are characterized by the existence of temporal constraints, induced by the dynamics of the controlled process. The validation of such applications does not only depend on the functional validation, which we assume to be done, but also on the temporal validation, which consists in proving that the application can be executed in such a way that all the temporal constraints are met.

Our general aim is to develop a complete model-driven methodology, using the formal model of Petri nets for the schedulability analysis of applications whose tasks can contain conditional statements. The tool must explicitly consider these conditional statements, examine the correlations which may exist between the behaviours of the different tasks, when the behaviours depend on the context, and produce a precise description of the effectively possible behaviours of the whole application.

Classically, each task $T_i = < r_i, C_i, D_i, P_i >$ is modelled by four temporal parameters [11]: **(1)** its first release time r_i; **(2)** its worst case execution time (WCET) C_i, which is the highest computation time of the task; **(3)** its relative deadline D_i, which is the maximum acceptable delay between the release and the completion of any instance of the task; **(4)** its period P_i. A task consists of an infinite set of instances (or jobs) released at times $r_i + k \times P_i$ where $k \in \mathbb{N}$.

When a task contains conditional instructions, the WCET corresponds to the longest execution path of the task [15]. It can be obtained either by simulation or by static syntactical analysis of the task code [15].

As to the schedulability analysis, two methods can be considered. A linear model can be deduced from each task, whose duration is the WCET of the initial conditional task. Then the classical scheduling results can be used. But we have proved in [8] that such an analysis can be rather pessimistic, and a yet valid application can be declared infeasible. Moreover, it is not clear how the different real-time primitives are taken into account during the schedulability analysis. Thus we propose an alternative method which consists in considering conditional statements explicitly during the analysis. This means on the one hand to consider conditional statements from the structural point of view: the notion of linear schedule will be replaced by the notion of **scheduling tree** [8]. And on the other hand, it also means to consider some semantic aspects: when the behaviours of different tasks are induced by their environment, the behaviours of these tasks may be correlated, and thus some theoretical combinations of behaviours may be actually impossible.

More precisely, we take into account: **(1)** the **intra-task** relations within a task: we consider the relations between the behaviours chosen in successive conditionals in which a same value is examined; **(2)** the **inter-tasks** relations between several tasks, which result from the fact that if they examine the same value coming from the context to determine their behaviours, these behaviours will be interdependent.

Considering these relations leads to more consistent schedulability results. In addition, the model is closer to the reality, and thus its direct analysis limits the over-sizing of the material, what in turn reduces the costs.

A. General context

In this paper, we consider real-time applications composed of interacting periodic tasks. They run on pre-emptive uniprocessor system. The applications may use real-time primitives: $L(R)$ (lock a resource R), $U(R)$ (unlock a resource R), $S(M)$ (send a message M) and $R(M)$ (receive a message M). We assume that the emission of a message is non blocking, but the reception is.

The tasks are given in a high-level language, and consist of sequential series of general blocks which can be: **(1)** blocks composed of imperative statements: $b(d)$ denotes an imperative block, with execution time equal to d; **(2)** conditional statements: if condition then S_1 else S_2 endif, where S_1 and S_2 are sequential series of general blocks; **(3)** real-time primitives, which assume to have a duration equal to 0, which in fact means that the actual durations are included in the durations of the adjoining blocks.

Two approaches are usually considered in order to solve the schedulability problem: **(1)** the on-line methods where a scheduling algorithm is implemented within the scheduler; **(2)** the off-line methods where a schedule, already computed, is stored within a table, used by the dispatcher. We use here an off-line approach, because in our context, there is no optimal on-line algorithms [6], where an optimal algorithm is an algorithm which computes a valid schedule for any feasible system. The main benefit of off-line strategies comes from the raise of the scheduling power compared to the on-line methods: on-line scheduling algorithms make decisions according to the instantaneous state of the system, meanwhile off-line methods are based on a complete knowledge of the tasks. In addition, these methods provide a precise description of the behaviour of the application. The price to pay is a high complexity, generally exponential in time and space, which can be improved by the use of good heuristics.
We adopt a model-driven approach, based on Petri nets. Indeed, Petri nets are well suited here since they enable to model and simulate the execution of interacting tasks. In addition, we have shown in [10] how the time and the timing constraints can be modelled and verified using some extensions of Petri nets: the maximal firing rule and terminal markings. These extensions will be presented in section 4. Then the labelled terminal marking graph will collect the valid efective behaviours of the application, and thus its analysis will lead to schedulability conclusions.

B. Related works

Several authors have taken the conditional statements into account, in order to get more realistic models, and to obtain more precise schedulability conclusions. The works of [12] propose to consider conditional task graphs to model complex precedence constraints between tasks: the precedence relations are associated to conditions. As to the tasks, they are assumed to be linear, and on-line non pre-emptive scheduling, based on dynamic priorities, is considered. The researches of [14] concern distributed environments. The authors represent a task as a set of conditional parallel branches, with each a probability of being executed. Then the authors propose an incremental off-line construction of schedules, which uses the probability of choosing a branch, as well as the relative deadlines. The works of [5] combines a probabilistic approach and fixed priority strategies, to schedule tasks with uncertain execution times. The worst case execution time C is replaced by a set of worst case execution c_k, with each a probability $P(C = c_k)$ to be obtained. Then, the authors use an approach based on the calculation of the worst response time for fixed priority algorithms. In the recurrent task model [2], each task is split into sub-tasks and is modelled as a directed acyclic graph, associated to a period. Each subtask has its own temporal parameters (execution time and relative deadline), and there are minimal delays between the activations of consecutive sub-tasks. A schedulability analysis based on the demand bound and request bound functions, for either dynamic or static priority strategies is proposed. Finally, in [1], a state transition graph is used to take the conditional statements into account. The nodes describe the observable states of the task, and the edges describe the transitions between two observable states. Then the synchronized product of the graphs modelling the tasks is built. Schedulability conclusions are deduced from the properties of the resulting graph, the considered strategies are priority driven.

C. Our contribution

The precedent approaches concern contexts which are different from our context, and the objectives are not exactly the same as ours. In these papers, tasks do not share critical resources while we aim to consider critical sections; tasks have the same first release time, we want to also consider late released tasks; tasks have due dates equal to the periods while we want to consider constrained deadlines; some papers deal with non pre-emptive systems, we consider pre-emptive ones. In addition, we found no approaches that take the semantic relations between the tests associated to the conditional statements into account. Finally, the schedulability analysis mostly concerns on-line priority-driven strategies. The analysis leads to feasibility conditions, but provides no precise description of the actual behaviour of the application. And when off-line techniques are used, they only produce (linear) schedules where we aim to get scheduling trees, which globally depict the effective behaviours of the application, and can take semantic constraints into account.
If conditional statements may be induced by any kind of tests, we restrict the semantic analysis to the tests on input values of the tasks. For example, they may be thresholding tests ($Temperature < 40$) or identification tests ($x = 5$). The tested values are assumed to be constant during the execution of any instance of the task. They are e.g. parameters IN of an **ADA** procedure that are read each time the task is released. It can also correspond to messages in OASIS with accurate visibility dates [1]. The other tests will only be considered during the modelling of the application. Any test t has a duration $d(t)$. For formalization reasons, we will model a test of duration $d(t) > 1$ as a block b of duration $d(t) - 1$ and a final test of duration 1.
Now, we are here interested in the temporal validation of the application which also requires to verify that the application is correct from the structural point of view. This means that within a task, each unlocked resource has previously been locked and each locked resource is then unlocked. We also require that each sent message is received and each message for which a task waits is actually sent.
The rest of the paper is as follows: in part 2, we present the execution model of a task, we introduce the notion of intra-task incompatibility and we present the temporal model of a task. In section 3, we present the model of an application, and we focus on the incompatibility relations, which model the inter-tasks relations. Finally, in section 4 we present our Petri net based model, which consists of three layers: one for the time management (the clock systems), one for the application itself, and one intermediate layer which supports the semantic

relations.

II. The task models

Classically, a task T_i which contains conditional statements can be modelled as a (labelled) tree (see figure 2). The different branches of the tree correspond to the different possible behaviours of the task. Now, if two blocks are conditional statements which test the same value (e.g. input of the task), the choices of the branch in both conditionals are correlated, and some combinations of choices are impossible in practise. We introduce the incompatibility relation to model this correlation.

A. Execution model of a task

We first introduce some notations: (**1**) $\mathcal{R}_i$ is the set of the resources used by T_i; (**2**) $\mathcal{EM}_i$ ($\mathcal{RM}_i$) are respectively the sets of the messages sent (received) by T_i; (**3**) a_i is an elementary unitary instruction ($b(d)$ corresponds to d instructions a_i); (**4**) $\mathcal{T}est_i$ is the set of the tests of the task T_i; (**5**) $\mathcal{R}esTest_i$ = $\{t_{ij}^+,\ t_{ij}^-\ /\ t_{ij} \in \mathcal{T}est_i\}$ the set of the results of the tests of $\mathcal{T}est_i$.
Then we define an alphabet $\alpha\beta_i$ as
$\alpha\beta_i$ = $\{a_i\}$ $\cup$ $\mathcal{R}esTest_i$ $\cup$ $\{L(R), U(R); R{\in}\mathcal{R}_i\}$ $\cup$ $\{S(M); M{\in}\ \mathcal{EM}_i\} \cup \{R(M); M \in \mathcal{RM}_i\}$.
The task T_i is modelled by a tree whose links are labelled by $\alpha\beta_i$. We also associate an integer $Temp(\eta)$ to each node η, which corresponds to the execution time of the portion of code that has led to the associated state:
$Temp$: $\{nodes\ of\ the\ tree\}$ $\rightarrow$ $\mathbb{N}$ with, $Temp(root)$ 0 and $Temp(\eta)$ = $Temp(Parent(\eta))$ + $dur(label(\ link(\ parent(\eta) \rightarrow \eta)))$
where the duration function dur gives the times required to complete the different actions:
$dur : \alpha\beta_i \rightarrow \mathbb{N}$, $dur(a_i) = 1$, $dur(t_{ij}^+) = dur(t_{ij}^-) = 1$ and $dur(L(R)) = dur(U(R)) = dur(S(M)) = dur(R(M)) = 0$.
The **structural execution tree of a task** is a tree such that each node η holds a key equal to $Temp(\eta)$, the links are labelled by the alphabet $\alpha\beta_i$ and the branches of the tree correspond to the possible behaviours of the task.

B. The intra-task relation and the effective execution tree

The **incompatibility relation** $\mathcal{RIT}_i$ models the incoherences that can exist along a structural behaviour of a task. If $\mathcal{T}estIn_i$ is the set of tests on some input parameters of T_i and $\mathcal{R}esTestIn_i$ the set of results of these tests, $\mathcal{RIT}_i$ is defined on $\mathcal{R}esTestIn_i$ by: $\forall\ t,\ t' \in \mathcal{R}esTestIn_i$, $t\ \mathcal{RIT}_i\ t'$ iff $t \Rightarrow \overline{t'}$ (where $\overline{t}$ = $not(t)$). For example, we have $(x <= 5)$ $\mathcal{RIT}_i$ $(x > 10)$.
A behaviour of a task is coherent and thus effective iff it does not contain two incompatible test results. The execution tree is thus reduced to these only behaviours. Intuitively, a branch of the **effective execution tree** corresponds to a behaviour of the task, i.e. the label of the branch is equal to one effective behviour, and there are as much branches as effective behaviours. In this tree, the outgoing link of a simple node is labeled either by an imperative statement, or by a real-time primitive, or by one alternative of a test, whose other alternative is incompatible with a previously encountered test. And the outgoing links of a double note are labelled by the two alternatives of a test. Let us consider the figure 1. The left tree is the complete tree. The test results $(t < 10)$ and $(x > 10)$ are incompatible, thus the behaviour $Comp_{1,2}$ is removed from the tree. Thus we get the effective tree on the right. A complete formal definition of the effective execution tree can be found in [9].

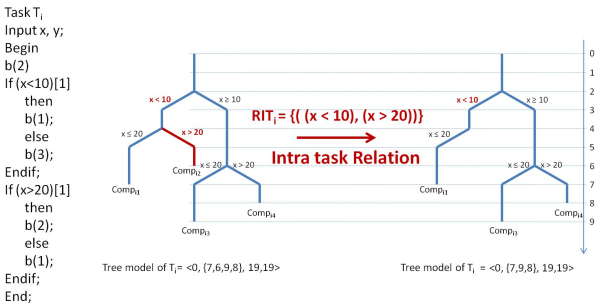

Fig. 1. Effective execution tree of a task. Each link corresponds to one processing time unit

C. Temporal model of a task

The temporal model of a task is derived from the task execution model. The execution time of a task is no longer modelled by its worst execution time as in classical linear approaches, but by a multi set of execution times ζ_i = $\{C_{ik}, k \in 1..l_i\}$ where l_i is the number of effective behaviours of the task. The C_{ik} correspond to the execution times of the effective behaviours of the task. These times are the keys of the leaves of the effective execution tree of the task. The others temporal parameters r_i, D_i and P_i remain unchanged. For the task of the figure 1, the multi set ζ_i is equal to {7, 8, 9}.

III. The model of the application

We consider an application composed of n tasks $(T_1, T_2, \ldots, T_n)$, for which the effective execution trees are known. Again, among the possible behaviours of the application, some may be structurally correct, but incoherent from the semantic point of view.
We aim to capture these incoherent behaviours in order to restrict the temporal analysis to the only effective ones. For that purpose, we first extend the incompatibility relation to $\mathcal{R}esTestIn$ = $\cup_{i=1}^{n} \mathcal{R}esTestIn_i$. Next we derive an incompatibility relation on the set of behaviours of the tasks, which express the inter-task relation. Since this derivation may contain redundancies, we propose a minimization algorithm of the incompatibility relation on the tests, in order to reduce the redundancies, what in turn will improve the efficiency of the implementation step.

A. The general incompatibility relation $\mathcal{RIT}$

The **incompatibility relation** $\mathcal{RIT}$ is defined on $\mathcal{R}esTestIn$ by: $\forall\ t,\ t' \in \mathcal{R}esTestIn$, $t\ \mathcal{RIT}\ t'$ iff: **(1)** let i and j be such that $i \neq j$, $t \in \mathcal{R}esTestIn_i$ and $t' \in \mathcal{R}esTestIn_j$, then $r_i = r_j$ and $P_i = P_j$; **(2)** $t \Rightarrow \overline{t'}$.
Point **(1)** assures that both tests consider the same values, thus it makes sense to look for a possible correlation between their result. It also enables to compare the behaviours of the tasks instance by instance.

B. The inter-task relation $\mathcal{RI}$

The inter-task relation is derived from the relation $\mathcal{RIT}$, what we denote $\mathcal{RIT} \triangleright \mathcal{RI}$. The relation $\mathcal{RI}$ is defined on the set of effective behaviours of a task in the following way: let $Comp_{ik}$ and $Comp_{i'k'}$ be two effective behaviours of the tasks T_i and $T_{i'}$, with $i \neq i'$, then $Comp_{ik}\ \mathcal{RI}\ Comp_{i'k'}$ iff $\exists\ t \in \mathcal{R}esTestIn_i$, $t' \in \mathcal{R}esTestIn_{i'}$ such that t appears in $Comp_{ik}$, t' appears in $Comp_{i'k'}$ and $t\ \mathcal{RIT}\ t'$.
The following example illustrates this notion. We consider a real-time system S composed of two tasks T_1 and T_2 described on figure 2. The relation $\mathcal{RIT}$ is equal to $\{((x \geq 10)_{T_1}, (x < 5)_{T_2}), ((y \geq 8)_{T_1}, (y < 4)_{T_2})\}$ (where $(x \geq 10)_{T_1}$ means that the test result $(x \geq 10)$ comes from the task T_1). From $(x \geq 10)_{T_1}\ \mathcal{RIT}\ (x < 5)_{T_2}$, we derive the relations $Comp_{12}\ \mathcal{RI}\ Comp_{21}$, $Comp_{12}\ \mathcal{RI}\ Comp_{22}$, $Comp_{13}\ \mathcal{RI}\ Comp_{21}$ and $Comp_{13}\ \mathcal{RI}\ Comp_{22}$, and from $(y \geq 8)_{T_1}\ \mathcal{RIT}\ (y < 4)_{T_2}$ we derive $Comp_{13}\ \mathcal{RI}\ Comp_{22}$. Now, we can notice that the relation $Comp_{13}\ \mathcal{RI}\ Comp_{22}$ is generated twice. And since the relation induced by $(y \geq 8)_{T_1}\ \mathcal{RIT}\ (y < 4)_{T_2}$ is included in the relation induced by $(x \geq 10)_{T_1}\ \mathcal{RIT}\ (x < 5)_{T_2}$, we can suppress the relation $(y \geq 8)_{T_1}\ \mathcal{RIT}\ (y < 4)_{T_2}$ without changing the induced relation.
The purpose of the section III-C is thus to define a minimal relation $\mathcal{RIT}_{min}$ which induces the same relation $\mathcal{RI}$.

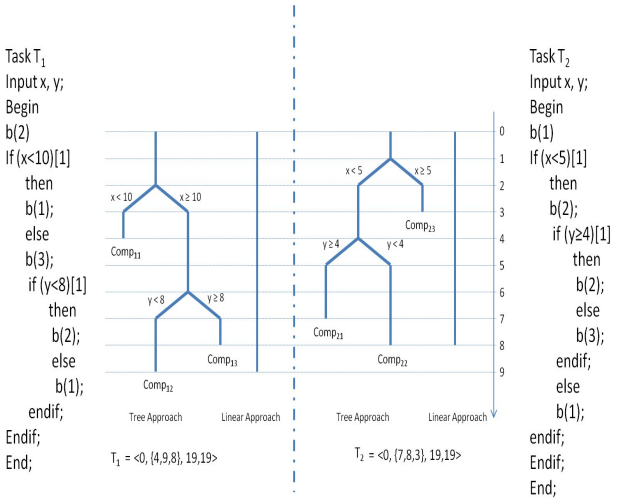

Fig. 2. The system S = $< T_1, T_2 >$

C. Minimization of $\mathcal{RIT}$

Let $\mathcal{RIT}$ be an incompatibility relation on $\mathcal{R}esTestIn$. A relation $\mathcal{RIT}_{min}$ minimizes $\mathcal{RIT}$ if it verifies the following properties: **(1)** $\mathcal{RIT}_{min} \subseteq \mathcal{RIT}$; **(2)** $\mathcal{RIT}_{min} \triangleright \mathcal{RI}$; **(3)** if we remove any element from $\mathcal{RIT}_{min}$, the generated relation $\mathcal{RI}'$ is strictly included in $\mathcal{RI}$ ($\mathcal{RI}' \subsetneq \mathcal{RI}$).
The following algorithm (A1) computes a minimized relation $\mathcal{RIT}_{min}$ from $\mathcal{RIT}$:
(1) $\mathcal{RIT}_{min} := \emptyset$: the relation $\mathcal{RIT}_{min}$ is initially empty;
(2) for every pair $(t^s_{im}, t^r_{jk}) \in \mathcal{RIT}$, there exist Pre_i and Pre_j such that $Pre_i.t^s_{im}$ and $Pre_j.t^r_{jk}$ are the prefixes of at least one effective behaviour of T_i and T_j respectively. They correspond to what the tasks have done before they reach t^s_{im} and t^r_{jk} respectively:
(2-a) if $\exists\ t^{s'}_{im'} \in \mathcal{R}esTestIn_i$ with $Pre_i = u.t^{s'}_{im'}.v$ and $(t^{s'}_{im'}, t^r_{jk}) \in \mathcal{RIT}$, then reject (t^s_{im}, t^r_{jk});
(2-b) conversely if $\exists\ t^{r'}_{jk'} \in \mathcal{R}esTestIn_j$ with $Pre_j = u'.t^{r'}_{jk'}.v'$ and $(t^s_{im}, t^{r'}_{jk'}) \in \mathcal{RIT}$, then reject (t^s_{im}, t^r_{jk});
(2-c) if $\exists\ t^{s'}_{im'} \in \mathcal{R}esTestIn_i$, and $\exists\ t^{r'}_{jk'} \in \mathcal{R}esTestIn_j$ with $Pre_i = u.t^{s'}_{im'}.v$ and $Pre_j = u'.t^{r'}_{jk'}.v'$, and with $(t^{s'}_{im'}, t^{r'}_{jk'}) \in \mathcal{RIT}$, then reject (t^s_{im}, t^r_{jk});
(2-d) else, add (t^s_{im}, t^r_{jk}) to $\mathcal{RIT}_{min}$: $\mathcal{RIT}_{min} := \mathcal{RIT}_{min} \cup \{(t^s_{im}, t^r_{jk})\}$.
The idea is that when a pair of behaviours are incompatible because of several couples in the relation $\mathcal{RIT}$, we only keep the upstream couples. Couples which appear later are useless for the construction of $\mathcal{RI}$. We have the following result [9]:

Proposition 1: The relation computed by the algorithm A1 minimizes the relation $\mathcal{RIT}$.

However, minimizing $\mathcal{RIT}$ reduces the redundancies, but some may still remain. This corresponds to cases where the incompatibilities induced by two distinct pairs of $\mathcal{RIT}_{min}$ have a non-empty intersection, but each also contains specific pairs.
The next step consists in the validation of the application. Our aim is to propose a methodology for the schedulability analysis which also takes the incompatibility relations into account, so that the feasibility conclusions are based on the only coherent behaviours. For that aim, we use a model-driven approach, based on Petri nets. We extend the approach proposed in [10] for linear tasks. The proposed method consists in modelling the application, and then to enumerate the possible valid schedules (by means of a marking graph). The simulation of the net must provide fully valid behaviours of the application, where validity concerns as well the structural validity as the temporal. The initial model is composed of two parts: the structural net which models the application, and the clock system, which models the time and together with the initial marking and the terminal set, it also models the temporal constraints. This model has then been enlarged to tasks with conditional statements by [4], but the coherence of the behaviours has not been taken into account.
Our aim is now to adapt the Petri net model so that it also

models the incompatibility relations. The model will then again enable to enumerate the valid scheduling trees of the application. The aim of section IV is to present our model.

IV. Our Petri net based approach

The model that we propose is meant as a tool for the validation of the application: its analysis will provide scheduling trees which will be both temporally and structurally valid, and which will obey all the coherence rules given by the incompatibility relations. The general structure of our net is given in figure 3.

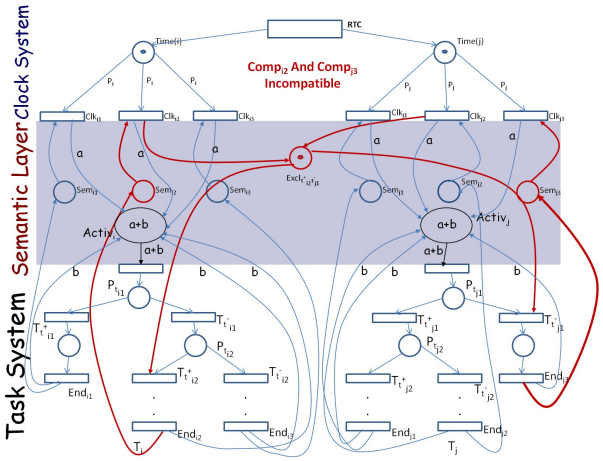

Fig. 3. General Petri net model of an application of two tasks with one incompatibility. The net is composed of three layers.

The net is composed of three layers. The task system models the different tasks and their interactions in a classical way. Each transition has the place processor, which contains one token since we consider uniprocessor systems, as input and output. The place *Processor* is not represented on the figure for clarity reasons. Then we have a temporal system where a source transition model the time, and local clocks manage the periodic reactivation of the tasks. These two parts were present in the previous models. We have added a new layer, which aims to assure the coherence of the behaviour of the application. Finally, the net runs under the maximal firing rule: only maximal (as to the inclusion relation) enabled transition sets fire.

A. The structural layer

Each task is modelled by a net which is a direct translation of the tree given figure 2. In order to reduce the number of places and transitions, an imperative block whose duration is 1 is represented by 1 place and 1 transition, a block whose duration is 2 by 2 places and 2 transitions, and a block whose duration is greater than or equal to 3 by 3 places and 3 transitions (see figure 4). A conditional statement induced by a test t_{ij} is modelled by a place with two output transitions $T_{t_{ij}^+}$ and $T_{t_{ij}^-}$. Messages are modelled by means of mailboxes. The last transition before the emission of a message produces a token in the mail box, and the first transition of a block following a reception consumes one token from the mailbox. Finally, there is one place for each critical resource, whose initial marking equals the number of instances of the resource. Here again, when a resource is locked (resp. unlocked), the previous (resp. following) block ends (resp. starts) with the consumption (resp. production) of a token from (resp. in) the resource place. Each task T_i starts with a coloured place $Activ_i$, to which two colours **a** and **b** are associated. When the place holds a token **a**, it means that a new instance has been activated, but has not yet started execution, and when it holds a token **b**, it means that the previous instance has completed execution. Finally, each last transition End_{ik} ($k \in 1 \dots l_i$) of the task T_i (there are as many such transitions as effective behaviours) verifies $W(End_{ik}, Activ_i)$= **b**. Then the task can start execution only if this place holds a token **a** and a token **b**. The initial marking is defined by: if $r_i = 0$ then $M_0(Activ_i)$ = **a** + **b**, else $M_0(Activ_i)$ = **b**.

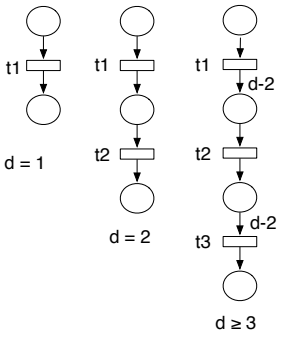

Fig. 4. Petri net model of a block b(d)

B. The temporal layer

The temporal systems consists of a source transition RTC and of a local clock for each task. RTC is the global clock of the system. Since the net runs under the maximal firing rule, it is fired each time a transition set is fired. Thus each firing of RTC models a time unit. Let T_i be a task. Its local clock is composed of a place $Time(i)$, which counts the elapsed time units since the last reactivation of the task, and of k_i transitions Clk_{ij}. If T_i is linear, then $k_i = 1$, if it admits l_i effective behaviours, then, if its first release time equals 0 then $k_i = l_i$ (j $\in 1 \dots l_i$) else $k_i = l_i + 1$ (j $\in 0 \dots l_i$). And finally we have $W(Time(i), Clk_{ij}) = P_i$ and $W(Clk_{ij}, Activ_i)$ =**a**. The firing of a transition Clk_{ij} models the reactivation of the task. Exactly one transition $Clk_{i,j}$ fires each period.
The initial marking of the place $Time(i)$ is defined by: $M_0(Time(i))$ = $P_i - r_i + 1$ if $0 < r_i < P_i$ and
$M_0(Time(i))$ = 1 if r_i = 0.
Finally, to consider the relative deadlines, we define a terminal set $\mathcal{I}$: a transition set can fire only if the resulting marking belongs to the terminal set. The terminal set is defined by: $M \in \mathcal{I} \Leftrightarrow$ (**1**) $M(Time(i)) > D_i \Rightarrow M(Activ_i)$ = **b** and (**2**) $M(Time(i)) = 1 \Rightarrow M(Activ_i)$ = **a+b** or $M(Activ_i)$ = **b** and (**3**) $M(Time(i)) \leq P_i$.

Point 1. means that after deadline, the pending instance must be completed. Point 2. means that at each reactivation date, the task must have completed the previous instance. The second case ($M(Activ_i)$ = **b**) corresponds to the initialization of the net when $r_i = P_i - 1$. And finally, point 3. assures that the reactivations are periodic, with period P_i.

C. The semantic layer

The aim of the semantic layer is to implement the incompatibility relation so that during the simulation, it is impossible for two tasks to choose, during the same period, two incompatible behaviours. For that purpose, we implement the incompatibility relation RIT_{min} which we know to be minimal. For each pair of incompatible tests $(t^s_{ij}, t^{s'}_{i'j'})$ in RIT_{min} with $i < i'$, we create a place $Excl_{t^s_{ij}t^{s'}_{i'j'}}$, which initially contains one token.

Then we have $W(Excl_{t^s_{ij}t^{s'}_{i'j'}}, T_{t^s_{ij}})$=$W(Excl_{t^s_{ij}t^{s'}_{i'j'}}, T_{t^{s'}_{i'j'}})$=1: the incompatible tests are in mutual exclusion. Then we have to restore the mutual exclusion token at the end of the tasks. We thus add a place Sem_{ik} for each behaviour $Comp_{ik}$ of a task T_i, initially empty. Then we have $W(End_{ik}, Sem_{ik})$=1= $W(Sem_{ik}, Clk_{ik})$. In this way, we mark the chosen behaviour. At the time of the next reactivation on the task, the transition Clk_{ik} will be fired. It must then refill the mutual exclusion places that have be emptied by the previous instance. For that aim, we consider the list L_{ik} of the results of tests used along $Comp_{ik}$, which are associated to a mutual exclusion place. For each t^s_{ij} in L_{ik}, for any place $Excl_{t^s_{ij}t^{s'}_{i'j'}}$, we have $W(Clk_{ik}, Excl_{t^s_{ij}t^{s'}_{i'j'}})$= 1 and for any place $Excl_{t^{s'}_{i'j'}t^s_{ij}}$, we have $W(Clk_{ik}, Excl_{t^{s'}_{i'j'}t^s_{ij}})$= 1.

Now, if the task is not linear, and is late released ($r_i > 0$), we add a last place Sem_{i0}, which initially contains one token. It verifies $W(Sem_{i0}, Clk_{i0}) = 1$. It assures the first release of the task, when no previous instance has been processed, thus no refilling is required. The figure 5 illustrates this case.

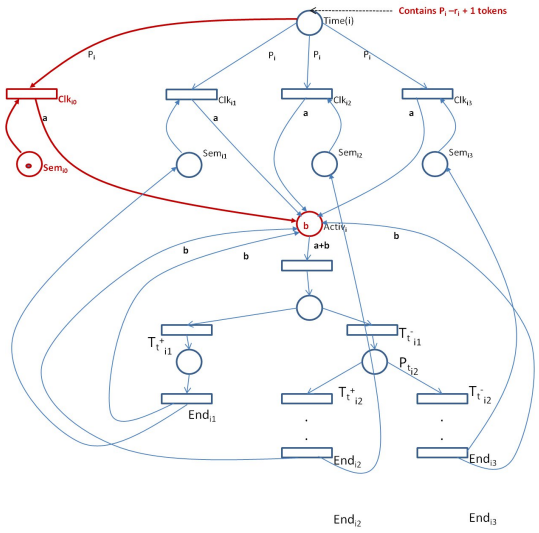

Fig. 5. Semantic layer of task T_i with $r_i > 0$

D. The idle task

In general, when $U < 1$, the processor remains idle part of the time ($P(1 - U)$ time units each hyperperiod P = lcm($P_1, \ldots, P_n$)). If only conservative scheduling is considered, the idle times take place only when there are no pending tasks. But in the context of the dependent tasks, conservative strategies are not optimal, and it is sometimes necessary to introduce idle times even when there are pending tasks in order to avoid further temporal faults [3]. For that aim, we complete the model and add a new task, the idle task T_0, which models the processor inactivity.

But, if the tasks have non deterministic execution times, the number of idle times is neither deterministic. Thus the duration of the idle task cannot be statically determined, it has to be adjusted dynamically. The least number of idle time units is $P(1 - U_{max})$ where U_{max} is obtained using the WCET for each task. Now, U_{max} may be greater than 1. In such a case, if the semantic coherence is not considered, the application is simply non feasible. But if we take the coherence into account, things may be slightly different. Indeed, it can be the case that the behaviours corresponding to a utilization factor greater than 1 correspond in fact to non effective behaviours. Thus, provided that $U_{min} < 1$, the application must be further analysed.

Thus, the problem is to determine the temporal parameters and the behaviour of the idle task. We set $r_0 = 0$ and $D_0 = P_0 = P = lcm(P_1, \ldots, P_n)$. Then initially C_0 is set to $P(1 - U_{max})$. Each time a task chooses an alternative in a conditional, the duration of the longest still possible behaviour is compared to the duration that has been taken into account for the current value of C_0. If it is shorter, then C_0 is increased by the difference.

To model the idle task, we first add $Time(0)$ and Clk_0 to the temporal system (defined as for any other task). Then we must consider two cases:

(1) if $P(1 - U_{max}) \geq 0$, it is modelled by one place $Activ_0$ and one transition $Idle_0$. Now, let $T_{t^s_{ij}}$ be a transition corresponding to the alternative of a test. Let d_1 be the longest execution time of the behaviours containing this test. If $T_{t^s_{ij}}$ is the first test in these behaviours, then d_2 is equal to the WCET of the tasks T_i, else, let $T_{t^{s'}_{ij'}}$ be the test just before $T_{t^s_{ij}}$. Then d_2 is the longest duration of the behaviours which contain $T_{t^{s'}_{ij'}}$. Finally, we set $v = d_2 - d_1$. If $v > 0$, then we have $W(T_{t^s_{ij}}, Activ_0) = v$, else there is no arc between $T_{t^s_{ij}}$ and $Activ_0$. And we have $W(Clk_0, Activ_0) = P(1 - U_{max})$, and $W(Time(0), Clk_0) = P$: the idle task is reactivated each hyperperiod;

(2) we have $P(1 - U_{max}) \leq 0$ but $P(1 - U_{min}) \geq 0$. In this case, we create 3 places ($Activ_0$, $Lateness$ and $Error$) and 3 transitions ($Idle_0$, $Empty$ and Clk'_0). Initially, $Activ_0$ and $Error$ are empty, and $Lateness$ holds $d = |P(1 - U_{max})|$ tokens. We also have $W(Clk_0, Lateness) = d$. The transition $empty$ is synchronized [13] on the always occurring event (e), and we have $W(Lateness, Empty) = W(Activ_0, Empty) = 1$. Each time a token is produced in the place $Activ_0$, if

there is still lateness, it is immediately consumed, and the lateness decreases of one unit. The arcs between the task system and $Activ_0$ are defined as in the case **1**. Finally, we have $W(Time(0), Clk'_0) = W(Time(0), Clk_0) = P$, $W(Lateness, Clk'_0) = 1 = W(Clk'_0, Error)$: if there is still lateness when the idle task is reactivated, these means that for the precedent hyperperiod, the actual value of U was greater than 1, thus a temporal fault had occurred. Finally, there is an inhibitor arc between $Lateness$ and Clk_0. Thus the transition Clk_0 which reactivates the idle task can only fire if the behaviour of the application during the previous hyperperiod corresponded to a value of U less than 1. And we finally add the property $M(Error) = 0$ in the definition of the terminal set. The figure 6 illustrates the modelling of the idle task. The validation of our modelling relies on the

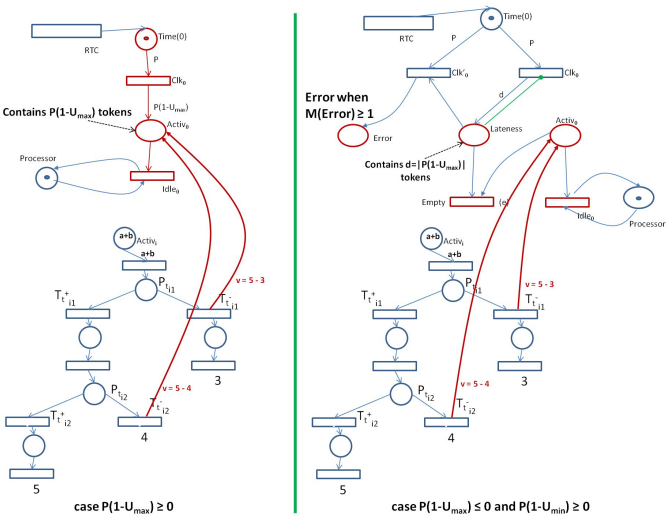

Fig. 6. Modelling of the idle task

next two results:

- During a same period, two tasks cannot execute two incompatible behaviours.
- Each time a task is reactivated, it disposes of all its effective behaviours, i.e. no possibly is missed.

The detailled proof of this result can be found in [9]. Thus simulation will only produce valid [10] and effective behaviours of the application.

V. Conclusion and perspectives

We have presented a formal model for real-time applications which takes the structure of the tasks, the temporal constraints and the semantic coherence constraints into account. Then, the analysis of the net, by means of the terminal marking graph will provide schedulability conclusions, and, if any exist, valid scheduling trees will be produced. One benefit of our approach is that it enables to assure the structural, the temporal and the semantic validities together.

Our model is got from the model proposed in [10] [4] to which a semantic layer has been added. For that aim, we have first defined an incompatibility relation induced by the semantics of the tests, and we have minimized this relation in order to minimize the number of places and transitions of the semantic layer. The further investigations will concern the extraction of the scheduling trees, and the associated heuristics which could help to limit the complexity of the extraction.

References

[1] Aussaguès C. and David V.: A method and a technique to model and ensure timeless in safety critical real-time systems. ICECCS'98, (1998)

[2] Baruah S.: Dynamic and static priority scheduling of recurring real-time tasks. Real-time systems, Kluver Academic Publishers, 93–128, (2003)

[3] Choquet-Geniet A. and Grolleau E.: Minimal Schedulability Interval for Real-Time Systems of Periodic Tasks with Offsets. Theoretical of Computer Sciences, 310(10), 117–134, (2004)

[4] Choquet-Geniet A. and Pailler S.: Off-Line scheduling of real-time applications with variable duration tasks. 7^{th} DEDS, 373–378, (2004)

[5] Cucu L.: Probabilistic real-time schedulability analysis: from uniprocessor to multiprocessor when the execution times are uncertain. Report, (2009)

[6] Dertouzos M.L. and Mok A.K.: Multiprocessor on-line scheduling of hard-real time tasks. IEEE T.S.E, 15(12), 1497–1506, (1989)

[7] Fotsing C. and Geniet A. and Vidal-Naquet G.: A realistic model of real-time systems for efficient scheduling. 33^{rd} IEEE SEW, (2009)

[8] Fotsing C. and Geniet A. and Vidal-Naquet G.: Tree scheduling versus sequential scheduling. CARS@EDCC, ACM, Valence, Espagne, (2010)

[9] Fotsing C.: Intégration d'éléments sémantiques dans l'analyse d'ordonnançabilité des applications temps-réel. Phd Thesis, (2012)

[10] Grolleau E. and Choquet-Geniet A.: Off-line computation of real-time schedules using Petri nets. Discrete Events Dynamic Systems (DEDS), Kluwer Academic Publishers, vol12(3), 311–333, (2002)

[11] Liu C.L. and Layland J.W.: Scheduling algorithms for multiprogramming in real-time environnement. J of the ACM, 20(1), 46–61 (1973)

[12] Lombardie M. and Milano M. and Ruggiero M. and Benini L.: Stochastic allocation and scheduling for conditional task graphs in multi-processor systems-on-chip. Journal of scheduling, 13(4), 315–345, (2010)

[13] Moalla M. and Pulou J. and Sifakis J.: Synchronized Petri nets: A Model for the Description of Non-autonomous Systems. Proc. of the 7^{th} MFCS, 374–383, LNCS 64 (1978)

[14] Santoshkumar I. and Manimaran G. and Siva C.: Static scheduling of object-based real-time task with probabilistic conditional branches in distributed systems. Parallel and Distributed Real-time, USA, (1998)

[15] Wilhelm R. and Engblom J. and Ermedahl A. and Holsti N. and Thesing S. and Whalley D. and Bernat G. and Ferdinand C. and Heckmann R. and Mitra T. and Mueller F. and Puaut I. and Puschner P. and Staschulat J. and Stenstrom P.: Scheduling algorithms for multiprogramming in real-time environnement. The worst case execution time problem–Overview of methods and survey of tools. ACM, 7(3), (2008)