ORDONNANCEMENT HORS LIGNE
D’APPLICATIONS TEMPS RÉEL
COMPORTEANT DES TÂCHES À DURÉES
VARIABLES.

OFF-LINE SCHEDULING OF REAL
TIME APPLICATIONS WITH
VARIABLE DURATION TASKS.

Stéphane Pailler 1 Annie Choquet-Geniet 2

LACS(LISI) - ENSMA
Téléport 2 - 1 Avenue Clément Ader BP 40109
86960 FUTUROSCOPE cedex, FRANCE

Résumé: Nous proposons une méthode d’analyse d’ordonnancement pour des
applications temps réel fortement couplées. Nous montrons que les variations
de durée d’exécution des tâches peuvent mettre en péril la stabilité du système
contrôlé. Pour réduire les risques d’erreurs et donc augmenter la fiabilité, nous
prenons en compte explicitement les instructions conditionnelles présentes dans
le code des tâches. Dans un premier temps, nous adaptions le modèlle temporel
de tâches pour ce contexte, puis nous modélisons ces applications à l’aide
de réseaux de Petri autonomes fonctionnant sous la règle de tir maximal et
munis d’ensembles terminaux. Nous définissons deux concepts d’ordonnancabilité :
l’ordonnancabilité locale et l’ordonnancabilité globale et nous définissons le concept
de graphe d’ordonnancement. Enfin, nous montrons comment obtenir un graphe
d’ordonnancement à partir du graphe d’ordonnancabilité globale.

Mots-clés: Ordonnancement hors-ligne, Réseau de Petri, instructions condition-
nelles, graphe d’ordonnancement.

Abstract: We propose a model oriented scheduling methodology for highly coupled
real time applications. We show that variations in the computation times of tasks
may hazard the safeness of the controlled process. For the sake of reliability we take
conditional instructions of tasks’ code explicitly into account, in order to reduce the
potential failures. We first adapt the task’s temporal model to this context, then
we model applications using autonomous Petri nets which run under the earliest
firing rule with terminal marking set. We define two concepts of schedulability:
the local schedulability and the global one and we define the concept of scheduling
graph. Finally, we show how to obtain a scheduling graph from the graph of global
schedulability.

Keywords: off-line scheduling, Petri Nets, conditional instructions, scheduling
graph.
1. INTRODUCTION

A constantly increasing number of either autonomous or assisted processes (cars, nuclear power station, plane, space probe) are controlled by real time systems interacting with their own environment. Yet these systems are not safe from failures that may put human beings’ lives in danger or jeopardize substantial economic value. That is why, it is necessary to develop methods to minimize those risks by validating the real time applications that control the processes. In most cases, the failures come from disregarded temporal constraints. Indeed, a real time application is defined as a multitask application in which each task is subjected to the inherent temporal constraints of the controlled processes. Thus, the main problem will be to choose a scheduling policy which would distribute each active job of task on the processor(s) so that the temporal constraints are respected. Two techniques can achieve that. The first one consists in executing a scheduling algorithm, which aim is to define at every moment the task to execute: it is the online scheduling. The second one consists in a previous analysis of the application and of the computation of valid schedules of the task executions within a temporal window of suitable size. Whatever the method, the schedulability analysis relies on a temporal model of the task to execute: it is the online scheduling.

2. SCHEDULING INSTABILITY PHENOMENA

In most real time applications, to find out a scheduling policy respecting constraints of the system proves to be a major issue. As an answer to that problem, we generally rely on a temporal model (Liu and Layland, 1973) that describes the temporal constraints of each task of the application in order to find and then validate a scheduling policy. But among the parameters that model a task, the execution time is probably the most complex to estimate. Those durations enable us to quantify the time allotted to the CPU to process each job of the tasks. It is thus an upper bound which is reckoned up from the task’s program code and the features of the hardware. Each tool of scheduling analysis being based on a pattern of the tasks, that upper bound (Worst Case Execution Time) represents then an input data whose accuracy of estimation will determine the quality of the scheduling policy’s validation. Therefore, researches on the WCET must take the two following preconditions into account (Puschner and Koza, 1989). First, the reckoned WCET must be safe, i.e. it has to be an upper bound whatever the execution context may be. Second, the estimation of WCET has to be as accurate as possible or instability phenomena may occur as shown in figure 1: the reduction of the duration of one of the job of the task can cause a temporal fault in spite of the validation of the scheduling policy (Deadline Monotonic 3 ). This kind of instability phenomena had been highlighted by (Graham, 1969).

1 pailler@ensma.fr
2 ageniet@ensma.fr
3 DM (Deadline Monotonic) refers to the priority allocation algorithm: the task with the shortest deadline (the smallest value of D) is assigned the highest priority.
3. CONDITIONAL TASKS AND SCHEDULING

3.1 The extended temporal model

We consider real time applications compounded of synchronous periodic tasks \((r_i = 0)\). However let’s point out that our method can take asynchronous periodic tasks into account, but it does at the cost of a combinatory explosion as far as the accessibility graph construction and the scheduling graph extraction are concerned. Optimization criteria need to be found to solve the problem.

Our objective is to refine the functional block description in order to take conditional instructions into account, which requires to modify the temporal model of tasks. We extend the Liu-Layland model in order to describe all the durations of the conditional branches: a task is represented by three deterministic parameters (the date of the first release, the relative deadline and the period) and by a multiset \(E\) of durations, each one corresponding to a possible behavior of the task. This new pattern of tasks is very close to the multi-frame model of (Mok and Chen, 1996). However, unlike the later, we don’t need the knowledge of the series of execution time for each task since each job of task may be processed with whatever execution time of the multiset. Let us note that if there is no conditionals instruction, the multiset \(E\) contains a single duration, which corresponds to the usual model.

In this temporal model, we have left the association between the real time primitives and the execution times aside. However, this information is not lost since it will be entirely integrated in the modeling of the application by Petri nets.

3.2 Scheduling graph

Considering on line scheduling strategies, the presence of conditional instructions matters only for the application validation, but most of these on line algorithms can be used without any adaptation even when conditional instructions are involved. On the contrary, off line scheduling strategies must be adapted. Indeed, if we consider tasks without conditional statements (durations are deterministic), the objective of an off line scheduling strategy is to build one or more valid schedules. This cannot be used if conditional statement are involved, because the various choices in conditional instructions will induce different behaviors of the application, which could not be described in a single schedule. In order to describe the behavior of a conditional application, we introduce scheduling graphs: a scheduling graph is a graph where each branch corresponds to a schedule of the application obtained by considering for each task only one of their paths. We call split sub-application each of these applications.

We then reformulate the scheduling problem, and define two concepts of schedulability:

- An application is said to be locally schedulable if each of its split sub-application is...
schedulable, i.e., there is for each one of them a valid schedule,

- An application is said to be globally schedu-
  lable if there is at least one valid scheduling
  graph, i.e., all deadlines are met whatever the
  conditional choices.

A globally schedulable application is obviously
locally schedulable (but the converse does not
hold if the application uses real time primitives).
This comes from the fact that each branch of a
scheduling graph is a valid schedule for one split
sub-application.

4. MODELLING BY MEANS OF PETRI NETS

The schedulability analysis methodology which we
proposed relies on (Choquet-Geniet et al., 1996;
Grolleau and Geniet, 2000; Grolleau, 1999). It
consists in modeling the application by a con-
strained marking colored Petri nets, under the
earliest firing rule with terminal marking set.
The feasible schedules are then obtained through
the construction of the state graph. The model
includes two parts: the task system which is
obtained through a classical modeling of the
functional description of the application, and a
clock system which models time (see figure 2).
We have adopted a discrete modeling of time
(Kopetz, 1992; Fohler, 1994): an external clock
(RTC) counts the time in each place Time, which
acts as a local clock used to release periodically
the related tasks. Let us note that each transition
corresponds to an action of duration one time
unit, and that all transitions of the task system
are in competition for obtaining the processor. It
follows that, according to the earliest firing rule,
at each time, one single valid transition is fired.

It results that only conservative schedules\(^4\) can
be produced. But for the sake of scheduling power
since there is no optimal conservative scheduling
algorithm in a scheduling context with resource
and synchronization, we need also to consider
non work conserving schedules. For that purpose,
we introduce in the task system a further task,
called idle task, which models the inactivity of
the processor. When transitions of this task fire,
the processor remains idle what allows to produce non
work conserving schedules too. The computation
time of the idle task is P(1-U)\(^5\). Moreover, the
knowledge of the number of idle times during the
metaperiod P help us for the detection of temporal
fault during marking graph construction.

\(^4\) in a conservative (or work-conserving) schedule, a task
never intentionally waits

\(^5\) where U is the processor utilization factor of the appli-
cation and P is the LCM of the periods of the tasks

5. OFF-LINE ANALYSIS

5.1 Construction of global accessibility graph

Once the model constructed, we are interested in
its exploitation in order to obtain valid scheduling
graphs. From this Petri net, we only need to build
a P depth final marking graph since the appli-
cation is cyclic with a period of P (metaperiod).
This means that, we have the same states at a
t-depth and at t+kP (k>0) in the marking
graph. We then remove from the marking graph

![Fig. 3. Idle time exchange between the Idle task
and a Conditional task with maximal dura-
tion policy. The multiset of the Conditional
 task is E={6,7,5}, so the number of token of
the Idle task is calculated with max(E)=7.
Each computation of the Conditional task
with a different duration increases the num-
er of idle times.

of the idle task becomes variable too. In order
to match the previously used rule (a transition of
the task system fires each unit time), we have
adapted our model so that the duration of the
idle task could be dynamically modified during
the simulation of the Petri Nets. To allow these
time units exchanges, the idle task is represented
by a special place, holding as many tokens as idle
time has been assumed within the metaperiod.

For that purpose, we have chosen a method which
has the advantage to minimize the number of
backtracking operations during the construction
of the accessibility graph. It consists in assuming
that the maximal durations are always chosen
within the multiset. Consequently, the duration
of the idle task is assumed to be minimal. When
a conditional task is computed, if the conditional
test doesn’t correspond to the maximum execu-
tion path, the duration of the idle task has to be
increased of the difference between the worst and
effective duration (cf. figure 3).

![Diagram showing the transition of time between the Conditional task
and the Idle task, with the notation of tokens and places in the Petri net.
5.2 Extraction of a scheduling graph

The last step of the analysis consists then in the extraction of the scheduling graphs. The main difficulty is to deal with the size of the scheduling graph which depends on the number of the possible execution paths of the tasks.

Property: We consider n tasks defined by 

\( < r_i, E_i, D_i, T_i > \) for 

1 \( \leq i \leq n \), which do not use real time primitives, so:

- The number of possible behaviors in a scheduling graph is

\[
\prod_{i=1}^{n} (\#E_i)^{\frac{P}{\tau}}
\]

- The global accessibility graph which contains all of the possible scheduling graphs is entirely hold in a hyper-cube which has

\[
\sum_{i=1}^{n} (\Phi(E_i)) + 1 \text{ dimensions},
\]

\# represents the number of elements of the multiset \( E_i \),

\( \Phi(E_i) \) is the maximum number of interleave conditional tests from all the behaviors of the task \( i \).

Nevertheless, we can note that in the case of highly coupled applications, the size of the graph decreases significantly. For a better result, we can choose to act in two ways. First, when we build the global accessibility graph, we can apply some constraints like successor constraints or define a threshold for jitter or response time for a task or a set of tasks (Grolleau, 1999). These kind of criterion are called \( a \) priori constraints and allow us to reduce the size of the global accessibility graph which is the most important drawback of our method. The second method is used to choose
from this reduced graph a optimized scheduling graph with some precise criterion: the better response time, the minimum jitter...(cf. figure 4). Anyway, both methods must be refined in the case of conditional task. When a task could have several behaviors, we can choose for example to only take into account the best response time for all of the different behaviours or at the opposite the worse. The result will be surely different. By default, we have choosen to consider the most pessimistic way for all the criterion.

Now that we can schedule periodic tasks with variable duration and by that way control the number of idle time in a scheduling graph, we have the possibility to optimize their position in the schedule. In the case some aperiodic tasks occur, we can look into the positioning of each idle time to simulate the on-line algorithms like Polling Server, Deferrable Server, Priority Exchange (Lehoczky et al., 1987).... It can be made by a special criteria that uses a weighting function (like sinusoidal function with a period equal to the period of the simulated server). It forces each idle time to occur periodically and then may reduce their response time. We can then extract scheduling graphs that can be used for mixed scheduler in the case of both periodic and aperiodic tasks occurrences.

![Graphs](image)

Fig. 4. (a) The global d’accessibilit graph; (b) a reduce graph with constraint of successor; (c) a scheduling graph which is extract from b.

6. CONCLUSION

We have proposed a method for scheduling analysis of highly coupled real time applications which uses a Petri net modeling. We have shown that the variations of execution time have to be taken into account. Indeed, the use of the WCET does not give inevitably good results. The indeterminism of the current processors, the use of the optimized compilers, the programming with particular instructions which increases the number of different behaviors from each task, makes it impossible to get an exact measurement of the execution times (Puschner, 2002). In spite of these difficulties, the analysis of WCET and schedulability strongly interests industrialists (Nielsen et al., 2002). It is the reason why we have to try to benefit from works on the WCET to integrate in our method the relevant conditional instructions that can be found in the code of tasks. After extending the temporal model of (Liu and Layland, 1973) to conditional tasks, we have defined two concepts of schedulability, coming directly from the implications of conditional tasks: local and global schedulability. We have shown that there is no equivalence between both. In addition, we have adopted the concept of idle task, because the duration of this task is closely related to the duration fluctuations of the application tasks. We have proposed a modeling of the idle task that allows on one hand a production of non work conserving schedules and on the other an optimization in the construction of the marked graph (the idle task enables to considerably reduce the number of non valid states). We have shown how to extract a scheduling graph with some optimization criteria for response time or jitter minimizing. The next step of this work is to extend our method to sporadic tasks, induced by conditional blocks within which new tasks are requested. Our method must be adapted in order to take into account this new kind of tasks model. Our goal is then, thanks to efficient idle time managing, to define a mixed scheduler for both periodic and sporadic tasks.

7. ACKNOWLEDGMENTS

A first version of this paper was published at IFAC Workshop on Discret Event Systems, WODES’04, 22-24 September 2004.

REFERENCES


