Design of real-time periodic control systems through synchronisation and fixed priorities

Daniel Simon and Fanny Benattar
INRIA Rhône-Alpes, Projet POP ART, 655 avenue de l'Europe, Montbonnot
38334 Saint-Ismier Cedex, FRANCE
phone : 33 4 76 61 53 28   fax : 33 4 76 61 54 77
Daniel.Simon@inrialpes.fr

February 20, 2004

Abstract

Control systems are often designed using a set of cooperating periodic modules running under control of a real-time operating system. A correct behaviour of the closed-loop controller requires that the system meets timing constraints like periods and latencies, which can be expressed as deadlines. The control system timing requirements are captured through a partition in control paths which priorities are assigned according to their relative urgency. Latencies are managed through precedence constraints and more or less tight synchronisation between modules. The implementation uses the fixed-priority based preemption service of an off-the-shelf real-time operating system.Such a system can be modelled with timed event graphs, and its temporal behaviour can be analysed using the underlying (max,plus) algebra. Examples coming from a uni-processor robot controller are provided.

keywords : Periodic control systems, real-time scheduling, software architecture, timed event graphs, timing analysis

1 Introduction

Most feedback control systems are essentially periodic, where the inputs (reading on sensors) and the outputs (posting on actuators) of the controller are sampled at a fixed rate. While basic digital control theory deals with systems sampled at a single rate, it has been shown, e.g. (Arzén, Bernhardsson, Eker, Cervin, Persson, Nilsson and Sha 1999), that the control performance of a closed-loop digital control system can be improved using a multi-rate and multi-tasks controller : some parts of the control algorithm, e.g. updating parameters or controlling slow modes, can be executed at a pace slower than the one used for fast modes. In fact, a complex system involves sub-systems with different dynamics which must be further coordinated (Törngren 1998). Therefore the controller must run in parallel several control laws with different sampling rates inside a hierarchy of more or less tightly synchronised layers. Among others, robot control gives an example of systems with complex dynamics and interaction which deserves using such multi-rate layered controllers (Simon, Castillo and Freedman 1998).

Digital control systems are often implemented as a set of tasks running on top of an off-the-shelf real-time operating system (RTOS) using fixed-priority and preemption. The performance of the controller, e.g measured by the tracking error, and even more importantly its stability, strongly relies on the respect of the specified sampling rates and computing delays (latencies) (Aström and Wittenmark 1990). In particular it has been observed, e.g. by (Chen, Armstrong, Fearing and Burdick n.d.) and (Wittenmark 2001), that sample-induced delays in synchronised multi-rate systems show complex patterns and can be surprisingly long. Therefore it is essential to check off-line that the implementation of the controller will respect the specified temporal behaviour.

Usually, real-time systems are modelled by a set of recurrent tasks assigned to one or several processors, as stated in the seminal work of Liu and Layland (Liu and Layland 1973) and further work (Audsley, Burns, Davis, Tindell and Wellings 1995). Each task τ_i is modelled by a tuple (C_i, T_i, R_i, r_i) where C_i is the worst case execution time of task τ_i, T_i is the period of τ_i, R_i is a deadline associated with the task and r_i is the instant at which the task becomes runnable for the first time.

Traditionally, a worst case response times technique is used to analyse fixed-priority real-time systems. The original basic analysis method assumes that all the tasks are periodic, run on a single processor, have a common first release instant, have a deadline equal to their period and that there are no precedence constraints between the tasks. These assumptions have been progressively released (Audsley et al. 1995) to compute results for more general systems, e.g. with precedence constraints, aperiodic tasks or release jitter. Among many papers, (Fidge 1998) gives a summary of schedulability tests for such real-time systems.

However these analysis tools fail to take into account the control engineering requirements and thus do not fit well with a 'system centric' approach gathering control and computing requirements (Arzén et al. 1999). Many algorithms rely on a particular scheduling policy, e.g. Rate Monotonic, or on a particular design language or framework, e.g. Ada (Burns and Wellings 1995), and they are designed to optimise CPU's related features rather than the control performance. Also dependence relations between tasks are still difficult to take into account, and using release delays between tasks, precedence based priority assignment or work arriving functions introduces some non trivial implementation constraints (Spuri and Stankovic 1994). Additionally the temporal analysis of the system most often relies on a simulation over its hyper-period which can be costly in time and memory since the complexity of such methods is exponential in the number of components of the system.

Finally the timing requirements of control systems w.r.t. the desired control goal expressed as a performance index do no fit well with purely computing-based scheduling methods. Reaching efficient control requires an adequate setting of periods, latencies and gains according to the available computing resource, e.g. as done through control/scheduling co-design in (Ryu, Hong and Saksena 1997) using off-line iterations. As computing-based scheduling theory basically tends to maximise the number of schedulable tasks with no other concern about the application's requirements, the blind use of scheduling policies like the popular Rate Monotonic or Earliest Deadline First can lead to very poor performance (Eker and Cervin 1999). The interest for control/implementation co-design is now increasing, e.g. (Nilsson, Wittenmark, Törngren and Sanfridson 1998) and (Törngren 1998) states the effort of the control community to better model the control related implementation constraints.

In this paper we introduce a model of periodic control systems where the control designer can arbitrarily assign priorities and synchronisations in the set of control modules. Such a system can be analysed through algebraic techniques and can be easily implemented using the basic features of an off-the-shelf RTOS.

Here each task is a synchronised endless loop, modelled by a Timed Event Graph, which execution time is the crossing time of a particular transition of the net. The period of a task is not necessarily given *a priori* but can depend on synchronisation with preceding tasks leading to period inheritance. Each task is triggered by events (e.g. a clock or a synchronising task). Besides synchronisations, some tasks have high priority and preempt the low priority tasks : priorities are set according to the relative importance of some control paths related to the control performance. The main scheduling constraint is that each task must run to completion before the next activation period[1].

This model naturally handles precedence constraints with respect to the control algorithm requirements, and is not bound to a particular scheduling policy. Moreover, Timed Event Graphs have a linear model in the underlying (max,plus) algebra (Baccelli, Gohen, Olsder and Quadrat 1992) : therefore this model can be used to compute relevant quantities such as the response time of the tasks and the respect of deadlines during both the transient and steady state (periodic) phases of the system's execution. The computational complexity of the analysis is polynomial, and has been implemented and experimented with real life robot controllers.

This paper is organised as follows : in the next section we present the model of real-time tasks used in the ORCCAD software, starting with closed-loop control requirements. Thus the tasks model is formalised using timed event graphs. In section 3.1 we recall a set of results using the (max,plus) algebra to compute the real-time behaviour of a timed event graph under preemption. As these results have been formally derived in (Baccelli, Gaujal and Simon 2002), their presentation here is very informal. Then section 4 shows how this model must be restricted to cope with a real-time system's behaviour, and the design and analysis of a robot controller running on a single processor is given as example. We finally summarise the work done and conclude with perspectives and further development. The main features of the (max,plus) algebra are recalled in appendix.

2 Design of a control system : the ORCCAD framework

ORCCAD[2] is a software environment dedicated to the design, the verification and the implementation of robotic control systems. It also allows for the specification and validation of robotic missions (Borrelly, Coste-Manière, Espiau, Kapellos, Pissard, Simon and Turro 1998).

The structure of the periodic tasks, which are called module-tasks (MTs), is as follows : after initialisation, an infinite loop is executed where all input ports are first read, calculations are performed on these inputs and finally results are

[1]Note that, conversely with a commonly shared idea, closed-loop control systems are not ´ hard real-time´ , since the timing constraints are assigned according to the desired control performance and can be occasionally missed with no catastrophic failure (Cervin 2003). However, due to the lack of underlying non-linear control theory, the assignment of adequate values for periods and deadlines in real-life systems is still a diffi cult problem mainly based on case studies and tedious experiments.

[2]http://www.inrialpes.fr/iramr/pub/Orccad/

posted on all the output ports (figure 1). At run-time the MTs are scheduled using the basic features of the RTOS (priority based preemption and synchronisation primitives) to meet the timing requirements of the control algorithm.

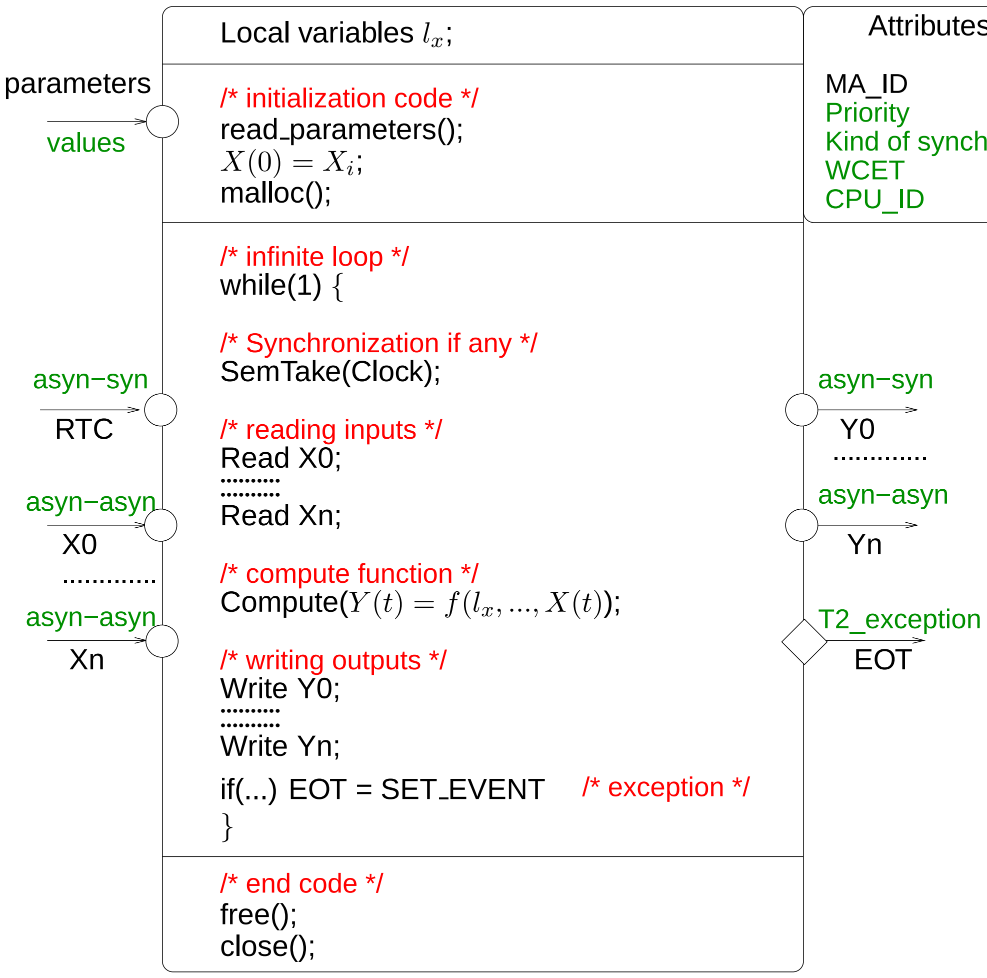

Figure 1: Structure of a Module-Task

2.1 Priority based preemption

Many robot controllers currently run on top of an off-the-shelf operating systems, where the basic way of interaction between tasks is preemption based on fixed priorities.

A control system for a robot, and more generally for process control, can be split into several calculation paths (Arzén, Cervin, Eker and Sha 2000) : the direct control path computes control set-points from tracking errors and must run with a small period and a low latency to ensure the process´ stability. The respect of the above timing constraints is critical w.r.t. the control performance, this part of the controller has a high relative urgency.

Other tasks are used to update slowly varying parameters of the non-linear plant model. These tasks are often data-handling intensive, e.g. using trigonometric functions or matrix inversion. Their duration can be far longer than the period assigned to the direct path, but delaying their ending instants has a weak effect on the controller performance, e.g. the control jitter or the system's stability. Thus they can be assigned a low priority so that their execution is preempted by every execution of the direct path calculation.

The whole system is run over a limited number of CPUs with a static partition of the tasks. All the MTs running on a same CPU are ordered according to their relative priorities. When a MT with high priority becomes runnable and starts its calculation cycle, the lower priority running MT is preempted and its context is stored. The activity of the runnable MT with the immediately lower priority resumes at the point where it was stopped as soon as the higher priority MT has finished its computation or is blocked waiting for a synchronisation event or an unavailable computing resource.

However, using only preemption is not enough to accurately specify the robot controller, in particular it cannot efficiently take into account the precedence constraints between subsets of the control algorithm.

2.2 Synchronisation

A partial synchronisation of tasks allows for the specification of precedence constraints and thus improves the control performance by decreasing the control latency[3]. Several point-to-point protocols are used on input/output ports in order to synchronise more or less tightly the set of MTs.

- ASYN-ASYN : Data id freely read and written, and communication block neither the reading nor the writing task. It must implemented using a truly asynchronous communication mechanism, e.g. using lock free multiple buffers (Simpson 1997), to avoid hidden and unwanted synchronisation[4].
- SYN-SYN : a communication of this type is a *rendez-vous*; the first task to reach the *rendez-vous* (either the writer or the reader) is blocked until the second one is ready for communication. Both tasks are unblocked after data updating.
- ASYN-SYN : the writing task runs freely and posts messages on its output ports at each execution; when reaching the input port, the reading task either reads the data if a new one is available or is blocked until the next data production.
- SYN-ASYN : symmetrical to the previous case : the reader runs freely, the writer is blocked until the next reading request except if a new one has been posted since the last reading.

Generally, the best data to be used in a *closed-loop* control algorithm is the last one produced, thus the buffers between ports have one slot, and the incoming data overwrites the old one. The ASYN-ASYN communication must be used between tasks with unrelated rates. The ASYN-SYN and SYN-ASYN ones are useful to specify dependencies between modules and to enforce the execution order of a pipe-line of tasks. The SYN-SYN rendez-vous must be chosen only when a very strong synchronisation is necessary, e.g. to merge data from a stereo vision pair of devices, as it can easily lead to a dead-lock due to a dependency cycle (Simon et al. 1998). These synchronisation mechanisms can be easily implemented, e.g. using a shared memory and synchronisation semaphores on a single processor, or through a session layer protocol on a field-bus (Mejia, Simon, Belmans and Borrelly 1989).

2.3 Modelling with timed event graphs

We need modelling and analysis tools to automatically check for timing and synchronisation inconsistencies in the network of synchronised MTs.

Modelling and analysis of discrete events systems are often done through Finite State Machines (FSM), timed process algebra and model checking, e.g. (Ermont and Boniol 2002), (Bouajjani, Echahed and Sifakis 1993). Besides their generality and large modelling power these methods are difficult to handle and rapidly suffer from computational complexity. Petri nets are other powerful tools which have been already used in the framework of real-time systems, e.g. in (Burns, Wellings, Burns, Koelmans, Koutny, Romanovsky and Yakovlev 2000) for the analysis of Ada tasking; however using general or coloured Petri nets leads to analyse the system's behaviour via simulations over the reachability graph rather than using algebraic techniques. In (Klaudel and Pommereau 2000) the semantics of Petri nets is extended to model priorities, preemption and abortion : analysing such a model again falls in model-checking with a high computational complexity.

Here, we adopt a particular modelling tool, timed event graphs (TEG), which provides a simple and efficient way to carry out temporal consistency tests.

Event graphs are a particular case of Petri nets (Murata 1989), also called marked graph. Recall that a Petri net is a bipartite graph, made of *transitions* and *places* connected by directed *arcs*. Places are marked by *tokens*, which represent some conditions in the discrete event system while crossing transitions represent actions performed by the system.

When all of the input places of a transition become marked (have a token), the transition is crossed ('fires'). The firing removes a token from each of the input places and deposits a token in each of the output places.

As shown in figure 2, the behaviour of the basic MT (reading an input port, computing, writing to an output port) can be modelled by a Petri net with three transitions. Of course, one transition must be added for each additional input and output port. Another transition is required to activate the MT subject to the periodic awakening provided by a real-time clock (RTC), also modelled by a Petri net. Since we are concerned with temporal analysis, we associate time intervals (also

[3]the latency is the delay between the instant of a measure q_n on a sensor and the instant when the control signal $U(q_n)$ is sent to the actuators (Astr¨om and Wittenmark 1990)

[4]shared data protection using some kind of priority inheritance must be avoided as these protocols dynamically jeopardise the initial schedule (carefully designed w.r.t. automatic control requirements) with unpredictable and potentially disastrous consequences for the controlled system

called 'crossing times') with some of the Petri net's transitions. Such Petri nets are called *timed Petri nets* (Ramamoorthy and Ho 1980). We have chosen to associate the duration $[d]$ of the MT with the computation transition, and thereby assume that reading and writing are instantaneous events, i.e. communication is atomic. A crossing time $[\tau]$ is also assigned to the transition associated with the RTC (Transitions associated with non zero duration are drawn with thick lines).

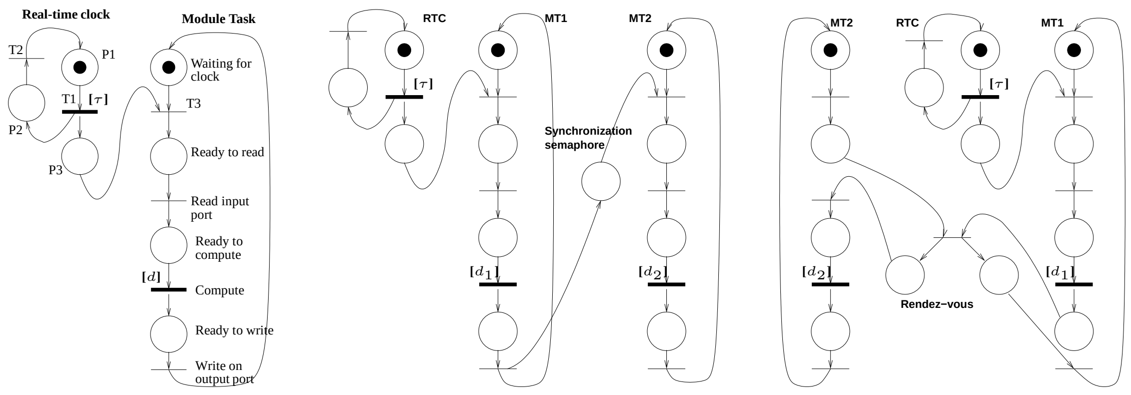

Figure 2: A Petri net model of : a periodic Module-Task, ASYN/SYN and SYN/SYN communications

Since each place have just one input transition and one output transition, the resulting Petri net is a so-called *event graph* (EG) (Murata 1989). The synchronisation and communication mechanisms described in section 2.2 also have EG models as depicted in figure 2. EGs can be used to model discrete event systems with synchronisation but with neither conflicts nor asynchronous interaction. Some of their mathematical properties (Baccelli et al. 1992) is used in the sequel.

The execution scheme of the set of EGs defines the so-called *synchronisation skeleton* of the real-time periodic process. Note that the MT model, where the whole module's computation takes place between reading all inputs and before sending all outputs, can still be used to describe more complex tasks, where synchronising I/O occur at intermediate points of the computation. As depicted by figure 3, such a task can be easily split into a set of basic MTs connected with ASYN/SYN links : such a conversion preserves the temporal behaviour of the tasks set and of their synchronisation skeleton while still using the basic MT model and associated analysis tools.

Conversely several MTs synchronised with ASYN/SYN links, running in sequence on the same CPU, can be clustered into a single thread of execution to avoid useless context switches at run time. Also note that, in any case, the execution of a timed transition can be preempted by a higher priority one at any point of progress of its firing, i.e. at any point of the modules' calculation process.

Studying the structural properties of such an EG, e.g. dead-locks avoidance, can be easily done using classical theory on Petri nets : every circuit of the EG must have at least one token in the initial marking (Murata 1989). Thus useless synchronisation or I/O ports mis-ordering can be checked.

Studying the temporal behaviour of the set of MTs is far more complex : classically this can be done through a more or less exhaustive exploration of the reachability graph of the TME, which is computed with an exponential complexity and can be costly in time and memory (Esparza and Nielsen 1994). Moreover, as the processor is a shared resource involving concurrency between the tasks, an event graph cannot model the priority based preemption provided by the scheduler. Therefore this model is refined in the next section.

3 A model for event graphs and preemption

3.1 Assumptions

The model consists of a set of tasks $\mathcal{T}_i$, $i = 1, ..., N$. Each task $\mathcal{T}_i$ is modelled by a strongly connected event graph $\mathcal{G}_i = (\mathcal{Q}_i, \mathcal{P}_i, M_i, \tau_i)$ (as in figure 2), where $\mathcal{Q}_i$ is the set of transitions, $\mathcal{P}_i$ is the set of places, $\tau_i = (\tau_{i_1}, \cdots, \tau_{i_Q})$ is the set of firing times, τ_{i_q} being the firing duration of transition q (we assume that these numbers are all integers), and $M_i(r, q)$ is the initial marking in the place between r and q when it exists.

The whole system is described by a set of tasks and clocks connected by synchronisation relations; in the EG framework, tasks and clocks are strongly connected components (SCC), which are partitioned in two sets :

- The set of initial components, denoted I_i. An initial component is called a *clock*. In most practical cases, this clock is composed of a single recycled transition but nothing forbids to consider more elaborate clocks.

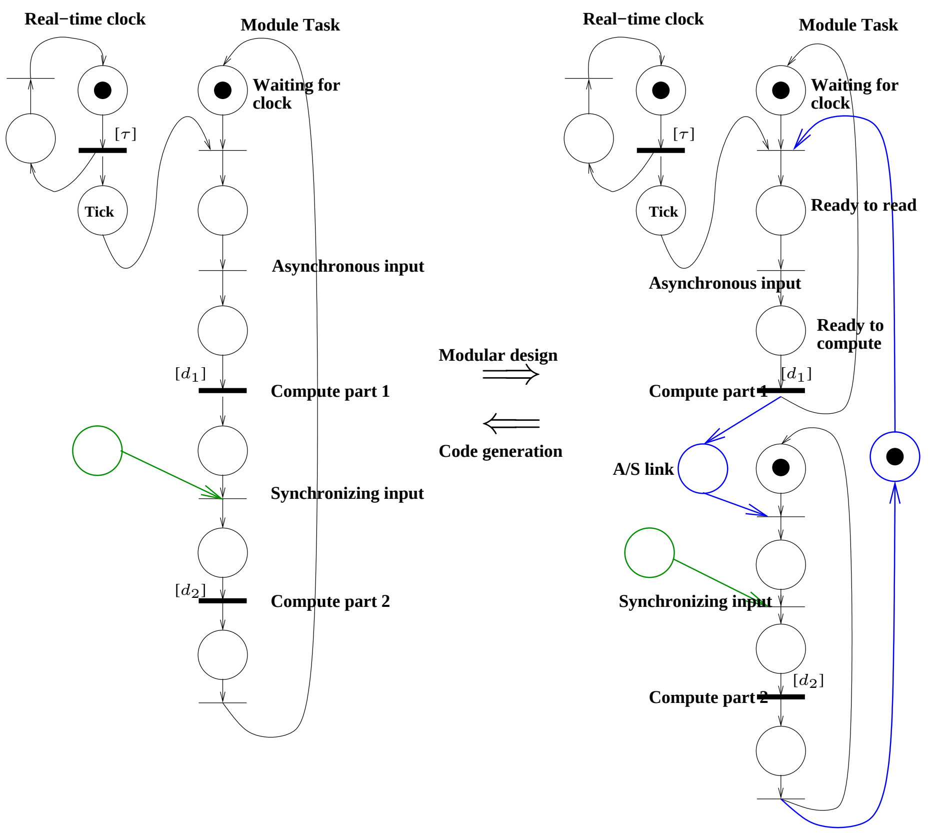

Figure 3: Splitting and gathering modules and executable threads

- All the other components, denoted O_i. They are often simple cycles for single task models but may be more complicated.

Clocks *cannot be preempted* and always deliver their ticks at the specified rate. Also, as they only emit events (the clock ticks), we consider that they do not load the computing resource. In consequence the timed transitions used to model the clocks must are not taken into account to compute the CPU's load.

In practise the clock generators are implemented either as high priority tasks in the system (running in kernel space) or provided by the hardware, while the application's modules run at low priority levels in user's space.

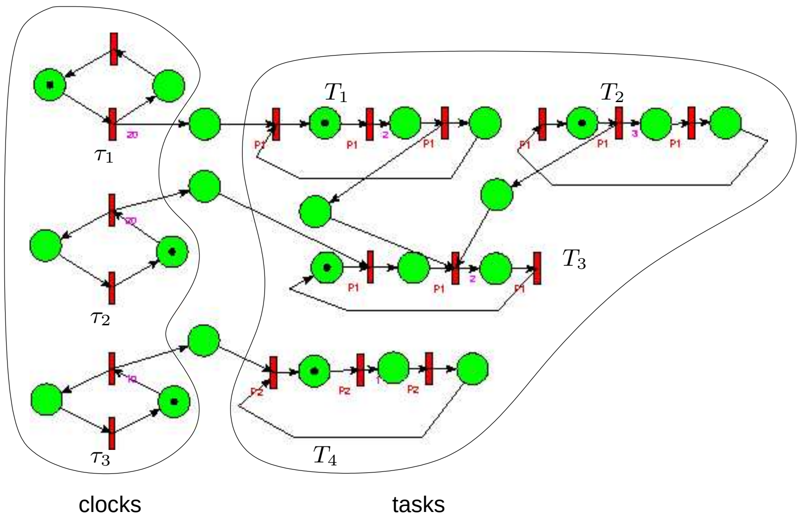

Figure 4: A graph with decomposition in its components I and O

The priority relation between tasks is given under the form of an order relation between the corresponding connected graphs $\mathcal{G}_i$. $\mathcal{G}_j \succ \mathcal{G}_i$ denotes that:

- all the timed transitions which belong to a given graph $\mathcal{G}_i$ have the same priority, which is the main restriction for our current solver. However some qualitative results have been established for a more general preemption scheme and are summarised in section 3.3.3 where the problem of arbitrary priority assignment is discussed;
- as soon as a transition in O_j fires, every timed transition in O_i being crossed is suspended and resumes its firing at its suspension point as soon as all activities in O_j stop.

Note that at this point the model is still quite general, and nothing prevents transitions with equal priorities to be crossed at the same time, as if there is enough computing resources to run all fire-able transitions in parallel. Additional restrictions are added to the mathematical framework in section 4.1 to cope with available computing resources and implementation constraints.

We first want to check that all tasks meet their time constraints, i.e. if each task is executed within the time slot given by its clock. Thus we check on the EG model that for all event graphs $\mathcal{G}_i$, the marking in the places that connect initial components I_i to any component O_i, is bounded by one, i.e. no new clock tick arrives before the O_i cycle has finished. This property is known as the *stability* of the discrete event system. Note that, as a strongly connected event graph is always stable (Baccelli et al. 1992), unbounded marking can only arise in places which connect SCCs.

We also may want to check a weaker property, where violations of the time constraints may only happen a finite number of time. In that case the marking in the places that connect initial components I_i to any component O_i may get larger than one for a finite number of occurrences. As this may happen only during the transient phase before the system reaches its steady state repetitive behaviour, we denote this property by *steady state stability*.

3.2 Modelling under contracted time

In this section, we introduce a representation of the system where we modify the time scale in the timed transitions according to their priority. Thus, starting from the highest priority level and real-time, we are able to analyse the temporal behaviour of event graphs which belong to the lower priority level with contracted time scale, and then recursively come back to real time. We consider the case where $N = 2$ which can be iteratively extended for an arbitrary number of priority levels.

For each SCC $\mathcal{C}$, we denote $\{X_q(n)\}$ the sequence of firing times of transition $q \in \mathcal{C}$. The set of all these sequences is called the behaviour of the system. By using the theory of timed event graphs (Baccelli et al. 1992), we get for all SCC $\mathcal{C}$ in isolation, a *cycle time* $\lambda_\mathcal{C} \in \mathbb{R}_+$, a *cyclicity* $s_\mathcal{C} \in \mathbb{N}_+$ and a *transient period* $k_\mathcal{C} \in \mathbb{N}$, such that for all transitions $q \in \mathcal{C}$ and all $k \geqslant k_\mathcal{C}$,

$$X_q(k + s_\mathcal{C}) = X_q(k) + s_\mathcal{C}\lambda_\mathcal{C}.$$

Note that the activity process of this SCC is therefore pseudo-periodic of period $s_\mathcal{C}\lambda_\mathcal{C}$. This period is an integer under the assumptions that we made on the firing durations. For each period there are $s_\mathcal{C}$ firings and $\lambda_\mathcal{C}$ is the average time between firings.

Let $\mathcal{G}_1 \succ \mathcal{G}_2$. We denote by $S_1(t)$ the activity process of $\mathcal{G}_1$, defined by $S_1(t) = 1$ if a transition in $\mathcal{G}_1$ is active at time t and $S_1(t) = 0$ otherwise. This function is assumed to be pseudo-periodic of period T_1, where T_1 is an integer.

We define the contraction function by $F_1(t) = \int_0^t [1 - S_1(\tau)]d\tau$, such that F_1 increases when $\mathcal{G}_1$ is inactive and constant when $\mathcal{G}_1$ is active F_1 is *pseudo-periodic* of period T_1 and increment $\Delta_1 = F_1(T_1)$. During each period T_1, $\mathcal{G}_1$ is sleeping for Δ_1 units of time, thus leaving the computing resource free to execute the tasks with a lower priority.

As the clocks' rate are fixed in real-time, they seems to be accelerated in contracted time by the contraction ratio Δ_1/T_1, and their cycle time λ'_C in contracted time are now $\lambda'_C = \lambda_C \Delta_1/T_1$ as illustrated in figure 6 (where (.)' stands for values computed in the contracted time scale).

On the other hand, tasks in $\mathcal{G}_2$ can be running at any instant of the remaining Δ_1 units of real-time, i.e. at any instant in the contracted time scale. Hence their cycle time is not modified (they just have less time to be active) and we have $\lambda'_{T_2} = \lambda_{T_2}$ for tasks. These quantities are the inverse of the firing rate of the transitions in contracted time.

Finally the system is *stable* if and only if for all the clocks and tasks in the preempted system $\mathcal{G}_2$:

$$\min_{clocks} \lambda'_{C_2} \geqslant \max_{tasks} \lambda'_{T_2}$$

For a simple system like the one in figure 7, this statement meets intuition where the clock's period must be larger than the duration of the tasks path it triggers.

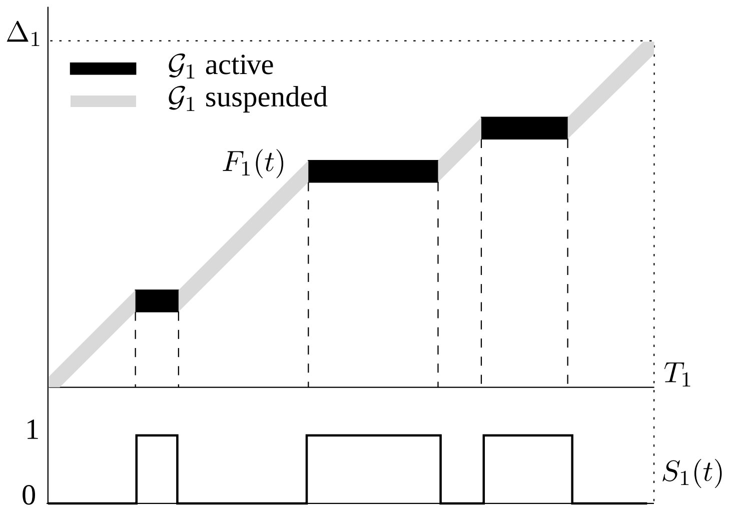

Figure 5: The contraction function F_1 associated with an activity S_1.

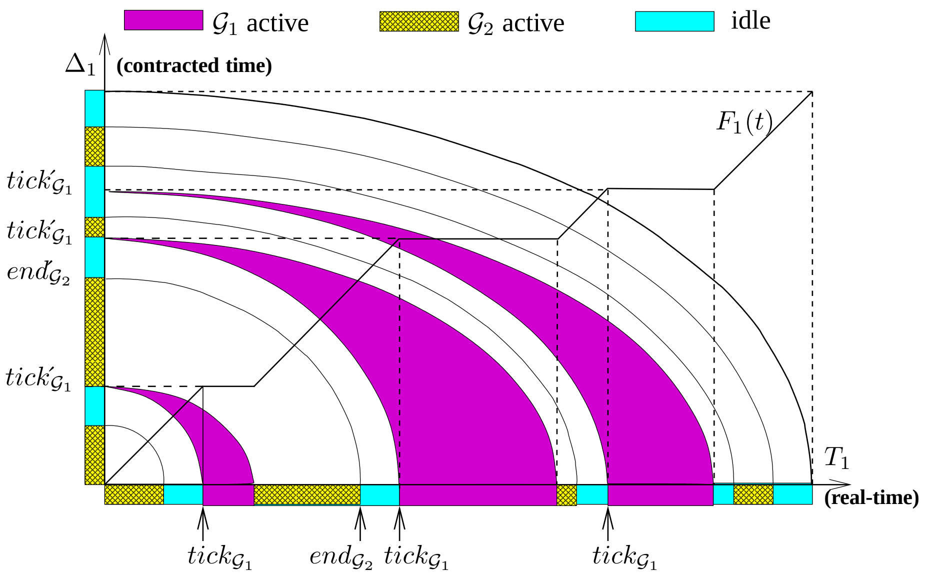

Figure 6: Contraction of clocks rate

In the stable case, the activity process becomes ultimately periodic both in contracted and in real-time : let $\min_{clocks} \lambda'_{C_2} = \lambda'_{C_{0_2}}$. Then the periodic regime is given by

$$X'_q(n + s'_{C_2}) = X'_q(n) + \lambda_{C_{0_2}} s'_{C_2} \Delta_1 / T_1$$

with cyclicity $s'_{C_2} = lcm(T_1, s_{C_2}\lambda_{C_2})/\lambda_{C_2}$.

By definition, such an event graph is stable iff the number of tokens is bounded in every place (Baccelli et al. 1992). In the particular case of figure 7, this means that after reaching the periodic behaviour, exactly one token must be extracted from the 'tick' place by the activation of the pending tasks for one token produced by the clock. According to the task model depicted in figure 2, where each task is modelled by a SCC, and where the first transition of a task is guarded by the end of the last one of the task, this means that the system is stable iff each task is completely executed between two activation ticks and thus meets its deadline.

When the number of priority levels N is larger than 2, contracted time scales are iteratively computed from the highest to the lowest priority levels. Real-time behaviours are then backward computed by applying the successive inverse contraction functions from the lowest to the highest priority level.

3.3 Algebraic formulation of real-time properties

Using the properties of linear algebra in the (max,plus) semi-ring allows one to check the event graphs system's stability via the computation of the sign of eigenvalues of transition matrices. Other quantities, like the time needed to reach a periodic steady state behaviour whatever is the initial phase between the system's components, can be further computed.

The following arguments are done in contracted time. We focus on a system with one clock $\mathcal{C}$ and one task $\mathcal{T}$. $\mathcal{C}$ is connected to $\mathcal{T}$ through a place (called 'tick' in figure 7). The output transition of place 'tick' belongs to $\mathcal{T}$ and is numbered q_1. This system is preempted by another one, with activity process S_1, period T_1 and increment Δ_1.

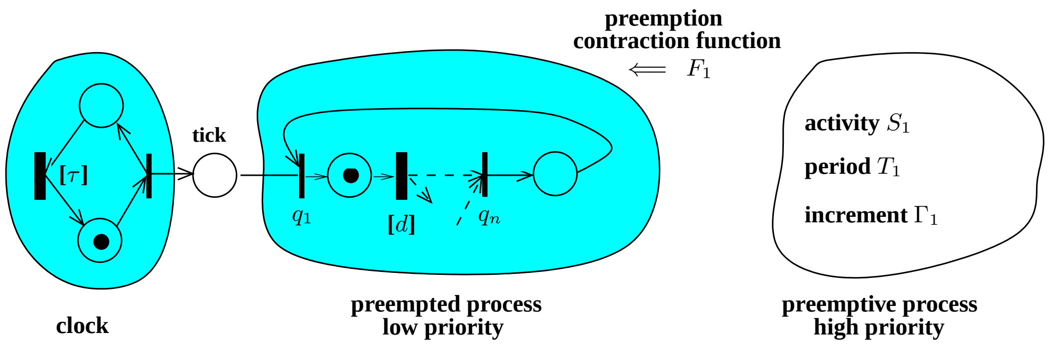

Figure 7: A clock and its synchronised circuit

Let us denote $u(n)$ the epoch of the nth arrival of a token in place tick, in contracted time. We have $u(n) = F_1(t_n)$, where t_n is the time of the n-th arrival in tick in real-time.

Note that, following section 3.2, $u(n)$ is pseudo-periodic with period $T = lcm(T_1, s_\mathcal{C}\lambda_\mathcal{C})/\lambda_\mathcal{C}$ and $u(n+T) = u(n) + \tau\Delta$ with $\tau = \lambda_\mathcal{C}$ and $\Delta = \Delta_1 T/T_1$.

The firing of transition q_1 is enabled by tokens arriving both from the clock and preceding transitions in the network, e.g. q_n, thus firing times for q_1 are given by

$X_1(k) = max[X_n(k-1), u(k)]$.

Using the (max,plus) notation where $\oplus$ and $\otimes$ stands respectively for the usual max and plus operations, this statement can be rewritten as $X_1(k) = X_n(k-1) \oplus u(k)$

And the whole system under contracted time can be represented as a (max,plus) system :

$$X(n) = A \otimes X(n-1) \oplus B \otimes u(n).$$

where $A \otimes X(n-1)$ denotes tokens crossing transitions along paths of the TEG and $B \otimes u(n)$ denotes inputs coming from the clocks.

We denote by n_0, c, γ the coupling, cyclicity and maximal eigenvalue of matrix A which is built from the EG description matrices (see the appendix where (max,plus) notations and their relations with the dynamics of event graphs are described).

Stability and steady-state stability properties can be respectively stated as

$$X_1(n) - u(n+1) < 0, \quad \forall n \geqslant 1$$

and

$$X_1(n) - u(n+1) < 0, \quad \forall n \geqslant n_0$$

where $X_1(n)$ is the n-th firing time of transition q_1 and n_0 is the time when the periodic regime is reached.

3.3.1 Stability

We consider that $X_1(1) = u(1)$ which means that the system is ready to start as soon as the clock emits its first signal.

Using computation in the (max,plus) algebra framework, some algebraic formulae to check quantitative properties of such models have been established in (Baccelli et al. 2002). A detailed exposition of these results is out of the scope of this paper and they are only summarised here :

Under the assumption given in section 3.1, the system is steady state stable iff the two following conditions hold:

$$\begin{cases} \tau\Delta/T > \gamma \text{ (maximal eigenvalue of matrix A)} & (1) \\ \text{coordinate 1 in } M^* \otimes C(s) \text{ is non-positive } \forall 0 \leqslant s \leqslant T-1 & (2) \end{cases}$$

(where $M = A^T \otimes D(-\tau\Delta)$, $M^* \overset{\text{def}}{=} \bigoplus_{i=0}^{\infty} M^{\otimes i}$ and $C(s) = \bigoplus_{i=0}^{T-1} A^i \otimes D(-u(kT+s+1)+u(kT+s-i)) \otimes B$)

As the eigenvalues of the (max,plus) system is related to the inverse of the firing rates of transitions, the first conditions is similar to the one given in section 3.2 for cycle times, stating that 'clocks must be slower than tasks'. If this condition does not hold the system can be neither stable nor steady-state stable.

The second condition is far more complex : it checks that during the periodic regime the number of tokens in the connexion places between clocks and tasks is never larger than one, and thus that there is no dead-line miss. Unfortunately no intuitive meaning can be given for this. The total complexity for this computation is $O(T|Q|^3 + T^2)$, using the max and plus operators on vectors and matrices with integers.

3.3.2 Initial phase and transient issues

It is often the case that the different tasks of the operating systems may have different initial phases each time the system is started anew. Although the preceding formula gives a simple test for steady-state stability for a fixed given phase, however, a test to ensure that the time constraint is satisfied for all possible phases between the preemptive system and the preempted one has been also derived.

It is useful to compute the length of the transient phase of the system before it reaches its periodic regime and begins to produce inputs to the actuators of the process, e.g. it can be used to set the timeout value of a watchdog checking for the good initialisation of a controller This value can be computed using a rather complex formulae, but still in polynomial time. Once this value has been computed, the stability of the system can be checked by computing the marking of the event graph during the transient phase.

Finally the activity process of each SCC, and therefore the sequence of control outputs times, can be computed whatever is the number of priority/preemption levels in the system as detailed later in example 4.5.

All these properties are expressed by formulae in the (max,plus) algebra with a polynomial computational complexity. Most of these algorithms are now efficiently implemented in the ERS software[5] to automatically compute the temporal behaviour of a real-time periodic system designed following the above synchronisation and preemption scheme.

3.3.3 Arbitrary priority assignment

Although this model allows the computing of the quantitative behaviour of the system, a main restriction is that tasks linked using synchronising communication must have the same priority.

A less restrictive model has been studied in (Baccelli et al. 2002) : in this extended model two sets of synchronised tasks are still ordered according to their relative priorities. However, only a subset of the preemptable set can be preempted by a subset of the preemptive set. Therefore, more complex systems can be modelled, e.g. where the non-preemptable subset of transitions in the low priority tasks set is located on a specific private computing resource. This model, which can also be generalised to an arbitrary levels of priorities, does not allow for computing quantitatives properties of the system, but two important qualitative properties have been established :

[5]http://www-sop.inria.fr/mistral/soft/ers.html

- if the system is stable it reaches a periodic regime which period does not depend on the initial phase of the system. Thus the system's stability can be checked via a simulation starting with an arbitrary set of initial firing instants up to reaching a periodic behaviour;
- during the periodic regime all the transitions which belong to a given SCC have the same cycle time and period, be they preemptable or not.

The later point is in fact not surprising since enabling the firing of transitions is propagated by the synchronising links. Therefore, from the real-time computing point of view (periodicity and latency) it does not seem useful to assign different priorities to tasks with synchronising communication links.

This statement can be further discussed for a fully general priority assignment, recalling that in our control framework synchronisation is used to manage (minimise) the input/output latency of the data flow paths used to implement the control algorithm. Assume now that T_1 and T_2 are successive computing activities in a synchronised data-flow, and that they are assigned different priority values :

- if $T_1 \prec T_2$, then T_2 anyway cannot start before the end of T_1 and the two values priority assignment is useless;
- if $T_1 \succ T_2$, the firing of T_2 can be preempted and delayed by an intermediary priority ready-to-run task T_3 (figure 8); at least this delay increases the input/output latency for the $\{T_1, T_2\}$ path and may lead to a control performance loss or instability. Moreover the preempting task T_3 may belong to another activity of the controlled plant, and the useless latency rise can be uncorrelated the control law timing requirement. Finally, if no extra caution is taken in the overall system design, this can lead to other kind of problems, e.g. an unprotected priority inversion, and finally a system failure (Sha, Rajkumar and Lehoczky 1990).

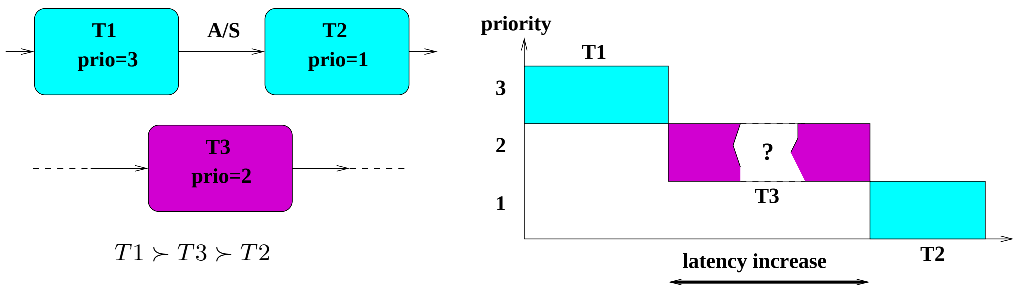

Figure 8: Latency increase by useless preemption

Although no results could be obtained for our model using a fully arbitrary priority assignment, the above remarks suggest that assigning different priorities to tasks linked by synchronising communication is useless and potentially dangerous, at least in the framework of periodic control systems.

4 Application to single processor designs

4.1 A model for implementation

In fact, all timed event graphs as described above do not represent the implementation of a multi-rate controller on a given computing architecture. Event graphs represent discrete event systems with potential parallelism, where nothing prevents timed transitions to be fired at the same time : the timed transitions of a TEG, which correspond with computing activities in our model, begin to be crossed as soon as their marking is fully enabled.

The controller must be implemented on a set of computing resources, where a sub-set of tasks is statically assigned to each processor. Thus the event graph model must be constrained so that, for each processor, only one timed transition can be running at a given instant. In the sequel we focus on single-processor controllers which are still of prime interest for embedded systems : therefore we now design models of sets of synchronised tasks, with additional modelling constraints to enforce mutual exclusion between tasks inside each priority level.

4.2 Some basic rules

The modular design of the control algorithm is usually specified through the chaining of elementary functions. An often used way of design consists in assigning a clock to the first module of the control path and synchronise the following modules using ASYN/SYN synchronisation. Therefore, the period of the whole control path is given by the clock, and the synchronisation enforces the execution order of the modules thus minimising the latency. As each module begins its execution as soon as the preceding one is finished through synchronisation and not through a presumed end-of-computing instant, it is expected that this method is more robust w.r.t. the execution time variations than releasing the execution time of the modules according to their worst case execution time.

A desirable feature for a real-time system is predictability w.r.t. the control algorithms requirements (sampling rate, latencies, relative urgency). The following design constraints are added to make the implementation deterministic and analysable at design time :

- each module must have at least one synchronisation source, either a clock or a synchronising link. The only task in the system allowed to run at its own pace is the idle task of the RTOS (the one which has the lowest priority in the system[6]);
- multiple synchronisation and links, e.g. using the rendez-vous protocol are allowed, but their appropriateness must be carefully considered w.r.t. the application requirements at design time.
- different priorities must be assigned to different clusters of synchronised tasks such that their scheduling does not rely on a particular policy taken by the scheduler for tasks with equal priorities. Priorities must be assigned according to the process and automatic control requirements.

In fact, in many practical cases more restrictive rules can be applied :

- each module must have exactly one synchronised i/o port [7];
- each set of synchronised tasks must have exactly one synchronisation source.

These latter rules ensure that the controller is dead-lock free and leads to simple schedulability analysis. They are used in example 4.5.

Additionally, we also consider that reading and writing on the physical devices drivers can be performed at any time and never block the controller.

4.3 Solving branching

Additional synchronisation must be added when the synchronisation flow forks as in figure 9, where TM_2 and TM_3 are both synchronised on the output of TM_1. As TM_2 and TM_3 have the same priority, their run-time behaviour rely on the policy of the scheduler for tasks with equal priority and is not predictable at design time. To avoid this undesirable behaviour the user is requested to choose what branch must be executed first : then a new synchronisation place P_{SYNC} (thick arc) is added to bind the starting of one synchronised flow to the end of the other one. Finally the initial useless synchronisation (between TM_1 and TM_4 here) can be replaced by an asynchronous link to handle the data exchange (dashed arc).

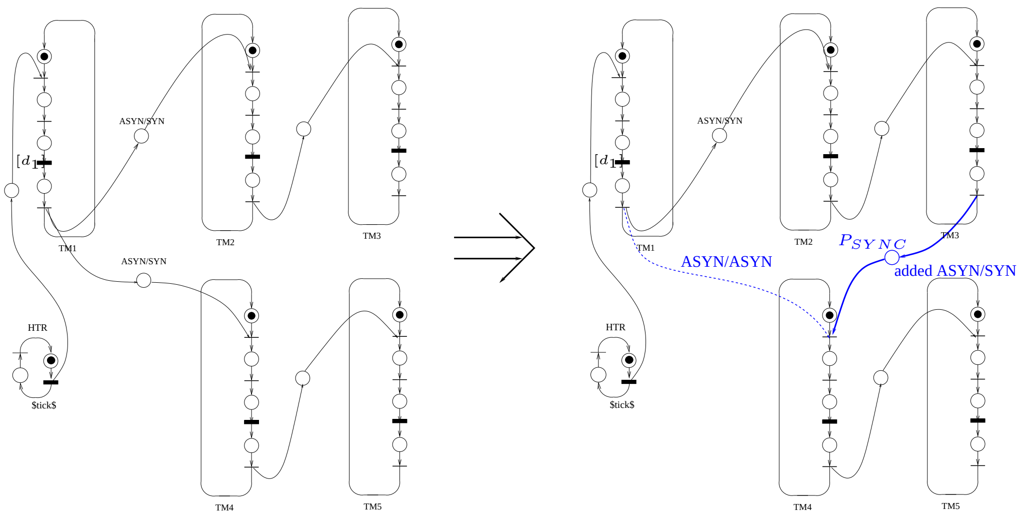

Figure 9: Synchronisation of a control fork : $\{T_2, T_3\}$ are executed first

[6]a very interesting case is Linux/RTAI where the idle task of the real-time kernel is Linux itself which can be used for useful non real-time activities like monitoring and display (`http://www.aero.polimi.it/~rtai/`)

[7]in particular this forbid using strong rendez-vous synchronisation on links

However synchronising the second path on the end of the first one is not always the right solution w.r.t. the control requirements, as enlightened in figure 9. Here, the forking link ensures that the Q sensor data on which the two filters work is measured at the same time. The filters output are then gathered in the fusion module so that the control flow must join on its inputs : thus choosing the adequate synchronisation scheme, i.e. adding or removing synchronising links, must be interactively done by the control designer.

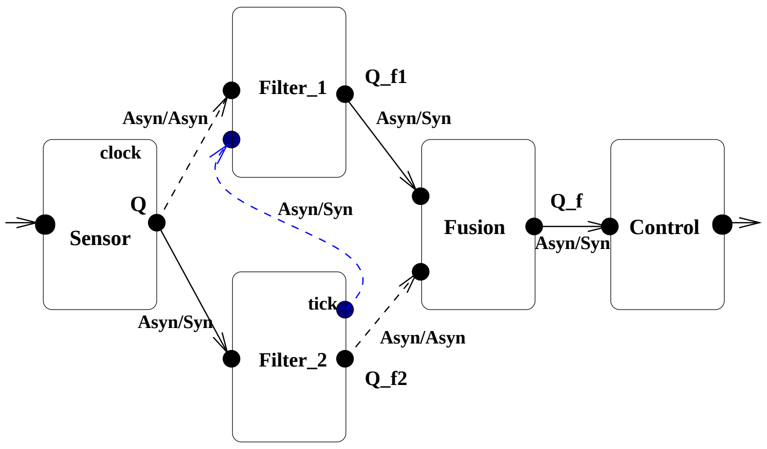

Figure 10: Fork and join control flow

4.4 Modelling exclusive access to the CPU

After solving indeterminism due to concurrent branches of equal priority, assigning the same priority to synchronised tasks is still not enough to model the behaviour of the tasks set on a given processor and to build a deterministic software : at a given time every ready task in the currently running priority level can potentially be executed by the processor. While mutual exclusion w.r.t. the processor between different priority levels is handled by the scheduler, the exclusive CPU access by tasks inside a given priority level must be modelled and implemented through additional synchronisation.

When the controller is mapped on a single processor, synchronising places (denoted P_{SYNC}) are added to loop between the end of the last task of a synchronised path (i.e. a task which synchronises nothing) and the starting transition of the first task of the path (which is fired by the clock), so that re-starting the first task is submitted to the ending of the last one of the synchronised path. This place is initially marked as in figure 11 to make the event graph live (i.e. the corresponding semaphore in the generated code must be initially free).

In fact, we create that way a new circuit which crosses all the synchronised tasks in the path. This circuit has only one token in the initial marking and, following a basic property of event graphs, its marking remains one for every firing. Therefore only one transition of the net can be enabled at a given time thus insuring the exclusive access to the computing resource.

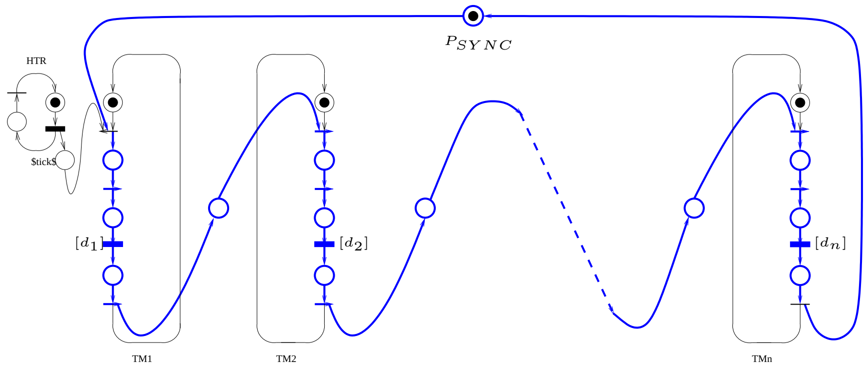

Figure 11: ASYN/SYN loop with mutual exclusion between transitions

Sketch of statically distributed control Although simple processor implementations remain of prime interest for small embedded systems a general more general case consists in using a static partition of the controller deployed over a set of control units, e.g. ECUs (Electronic Control Units) connected through a CAN bus for the automotive domain. Hardware and software redundancy is also s key feature for fault tolerance. We assume that most a the preceding ideas can be extended to the case of controllers statically mapped on a distributed hardware architecture following the next steps :

- Modules are mapped on the architecture first according to functional constraints, e.g. dedicated I/O boards, fast DSPs or low level vision processing embedded in cameras. Heuristics can be further used to optimise the implementation or take into account fault tolerance, e.g. (Girault, Lavarenne, Sighireanu and Sorel 2001).
- The control flow can fork along synchronised links between two tasks running on different processors as these two tasks can be truly run in parallel.
- Synchronising links ensuring mutual exclusion inside a given priority level must be added according to every subset of synchronised tasks assigned to a given processor.

4.5 Example 1: the computed-torque controller

The first example considers the computed-torque robot controller : its goal consists in position control of the robot tip including compensation for gravity, centrifugal and Coriolis forces and inertia variations during the robot motion. It is made of several modules ($MT1$ to $MT7$) specified using the ORCCAD GUI.

Partitioning The control algorithm can be partitioned in three groups of modules :

- MT3–MT6 is the direct control path; it computes the desired trajectory, the tracking error, the control theoretic torque and the filtered controller's output torque U_f;
- the long duration MT7 module computes an explicit model of the robot arm dynamics, i.e. the inertia matrix M and the vector of disturbing forces N from the robot's joints positions and velocities measurements (Q);
- MT1 and MT2 are observers checking for safety conditions, thus they must react very fast to signal a $joint\_limit$ exception.

Synchronisation and priority assignment It has been show using realistic simulations that MT7 can be computed at a rate several times slower than MT3–MT6, and that its output can be delayed, e.g. by preemption from others control modules, with a small effect on the control performance measured by the tracking error (Simon, Espiau, Castillo and Kapellos 1993). As the latency of the {MT3–MT6} path is critical w.r.t. the robot's stability, these tasks are synchronised using asyn/syn links to minimise the latency between the Q measurement and the emission of the U_f control signal ; they must run with a priority higher than MT7, so that this path cannot be preempted for a long time by MT7. As MT5 and MT7 run with different and unrelated frequencies (and because the duration of MT7 may have large variations), the M and N connexions must use asyn/asyn links (and the system automatically inserts a lock-free multiple buffer to ensure data integrity and avoid synchronisation side effects which occur when using a mutex).

Checking for joint-limits must run with a high frequency and, as any delay or deadline miss for this exception can lead to a mechanical failure of the robot MT1 and MT2 must be assigned with a high priority.

Thus the priorities are set according to the relative urgency of the tasks, such that

$$\{MT1, MT2\} \succ \{MT3, MT4, MT5, MT6\} \succ \{MT7\}$$

Recall that the automaton runs with a priority higher than any algorithmic module. Note that in this particular case it appears the rate-monotonic scheduling policy meets the control requirement, but this is not the general case, e.g. (Eker and Cervin 1999) because such general purpose priority assignment policies mainly try to optimise the CPU use and do not care about the controlled process requirements. To be efficient the priority assignment must take into account the characteristics of the controlled system and of the control goal : obviously it means that the designer must have a good knowledge about the process, its control algorithm, its environment and its failure modes.

Two modules in the design do not represent periodic computations. The `robot` module represents the real robot, and its input/output port provide gateways to the driver functions connected to the sensors and the actuators : it is assumed that reading and writing on these drivers never block the calling module-task, and that they can be read or written at any instant. The `dyn_Atr` block is a finite state machine in charge of the temporal management of the control law, i.e. starting, stopping and exception handling. It is event-driven and it is not involved in the repetitive computation of the control law. Therefore these two modules are not modelled in the synchronisation skeleton.

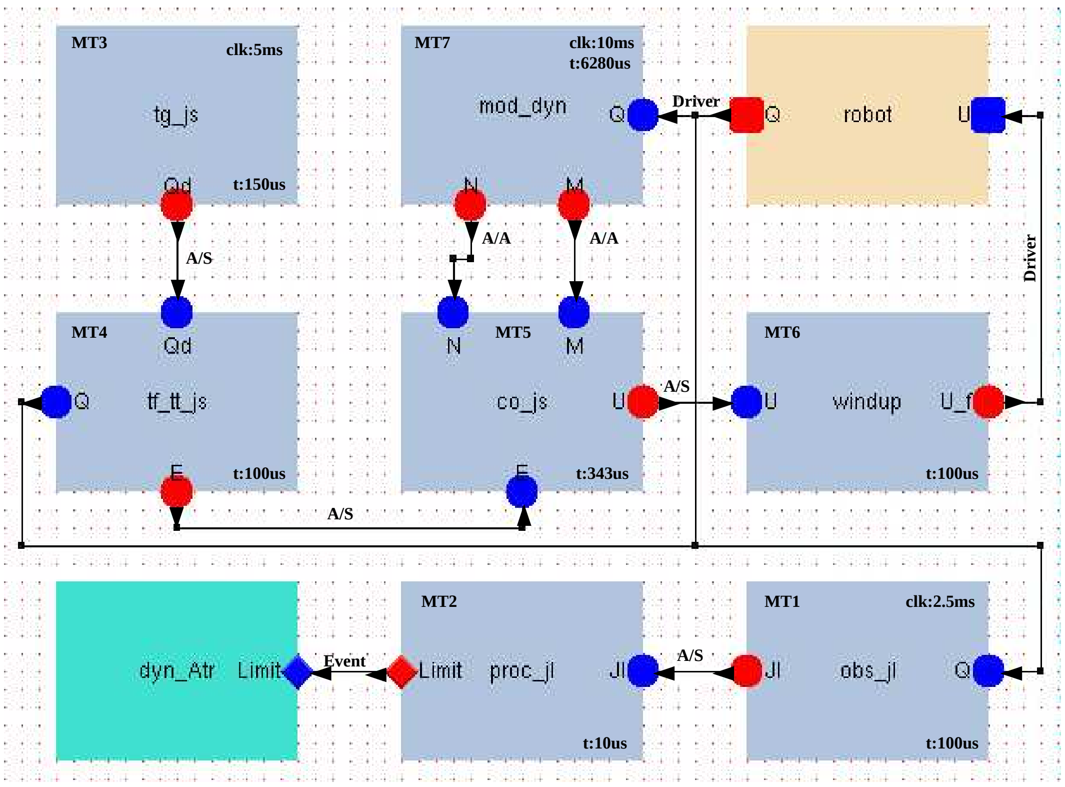

Figure 12: Computed-torque robot controller

The incidence matrix, initial marking vector and timing vector of the event graph model of this system, depicted by figure 13 in graphical form, is actually generated by the Orccad verifier from the block-diagram oriented GUI. The result in `egl` (event graph list) format feeds the corresponding solver in ERS (Benattar 2001).

The event graph is built from the elementary models of module-tasks and synchronisation links depicted by figure 2. As there is only one processing unit then all tasks in $\{MT1, \ldots, MT7\}$ must run in mutual exclusion. Thus the initially event places $P_{sync}1$ and $P_{sync}2$, inside the S_1 and S_2 synchronised paths, have been added to serialise the activities inside the clusters of tasks with equal priorities[8]. Exclusion between the different clusters of tasks is handled by the preemption.

- cycle times : the sub-system S_1 made of clock 1, MT1 and MT2 is stable : $(2500 > (100 + 10))$.
  Now, we consider the second connected component S_2, made of Clock 2, MT3-…-MT6. This sub-system is preempted by the first component, which have an activity of period 2500, with $T = 2500$ and $\Delta_2 = 2390$. Stability of S_2 holds since $5000 \times \frac{2390}{2500} = 4780 > (150 + 100 + 343 + 100)$.
  The last component S_3 is preempted by both sub-systems S_1 and S_2. The whole preemption process have period $T = 5000$, and a total non busy time of $\Delta_3 = 5000 - ((100 + 10) \times 2 + 150 + 100 + 343 + 100) = 4087$.
  The stability property holds : $10000 \times \frac{4087}{5000} = 8174 > 6280$, and the whole system is stable.
- Steady-state stability : Sub-system S_2 is preempted by S_1. However, it also have period one (in terms of number of firings). The input under contracted time is : $u(n) = u(n-1) + 2\Delta_2$, with $2\Delta_2 = 4780$. Its structure can be reduced to a scalar version of Equation (3.3): $z_2(k+1) = x_{2_1}(k) - u(k+1) = (a_2 - 2\Delta_2) \otimes z_2(k) \oplus -2\Delta_2$, with $a_2 = 693$. The solution is $z_2(k+1) = (a_2 - 2\Delta_2)^* \otimes -2\Delta_2 = -2\Delta_2 = -4780 < 0$, once the periodic regime of z_2 is reached.
  As for sub-system S_3, we get similarly, a periodicity equal to 1, and a solution $z_3(k) = (a_3 - 2\Delta_3)^* \otimes -2\Delta_3$, with $a_3 = 6280$. and $2\Delta_3 = -8174$. The solution is $z_3(k) = -8174 < 0$.
- Transient regime : all systems have period 1 (in terms of number of firings) as well as a transient regime of length 1. The periodic regime is reached immediately and the stability property is also satisfied immediately.

The whole computation process is performed by ERS in 0.09 sec. on a Pentium II 300 MHz. A sample of results is displayed down below :

[8]In fact during code generation tasks $\{MT1, MT2\}$ (resp. $\{MT3, MT4, MT5, MT6\}$) can be clustered in a single thread to avoid useless context switches, while preserving the synchronisation skeleton and temporal behaviour of the modular design.

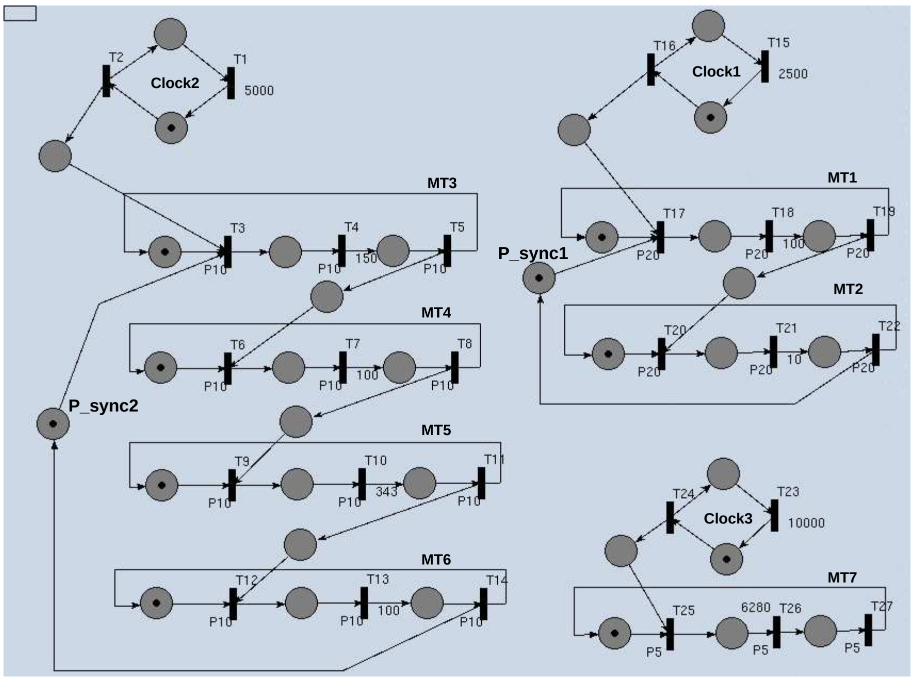

Figure 13: Event graph of the computed-torque robot controller

`activity of systems` S_1, S_2 `and` S_3 `(1(n) means active for n time units, 0(n) means inactive and [.] is a repetitive pattern):`

S_1 `:  1(110)[0(2390)1(110)]`

$\{S_1 \cup S_2\}$ `:  1(803)[0(1697)1(110)0(2390)1(803)]`

$\{S_1 \cup S_2 \cup S_3\}$ `:  1(8106)[0(1894)1(8106)]`

from which we can easily extract the activity diagram displayed by figure 14 for the first period of the system. As expected we see that the start-up of S_2 is delayed by S_1, and that the low urgency S_3 task is preempted by both S_1 and S_2.

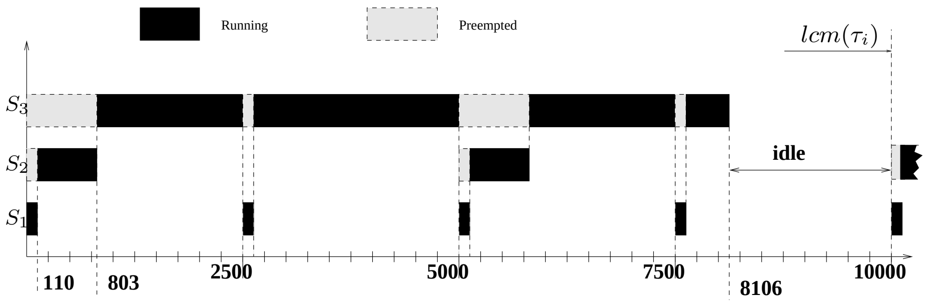

Figure 14: Activity of the computed torque controller

4.6 Example 2 : a stereo-vision based controller

A more complex controller using data coming from a stereo-vision sensor can be derived from the computed-torque controller designed in the previous section. The new controller depicted in figure 15 needs additional modules and a more complex synchronisation scheme : `LeftFrame` and `RightFrame` are frame grabbers and take snapshots of the scene; `Extract3L` and `Extract3D` extract segments from the raw images and `Stereo` builds a 3D reconstruction from the extracted features. The vision sub-system has the following temporal behaviour :

- to enhance the quality of 3D reconstruction the left and right images must be frozen at the same time, thus the two frame grabbers are strongly synchronised using a syn/syn link between their `synchro` ports;

- the modules which extract features from the raw images are assumed to provide three segments during their activity process, but the order at which they are available is not known in advance : the corresponding output ports of the `Extract` modules trigger the `Stereo` module through multiple asyn/syn links. That way we avoid polling on the Extract output and ensure that the Stereo module gets all its input data before performing its computation. This is an example where multiple synchronisation can be useful and thus must be allowed.

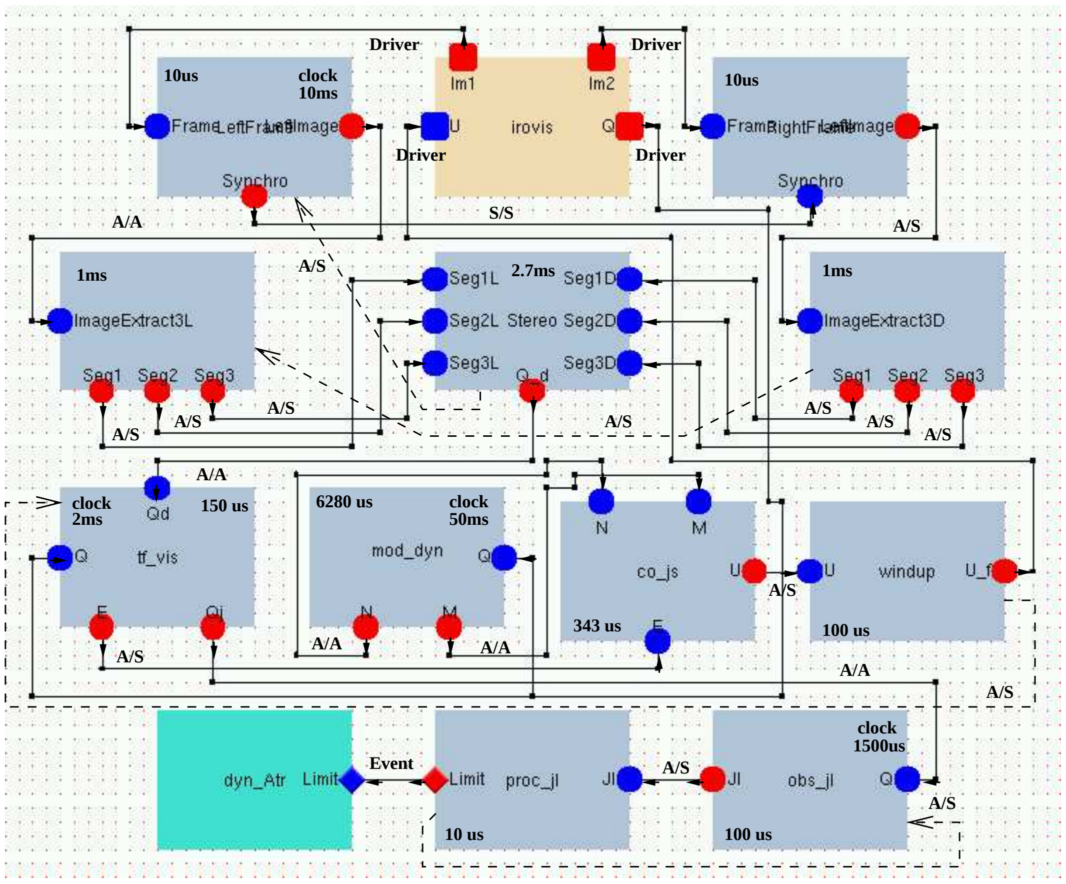

Figure 15: Stereo vision based controller

The rest of controller is the same as in example 4.5 and leads to the following partition, from high to low priority :

- $\{obs\_jl, proc\_jl\}$ with a 1500 μs clock triggering $\{obs\_jl\}$;
- $\{tf\_vis, cp\_js, windup\}$ with a 2 ms clock triggering $\{tf\_vis\}$;
- $\{LeftFrame, RightFrame, Extract3L, Extract3D, Stereo\}$ with a 10 ms clock triggering $\{LeftFrame\}$;
- $\{mod\_dyn\}$ with a 50 ms clock.

After a mixed automatic/interactive work to add and remove links to solve forks and mutual exclusion a possible synchronisation scheme (among others possible schedulable schemes) is finally set as indicated in figure 15, where dashed links are added to the initial design (but are hidden on the GUI). The analysis is performed as usual using the EG model and the (max,plus) algebra solver. Due to the increased computing load the clock of the dynamic model computation must be lowered to 50 ms to preserve the stability of the controller during the steady state phase. Also, due to the more complex synchronisation scheme, and because the clocks are not harmonic leading to a 3 cyclicity, the temporal analysis shows a rather complex activity pattern for the four synchronised task paths (times are in μs) :
S_1 : 1(110)0(1390)1(110)[0(1390)1(110)]
$\{S_1 \cup S_2\}$: 1(703)0(797)1(110)0(390)1(593)[0(407)1(110)0(890)1(703)
0(1297)1(703)0(797) 1(110)0(390)1(593)
$\{S_1 \cup S_2 \cup S_3\}$: 1(7742)[0(258)1(593)0(407)1(110)0(890)1(7632)0(368)1(703)0(797)
1(110)0(390)1(7632)0(368)1(703)0(1297)1(7742)]
$\{S_1 \cup S_2 \cup S_3 \cup S_4\}$:1(39950)[0(50)1(7632)0(368)1(703)0(797)1(110)0(390)
1(39840)0(160)1(7742)0(258)1(593)0(407)1(110)0(890)1(39950)

0(50)1(7632)0(368)1(703)0(1297)1(39950)]

Anyway iterative runs of the method and associated tools rapidly leads to find a dead-lock free, deterministic and schedulable implementation of the controller on a single processor.

5 Summary and future research

In this paper we have shown how following some simple guidelines can lead to the implementation of a real-time software which is efficient w.r.t. to a control goal. We assume that the tasks are scheduled using preemption and fixed priorities, and that their deadline equals their period. The model also takes into account synchronisations between tasks enforcing precedence constraints and thus can be used to manage the input/output latencies of the controller. Under these assumptions, the set of tasks can be modelled by Timed Event Graphs and its temporal behaviour can be analysed with a polynomial complexity using the underlying linear model in the (max,plus) algebra. Improvements in modelling will allow to handle systems where deadlines are not equal to periods.

Using this model we derived tests to check temporal properties of the system such as periodicity, cycle time, response time and respect of deadlines, both for the transient regime and for the steady state regime. The method is quite general and is not limited to a particular scheduling policy, thus priorities can be assigned to tasks according to a *controlled process* based relative urgency. Then the associated ORCCAD software automatically generates the run-time code for a single processor and the method can be easily extended to implement controllers statically mapped on a multi-processor computing resource. The controller only uses the basic features of off-the-shelf real-time operating systems, i.e. a fixed-priority preemptive scheduler, periodic timers and semaphores. Note that anyway the design of an controller efficient w.r.t. the end-user's requirements needs some knowledge about the process, its control laws, its environment and its failure modes and that using a schedulability design/analysis method must not be done blindly.

However, like other existing schedulability analysis tools for real-time systems, the temporal analysis assumes the *a priori* knowledge of the worst case execution time of the tasks. This is always difficult to measure (Puschner and Burns 2000) as tasks durations depends on both the software tools (programming language and compiler) and the target hardware (processor speed), and may vary according to the run-time state (e.g. cache effects and network induced delays). The computing load may also have large variations due to the algorithms themselves, e.g using vision processing or recursive filters with unpredictable convergence rates. Hence the design of a strictly hard real-time systems may lead to a severe under-use of the computing resources, while adding fault tolerance requires specialised and costly hardware and software components (Chevochot and Puaut 2001).

Besides safety critical systems embedded computers are of growing interest and are forecast to be integrated in many commonly used devices such as smart home, office or transport systems[9]. Many of these systems will be executed in dynamic environments with timing uncertainties and unpredictability. Therefore the *hard* real-time assumption must be carefully studied and should be relaxed when possible to provide adaption and cost-effective solutions.

In particular control systems are often claimed to be hard real-time systems where any violation of the timing constraints leads to a catastrophic failure. Setting adequate values to these timing constraints is a difficult task, mainly based on rules of thumb, e.g. (Aström and Wittenmark 1990) or case studies, and in many cases the hard real-time assumption is accepted *as is* with no further study : this may lead to unnecessarily over-constrained systems, however with the advantage of quite simple schedulability tests and timing exception handling as the answers are binary.

This is even more the case for closed-loop control systems : in practise a closed-loop system must have a *stability margin* to be robust against the parameters uncertainties of the controlled plant. It has been recently recognised that robust closed-loop control also exhibits robustness against timing uncertainties, i.e. the controller performance and stability are submit to graceful degradation rather than abrupt failure in case of sampling jitter and occasional deadlines miss (see for example the examples in (Cervin 2003)).

Thus such control systems are robust w.r.t. timing constraints and may be subject to deviations from nominal timing parameters, e.g. tasks durations and sampling rates. Besides the traditional strict real-time approach, alternative and more flexible scheduling methods may be investigated to exploit this robustness : it is expected that rather than enforcing timing predictability for a system which is naturally timely uncertain a better way consists in learning how to live with uncertainty.

Based on the control/scheduling co-design structure of controlled explained in this paper a soft real-time and adaptive fault-tolerant approach can be sketched as :

- starting from control algorithms requirements, the control laws structures are partitioned into subsets of synchronised modules for which priorities are assigned w.r.t. to their relative urgency as shown in this paper. Timing con-

[9]see for example the proposals about embedded systems and hybrid control in the FP6-IST call for Expressions of Interest `http://eoi.cordis.lu/` and subsequent call for proposals FP6-2003-IST-2

straints (tasks periods, deadlines and latencies) are off-line identified for the controllers. The proposed method can be used to check the structural correctness of the design and to evaluate the scheduling feasibility using the nominal timing parameters. To take into account the imperfections of the implementation, allowed variations around nominal values should be ideally provided. However finding such bound remains difficult an deserves further theoretic investigation (Nilsson et al. 1998);
- These constraints may be expresses in several ways, e.g. as a set of nominal values with associated gains to perform gain scheduling, e.g. (Ryu et al. 1997), as weakly hard constraints combining allowed deadlines violations and associated exception handling (faulty tasks can be delayed, skipped or aborted) as in (Bernat, Burns and Llamosí 2001) or as complex temporal attributes (e.g. nominal period variation) used by an heuristic procedure to assign the scheduling parameters (Sandström and Norström 2002). A given control functionality can be also implemented using more or less costly variants of the control algorithms, which can be weighted w.r.t. their respective computing cost and control performance.
- as the controller is no longer considered as strongly hard real-time, timing constraints violations must not be considered as fatal but must be processed by a fault-tolerant layer. Control execution related exceptions are catch from the control tasks and passed to a *scheduling manager* in the coordination layer (recall that it is the duty of the 'dynAtr' FSM module in Figure 12). According to incoming exceptions and events it manages the set of concurrent control modes to optimise a global (application level) Quality of Service criterion (Sanfridson 2000). Indeed this discrete-events based manager may act at different levels according to the received signals :
  - occasional violations of timing constraints (data loss or deadlines miss) lead to scheduling decisions (e.g. skipping or delaying the faulty task);
  - more constant violations lead to either decrease the frequency of at least one of the controllers (and hence also its gains and performance) or to switch for a variant with lower computing cost if available.
  - in case of unresolved constant overload, the scheduling manager also may act as an 'admission controller' to cancel the lower weight activity and maintain the QoS level as high as possible;
- an even more adaptive approach consists in *feedback scheduling* where, as pioneered in (Arzén et al. 2000) and (Lu, Stankovic, Abdelzaher, Tao, Son and Marley 2000), the scheduling parameters are continuously computed in an outer closed loop (the scheduling controller) which goal is to control the global QoS criterion, e.g. minimising a weighted sum of tracking errors under constraint of limited computing power.

It is expected that such flexible scheduling methods will be able to provide a cost effective use of the computing resource w.r.t. closed-loop control requirements in presence of timing uncertainties and jitter. The control architecture described in this paper is assumed to be a safe starting platform to implement such flexible control/real-time systems.

Appendix : (max,plus) algebra and event graphs

In this section we list several properties of the so-called (max,plus) algebra. All these results can be found in (Baccelli et al. 1992), where they are presented in full details.

$\mathbb{R}_{max}$ is the semi-ring $(\mathbb{R} \cup \{-\infty\}, \oplus, \otimes)$, where $\oplus$ stands for the $\max$ operation and $\otimes$ stands for the $+$ operator. These operations are extended to vectorial operation in the canonical way.

To any event graph $\mathcal{G}$ with an initial marking bounded by one, one can associate matrices, $A(k)$, $k \in \{0, 1\}$, of size $Q \times Q$, where the entry (i, j) in matrix $A(k)$ is σ_j, the delay or lag time of transition j, if there exists a place between transitions q_j and q_i with k initial tokens, and $-\infty$ otherwise.

Let $A(0)^* = \bigoplus_{i=0}^{\infty} A(0)^{\otimes i}$, and $A = A(0)^* \otimes A(1)$. Let $X_q(k)$ be the epoch when the k-th firing starts in transition q.

If there is an input transition with arrival process u, where $u(n)$ gives the epoch of the nth release of a token by the input, then

$$X(n) = A \otimes X(n-1) \oplus B \otimes u(n),$$

where $B_i = 0$ if there is a place between the input and transition q and $-\infty$ otherwise.

By using the spectral theory with timed event graphs, we get the following result.

Lemma 1. *For all SCC $\mathcal{C}$ in isolation, there exists a cycle time $\lambda_{\mathcal{C}} \in \mathbb{R}_+$, a cyclicity $s_{\mathcal{C}} \in \mathbb{N}_+$ and a transient period $k_{\mathcal{C}} \in \mathbb{N}$, such that for all transitions $q \in \mathcal{C}$ and all $k \geqslant k_{\mathcal{C}}$,*

$$V_q(k + s_{\mathcal{C}}) = V_q(k) + s_{\mathcal{C}} \lambda_{\mathcal{C}}.$$

As for the whole system $\mathcal{G}$ (all SCC considered together), in the case with no input, we have the following result : we denote $\mathcal{C} \to \mathcal{C}'$ if the SCC $\mathcal{C}$ precedes the SCC $\mathcal{C}'$ for the topological ordering. If

$$\max\{\lambda_{\mathcal{C}} | \mathcal{C} \to \mathcal{C}'\} > \lambda_{\mathcal{C}'}, \tag{1}$$

then the SCC $\mathcal{C}'$ have the same cycle time as the preceding SCC achieving the maximum in Equation (1)*. If*

$$\max\{\lambda_{\mathcal{C}} | \mathcal{C} \to \mathcal{C}'\} < \lambda_{\mathcal{C}'},$$

then the SCC $\mathcal{C}'$ (and hence the whole system) is said unstable (the marking in some places will grow to infinity).

A similar result holds for a event graph with a pseudo periodic input u. In particular, if the inverse of the input rate is larger than or equal to the maximal cycle time of all SCCs in isolation, then the system is stable. Otherwise, it is unstable.

Acknowledgement The authors thanks A. Jean-Marie and B. Gaujal for the integration of the solver for preemptive graphs in ERS.

References

Arzén, K.-E., Bernhardsson, B., Eker, J., Cervin, A., Persson, P., Nilsson, K. and Sha, L.: 1999, Integrated control and scheduling, *Technical Report ISRN LUFTD2/TFRT–7686–SE*, Dept. of Automatic Control, Lund Institute of Technology.

Arzén, K.-E., Cervin, A., Eker, J. and Sha, L.: 2000, An introduction to control and scheduling co-design, *39th IEEE Conference on Decision and Control*, Sydney, Australia.

Aström, K. and Wittenmark, B.: 1990, *Computer-Controlled Systems - Theory and Design*, Prentice Hall, Englewood Cliffs, NJ.

Audsley, N., Burns, A., Davis, R., Tindell, K. and Wellings, A.: 1995, Fixed priority preemptive scheduling: An historical perspective, *Real-Time Systems* **8**, 173–198.

Baccelli, F., Gaujal, B. and Simon, D.: 2002, Analysis of preemptive periodic real time systems using the (max,plus) algebra with applications in robotics, *IEEE Trans. on Control Systems Technology* **10**(3), 368–380.

Baccelli, F., Gohen, G., Olsder, G. J. and Quadrat, J.-P.: 1992, *SynchronizatioN and LinearitY : An Algebra for Discrete Event Systems*, Wiley Series in Probability and Mathematical Statistics. http://www-rocq.inria.fr/scilab/cohen/documents/BCOQ-book.pdf.

Benattar, F.: 2001, *Programmation et vérification de tâches robotiques multicadences*, Master's thesis, DEA Imagerie, Vision et Robotique, INP Grenoble.

Bernat, G., Burns, A. and Llamosí, A.: 2001, Weakly hard real-time systems, *IEEE Transactions on Computers* **50**(4), 308–321.

Borrelly, J., Coste-Manière, E., Espiau, B., Kapellos, K., Pissard, R., Simon, D. and Turro, N.: 1998, The ORCCAD architecture, *Int. Journal of Robotics Research* **18**(4), 338–359.

Bouajjani, A., Echahed, R. and Sifakis, J.: 1993, On model checking for real time properties with durations, *8th Symposium on Logic in Computer Science (LICS 93)*.

Burns, A. and Wellings, A.: 1995, *HRT-HOOD: A Structured Design Method for Hard Real-Time Ada System*, Elsevier.

Burns, A., Wellings, A., Burns, F., Koelmans, A., Koutny, M., Romanovsky, A. and Yakovlev, A.: 2000, Towards modelling and verification of concurrent Ada programs using Petri-nets, *Workshop on Software Engineering and Petri Nets,21st Int. Conf. App. Theory of Petri Nets*, Aarhus, Denmark, pp. 115–134.

Cervin, A.: 2003, *Integrated Control and Real-Time Scheduling*, PhD thesis, Department of Automatic Control, Lund Institute of Technology, Sweden.

Chen, J., Armstrong, B., Fearing, R. and Burdick, J.: n.d., Satyr and the nymph: Software archetype for real time robotics, *IEEE-ACM Joint Computer Conference*, Dallas, U.S.A.

Chevochot, P. and Puaut, I.: 2001, Experimental evaluation of the fail-silent behavior of a distributed real-time run-time support built from cots components, *Int. Conf. on Dependable Systems and Networks (DSN'01)*, Göteborg, Sweden, pp. 304–313.

Eker, J. and Cervin, A.: 1999, A Matlab toolbox for real-time and control systems co-design, *6th Int. Conf. on Real-time Computing Systems and Applications*, Hong Kong, pp. 320–327.

Ermont, J. and Boniol, F.: 2002, TPAP: an algebra of preemptive processes for verifying real-time systems with shared resources, *Workshop on Theory and Pratice of Timed Systems, ETPAS'2002*, Grenoble, France.

Esparza, J. and Nielsen, M.: 1994, Decibility issues for Petri nets - a survey, *Journal of Informatik Processing and Cybernetics* **30**(3), 143–160.

Fidge, C.: 1998, Real-time schedulability tests for preemptive multitasking, *Real-Time Systems* **14**(1), pp 61–93.

Girault, A., Lavarenne, C., Sighireanu, M. and Sorel, Y.: 2001, Fault-tolerant static scheduling for real-time distributed embedded systems, *21st International Conference on Distributed Computing Systems, ICDCS'01*, Phoenix, USA.

Klaudel, H. and Pommereau, F.: 2000, A concurrent and compositional Petri net semantics of preemption, *in* T. S. W. Grieskamp and B. Stoddart (eds), *Integrated Formal Methods*, Vol. 1945 of *Lecture Notes in Computer Science*, Springer, pp. 318–337.

Liu, C. and Layland, J.: 1973, Scheduling algorithms for multiprogramming in hard real-time environment, *Journal of the ACM* **20**(1), 40–61.

Lu, C., Stankovic, J., Abdelzaher, T., Tao, G., Son, S. and Marley, M.: 2000, Performance specifications and metrics for adaptive real-time systems, *Real-Time Systems Symposium*.

Mejia, M., Simon, D., Belmans, P. and Borrelly, J.: 1989, Mécanismes de synchronisation dans un système robotique réparti, *Séminaire franco-brésilien sur les systèmes informatiques répartis*, Florianópolis, Brazil.

Murata, T.: 1989, Petri nets: Properties, analysis and applications, *Proceedings of the IEEE* **77**(4), pp 541–580.

Nilsson, J., Wittenmark, B., Törngren, M. and Sanfridson, M.: 1998, Timing problems in real-time control systems, *Technical Report DICOSMOS project ISRN KTH/MMK–98/20–SE*, KTH, Stockholm.

Puschner, P. and Burns, A.: 2000, Guest editorial : A review of worst-case execution-time analysis, *Real Time Systems* **18**(2-3), pp 115–128.

Ramamoorthy, C. and Ho, G.: 1980, Performance evaluation of asynchronous concurrent systems using Petri nets, *IEEE Trans. on Software Engineering* **6**(5), 440–449.

Ryu, M., Hong, S. and Saksena, M.: 1997, Streamlining real-time controller design: from performance specifications to end-to-end timing constraints, *IEEE Real Time Systems Symposium*.

Sandström, K. and Norström, C.: 2002, Managing complex temporal requirements in real-time control systems, *9th IEEE Int. Conf. and Workshop on the Engineering of Computer-Based Systems (ECBS'02)*, Lund, Sweden.

Sanfridson, M.: 2000, Problem formulations for qos management in automatic control, *Technical Report TRITA-MMK 2000:3, ISSN 1400-1179, ISRN KTH/MMK–00/3–SE*, KTH, Stockholm.

Sha, L., Rajkumar, R. and Lehoczky, J. P.: 1990, Priority inheritance protocols: An approach to real-time synchronization, *IEEE Transactions on Computers* **39**, pp. 1175–1185.

Simon, D., Castillo, E. and Freedman, P.: 1998, Design and analysis of synchronization for real-time closed-loop control in robotics, *IEEE Trans. on Control Systems Technology* **6**(4), 445–461.

Simon, D., Espiau, B., Castillo, E. and Kapellos, K.: 1993, Computer-aided design of a generic robot controller handling reactivity and real-time control issues, *IEEE Trans. on Control Systems Technology* **1**, 213–229.

Simpson, H.: 1997, Multireader and multiwriter asynchronous communication mechanisms, *IEE Proceedings-Computer and Digital Techniques* **144**(4), 241–244.

Spuri, M. and Stankovic, J.: 1994, How to integrate precedence constraints and shared resources in real-time scheduling, *IEEE Transactions on Computers* **43**(12), 1407–1412.

Törngren, M.: 1998, Fundamentals of implementing real-time control applications in distributed computer systems, *Real Time Systems* **14**(3), 219–250.

Wittenmark, B.: 2001, A sample-induced delays in synchronous multirate systems, *European Control Conference*, Porto, Portugal, pp. 3276–3281.