Design and Analysis of Synchronization for Real-time Closed-loop Control in Robotics

to appear in *IEEE Transactions on Control Systems Technology*

Daniel SIMON
INRIA Sophia-Antipolis, B.P. 93
06902 SOPHIA-ANTIPOLIS Cedex, FRANCE
(corresponding author. e-mail: dsimon@sophia.inria.fr)

Eduardo CASTILLO CASTANEDA
Universidad Autonoma de Queretaro, Facultad de Ingenieria, Centro Universitario
76010-Queretaro, Qro., MEXICO
(work performed during PhD studies at INRIA under grant of CONACYT)

Paul FREEDMAN
CRIM, 1801 Av. McGill College, Bureau 800
H3A 2N4 MONTREAL, Québec, CANADA

Abstract

In the framework of the ORCCAD system, periodic and multi-rate control laws are implemented in terms of a set of computing tasks to be executed under a real-time operating system. Simulations and experiments demonstrate that partially synchronizing such tasks can improve the practical performance of the implementation. However, using synchronizations may lead to deadlocks or temporal inconsistencies. In this paper, we examine the consequences of introducing such synchronization in terms of structural and temporal problems which may occur and how they may be detected using Petri net modeling and analysis. We conclude with some guidelines about how to add such synchronization to design deadlock-free and efficient implementations of real-time periodic control laws.

Keywords: Robots, real time systems, multitasking, synchronization, Petri nets, software verification and validation

1 Introduction

ORCCAD is a set of design, programming and verification tools aimed to ease the development of efficient and safe robotic tasks and applications[1] [28]. It is based on a bottom-up approach, where a basic assumption is that many complex robotic actions can be stated and efficiently solved using Automatic Control theory, e.g. the Task function approach [23]. Therefore, the first items to be designed and verified are the so-called Robot-Tasks (RT), i.e. closed-loop control laws encapsulated in a reactive logical behavior. These RTs are then logically and temporally composed to build more complex actions and full robotic missions as explained in a complementary paper [9].

Usually, the design of control laws for non-linear systems like robots is made in continuous time and allows for checking mainly qualitative properties like stability. At run time, these control laws are generally implemented as multi-tasks programs controlled by a Real Time Operating System (RTOS) on a single or multiprocessors target. Multi-tasks programs permits modular programming, software reusability and design of multi-rate controllers. This last feature is useful when the plant exhibits subsystems with different dynamics [29], when feed-forward paths are used to update some parameters of the controller or when measurements come from sensors of different kind running at different rates [19]. Also, it can be useful to optimize computing resources, e.g. for un-tethered underwater vehicles or planetary rovers where both on-board space and energy are strongly limited.

Actual control laws must meet end-users' requirements like the maximum value of tracking errors, time of response and perturbation rejection. Besides the algorithm in use, these quantitative performance indices strongly depends on the actual implementation and in particular on sampling rates and computing latencies. Unfortunately, non-linear systems control theory do not provide tools to analyze or synthesize such sampled control laws with respect to output performance indices. On the other hand, research on real-time operating systems deals with important issues like schedulability, fault tolerance or liveness but do not measure the impact of the organization of the controller on the controlled process. In fact, rather few such work is reported.

In [6], Chen et al make a first attempt to analyze the impact of synchronization between concurrent control tasks on the tracking accuracy of an industrial manipulator. In a further report [1] Armstrong computes the average latency in a two tasks controller and concludes that the way they must be synchronized depends on the ratio between their respective durations. In [14] Khosla reports experiments on a direct driven arm while varying the sampling rate of single loop controllers and shows that higher sampling rates allows for using higher gains with respect to stability. Whitcomb et al [30] computes the cross latency matrix of a multi-variable control system for a robot arm implemented on a network of Transputers where both asynchronous and handshaking communications are available between the processors. In a more recent paper [26], Shin et al refers to delay and loss problems according to the respective values of the sampling rate and of the latency and computes upper bounds for the values of the gains of the controller of a robot arm.

In this paper we report some results about the liveness of sets of synchronized real-time tasks in the framework of the ORCCAD system. In the next section we define the main entities of the lower level of our system, i.e. tasks and synchronization protocols together with the temporal properties we want to check. In section 3 we design a Petri net model of the network of communicating tasks and show in section 4 how it can be used to verify deadlocks freedom. Extending the model in section 5 allows to check for some kind of temporal inconsistency. A synthesis method to build deadlock free, partially synchronized control laws is given in section 6. Further possible improvements are outlined in the conclusion.

[1] http://www.inrialpes.fr/iramr/pub/Orccad/orccad-eng.html

2 From control laws to multi-tasks programs

2.1 Robot-tasks and Module-tasks

The *Robot-Task* (RT) in ORCCAD is the minimal granularity seen by the end-user at the application level, and the maximum granularity considered by the control systems engineer at the control level. It characterizes in a structured way continuous time closed loop control laws, along with their temporal features related to implementation and the management of associated events. More formally, a RT is the entire parameterized specification of:

- an elementary servo-control task, i.e. the activation of a control scheme *structurally invariant* along the task duration;
- a logical behavior associated with a set of signals (events) which may occur before or during the task execution.

The translation of the continuous-time specification into a description taking into account implementation aspects is made by adding *temporal properties*, i.e. discretization of the time, durations of computations, communication and synchronization between the involved processes. This is done by defining each RT in terms of communicating real-time computing tasks called *Module-Tasks* (MT) which each implement an elementary part of the control law.

Most of the MTs are periodic tasks: some perform the calculations involved in the computation of the control algorithm, e.g. the task Jacobian matrix or the control torque. Others, called *observers*, monitor conditions like reaching a joint limit or a Jacobian singularity. These observers may trigger events which are classified into preconditions, postconditions and exceptions and participate in the reactive management of the RT. The non-periodic *reactive* behavior of the RT is handled by a special MT called the Robot-task Automaton (RTA) which may be awakened by events coming from the RT itself or from an embedding *Procedure*. These events represent the external view of the RT and are further used to logically and temporally compose RTs into more complex Procedures up-to the mission level [9].

Since we expect that in most cases, the MTs will be distributed over a multiprocessor target architecture, ORCCAD makes available various message passing mechanisms over typed ports. Moreover, in order to ease the automatic code generation from the dedicated Graphical User Interface (GUI), the structure of the periodic MTs is as shown in Figure 1. Such a structure clearly separates calculations, related to control algorithms issues, and communications, related to implementation aspects and calls to the underlying operating system.

The temporal attributes of a MT are:

- its (nominal or worst case) duration d_i, which mainly depends upon the algorithm, the programming language used to encode it (generally C or C++), and the target microprocessor. It is estimated for simulation and schedulability analysis purpose.
- its activation period τ_i (the reciprocal of sampling rate measured in Hz).
- the set of input and output ports of the MT and their associated communication protocols.
- their priority, allowing them to be scheduled at run-time by the operating system
- their assigned processor in case of a multiprocessors target

Note that the ports of each MT (except the RT Automaton) are read or written in the order they are declared.

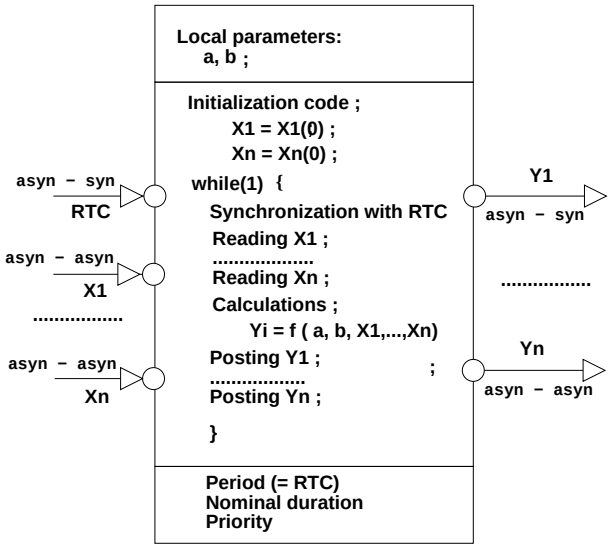

Figure 1: Basic structure of a periodic Module-Task

2.2 Performance versus synchronization

In complex robotic control systems, there are often closed loops (in the sense of control laws) executing at different sampling rates. For example, the data associated with feedback paths may be updated more frequently than data associated with feed-forward paths. Indeed, controller performance may also be influenced by how tightly cooperative tasks are coupled together, according to their respective durations ([1] [6] [30]).

Simulation and experiments are the only way to estimate the quantitative performance of a non-linear system controlled by a multi-tasks system. We show now some simulation results using the GUI of ORCCAD and its underlying simulation software SIMPARC [2], an hybrid simulator able to run concurrently a numerical integration of the dynamics of the plant and the execution of the control code, including the temporal features of the MTs.

The first example deals with the control of a robot arm for trajectory tracking in the joint space. Two algorithms were compared. The first one uses simple fixed gains Proportional-Integral-Derivative (PID) decoupled control. The second one is more elaborated and is known as computed torque control: in fact, it uses the additional MOD.DYN MT to compute in real-time estimates of M, the inertia matrix of the arm and N, the vector of Coriolis, centrifugal and gravity forces. The corresponding *Temporal-Constrained Specification* (TCS) is given by Figure 2 and simulation results are shown by Figure 3. The TCS combines the control block-diagram to which are added the periodicity, duration and synchronization constraints[2], excluding the description of implementation on a target architecture: thus it may be simulated on an arbitrarily fast fictitious execution machine for control investigation purpose.

Figure 3a shows the tracking error for the first joint, using the PID control law with different sampling periods for the control computation task. As expected from sampling theory, we can see that decreasing the computing period leads to decrease the tracking error, even if the gains remain constant.

More interesting is the comparison between these two control laws given by Figure 3b. As hoped, using an explicit model of the arm dynamics allows to strongly decrease the tracking error at high speed while using smaller gains, thus allowing for a reduction of the control noise. We can also check an old assumption: the computation of the dynamic model of the arm can be run at a period slower than the one of the main loop of the control law, with small effects on the global performance. This way, we can optimize the use of the computing power but obviously, as the periods are different, this task *must not* synchronize with the others.

[2]Due to the lack of an underlying control theory, finding convenient values for these temporal attributes remains art work based on experience and simulations.

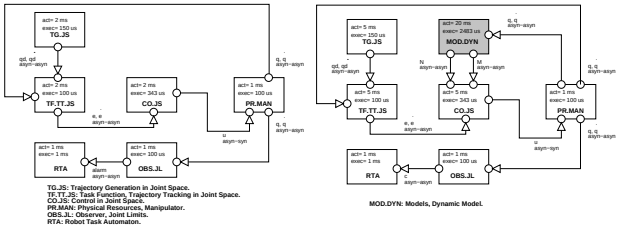

Figure 2: TCS block-diagrams a) PID control b) Computed torque control with the additional MOD.DYN MT

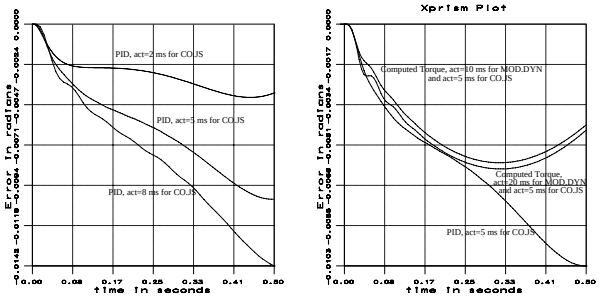

Figure 3: Tracking errors: a) PID control with different sampling rates – b) Computed torque control vs. PID control

At first glance, the simplest way to run the control algorithm should be to not synchronize the MTs, using for example a client/server mechanism e.g. like in [10]. In fact, if we only use asynchronous communication between MTs having the same priority, the RTOS's scheduler will trigger them in an arbitrary order, e.g. the order MTs were drawn through the GUI, without respect to data and time dependencies. For example, in the upper part of Figure 4 we design a pipe-line of four MTs we expect to run according to the sequence {MT1, MT2, MT3, MT4} at each period of the RT. Actually, since they have the same priority and are not synchronized, the RTOS is left to schedule them according to some hidden internal mechanism. The worst case is shown by the lower part of Figure 4 where the execution order of the computing path is completely reversed, with MT4 running first and MT1 running last : although control outputs are sampled at the expected rate, the latency between the k^{th} measure of sensor q and the instant when this measure is used in the corresponding output $U[q(k)]$ is several time larger than the sampling period. It may also cause initialization problems during the execution of the first loops. Conversely, using adequate synchronization can enforce a MTs execution ordering compatible with data and time dependencies between algorithmic modules.

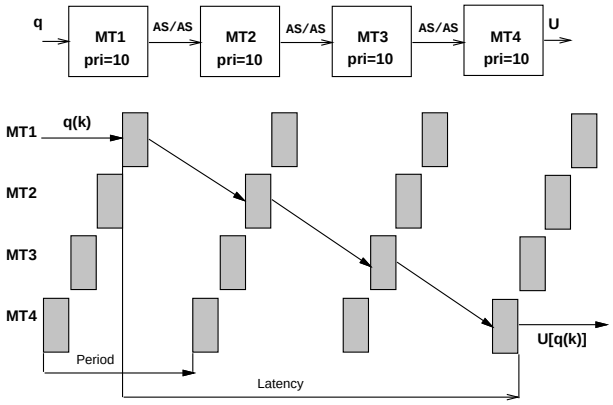

Figure 4: Reverse ordering in a MTs pipe-line: *top*: Computing path design *bottom*: Actual execution

Another example given in Figure 6 shows simulation results concerning a trajectory tracking algo-

rithm in SE^3, the 6-dimensional space of frames (i.e. the operational space for the arm's tip), which TCS is given by Figure 5. In the first case (sim1), the tasks were not synchronized leading to large latencies, e.g. due to reverse pipe-line execution, and thus large tracking errors and unstability. In the second case, some tasks were synchronized (thick lines), leading to far better performance. It is worth noting that we only changed the organization of the implementation while both simulations used the same gains and the same amount of simulated computing power [5].

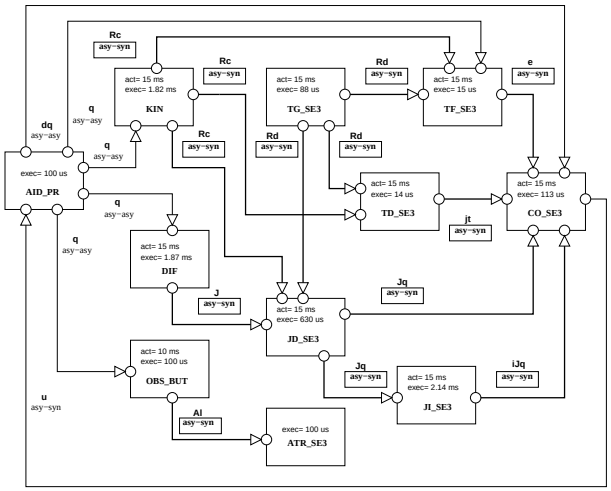

Figure 5: TCS of trajectory tracking in the SE^3 operational space for a robot arm

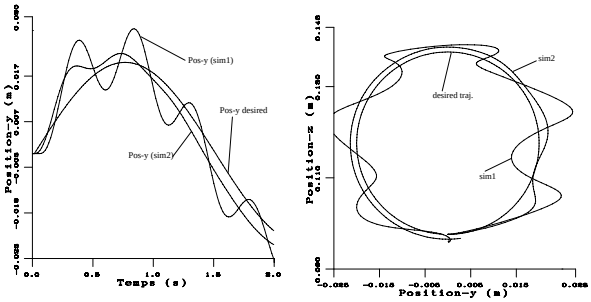

Figure 6: Tracking errors: a) tip position vs. time b) tip position in the y-z plane

2.3 Message passing and synchronization

ORCCAD makes available different communication and synchronization mechanisms to link pairs of MTs via their typed ports. In the sequel, we shall be concerned with the following four [17]:

- ASYN-ASYN: Each task is running freely and the communication does not add synchronization.
- SYN-SYN: The first task to reach the *rendez-vous* is blocked until the second one is ready.
- ASYN-SYN: The writer is running freely and posts messages on its output ports at each period. The reader either reads the message if a new one is available or is blocked until the next message is posted.
- SYN-ASYN: A symmetrical of the previous one. The reader runs freely, the writer is blocked up to the next request except if a new one was posted since the last reading.

Since we are mainly concerned with closed-loop control, we shall often consider that the best data is either the last or the next available one. Thus, these protocols do not provide data queuing and

generally allows loss of data, with exceptions for those which are used to send signals to the Robot-task and application automata to manage the logical behavior of control laws.

In any case, we will now focus only on the periodic behavior of the set of MTs used to perform the algorithmic part of the RTs.

2.4 Temporal behavior

The practical efficiency of a closed-loop control algorithm is essentially determined by its implementation. The two major items to be considered are the sampling rate of the control law and the latency between a sensor measurement and its effective use in the control applied to the robotic system. Increasing the sampling rate and decreasing the latency makes it possible to increase control gains, without creating unstability. Thus, the performance of the control as measured by tracking accuracy and robustness with respect to modeling errors can be improved e.g. [23] [13].

Clearly, the overall temporal behavior of a RT depends upon the temporal attributes of its MTs and how they communicate (how they are synchronized). Since this is of critical importance for RT performance, some temporal analysis must be performed. In addition, certain kinds of analysis will also later facilitate the assignment of MTs to processors, by identifying those MTs which, for example, never execute at the very same time.

While simulations using Simparc [2] provide information useful for tuning, for example, control parameters such as the minimum period τ required for a control loop, simulation results are somewhat difficult to examine and more formal methods can provide complementary kinds of analysis. Verifications are carried out using the *synchronization skeleton* of the RT, i.e. a description of the temporal and synchronization constraints existing inside the set of MTs. This description takes into account the MTs period and duration, the *ordered* list of associated communication ports and the kind of synchronization associated to connected ports.

At the very least, we suggest there are two temporal aspects to RT behavior which must be checked:

- Compatibility between a task duration and its activation period.
- Deadlock avoidance and consistency of temporal constraints in the resulting synchronization skeleton.

The first one is easy to check at the GUI level for a MT with an explicit period: we must have a necessary condition $d_i \leq \tau_i$. But whenever a MT engages in synchronous communication, it necessarily binds its real-time behavior to other MTs and therefore other activation periods. For MTs whose period is implicitly defined in terms of synchronization with other MTs, this value must be recovered from the synchronization skeleton. Finally, a necessary schedulability condition for a control path where all the MTs are run in sequence is $\tau \geq \sum_{i=1}^{n} d_i$.

Design inconsistencies may arise in several ways. *Structural* deadlocks are due to the synchronization structure itself whatever are the numerical values of temporal attributes. In addition, badly chosen numerical values of temporal attributes like tasks period and duration may lead to *temporal inconsistencies* and unsafe (e.g. non-periodic) behavior of the RT, even if it is free from structural dead-locks.

Structural deadlocks may be associated with circularity in inter-MT communication. For the example shown in Figure 7 on the left, each of the three MTs is blocked while waiting to "read". Others structural deadlocks are more subtle and emerge when the order in which a MT communicates over multiple ports is incorrectly specified. In the example shown in Figure 7 on the right, MT2 blocks while waiting to "read" from MT3 which blocks MT1 from "writing" which in turn blocks MT3 from "reading". Clearly, this structural deadlock may be eliminated simply by having MT2 "reading" first from MT1 and then from MT3, but *identifying* such deadlock on the GUI screen is usually difficult because of the complexity of some control laws where many synchronizing links are interleaved as shown in the TCS depicted by Figure 5.

Therefore, we need modeling and analysis tools to automatically check for deadlock avoidance in the network of synchronized MTs. While several methods and associated computing tools have been

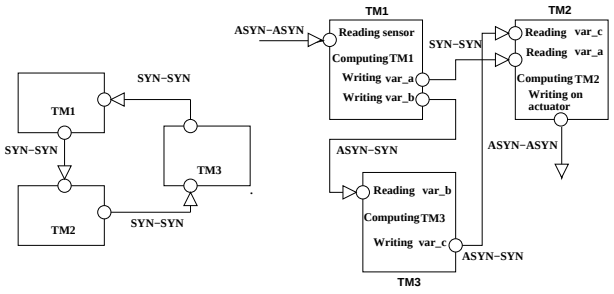

Figure 7: Two kind of structural deadlocks: a) Synchronization circularity b)Wrong ordering of the input ports of MT2

investigated, like model checking on Timed Graphs [16] and on data flow synchronous programs encoded with Signal [15], Petri nets theory provided a simple and efficient way to carry out this task.

In the next section, we introduce a Petri net (PN) model of the MTs behavior which makes it possible to address such structural analysis in a formal way and in particular, verify that the RT as defined by the control systems engineer in terms of communicating MTs is deadlock-free.

3 Modeling MTs and synchronization using Petri Nets

Petri Net (PN) theory [18] is gaining increasing importance in the robotics literature for the discrete event modeling and analysis of robotic systems e.g. [11], since it offers a convenient way of expressing system behavior which is both parallel, asynchronous and distributed. In addition to the precedence constraints among robotic actions, looser couplings associated with shared resources can also be directly expressed, as well as the repetition of certain actions (or sequences of actions). Besides modeling capabilities and perhaps most importantly for our concern, the PN can be analyzed in a *formal* way to obtain information about the dynamic behavior of the modeled system.

In this section, we shall develop a particular kind of PN suitable for the modeling of MT behavior which is periodic, and then demonstrate how the PN model of a RT composed of communicating MTs may be analyzed for the presence of deadlock. Our presentation throughout will be rather informal and uses existing results taken from [7] and [18].

3.1 A PN model of periodic MTs

The simplest PN model associates events with *transitions* and conditions (which may be true or false) with *places*. Places and transitions are linked by directed edges called *arcs*. The presence of a *token* in a place indicates that the corresponding condition is true. The *marking* therefore describes the state of the PN in terms of conditions which are true and those which are false. When all of the input places of a transition become marked (have a token), the transition 'fires', i.e. the event occurs; the firing removes a token from each of the input places (making the pre-conditions false) and deposits a token in each of the output places (making the post-conditions true).

As shown in Figure 8, the sequential behavior of the simplest periodic MT (reading an input port, performing a calculation, writing to an output port) may be modeled by a condition/event PN with three transitions. (Of course, when a MT has multiple input and output ports, we must be careful to associate a distinct transition with each read and each write.) A fourth transition is required to activate the MT subject to the periodic awakening associated with a real time clock (RTC), also modeled by a PN. As we shall be further concerned with temporal analysis we may add some temporal properties to the model to obtain a *timed Petri net*, i.e. a PN where durations are associated with some transitions or places. Following [21] and [22], we have elected to associate the duration [d] of the MT with the calculation step transition, and thereby assume that reading and writing are instantaneous events, i.e. events of zero duration. 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).

Since each place has just one input transition and one output transition, each associated condition may only become true in one way and can only have one consequence. The MT behavior is therefore deterministic, and the resulting PN is a so-called connected[3] *marked graph*. Properties of marked graphs shall be of particular importance when we turn to the question of deadlock analysis.

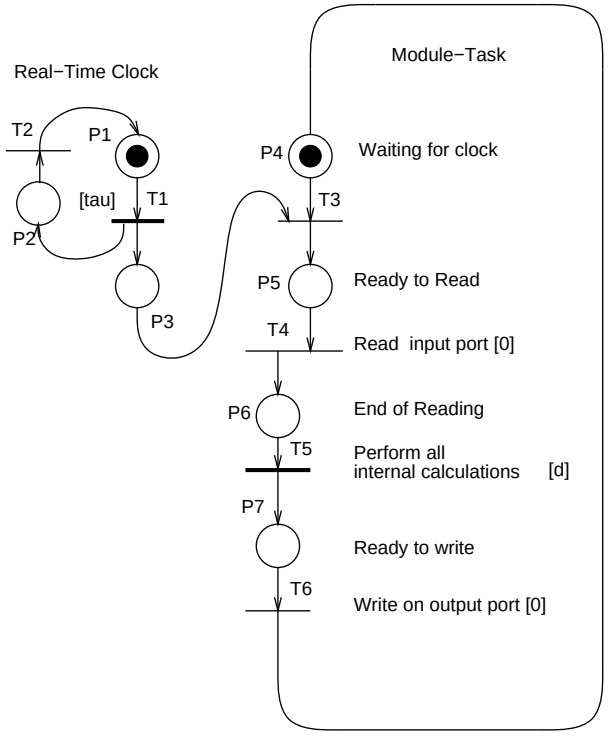

Figure 8: A Petri net model of a periodic Module-Task.

ASYN/SYN communication between two MTs is modeled as shown in Figure 9 on the left and SYN/SYN communication is modeled as shown in Figure 9 on the right. Note that the transition associated with the periodic awakening of MT2 is no longer present, since the temporal behavior of MT2 is bound to that of MT1. Once again, the combination of the two PNs is a connected marked graph.

Note that ASYN/ASYN communication do not add synchronization constraints and thus have no model in this synchronization skeleton, i.e. MTs communicating using this protocol have disconnected models.

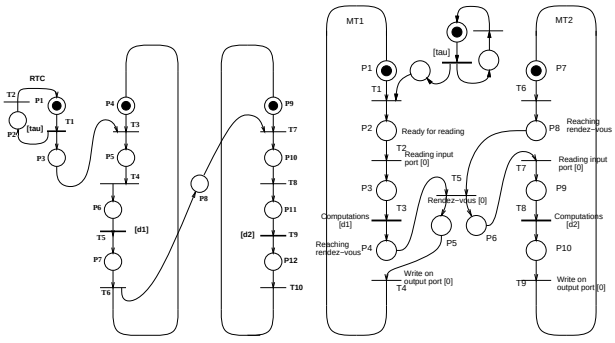

Figure 9: PN models for ASYN/SYN and SYN/SYN communications

Using these PN models for the MTs and synchronization protocols, we are now able to define the

[3]A PN is connected if there exists a non-directed path between any two elements (place or transition) of the PN. It is strongly connected if these paths are directed, i.e. respect the direction of the arcs

PN model of the set of synchronized MTs of Figure 7b. This model is shown by Figure 10.

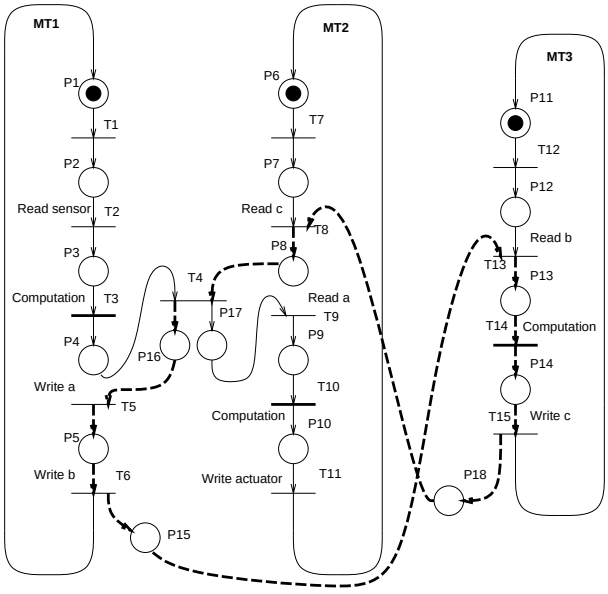

Figure 10: A Petri net model of the three MTs with deadlock.

4 Structural analysis

A basic property we want to check is structural dead-lock freedom. More precisely, we would like to ensure that given any sensible initial marking, deadlock cannot occur. This is important because we are interested in modeling repetitive behavior of periodic control laws. This property has corresponding definitions in the PN and marked graph theory [18].

Definition 1 *A PN is live (L) with respect to an initial marking M_0 if from any marking in $[M_0]$, there exists for each transition a legal firing sequence leading to a marking in which that transition is (necessarily) enabled.*

As deadlock arise when nothing happens, i.e. no transition can be longer enabled, liveness implies deadlock freedom and allows to use two important theorems:

Theorem 1 *For a marked graph, the token count in a directed circuit is invariant under any firing, i.e.$M(C) = M_0(C)$ for each directed circuit C and for any M in $R(M_0)$, where $M(C)$ denotes the total number of tokens on C and $R(M_0)$ is the set of markings reachable from M_0.*

And therefore:

Theorem 2 *A marked graph (G, M_0) is live if and only if M_0 places at least one token on each directed circuit in G.*

For example, the directed circuits of the marked graph shown in Figure 9 on the left are $s_1 = \{p_1, p_2\}$, $s_2 = \{p_4, p_5, p_6, p_7\}$ and $s_3 = \{p_9, p_{10}, p_{11}, p_{12}\}$. Therefore, any marking which assigns at least one token to each of s_1, s_2, s_3 is live. Moreover, given a live initial marking, the firing behavior of a marked graph is repetitive. In other words, in a deadlock free network of MTs, every directed circuit of its model is assigned at least one token in the initial marking. But identifying all of the simple cycles in a more complicated marked graph such as the one shown in Figure 10 cannot be done by visual inspection.

Rather than finding circuits on the graphical representation of PNs we check liveness using invariant structural properties of the net. Informally, invariants are properties of the logical structure of a PN,

and therefore characterize in some way all possible system behaviors. Invariants associated with places are usually called (in the English literature) *p-invariants* and represent unchanging truths about sets of conditions, such as mutual exclusion. Typically, we are most interested in the sets of minimal linearly independent positive invariants which are characteristic vectors, i.e. whose entries are all 0 or 1. When the PN is a marked graph, it is well known e.g. [18] that the set of arcs in a directed circuit is a support for a minimal p-invariant and therefore an initial marking is live when each minimal p-invariant is assigned at least one token.

We now present a mathematical representation of PNs allowing an automatic search of the p-invariants. For a p places and t transitions PN, its incidence matrix $W(p,t)$ is a $p\times$lines and $t\times$columns matrix where[4]:

- $W_{ij} = 0$ if there is no arc between place P_i and transition T_j
- $W_{ij} = 1$ if P_i is an output place of T_j
- $W_{ij} =$ -1 if P_i is an input place of T_j

The marking vector M is a p dimension vector, describing the marking of the PN during its evolution. The value n of M_i denotes the number of token that place P_i owns at a given instant. The initial marking vector M_0 is of prime importance for deadlocks checking. It is worth noting that, thanks to the structure of our MTs, tokens in the initial marking are always in the starting place of the PN, i.e. the task is ready to read its first input port. For example, the elementary PN described by Figure 8 is also described by the following incidence matrix and initial marking vector:

$$W(p,t) = \begin{bmatrix} -1 & 1 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 1 \\ 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix} \qquad M_0 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Let us now consider the following equation:

$$W(p,t)^t \;.\; X \;=\; 0 \tag{1}$$

where $W(p,t)^t$ is the transpose of the incidence matrix and X a p-dimension vector, and assume that $X = V$ is one of its solution. According to [7], it can be shown that the form $V^t \;.\; M_i$ is constant for every accessible marking, and in particular for the initial marking M_0. In consequence, the p-invariants are the solution of:

$$V^t \;.\; M_i \;=\; \sum_{i=1}^{p} v_i \;.\; M_0(P_i), \qquad V^t \;=\; [v_i, v_2, ..., v_p] \tag{2}$$

A simple algorithm using linear programming with integers, also given in [7], allows to compute the p-invariants, and therefore to conclude about the liveness of the PN model of the set of MTs. This algorithm has been encoded inside a C package allowing to automatically build the PN model from the GUI of ORCCAD, search the p-invariants and perform the liveness analysis. The complexity of this algorithm is polynomial in the number of elements of the net [27]. For a medium set of MTs like the one given Figure 5 the whole process runs in less than 0.2 seconds on a Sparc 5 workstation.

Using this package, the minimal p-invariants of Figure 10 may be identified as follows:
$s_1 = \{p_1, p_2, p_3, p_4, p_{16}, p_5\}$, $s_2 = \{p_6, p_7, p_8, p_{17}, p_9, p_{10}\}$, $s_3 = \{p_{11}, p_{12}, p_{13}, p_{14}\}$, and
$s_4 = \{p_{15}, p_{13}, p_{14}, p_{18}, p_8, p_{16}, p_5\}$. If we consider the initial marking corresponding to all three MTs waiting for activation (places p_1, p_6, p_{11} marked), the reader may verify that s_4 (in doted line) has no token and therefore the marking is not live.

[4]there is no loss of information if the PN is pure, i.e. no place have a same transition as both input and output.

By modifying Figure 10 so that MT2 reads from MT1 first and then from MT3, we obtain the marked graph model as shown in Figure 11. P-invariants s_1, s_2, s_3 are as before but now $s_4 = \{p_{15}, p_{13}, p_{14}, p_{18}, p_9, p_{10}, p_6, p_7, p_{16}, p_5\}$. The reader may verify that the initial marking $\{p_1, p_6, p_{11}\}$ is now live as required, since p_6 appears in s_4.

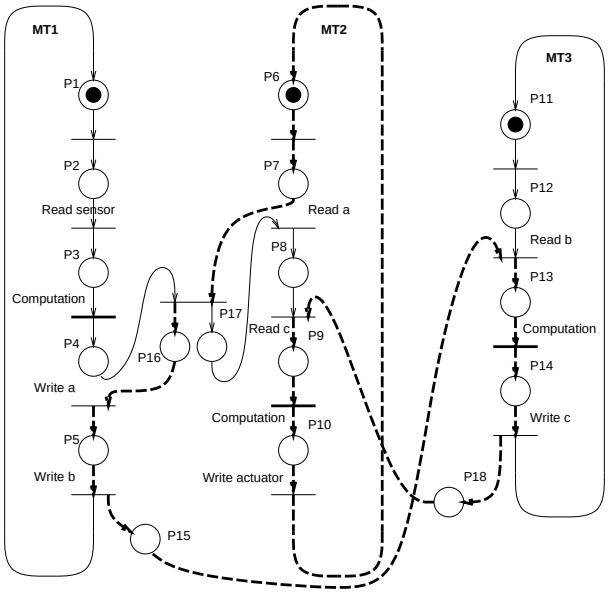

Figure 11: A deadlock-free Petri net model of the three MTs.

5 Temporal analysis

Temporal inconsistencies can occur in complex networks of MTs, where the interleaving of computing paths can hide, for example, multiple synchronization. As a result, the beginning of execution of a MT can be delayed at each activation of the control law, leading to a temporal mismatch and a possible system crash.

Figure 12 shows such an incoherent specification due to a double synchronization of task MT2, one by a synchronizing reading on the output of MT1 and the other one by an explicit period declaration. After its third execution, the operating system tries to awake it while the previous execution was not finished and the system goes down.

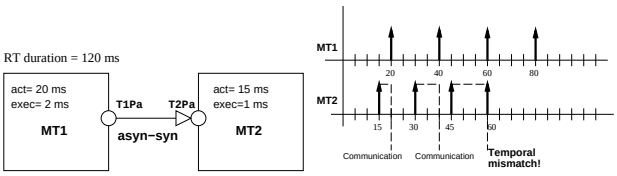

Figure 12: Temporal specification inconsistency due to double synchronization

To detect such situation, we added additional places and transitions to the previous Petri net model, as depicted by Figure 13. In fact, these places are only added to explicitly periodic tasks, using a synchronizing input port connected to a physical clock.

These additional places and transitions allows to catch tokens in a sink place in case of the aforementioned failing behavior, i.e. the clock tries to restart a task which did not achieve its computations. Looking at the model of MT2 in Figure 13, we see that the first transition T_8 is triggered when the real-time clock (RTC) deposits a token in P_{10} while the task is ready to start (token in P_{11}). Thus, these two token are removed, and one is added in P_{17}, allowing to fire the very last transition of MT2

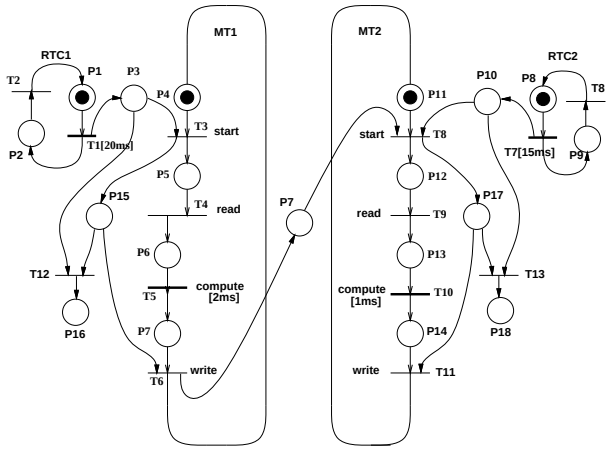

Figure 13: Extended model of module-tasks with additional sink places P16 and P18

before its starting place. In case of temporal inconsistency, this token will be still there at the next clock tick: T_{13} will be fired, a token will be locked in the sink place P_{18} and MT2 will never restart since T_{11} cannot be longer enabled.

Such situation can be checked by observing the evolution of the reachability graph (RG) of the PN until either the normal completion of the Robot-Task or the occurrence of such a deadlock. Unfortunately, enumerating all the states of the marking can be very expensive and time consuming, especially if the PN is complex and the nominal duration of the RT is long. In fact, a connected timed PN will reach a steady state (periodic) behavior after a finite time [7]: thus the PN model of a consistent RT will reach this periodic behavior before being trapped in a sink place. Searching for this steady state behavior consists in going through the reachability graph until a marking already visited is once again reached. This operation is usually faster than a systematic exploration of the RG up to the completion of the RT. Figure 14 shows such a RG where the repetitive behavior is reached after 75ms (most often the duration of the transient phase before reaching a periodic behavior is not equal to the steady state period). These new models and analysis methods have been integrated in our aforementioned software.

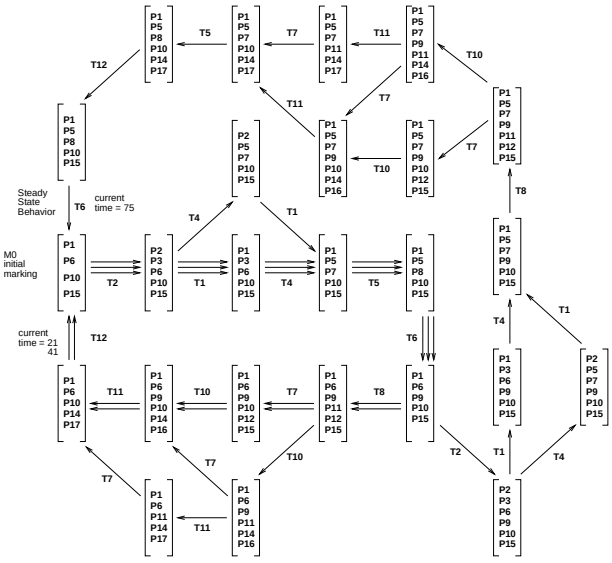

Figure 14: Reachability Graph with a Periodic Steady State Behavior

6 Sufficient conditions to design deadlock-free Robot-tasks

Many lessons have been learned from extensive mathematical modeling, analysis, simulations and experiments in ORCCAD about the design of computing architectures for implementing multi-rate closed loop control laws. Hereafter we present sufficient (but not necessary) conditions about a synchronization scheme that can be adopted by the control engineer for ensuring that the RT is deadlock free. Moreover the organization of MTs we propose allows to minimize computing latencies in order to improve the control performance.

First we begin by summarizing key aspects of the "translation" of control algorithms (closed loop control laws) into computing tasks.

- As we deal with control of physical processes, data always flow from sensors to actuators along several, and sometimes cross-coupled, control paths. It is assumed that data are always available from the sensor device drivers and can be posted at any time to actuators, i.e. communication with the physical plant never blocks.

- It is often the case that all parts of the control algorithms need not be computed at the same rate. In particular, feed-forward paths and parameter adjustments are typically executed more slowly (less frequently) than feedback error computations. Using multi-rate sampling can also optimize computing power. Stated more simply, control algorithms can often be decomposed into *groups of cooperating tasks* running in sequence and executing with the same period.

- In order to implement (robotic) control laws efficiently, it is desirable to impose a certain ordering on the execution of sequential groups of tasks along the various control paths, i.e. synchronization among data-dependent MTs. Just using asynchronous (ASYN/ASYN) communication can lead to an unexpected (and undesirable) ordering of tasks at run-time by the operating system.

- Using MTs to design RTs promotes a modular approach and reuse of previously encoded functions and models. However, mapping each MT to a thread (or tasks in the RTOS terminology) at run-time increases operating system overhead due to numerous context switches and inter-process communication. Thus, assigning MT groups which may be sequentially executed to threads by adding a certain amount of synchronization is desirable in order to reduce such overhead and use CPU resources more efficiently. As ASYN/ASYN communication between groups does not add synchronization constraints the synchronization skeleton of the RT is not changed.

- The control engineer's requirements are usually expressed as performance indices about the outputs of the physical system, e.g. maximum permissible tracking error. The synchronization constraints added to the specification of the control block-diagram to better organize the execution of the control algorithm are just a mechanism for better achieving such performance requirements. Decreasing the computing latencies may be addressed at design time through these mechanisms.

- Increasing the number of instances of synchronization increases the risk of deadlock and temporal inconsistency. Given the kinds of control laws typically found in a robotics context, we have found that few strong synchronization using SYN/SYN mechanisms are necessary and must be avoided as far as possible.

- Temporal inconsistencies arise when multiple synchronizations are in conflict. In fact, in most cases it is meaningless to assign a task an explicit period (using synchronization with a clock) and at the same time binding its temporal behavior through synchronization to other tasks. Looking at the previous example given Figure 12, even the notion of period for MT2 is questionable in such a case. The system of tasks must behave in a deterministic manner for schedulability analysis and run-time safety purpose.

Given such a context, we now turn to guidelines about the design of multi-tasking computing architectures which exhibit no deadlocks, no temporal inconsistency and tend to minimize the computing latencies.

While our PN model may be used to check for deadlocks in the inter-task communication, we suggested in a recent work [4] how the communication ports of each MT might be systematically reordered until we either obtain a deadlock-free network of tasks or we exhaust all possible choices and conclude that the control specifications are in error. A drawback of this method is that it may be unsuccessful if too many synchronizations were used in the design. Another one is its exponential complexity, even if in practice the sizes of problems to be solved are not enormous. More importantly, this approach fails to take into account the constraints associated with performance optimization, i.e. reducing the latency between measurements and corresponding changes to control set-points.

Therefore, under these assumptions we suggest the following sufficient (but not necessary) conditions useful for systematically designing multi-tasking computing architectures (networks of communicating MTs) which are free from deadlock and temporal inconsistency and fit closed-loop control requirements as well as possible.

- *Rule 1:* Use one and only one synchronization for each MT, i.e. there is just one synchronizing communication port, either connected to another MT or to a physical clock (when an explicit activation period is specified).
- *Rule 2:* Synchronize cooperating MTs with ASYN/SYN and SYN/ASYN connections to impose execution order (such synchronized MTs will belong to a synchronized group).
- *Rule 3:* There should be *one and only one* synchronization with a physical clock for each group of synchronized MTs.
- *Rule 4:*Assuming that the real-time operating system uses a preemptive and fixed priority scheduler, assign the same priority to every task in a synchronized group and different priorities to different synchronized groups.

To illustrate these conditions in practice, we present in Figures 15 and 16 the computing architecture previously described in Figure 2b, this time synchronized following these guidelines, along with the corresponding PN description (restricted to the structural analysis part)[5].

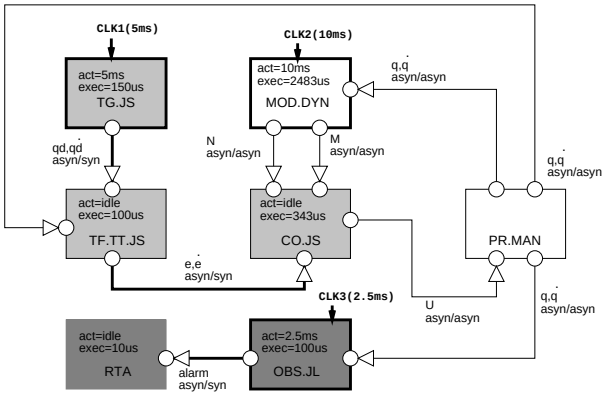

Figure 15: Correctly Synchronized Computed Torque Control Diagram

Since ASYN/ASYN communication does not add connectivity to the PN model, the global PN is now split into several *connected components*, each corresponding to distinct groups of cooperative MTs bound by synchronized computing paths.

[5]There is no PN model for the PR.MAN block which is not a MT but represents either the actual robot or its simulated model.

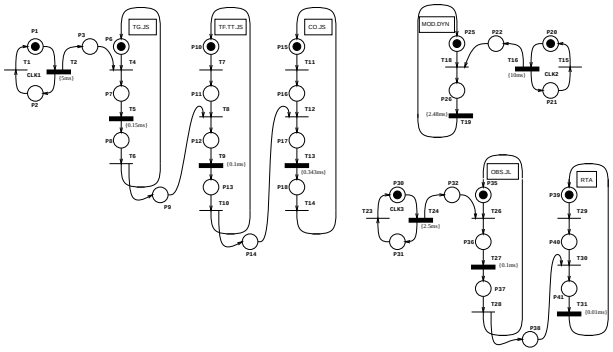

Figure 16: Petri net model of the Synchronized Computed Torque RT

Following the 1^{st} rule, no MT has multiple synchronizations, and thus the computing architecture may admit no temporal inconsistency. In that way, each MT has a well defined period inherited from the synchronizing clock of its connected component, which thus can be easily computed[6].

Following the 2^{nd} rule, communication inside every connected component only use SYN/ASYN and ASYN/SYN mechanisms. These communication mechanisms add PN connectivity in just one direction (from the asynchronous port to the synchronous port). On the other hand and according to rule 3, assigning a physical clock to the *unique* synchronizing port of one of the MTs prevents the set of MTs to loop itself. Thus, it avoids deadlocks due to synchronization circularity like the one depicted by Figure 7a. It must be remarked that, conversely to MTs, the clock cannot be preempted nor blocked[7] and in any case regularly delivers ticks to awake the set of MTs.

Therefore, using the first three rules implies that the synchronization constraints *cannot add any directed circuit* (and thus no blocking circuit) to the synchronization skeleton. In that way the only remaining minimal p-invariants correspond to the basic structure of the MTs. Since each one is assigned exactly one token in the initial marking, each connected component is live and the computing architecture is deadlock free.

These rules define sufficient but not necessary conditions to design a deadlock free RT since a counter-example can be found looking back at Figure 7b: if we reorder the input ports of MT2 we get a deadlock free set of MTs while violating the rules. In fact, these rules may not apply to all real-time applications but are efficient in the ORCCAD approach where the behavior of the RTs is dominated by the periodic execution of closed-loop control while reactivity to asynchronous events (often leading to switch the current RT to another one e.g. for graceful degradation) is handled by the RT automaton which therefore acts as a scheduling supervisor [8].

Due to asynchronous communication between the different connected components, these synchronization constraints are not sufficient to completely define the temporal behavior of the set of MTs. Choosing an adequate scheduling policy, based on priority assignment according to rule 4, allows to completely define this temporal behavior at *design time*.

Assigning one and only one physical clock to each connected component ensures that all threads are synchronized (no one is free running) and allows us to compute their common period. This is a key point for further schedulability analysis according to a chosen scheduling policy. A scheduling policy may be derived from the rate monotonic one [24], i.e. higher priority is given to threads running at higher sampling rates, which is optimal for single processors using preemption and fixed priorities.

Assigning the same priority for all the threads associated with a same connected component ensures that this synchronized group will be entirely computed before the next one, which is assigned a lower priority, is started. In fact, we must be careful to give different priorities to different groups running with the same period or to branches of a connected component which has forked to avoid undesirable switches of activity between these computing paths. Also, we must give the highest priority in the

[6]In practice, the synchronizing port will be either the first reading or the last posting one so that the MT computes with the last available data as inputs.

[7]In some cases, this clock might be an interrupt signal coming from sensor data processing [20].

network of threads to the RT automaton which manages the set of MTs' reactive behavior. According to such a policy, the timing diagram of the control law becomes predictable at specification time for a single board implementation.

In a tightly coupled multi-processor context, as we do not intend to perform dynamic task migration and thanks to the static kind of the RTs the structure of which is invariant along their duration, pools of threads will be statically assigned to processors and scheduled according to the same policy [25].

Figure 17 shows the timing diagram of a single CPU implementation of the computing architecture given by Figure 15, assuming that the three clocks have a common starting instant[8] and that the synchronizing port of each task is the first reading one. According to the rate monotonic policy, a high priority is given to the OBS.JL MT, a medium one to the {TG.JS,TF.TT.JS,CO.JS} set and a low one to MOD.DYN. The highest priority is assigned to the RTA which is awakened under interrupt upon occurrence of events on its input port.

Before the occurrence of the first clock tick all threads are locked on their synchronizing input port. Then, OBS.JL is unlocked upon occurrence of the first clock tick, runs to completion and possibly triggers the RTA, for which provision is given in the timing diagram. The set of medium priority tasks then run in sequence in the right order. Finally, MOD.DYN is started, is further temporarily suspended by the next execution of the higher priority OBS.JL observer and finishes execution during the second basic cycle of the RT. As shown by the timing diagram, this threads system is predictable and schedulable. Moreover a schedulability condition may be easily derived.

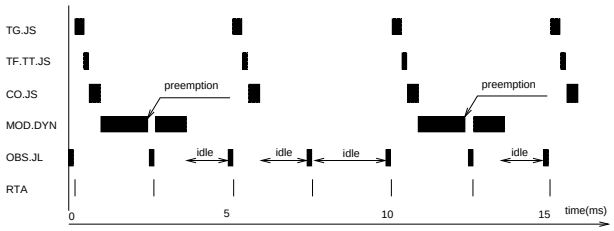

Figure 17: Timing diagram for the Synchronized Computed Torque RT

Each MT now has a well defined period which is the one associated with its connected component. Let n_i be the number of MTs in the i_{th} connected component, m the number of connected components in the Robot-Task, $d_{i,j}$ the worst case duration of the i_{th} MT in the j_{th} connected component, τ_j and D_j respectively the period and worst case duration of the j_{th} connected component and T the base period (or Hyperperiod [12]) of the RT, i.e. the smaller common multiple of the τ_j.

The schedulability condition for a connected component is simply computed as $\tau_i \geq \sum_{j=1}^{n_i} d_{i,j}$ while the schedulability condition for a set of connected components running on a given processor is:

$$\sum_{i=1}^{m} D_i \frac{T}{\tau_i} \quad \leq \quad T$$

where $\frac{T}{\tau_i}$ represents the number of execution of the i_{th} connected component during a base period of the RT. (If a connected component is split over several processors only the modules running on the processor under analysis must appear in the computation of D_i).

Finally, it becomes obvious that the MTs belonging to a given connected component like the set {TG.JS,TF.TT.JS,CO.JS} in Figure 17 (in fact those which are implemented on a same processor) can be collapsed in a single real-time thread at compile time, thus cancelling useless calls to the RTOS due to context switches and communication while preserving the global structure of the synchronization skeleton.

[8] if the subnets are synchronized by the system clock the global timed PN still will have a steady-state periodic behavior even if it is not fully connected

7 Conclusions

In this paper, we have addressed some problems related to the execution of periodic control laws implemented as multi-tasking programs, with robotics as a privileged application field. Simulations and experiments have shown that using multi-rate sampling and partial synchronization of real-time computing tasks called "module tasks" (MTs) which collectively form "robot tasks" (RTs), can improve the practical efficiency of the controller. Unfortunately, multiple synchronization can lead to deadlocks or temporal inconsistencies. Therefore, we developed Petri Net (PN) models of the periodic behavior of the set of synchronized tasks, and then used this model to perform the formal verification of their temporal behavior. Finally, under some assumptions usually met by robotic control laws we proposed a synthesis method insuring both deadlock avoidance and latency minimization in order to obtain safe and efficient multi-tasking implementations whose runtime behavior is defined using both partial synchronization and the prioritized scheduling provided by the underlying real-time operating system.

In this work, we limited our focus to the periodic behavior of control tasks, without addressing the initialization of the multi-tasking implementation. As a result, liveness and deadlock freedom are equivalent properties and basic PN theorems and simple algorithms are used to perform the temporal analysis. More precisely, we have simply assumed that the marking associated with the initialization state (all MTs waiting for their first execution) is reachable. Clearly, the case of tasks with "memory" e.g. recursive filters which must execute several times before producing their first outputs, deserves special attention. In fact, since during initialization these kinds of tasks read their input ports just once and then perform the same calculation several times before posting their results at their output ports, the structure of the PN model (i.e., of the synchronization skeleton) remains unchanged and thus our structural dead-lock analysis remains valid. Insofar as initialization is concerned, the first usable output is simply delayed in time, increasing the duration of the transient behavior before reaching the steady state periodic behavior.

While structural analysis of our PN model makes it possible to systematically detect structural deadlocks, the temporal analysis presented in section 5 only addresses one kind of temporal inconsistency. Moreover, this method relies upon the exploration of the reachability graph of the extended PN model for which the analytical "overhead" cannot be predicted (since it is dependent upon the value of temporal attributes) and and may be high. As an alternative, we note that algebraic methods to analyze timed PNs are now emerging e.g. algebra on the (max,+) semi-ring [3]. Indeed, since our basic PN models are timed event graphs, they can be translated into a *linear model* in the (max,+) algebra, thus allowing us to exploit the (max,+) counterparts of classical concepts in system theory such as state-space recursive equations, transfer functions or feed-back loops. In this way, it would be possible to analyze both the transient and asymptotic behavior of our multi-tasking controller implementations within the framework of discrete event dynamic systems, and to evaluate their performance e.g. as expressed by token production rate. We are currently pursuing such research and we expect that the temporal properties of interest to us, associated with both the transient and steady-state periodic behaviors of the RT, will be analyzable in polynomial time, and that taking into account the effects of priority-based thread preemption at runtime will be tractable.

In the work described in this paper, we have also assumed that the control law implementation was designed for a mono-processor target platform e.g. a single board computer. Of course, controlling complex systems often requires multiprocessor implementations. In particular, when there are loosely coupled elements, the assumption that the overhead associated with inter-task message passing is negligible, may no longer be valid. This, in turn, raises the issue of how best to distribute the set of MTs, RTs, and reactive elements of the controller over multiple processors so as to minimize such overhead. A related problem is how to optimize computing resources when several RTs (e.g. corresponding to different control modes) are scheduled to carry out a complex mission. We are currently studying this topic [9] using a synchronous programming language to specify the reactive behavior associated with changing RTs as part of our experimental work with an underwater vehicle equipped with a robot arm [29]. New developments such as a new graphical user interface, automatic means to verify RT properties, and code generators for both simulation and real-time target platforms, are currently being

incorporated into the ORCCAD toolbox.

Acknowledgments The authors gratefully acknowledge the collaboration of Professor Michel Barbeau at Université de Sherbrooke and Dr. Konstantinos Kapellos at Inria for careful reading and fruitful comments and suggestions. Exchanges between France and Canada were made possible thanks to the help of the Commission permanente Franco-Québecoise, CRIM, INRIA, and the Canadian Natural Sciences and Engineering Research Council.

References

[1] B. Armstrong-Helouvry, "Latency Analysis of Asynchronous Distributed Control", Internal Report, Department of Electrical Engineering, University of Wisconsin, Milwaukee, 1992

[2] C. Astraudo, J.J. Borrelly, "Simulation of Multiprocessor Robot Controllers", *Proc. IEEE Int. Conf. on Robotics and Automation*, Nice, May 1992

[3] F. Baccelli, G. Cohen, G.J. Olsder and J.P. Quadrat: *"Synchronization and Linearity"*, Wiley Series in Probability and Mathematical Statistics, John Wiley & Sons, 1992.

[4] M. Barbeau, P. Freedman and D. Simon, "Synthesis of safe and live Robot-Tasks", Research report # 145, Département de mathématiques et d'informatique, Université de Sherbrooke, Québec, Février 1995.

[5] E. Castillo, "Principes, techniques et outils de simulation, vérification et exécution d'actions robotiques", PhD dissertation, INPG Grenoble, France, November 1994

[6] J.B. Chen, B.S.R. Armstrong, R.S. Fearing and J.W. Burdick, "Satyr and the Nymph: Software Archetype for Real Time Robotics", *Proc. IEEE-ACM Joint Computer Conference*, Dallas, November 1988.

[7] R. David and H. Alla, *"Du GRAFCET aux réseaux de Petri"*, Editions Hermès, Paris, 1989.

[8] J. Delacroix, "Stabilité et régisseur d'ordonnancement en temps réel", *Technique et science informatiques*, vol. 13, no 2, pp223-250, 1994.

[9] B. Espiau, K. Kapellos, M. Jourdan and D. Simon, "On the Validation of Robotics Control Systems. Part I: High Level Specification and Formal Verification", Inria Research Report #2719, November 1995

[10] S. Fleury, M. Herbb and R. Chatila: " Design of a modular architecture for autonomous robots", *IEEE Int. Conf. on Robotics and Automation*, San Diego, May 1994.

[11] P. Freedman, "Time, Petri Nets, and Robotics", *IEEE Trans. on Robotics and Automation*, Vol. 7, No. 4, August 1991.

[12] L. George, N. Rivierre and M. Spuri: "Preemptive and Non-Preemptive Real-Time UniProcessor Scheduling", Inria Research Report #2966, September 1996.

[13] A. Jaritz and M. Spong: "An Experimental Comparison of Robust Control Algorithms on a Direct Drive Manipulator", *IEEE Trans. on Control Systems Technology*, vol. 4, no 6, November 1996.

[14] P.K. Khosla, "Choosing Sampling Rates for Robot Control", *Proc. IEEE Int. Conf. on Robotics and Automation*, 1987

[15] P. Le Guernic, T. Gautier, M. Le Borgne and C. Le Maire, "Programming real time applications with Signal", *Proceedings of the IEEE*, vol. 79, no 9, September 1991.

[16] M. Jourdan and F. Maraninchi, "Static Timing Analysis of Real-Time Systems", *ACM SIGPLAN Workshop on Languages, Compilers and Tools for Real-Time Systems*, La Jolla, California, June 1995.

[17] M. Mejia, D. Simon, P. Belmans and J.J. Borrelly, "Mécanismes de synchronisation dans un système robotique réparti", *Proc. Séminaire Franco-Brésilien sur les systèmes informatiques répartis*, Florianopolis, Brazil, September 1989.

[18] T. Murata, "Petri Nets: Properties, Analysis and Applications", In *Proceedings of the IEEE*, Vol. 77, No. 4, April 1989.

[19] M. Nemani, T.C. Tsao and S. Hutchinson, "Multi-Rate Analysis and Design of Visual Feedback Digital Servo-Control System", *Journal of Dynamic Systems, Measurement and Control*, vol. 116, pp 45-55, March 1994.

[20] R. Pissard-Gibollet, K. Kapellos, P. Rives, J.J. Borrelly, "Real-Time Programming of Mobile Robot Actions Using Advanced Control Techniques" *4th Int. Symp. on Experimental Robotics*, Stanford, USA, June 30- July 2, 1995

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

[22] C. Ramchandani. "Analysis of Asynchronous Concurrent Systems by Petri Nets" PhD thesis, L.C.S./M.I.T., February 1974.

[23] C. Samson, M. Le Borgne and B. Espiau, *"Robot Control: the Task-Function Approach"*, Clarendon Press, Oxford Science Publications, U.K. , 1991.

[24] L. Sha, R. Rajkumar and S. Sathaye, "Generalized Rate-Monotonic Scheduling Theory: A Framework for Developing Real-Time Systems", *Proceedings of the IEEE*, Special Issue on Real-Time Systems, vol. 82, n^o 1, January 1994.

[25] K.G. Shin and P. Ramanathan, "Real-Time Computing: A New Discipline of Computer Science and Engineering", *Proceedings of the IEEE*, Special Issue on Real-Time Systems, vol. 82, n^o 1, January 1994.

[26] K.G. Shin and X. Cui, "Computing Time Delay and its Effects on Real-Time Control Systems", *IEEE Trans. on Control Systems Technology*, Vol. 3, n^o 2, June 1995.

[27] M. Silva and J.M. Colom, "On the Computation of Structural Synchronic Invariants in P/T Nets", *Advances in PN'88, LNCS 340*, Springer-Verlag, 1989

[28] D. Simon, B. Espiau, E. Castillo and K. Kapellos, "Computer-Aided Design of a Generic Robot Controller Handling Reactivity and Real-Time Control Issues", *IEEE Trans. on Control Systems Technology*,Vol. 1, n^o 4, December 1993.

[29] D. Simon, K. Kapellos and B. Espiau: "Control Laws, Tasks and Procedures with ORCCAD: Application to the Control of an Underwater Arm", in *Proc. of 6th IARP workshop on Underwater Robotics*, Toulon, France, March 1996.

[30] L. Whitcomb and D. Koditschek, "Robot Control in a Message Passing Environment: Theoretical Questions and Preliminary Experiments", *Proc. IEEE Int. Conf. on Robotics and Automation*, USA, 1990.