Using the Time Petri Net formalism for specification, validation and code generation in robot control applications

Luis Montano[†], Francisco José García[*] and José Luis Villarroel[†]

[†] *CPS, Universidad de Zaragoza*
Dpto. de Informática e Ingeniería de Sistemas
C/ Maria de Luna 3, 50015 Zaragoza, Spain
{montano,jlvillarroel}@posta.unizar.es

[*] *Universidad de La Rioja*
Dpto. Matemáticas y Computación
C/ Luis de Ulloa s.n., 26004 Logroño, Spain
fgarcia@siur.unirioja.es

September 2, 1999

Abstract

The main objective of this paper is to show the advantages of using the Time Petri Net formalism for specification, validation and code generation in robot control applications. To achieve this objective we consider as application the development of a control system for a mobile robot with a rotating rangefinder laser sensor with two degrees of freedom to be used in navigation tasks with obstacle avoidance. It is shown how the use of the Time Petri Net formalism in the whole development cycle can fulfil the reliability requirement of real-time systems, make the system development easy and quick, strongly reduce the time for testing and the tuning

phase and, therefore, reduce the development cost significantly. It allows verification of functional and temporal requirements, error detection in the early stages of the development cycle and automatic code generation avoiding coding mistakes. Experimental tests show that the theoretical results obtained from the analysis of formal system models match the real-time behaviour of the robotic system.

Keywords: Time Petri Nets, Real Time, Software Life Cycle, Mobile Robots, Navigation.

1 Introduction

Most of the papers concerned with mobile robot navigation present methods or techniques for controlling the robot motion or for integrating sensorial information, but do not explicitly consider either the real-time implementation aspects, or any methodologies to model and develop such a complex control system. Those systems are concurrent by nature, because there are different tasks to be performed simultaneously: robot control, image processing, data from rangefinder processing, decision making, planning, etc. Moreover, to properly and safely operate a mobile robot in a working environment they must fulfil a set of timing requirements: bounded response times to external events as collision or remote obstacle detection, maintain an adequate control sample period suitable for the robot velocity, guarantee the recency of environment information obtained from sensors, implement communication time-outs for fault detection, etc. Mobile robots can be classified as complex hard real-time systems.

The difficulty in analysing and building up real-time systems is well-known. Moreover, in many real-time systems both software and hardware reliability are critical aspects owing to the possible catastrophic effects that a failure can produce. The use of formal methods in the whole development cycle can fulfil the mentioned reliability requirement, since these kinds of methods can allow the verification of functional and temporal requirements. Moreover, the use of an automatic tool in the coding phase of the life cycle will prevent us from making coding mistakes, it will simplify the development of the system and, therefore, reduce its cost significantly.

Not many papers in robotics have dealt with the problem of using a formal methodology to develop complex control systems considering real-time aspects. In (Hu, Brady, Du and Probert 1995) a distributed real-time architecture for controlling a mobile robot is presented. In it, attention is focused on maximising the amount of parallel information flow from sensing to action. However, they do not use a formalism to model, implement and validate the control system developed. In (Caloini, Magnani and Pezze 1998) Petri nets augmented with *Building Blocks*, called *Control Nets*, are used in the design phase of robotic control systems. The aim, is to propose the Control Nets and an associate methodology to asses the design phase of the system life cycle. Aspects, related to the implementation phase, as real time analysis and planification or code generation are not treated in this work. Petri nets are also used in (Oliveira, Pascoal, Silva and Silvestre 1998) as a formal language to describe the control structure of the mission control software of a autonomous underwater vehicle.

From this description an automated code generation is performed. However, any real time aspect are considered in this work. In (Coste-Manière and Turro 1997) a programming language for robotic systems is presented; it is based on the use of proper formal methods for the development. The approach allows both, logical and temporal analysis of the specification. The system also allows logical and dynamic validation by means of simulations, and generates an executable program. It is connected to an environment for programming robotic systems described in more detail in (Simon, Espiau, Kapellos and Pissard-Gibollet 1997) which allows the users to develop, validate and encode robotic applications. In this system, the robot control software is seen as a hybrid system, composed of the continuous-time part and the discrete event controller.

In this paper we also consider a robotic system as a hybrid system and, differently from other referenced papers, we propose the use of an existent formalism, Petri Nets, for the whole cycle of life of a robotic control system, from specification to automatic code generation. Using such a formalism we can take advantage of all the theory and tools developed around it, making the whole development easier.

It is features like the possibility of modelling concurrency, resource sharing, synchronisations, etc. that have made Petri Nets so widely used for modelling and analysing discrete event systems. Petri Nets have a strong mathematical foundation which allows the validation and verification of a wide set of correctness and liveness properties. In this paper we assume that the reader knows the basic concepts of Petri Nets (see Murata's work (Murata 1989) for a sur-

vey). However, classical Petri Nets are not suitable either for the modelling or the analysis of real-time systems, owing to the impossibility of including time features and constraints in the model. To avoid these problems many authors have proposed extensions which add time characteristics to the basic Petri Nets. There are two main ways to incorporate time to Petri Nets: associating a *delay* (from the enabling instant) with the instantaneous firing of a transition or associating a *duration* with the firing. The first of the extensions, *Timed Petri Nets* (Ramachandani 1974), considers a deterministic and fixed *duration* associated with each transition in the net. From the modelling point of view, it can be seen as the duration of some activity. This kind of net does not consider the fact that the duration of some tasks in real-time systems is not fixed, and that, depending on the system or environment state, it takes different values. One way of modelling this feature is to associate a duration (or delay) which obeys some probabilistic distribution. This leads to the *Stochastic Petri Nets* and *Generalized Stochastic Petri Nets*. These nets have been commonly used for performance evaluation, analysis of parameters such as throughput, service time, number of tasks in the system, etc. All of them are mean values of the parameters.

Nevertheless, an stochastic treatment for a real-time system is highly inadvisable since it is impossible to guarantee absolute time properties. To avoid these problems *Time Petri Nets* (Merlin and Faber 1976) (Berthomieu and Diaz 1991) were proposed. In Time Petri Nets, an interval specifying an upper and a lower bound for the firing *delay* is associated with each transition. In this case, the

enabling of a transition can model the start of an activity, and the firing, the end of the activity. This approach is more general than Timed Petri Nets, since these can be modelled using a Time Petri Net, but the opposite is not true. The bounds associated with a transition can be used to verify time properties and constraints. Time Petri Nets were initially defined by Merlin and Faber (Merlin and Faber 1976) and other authors have proposed different ways to integrate temporal intervals in Petri Nets: *Place/Transition Nets with Timed Arcs* (Hanisch 1993); *High-Level Timed Petri Nets* (Felder, Ghezzi and Pezze 1993); *Interval Timed Coloured Petri Nets* (Aalst 1993). The latter considers tokens of different colours (representing different kinds of resources, tasks, ...) and with an associated timestamp (time when the token was generated) which is considered while studying the enabling of the transitions. Considering only temporal aspects, we have chosen Time Petri Nets (hereafter TPN) because their expressive power is equal to or greater than the rest of the extensions of Petri Nets and because, from our point of view, they are more intuitive and suitable for the specification of the systems which are the object of our study.

TPNs are useful in order to develop reliable real-time software because they enable modelling timeouts, periodical activities, synchronizations and concurrency. We propose the use of the TPN formalism throughout the whole life cycle. It will allow the detection of bad properties and malfunctions in the early stages of the cycle. The use of the TPN formalism also allows us not to restrict the structure of systems in order to analyze their temporal constraints. In this sense, the design flexibility is increased with respect to the use of classi-

cal analytic techniques such as Rate or Deadline Monotonic Analysis. In these approaches, for example, in order to allow the analysis, the communications between the periodical tasks must take place through an intermediate server with no guarded entry. The use of TPNs for the analysis of real-time systems eliminates this kind of restrictions.

In this paper we apply these ideas to develop a robot control system; this robot is composed of a mobile platform and several sensors that get information from the environment. More precisely, a 3D laser rangefinder is used in the experiment to illustrate the proposed techniques. In that system several processes run in concurrently. We propose the use of TPNs in the whole life cycle of the system. Petri Nets based techniques has been used very often in specification, analysis and design (see for example the previously cited papers (Caloini et al. 1998) or (Oliveira et al. 1998) in the robotics field). However, the present paper is mainly devoted to the implementation stages, including the real-time analysis, planification and code generation.

In relation to robotic aspects, we use a potential field technique (Khatib 1986), applied to a non holonomous robot so that the trajectories and motions generated by the controller are compatible with the kinematic and dynamical constraints of the robot. The robot trajectories are smooth when the nominal trajectory is corrected to avoid an obstacle whose location is not known a priori. For this, a motion generator based on a dynamic model of the robot and techniques to treat the information obtained from the sensors to avoid obstacles in real-time have been developed (Montano and Asensio 1997).

This paper is structured as follows. Section 2 presents the Time Petri Net formalism and how it can be used to model software real-time systems. The mobile robot application developed in this paper is stated in section 3. Next, in section 4, the robotic system is modelled by a Time Petri Net. Based on this model, a qualitative and timing analysis is performed to guarantee the accomplishment of real-time constraints in execution. Section 5 shows techniques for automatic code generation from Time Petri Net formal models and their application to the implementation of the robotic system. In section 6 the operation of the developed system and the validity of results of theoretical analysis are examined by means of a real experiment. Finally, the conclusions are presented in section 7.

2 The Time Petri Net formalism

A Time Petri Net (Berthomieu and Diaz 1991) is a tuple (P, T, B, F, M_o, SIM), where P is a finite nonempty set of places p_i; T is a finite nonempty set of transitions t_i; B is the backward incidence function $B : T \times P \to \mathbb{N}$; F is the forward incidence function $F : PCT \to \mathbb{N}$; M_o is the Initial Marking function $M_o : P \times \mathbb{N}$, (P, T, B, F, M_o) defines a Petri Net, the underlying Petri Net; and SIM is the mapping called static interval $SIM : T \to \mathbb{Q}^* \times (\mathbb{Q}^* \cup \infty)$, where $\mathbb{Q}^*$ is the set of positive rational numbers. So, we can see TPNs as Petri Nets with labels: two time values (α_i, β_i) associated to transitions. The first time value represents the static Earliest Firing Time (static EFT), the minimum

time starting from t (time at which transition t_i is enabled) that a transition has to wait until it can be fired, and the second is the static Latest Firing Time (static LFT), the maximum time that a transition can be enabled without firing. Assuming that transition t_i was enabled at time t, and is being continuously enabled, these two time values allow the calculation of a firing interval for each transition t_i in the Net. The firing of t_i must occur in the interval $(t+\alpha_i, t+\beta_i)$. Once the transition is to be fired, the firing is instantaneous. Unfortunately, there are still few results applicable to TPNs. Most of the existing approaches of analysis of TPNs use enumerative methods which involve the computation of the reachability graph (see (Berthomieu and Diaz 1991) and (Popova 1991)). A problem of these approaches is the state explosion problem: the state space of a TPN can be quite large, even for relatively small models. In order to make reachability analysis practical, some techniques, such as reduction rules (Sloan and Buy 1996) have been developed. However, a large amount of effort must still be made in this field.

TPNs have a time interpretation that associates a delay with transitions. The opposite interpretation that associates a duration (non instantaneous, three phase firing as Timed Petri Nets (Ramachandani 1974)) instead a delay can be simulated by the TPN model. In addition, the duration interpretation does not allow to model activities that can be interrupted (e.g., due to a time-out expiring), because this contradict the model semantics.

We use (García and Villarroel 1996) TPNs to model systems consisting of a set of concurrent activities with temporal constraints, i.e. real-time systems.

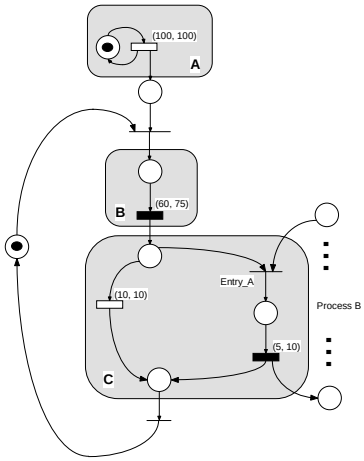
We can model periodic or aperiodic processes which communicate with each other. For example, in figure 1 we can see a TPN model of a periodic process that executes a piece of code and communicates with another process. This communication has an associated time-out. Three elements in figure 1 have been highlighted (a piece of Ada code with the same behaviour of the TPN model is provided for a better understanding of the model):

- Box B shows an action, i.e. code, to be executed by the process. The execution starts when the input place becomes marked. The execution must finish at a time between $(60, 75)$, i.e. the computation time of the code is between $(60, 75)$ time units. When the execution ends, the transition is fired.

- Box A models the periodic activation of the process. Every 100 time units the transition fires and promotes the execution of the process.

- Box C shows a time-out in a communication with another process. Let us suppose that the place on top of box C is marked at time t. If the transition labelled with `entry_A` does not fire (starts the communication) before $t+10$ (expiration time of the time-out), then transition $(10, 10)$ will fire, aborting the start of the communication.

Transitions in Merlin's model are all of the same type. They all have the same functionality. But in a real-time system there are different situations that are suitable for being modelled as a transition. In order to highlight the different roles that a transition can play and with the aim of implementing the model, in




```
loop
  CODE;                    -- Box B
  select
    Process_B.entry_A;     -- Box C
  or
    delay 10.0;
  end select;
  delay until Next;        -- Box A
  Next := Next + 100.0;
end loop;
```

Figure 1: Example of TPN model

our models we distinguish three kinds of transitions:

1. CODE-Transitions (CODE-T). One of these transitions, together with its input place, represents the code associated to one activity. This activity starts its execution when the transition gets enabled. These transitions are tagged with two time values (α, β), in the TPN fashion. In the model, the meaning of these time values is associated with the computation time

of the activity. At best, the code computation will have finished at time α, and at worst, the computation will last β. Thus, the computation takes a time between (α, β). The termination is represented by the transition firing. We draw a CODE-T as a thick segment.

2. TIME-Transitions (TIME-T) are transitions with an associated time event, e.g. a time-out or the next periodical activation of a process. These transitions also have time information associated, described with an interval (α, α), where α represents the event time. The firing of this kind of transition represents the occurrence of the event, which causes control actions to take place on the system. If a time-out related to an action occurs, the action must be aborted and the resources used by it released. If a periodic activation event occurs, the related periodic process must start its execution again. We draw a TIME-T as an empty thick segment.

3. SYCO-Transitions (SYCO-T) are transitions with no temporal meaning. They are used to perform synchronizations (SY) and control (CO) tasks. The firing of a transition of this kind leads to plain state changes or synchronizations among activities. We draw a SYCO-T as a thin segment.

3 Problem statement

In this paper we will show how a design based on TPNs can make it easy to analyze and to implement a control system for a mobile robot in which several processes related to the motion control and the sensorial systems may be in

concurrency. To focus on this problem, we consider a mobile robot with a rotating rangefinder laser sensor with two degrees of freedom to be used in navigation tasks with obstacle avoidance. The laser information allows us to modify the nominal trajectory while the robot moves near an obstacle. Several processes are involved:

- the robot control process, which controls the robot motion and is periodic.
- the laser process, which provides proximity information used by the robot control process to avoid the obstacles. This process is also periodic.
- the supervisory process, which supervises the whole robotic tasks to detect if a goal or subgoal in the trajectory is reached, updates the current goal point and manage the system alarms. This process is not periodic.

Considering the laser process, the locations of sensed points are corrected taking into account the motion of the robot while the rotating sensor gathers the points. In this way, all the points of a sensor lap are referred to the same time, the time corresponding to the last point sensed in the lap. As the sensor makes a complete 3D scan of the scene in ten laps, the system must integrate the points of the ten last laps in each control sample period in order to have a complete updated information about the obstacles. This integration task is performed by the control process using the most recently updated sensed points.

The real-time constraints of the system are the following. The control loop has a sample period which will be established at the analysis phase. However, owing to the time constraints of the robot's internal control system, this sample

period must be greater than 0.18 seconds. The communications between the robot and the controller have defined time-outs: 0.1 sec. in position reading and 0.1 sec. in setpoint sending. At this point of the application development, only alarms related to the communication time-outs have been taken into account. The firing of a communication time-out must stop the system within 0.1 sec. The 3D laser sends a new scan every 100 ms. The controller must be capable of accepting and processing the sensor data at this rate. The communication with the laser has a 0.2 s time-out.

In the following subsections we describe briefly the robotic aspects of the problem: the technique used to control the motion and to avoid obstacles, and the correction and integration techniques for the sensed points.

3.1 Motion Generation

The technique proposed to navigate in a partially known environment is based on *artificial potential field* techniques (Khatib 1986). The robot moves in a field of forces: it is subjected to an attractive force from the goal, and to repulsive forces from the walls and obstacles found during the motion.

To implement the method, we have developed a virtual controller in which linear and angular velocities are the setpoints in each sampling period. This allows the method to be independent from a particular robot, being needed only to develop a robot velocity controller for each particular robot (Crowley 1995).

All the forces acting on the robot are considered to be applied on one point located in the middle of the robot front. The controller computes the resultant

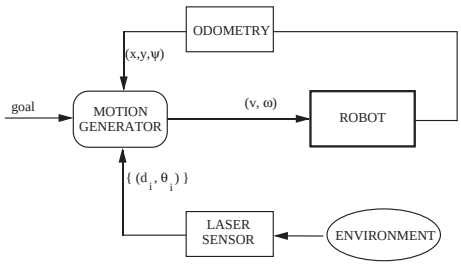


Figure 2: Control scheme for navigation.

force **F** applied on that point from the attractive and repulsive forces. The latter are calculated using the information provided by the laser sensor about the obstacles which appear in the robot environment. We model the system as a real mobile robot and thus, the robot motion is computed from a robot dynamic model.

Considering the constraints due to the contact wheels-floor and applying the theorems of the dynamics, we reach a differential state equation for the system, whose parameters are a function of the geometric and dynamic parameters of the robot model. From the dynamic equations we obtain a discrete time model used as a motion generator for the robot. For more details see (Montano and Asensio 1997).

In figure 2 we represent the general control scheme. The motion generator computes each period the velocity commands (linear and angular) taking into account the information provided by odometry and the laser sensor. The laser information is processed to compute the points belonging to the environment and to the obstacles and thus, modifying the motion computed in the previous period.

3.2 Real-time obstacle avoidance

From the model explained above, we only need to compute one force applied on the frontal point of the robot to calculate $\mathbf{u}_k = (Fsin\theta, Fcos\theta)^T$, being F the force module and θ the force angle with respect to a reference system associated to the robot. If there are no obstacles, a pure attractive force will act from the goal point. If there are obstacles or walls, the force will be computed as the resultant from the attractive force and the repulsive forces which act from the obstacles and walls around the robot.

The laser sensor can provide many points from the detected obstacles. To avoid a long time repulsive force computation we choose only a few significant points. The robot environment is sectored in angles (see Figure 3), and for each sector the nearest sensed point is selected. The selected point is used to represent the whole sector. So, the repulsive force for the sector is computed as the repulsive force for the unique selected point.

3.3 Real-time implementation

Laser information is updated every 100 ms. with the arrival of the data belonging to a lap. In order to compute the repulsive resultant force, the system integrates the last 10 scans (corresponding to a complete sweep) taking into account the robot movements, using the following method:

- In each lap, the points belonging to the sectors are selected as explained in section 3.2. The location of each of these points is corrected taking into account the robot velocity during the sample time in which they were

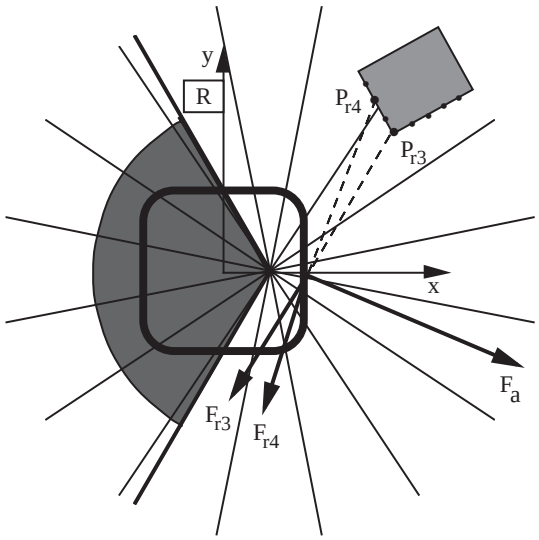


Figure 3: Sectors chosen for the experiments. The laser cannot scan in the shadow zone at the rear of the robot.

gathered, and considering that the reference time is the time corresponding to the last point in the lap.

- In each robot control period T, the location of the selected points of these last 10 laps is transformed into the current robot location, taking into account the robot linear and angular velocities for each sensor lap. These velocities depend on the sample period in which the scan was made.

4 Modelling and analysis of the robot navigation task

This paper focuses on the use of Time Petri Nets in the development of real time systems. However, the proposed techniques are independent of the design methodology. There is considerable work in the literature dealing with the integration of Petri Net formalisms and modelling methodologies, see for example (Giovanni 1990) and (Muro, Banares and Villarroel 1998) for the use of Petri Nets in object oriented methodologies. Thus, we will build the TPN model of the system directly, to highlight the net model characteristics without assuming any modelling methodologies.

4.1 Modelling the system

In section 3 it has been established that robot control systems have three main processes: the robot control process, the laser process and the supervisory process. The first step in the modelling will be to build a subnet for each process and then, the intercommunication mechanisms will be established.

1. *Control process.* The period T of the robot control process is not fixed, it must be as small as possible and will be established in the real-time analysis phase. Each period, the following control loop must be executed:

   (a) *Read Location.* The control process communicates with the robot to obtain the current location. All communications with the robot involve a protocol on a RS-232 serial line.

(b) *Scans Integration.* The control process integrates the points of the last ten laps in each control sample period to have complete updated information about the obstacles (see section 3.3).

(c) *Compute Setpoint.* Based on the current location, the points detected by the laser and the current goal, the action to be sent to the robot is computed by a potential field technique (see section 3.2).

(d) *Send Setpoint.* The control process communicates with the robot to send the new velocity setpoint (the action, from the control point of view).

To integrate the points detected by the laser, the corrected scans must be accessed. This information is also accessed by the laser process in order to update it. Thus, the mutual exclusion in accessing the scans must be guaranteed. The current goal is shared by the robot control process and the supervisory process. When the current location is obtained from the robot, a protected global variable, shared by all processes, is updated. That is, following *Read Location* the activity *Store State* is executed. A possible blocking of the control process caused by loss of communication with the robot can be avoided assigning time-outs (0.1 s, see section 3) to *Read Location* and *Send Setpoint* activities.

2. *Supervisory process.* The supervisory process is not activated by a timed periodic event; it must be executed each time that a new location comes from the robot. The activities of the supervisory process are:

(a) *Store Goal.* Updates the goal to be used by the controller.

(b) *Trace Location.* Stores the current state of the robot in the file system. In order to do this, the shared variable *State* must be accessed. Thus, the *Supervisor Read State* activity is executed before creating a local copy of the current state.

(c) *Goal Test.* Tests if a goal or subgoal in the trajectory is reached by the robot. If a subgoal is reached, a new subgoal must be established in the system. If the final goal is reached the robot must be stopped.

(d) *Alarm.* Executes a code of alarm treatment if an alarm event comes from the control or laser processes. It is not considered a possible error recovery from an alarm and the system is stopped.

3. *Laser process.* Finally, the laser process is not activated by a periodic event either. However, it is executed when a scan comes from the laser and the scans have a periodic timing pattern. Thus, the laser process behaves as a periodic process. Its activities are:

   (a) *Read Scan.* Reads a scan from the laser. This communication activity is protected from blocking by a time-out (0.1 s, see section 3).

   (b) *Correct Scan.* Computes the time correction of the points of a scan. This activity accesses robot state and scans shared variables.

Figure 4 shows the Time Petri Net that specifies the full real-time robot control system (time information has been removed for clarity), in which the three processes are highlighted. The next step in the modelling is to establish the

communication and synchronizations between processes. The following aspects must be considered to establish communication between the processes:

- The shared variables must be protected from concurrent access. A mono-marked place which prevents the concurrency in accessing primitives, `State`, `Scans` and `Goal`, is assigned to each variable.
- The arrival of a new robot location to the control system must be signalled to the supervisory processes. This is carried out by means of a binary semaphore, the place `CC4`, from the control to the supervisory process.
- Each time that a time-out is completed, an alarm signal must be sent to the supervisory process. This is carried out by semaphores `CC3` and `LC2`.
- When an alarm signal is processed by the supervisor, all system processes must be stopped. In this way, a signal is sent to any active process and then, the supervisor ends. This signalling is implemented by places `CC1` to stop the periodical activation of control, `CC2` to stop the control process and `LC1` to stop the laser process.
- When the supervisor detects that the robot has reached the final goal of the trajectory, it sends stop signals to the rest of the processes of the system. This signals are implemented by places `CC1` to stop the periodical activation of control and `LC1` to stop the laser process as in the previous point. However the stopping of the control process is carried out by place `CC5` because a final movement must be accomplished to reach the goal effectively before of robot stops.

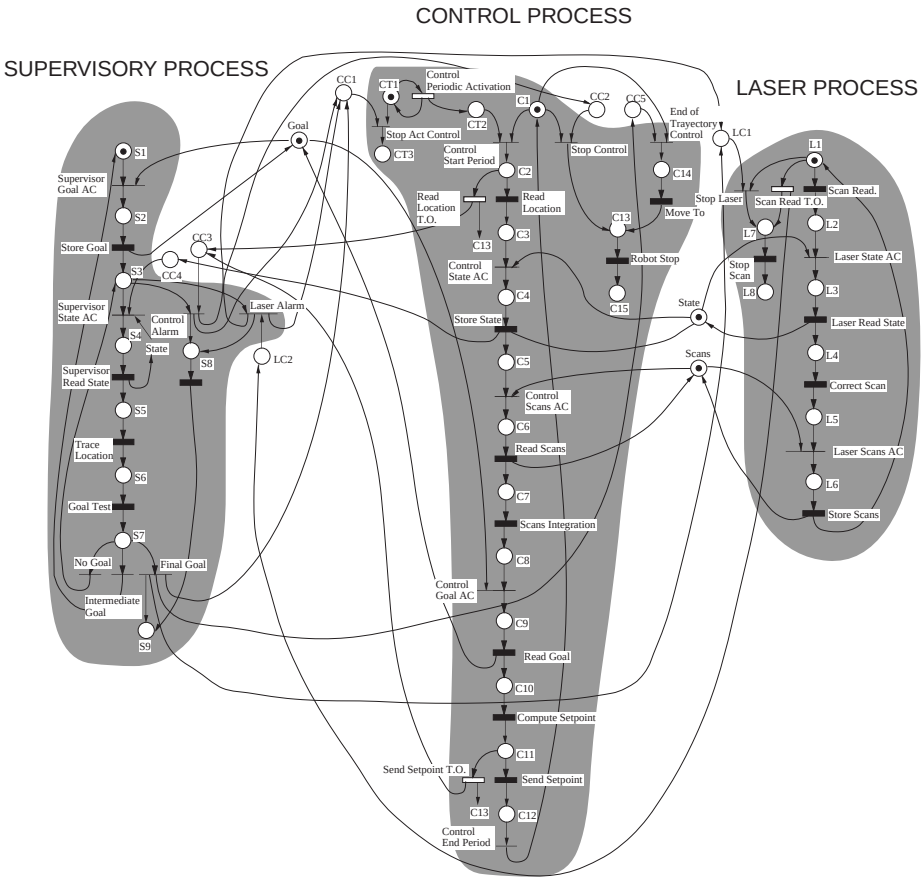


Figure 4: Time Petri Net which models the mobile robot application. Time Petri Nets which model: the supervisor process, the robot control process, the laser process and synchronizations between processes.

To complete the model, it is necessary to include the best and worst case computation times for code transitions. These computation times have been measured directly by executing the associated pieces of code on the target platform (a SPARC CPU-5V running SOLARIS in Real-Time Mode). The TPN model (figure 4) can be used directly to code generation. However, to carry

out the time analysis of the execution in a monoprocessor architecture, some changes must be made.

Obviously, the modelling stage must be assisted by the appropriated CASE tools and methodologies, for example based on building blocks as in (Caloini et al. 1998) or based on a hierarchical object oriented approach as in (Giovanni 1990). The present paper is mainly devoted to the implementation stages, including the real-time analysis, planification and code generation. In this sense, the TPN result of the modelling phase that can be of a large size and in some cases unintelligible, will be the input of automatic or semiautomatic techniques.

4.2 Planification and real time analysis

The final aim is to generate Ada95 concurrent code for a monoprocessor architecture. The real time execution of an Ada code is controlled by priorities (static or dynamic). In this sense, the main objective of the planification and real-time analysis phase is to determine a priority assignment (or a strategy for dynamic priority assignments) that guarantees the accomplishment of the real time constraints of the system.

Most research work on scheduling and Petri Nets has been focused, mainly, on the static planification of Timed Petri Net models, see for example (Carlier and Chretienne 1988) or (Bruno, Castella, Macario and Pescarmona 1992). It is not an objective of this paper to present a planification technique based on priority assignments. It is still under study. However, we have used a heuristic method based on the Rate Monotonic Scheduling (RMS). From the problem

stated in section 3 there are two periodic processes (the robot control and the laser process) and a sporadic process (the supervisor). The period of the robot control process is a design variable, but greater than the laser period because of the computation time of the code that must be executed. From the RMS point of view, the transitions involved in the laser process will have a priority greater than the transitions of the robot control process. In order to minimize the response time of supervision decisions, the supervisor process will have the maximum system priority. The next step of the planification phase is to analyze the behaviour of the Time Petri Net model with the assigned priorities executed in a monoprocessor architecture to verify the real time restrictions.

The analysis methods for Petri Nets extended with time are based mainly on enumerative techniques related to reachability analysis methods for usual Petri Nets. For example, in (Carlier and Chretienne 1988) the Earliest State Graph is presented in order to analyze Timed Petri Net schedules. The building of the state Graph of a Time Petri Net is more complicated because the firing intervals increase the number of states (infinite with dense time). This problem can be solved by means of time discretization as in (Popova 1991) or by means of state class definition as in (Berthomieu and Diaz 1991).

In our robotic application there are two kinds of activities:

1. Activities with fixed computation time, which can be modelled by a code transition which has the same EFT and LFT, $[t, t]$. In this sense, the algorithms involved in our robotic application has been forced to have a fixed computation time.

2. Activities which involve external communication. If there are no problems in the communication, it takes a fixed time, otherwise the communication is blocked. These activities are modelled by a transition with an infinite LFT: $[t, \infty)$

This characteristic allows us to avoid the difficulty of analyzing a Petri Net model with firing intervals. The model can be split into several models with fixed firing delays, a main model which does not consider bad communications and a model for each possible communication time out (see, figure 5). Thus, an automatic state graph builder for models with fixed firing delays has been developed for the analysis of the application. This analysis consists in the separate analysis of *each model* generated by splitting the original model with intervals. If a property or requisite is accomplished in all analysis models, it is accomplished in the original one also. The graph builder, which works on textual specification of the TPNs, considers static priority assignments to transitions, the preemptive behaviour of the Ada95 kernel and also the monoprocessor architecture.

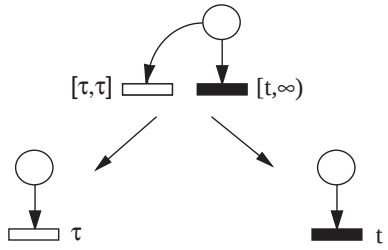


Figure 5: An activity with fixed computation time but which can be blocked and is protected by a time-out can be split in two test cases with a fixed firing delay

To build the state graph of the system, the firing rules of TPNs have been

updated with the behavioural rules of a preemptive kernel with static priorities. The fundamental rule is that the code ready for execution with greater priority gains the processor. From the TPN point of view, this means that only the code represented by the enabled transition with higher priority is running. It can be represented by a clock associated to each CODE-T. This clock advance only if the underlying code is running in the processor. When a transition with greater priority becomes enabled, a preemtion is performed, the clock of the running transition is stopped and the clock of the new transition is started. When a transition regains the processor its clock continues at the same instant of the stop. TIME-T and SYCO-T can be viewed as kernel control actions, thus their execution is immediate at the highest priority.

Each time a transition is fired or a preemtion is performed, the system enters in a new state and thus a new node is generated in the state graph. The main information attached to each node are the marking and the clocks associated with enabled transitions.

To carry out the real-time analysis of the system, some modifications have been made in the model. Mainly:

- A periodical activator has been added to the laser process. It models the periodical arrival of a laser scan and makes the model autonomous.
- An implicit place has been added to both periodical processes. This place is marked when the periodical activity starts and demarked when the activity ends.

One objective of the analysis process is to find the lower bound of the robot control period. To achieve this objective, the state graph builder has been programmed to detect states with no binary places and with specified pairs of places simultaneously marked. The existence of a non binary place means that there exists in the system an event which cannot be processed at the required rate, thus the representative tokens are accumulated in a place. On the other hand, the simultaneous marking of an activity place and the corresponding activation place means that the periodical activity has not finished before the next activation starts, that is, the deadline is not met.

The system has been analyzed iterating over the control period with a step of 10 ms. Following this methodology, it has been established that, for the imposed priority assignment, 0.2 seconds is the minimum period that makes that the deadlines are met. Based on this result, the control period will be 0.25 seconds.

5 Implementation of the application

Once a model of the system has been obtained, a model which meets all the functional and temporal requirements, and even considers the constraints of the actual implementation platform, is necessary to generate the implementation. If a software implementation of a system is a program that satisfies every functional requirement of the system, with reference to Petri Nets, an implementation is a program which simulates the firing of the net transitions,

observing the marking evolution rules. We have performed an automatic code generation using adaptations of classical Petri Net implementation techniques, which consider the addition of time information to the net transitions. As for classical implementation techniques (Colom, Silva and Villarroel 1986), we can distinguish between centralized and decentralized techniques. The former use a single coordinator process responsible for the control and evolution of the net. The latter split the control between several sequential processes, each one implementing a subnet extracted from the original net. Ada 95 is used as target language for the implementation.

5.1 Centralized implementation

The special features of centralized implementations make them especially suitable for the system prototyping. One process encapsulates the whole state of the system and performs all control actions making the system debugging easier. In this sense, we will describe the centralized implementation technique for the prototyping stage of the development of the mobile robot application. This technique was the object of study in (García and Villarroel 1996). As in any centralized implementation, the operational and control parts of the system are separated. This is a direct consequence of the architecture of the centralized implementations which considers two types of processes (hereinafter we will talk about Ada-tasks when we refer to a process):

- Each unit {place + CODE-Transition}, which represents each code execution of the net, will be implemented as an Ada-task, called *CODE task*.

In this way, we maintain the concurrency of the model in the implementation. Each CODE task will execute the code associated to the unit. These codes must be developed separately, and later linked together.

- Control and timing supervision will be performed by another task called *coordinator*. TIME-Ts and SYCO-Ts are considered by the coordinator to perform the control of the net implementation. Every CODE task communicates with the coordinator, which is responsible for taking decisions as to when a transition must fire.

We can see the coordinator as the kernel of an operating system, and the CODE tasks as the processes managed and executed in it. Therefore, the operational part of the system is performed by the CODE tasks and the control part, by the coordinator (see the schema in figure 6). Details of this implementation technique can be found in (García and Villarroel 1996).

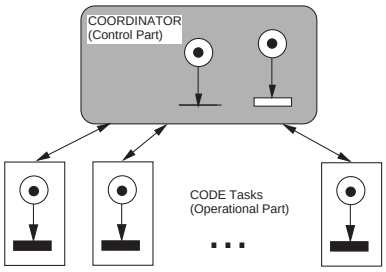


Figure 6: Schema of centralized implementation

The priority assignment performed in section 4.2 will be taken into account during a centralized implementation in two different ways. The first is to assign static priorities to the CODE tasks. These priorities are used by the Ada-Kernel

to schedule the different tasks. The second is used by the net coordinator to solve the conflicts among transitions. The highest priority transition will always be chosen to fire. Apart from these priorities computed in the real time analysis phase and associated to transitions, the coordinator must be considered as the highest priority task in the final implementation since it is responsible for the activation of the rest of the system and of the timing and control actions that must be performed immediately. The preemptive mechanism provided by the Ada Kernel is used for the implementation.

The coordinator is built as an interpreter which interprets a data structure representing the net structure. Therefore, this technique is especially suitable for the prototyping and the simulation of systems (just the way it was used during the mobile robot application development), since changes in the system involve only changing the data structure interpreted by the coordinator. The technique is easily automatizable, preventing us from making coding mistakes, simplifying the coding stage, and so, reducing the cost of the system. Nevertheless, despite the simplicity in the implementation, the technique presents several problems. As shown in (García and Villarroel 1996) the presence of the coordinator, which acts in every transition firing updating the marking and reevaluating the enabling conditions of the transitions of the net, introduces an overload which reduces the maximum schedulable utilization. In addition, the coordinator alone is responsible for the control of the implementation. So, it sequencializes the control of the implemented system, which is in fact concurrent. Moreover, the whole evolution of the system depends on the coordinator.

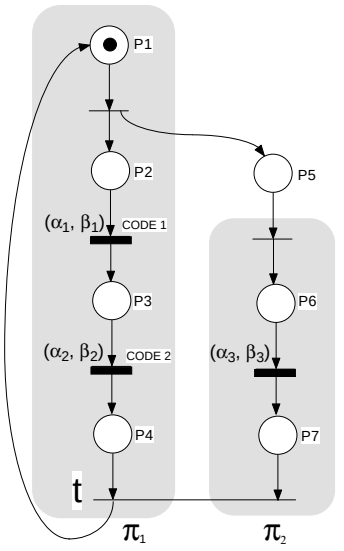
That is, the implementation is sensitive to faults, since if the coordinator fails, the whole system fails too. And, finally, the number of concurrent tasks in the implementation may be greater than the actual concurrency of the modelled system. See for example figure 7, a process which communicates with another one is represented. However, the implementation generates four processes: three CODE tasks and the coordinator).

5.2 Decentralized Implementation

In this section, we describe the definitive implementation of the mobile robot application. The idea of decentralized implementations is quite simple. In order to avoid the problems of centralized implementations the control of the net is split into several sequential subnets, each of which is implemented in a separate process, concurrent with the others. This way, the actual concurrency of the system is respected and the presence of the coordinator is not needed, avoiding the overload and making the system more fault tolerant (a part of the system can fail without implying the total unavailability of the implementation). Each process integrates both control and operational parts of the subnet, just the opposite to the centralized technique. This technique was the object of study in (García and Villarroel 1998) and in (García and Villarroel 1999).

Clearly, the first step in any decentralized implementation is the recognition of the sequential processes (or state machines) embedded in the net and interconnected through a communication mechanism like buffers or rendezvous. The basis is to merge a set of transitions which are in mutual exclusion into a single

process (t_i, $t_j \in T$ are in mutual exclusion, $t_i ME t_j$, if there does not exist any reachable marking for which both transitions are fireable). A set of transitions which are in ME relationship are not concurrent, so they can be in the same process (π) without reducing the actual concurrency. A place p, with respect to a process, can be either private (every input and output arc of p are connected to transitions belonging to the process, ${}^\bullet p \cup p^\bullet \in \pi$), external ($p$ is only connected to transitions not belonging to π), or shared (p is connected both to transitions belonging to π and transitions not belonging to π). In fact, a shared place models an asynchronous communication (buffer). See figure 7 for an example.




```
loop
  P5.Mark;
  CODE1;
  CODE2;
  accept t;
end loop;
```

Figure 7: A Petri Net decomposed into two sequential processes. Place $P5$ models an asynchronous communication between the processes, whereas, transition t models a synchronous communication. The Ada code implementing π_1 is included

For the computation of transition sets in mutual exclusion, the underlying

Petri Net (the net without time information) is used. The reason is that two or more transitions which are in ME in the underlying Petri Net, remain in ME in the TPN, but the opposite it is not true. Several techniques are known in classical Petri Net theory in order to determine the mutual exclusion of transitions. It can be achieved with the reachability graph or using structural techniques like the computation of monomarked p-invariants in the net (Colom et al. 1986) (García and Villarroel 1998). A p-invariant establishes a weighed ratio among the markings of a set of places which is maintained for any reachable marking. e.g., in figure 7, m(p1) + m(p2) + m(p3) + m(p4) = 1 is one of the p-invariants of the net. Monomarked p-invariants are especially interesting because they describe a set of places in ME. In the example in figure 7, either p1, p2, p3, or p4 is marked, but never two or more of them are marked at the same time. Obviously, a set of places in ME implies a set of transitions in ME, the input and output transitions of the places. Unfortunately, it is not always possible to cover a Petri Net with a set of monomarked p-invariants. To solve this problem (Villarroel 1990) proposes a decomposition technique based on the concept of pipeline. The objective is to build, if necessary, monomarked p-invariants by means of the addition of implicit places.

However this is achieved, a partition of the underlying Petri Net, which covers every transition of the net, can be found. This partition is made up of a set of state machines, implementable as sequential processes. Places included in a p-invariant can be used to describe the control flow of the process to which they belong. In this way, those transitions whose input place belonging to the

p-invariant is marked, are able to fire. The remaining places which are private with respect to a process and not belonging to the p-invariant or pipeline act as local variables for correctness checking or taking decisions. Each process can be implemented in an Ada task, using a case structure. Each place of the p-invariant or pipeline describes a state which will be implemented at each branch of the case. The code associated with each branch depends on the structure associated with the place (consider the fragments of nets in figure 8), e.g., the kind of output transition. It is possible to improve the implementation using the single token of the p-invariant or pipeline as if it were the program counter of the process. The flow of the token through the p-invariant defines the execution order of the transitions, avoiding the use of the case structure (e.g. see the code shown for process π_1 in the net in figure 7). Implementing each process in an Ada task, both the control and the operational parts of the set of transitions are integrated in the same process avoiding the use of a coordinator.

There are three kinds of transitions in our modelling approach and, from the software implementation point of view, they will be implemented in different ways:

1. SYCO-Ts (used for representing control and synchronization actions) will be taken into account to make decisions inside a process or, when it is shared between two or more process, to perform synchronous communication between processes (see figure 8.a for an example).

2. CODE-Ts involve the execution of their associated code.

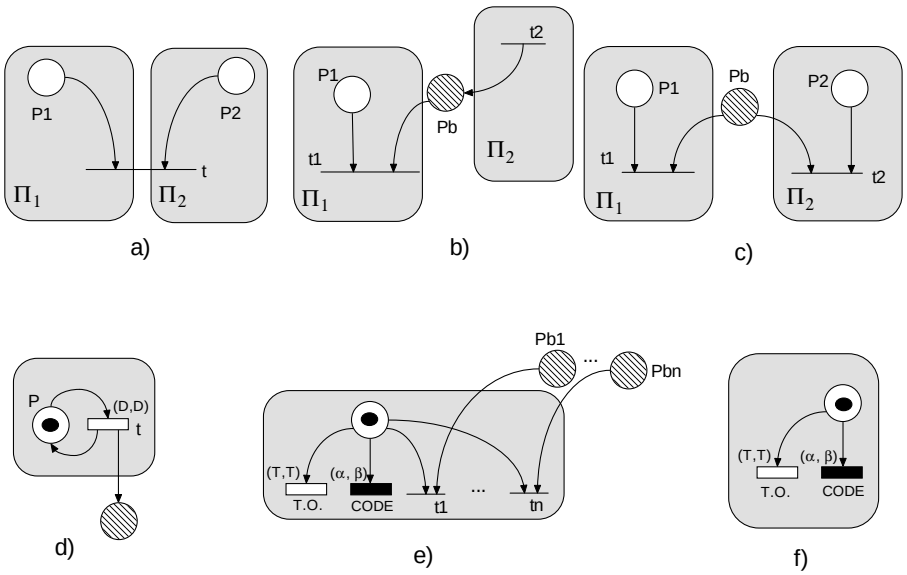


Figure 8: Common structures in TPNs: a) Synchronous communication, transition t; b) asynchronous communication, place p_b; c) simple conflict; d) Periodical activator; e) Inner conflict inside a process; f) Code execution with a time-out.

3. TIME-Ts represent some time activity such us a delay, time-out, periodical activation, ... As a first approximation, this kind of transitions can be implemented with an Ada delay which starts when the input place of the transition gets marked. But this situation can provoke accumulative drift in the processes. In order to avoid the drift, a time variable (from Ada.Real_Time) is associated with each process with TIME-Ts. This variable, called Last_update, records the time at which the last marking update occurred. This time is used in the computation of the expiration time of the delays. For example, consider these implementations for the

TIME-T in figure 8.d. which represents a periodical activation. The implementation on the left presents accumulative drift, solved on the right by using the Last_update variable.

```
loop                Last_update := CLOCK;
  delay D;           loop
  P.Mark;             next := Last_update + D;
end loop;             delay until next;
                      Last_update := next;
                      P.Mark
                     end loop;
```

Sometimes a transition can be shared between two processes. This situation represents a synchronous communication between both processes, and will be implemented with an Ada rendezvous (see figure 8.a for an example).

So far we have not referred to the places shared between processes. As mentioned above, a shared place represents an asynchronous communication between the processes. One of these processes can mark it and the others use this token later. Two kinds of shared places must be distinguished: those whose destination is a single process (private destination), and those whose destination can be several processes (non private destination). The latter sets up a conflict, which is modelling a resource (see figure 8.c) that must be implemented as a shared variable inside an Ada protected object which guarantees the access to the resource in mutual exclusion. The former (see figure 8.b) will be implemented with a buffer or a relay process.

```
task body RELAY_PLACE is
  Last_update: TIME;
begin
  loop
    accept Mark ;
    Process.Demark ;
  end loop;
end RELAY_PLACE;
```

```
protected body BUFFER_PLACE is
 procedure Mark is
 begin
   M:= M + 1;
 end;
 entry Demark when M>0 is
 begin
   M:= M - 1;
 end;
end;
```

Transition	I1	I2	I3	I4	I5	I6	I7
Store_Goal	x				x		
Supervisor_Read_ State	x					x	
Trace_Location	x						
Goal_Test	x						
Alarm	x						
Supervisor_Goal_AC	x				x		
Supervisor_State_AC	x					x	
No_Goal	x						
Intermediate_Goal	x						
Final_Goal	x						
Control_Alarm	x						
Laser_Alarm	x						
Move_To		x					
Robot_Stop		x					
Read_Location		x					
Store_State		x				x	
Read_Scans		x					x
Scans_Integration		x					
Read_Goal		x			x		
Compute_Setpoint		x					
Send_Setpoint		x					
Control_Scans_AC		x					x
Control_Goal_AC		x			x		
Control_State_AC		x				x	
Control_Start_Period		x					
Stop_Control		x					
Control_End_Period		x					
End_Trajectory_ Control		x					
Read_Location_TO		x					
Send_Setpoint_TO		x					
Stop_Act_Control			x				
Control_Periodic_ Activation			x				
Scan_Read				x			
Laser_Read_State				x		x	
Correct_Scan				x			
Store_Scan				x			x
Stop_Scan				x			
Laser_State_AC				x		x	
Laser_Scans_AC				x			x
Stop_Laser				x			
Scan_Read_TO				x			

Figure 9: Covering table of transitions

For the sequential process recognition in the mobile robot application, a

p-invariant computation has been done. There are seven p-invariants: (I_1) Supervisor $\{S1..S9\}$, (I_2) Control $\{C1..C15\}$, (I_3) Activation $\{CT1, CT3\}$, (I_4) Laser $\{L1..L8\}$, (I_5) Protected Goal $\{Goal, S2, C9\}$, (I_6) Protected State $\{State, L3, C4, S4\}$, (I_7) Protected Scans $\{Scans, L6, C6\}$. With this p-invariants the transition coverability problem can be solved. In Figure 9 the covering table is shown, a transition is covered by an invariant if the transition is descending from a place of the invariant. It can be seen that there are four essential p-invariants (I_1, I_2, I_3, I_4), which cover all the net transitions. The first set of transitions in EM (covered by I_1) corresponds to the Supervisor process. The second (covered by I_2) corresponds to the Control process. The third (I_3), is the periodical activator of the Control process, the Control_Activation process. And the fourth (I_4), is the process which deals with the laser, the Laser process. These processes will be implemented in Ada95 tasks, which contain the previously mentioned structures. There are several places left which do not belong to any processes. These places model asynchronous communications between the processes. The places are CC_4, CC_3, LC_2, CC_1, CT_2, LC_1, CC_2, CC_5, all of them of private destination, and $Goal$, $State$, $Scans$, of non-private destination. The former will be implemented as relays and the latter implemented as protected objects. Several pieces of code corresponding to this implementation are shown in Appendix A.

6 Experimental results

We have tested the real-time robot control system and the navigation techniques above explained making the robot navigate in an unstructured indoor environment in which the robot moves avoiding obstacles following a nominal trajectory. The environment is composed of two corridors in which there are two doors that the robot must cross and several obstacles that the robot must avoid.

We have studied the behaviour of the real-time control system, which works correctly, synchronizing the parallel processes involved in the system. The theoretical minimum sample period obtained from the analysis has been tested in the real implementation.

In order to verify the accomplishment of the real time restrictions in the execution of the control system, supervisor elements that allow the detection and treatment of deadline violations have been included in the TPN model. Figure 10 shows how to detect when a periodic process does not meet its deadline (supposing the deadline equals the period). Based on this supervision structure, the minimum robot control period has been experimentally established. As in the theoretical analysis, the method has been iterating over the control period until no deadline violations have been detected. Two different results have been obtained: 0.25 seconds for centralized implementation and 0.20 seconds for decentralized implementation. In Figure 11, two traces of execution using the decentralized implementation are shown. In the first one, all the processes accomplish the execution deadlines. In the second one, a deadline violation is

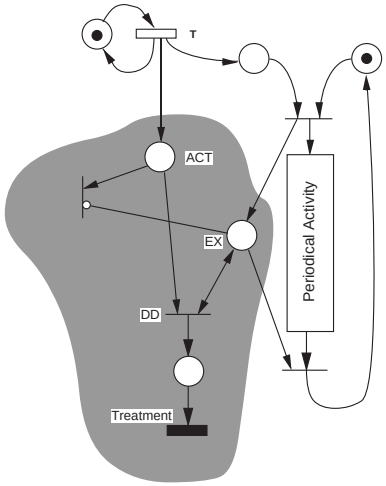


Figure 10: Supervision structure to detect deadline violations in periodic processes when the deadline is equal to the period. The simultaneous presence of a token in places ACT and EX means that a new activation has arrived before the computation of the previous one finishes. With this marking, transition DD is fired and the treatment of deadline violation started.

detected, due to the sample period is smaller than the theoretical one. The result obtained for decentralized implementations demonstrate the validity of theoretical analysis. However, in the case of centralized implementation, the coordinator overload has not been taken into account in the analysis phase. This overload which reduces the maximum schedulable utilization justifies the difference of 50 milliseconds with respect of theoretical result.

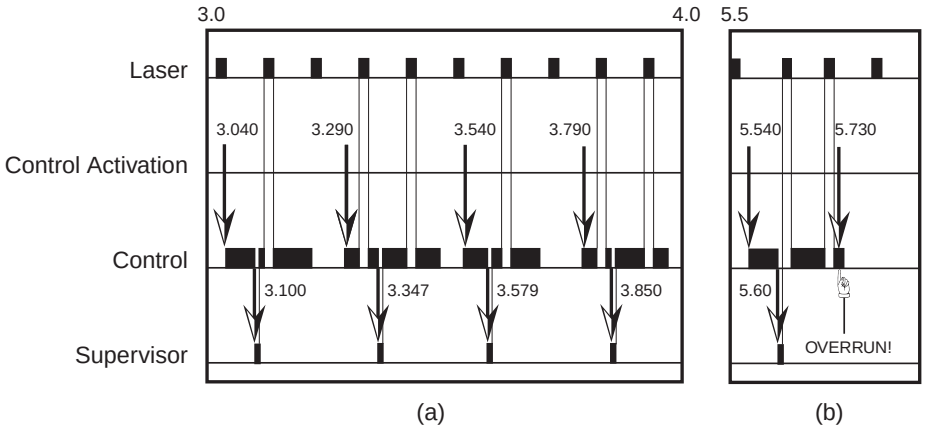


Figure 11: Trace of the execution of processes generated in the decentralized implementation in two differents cases: (a) with a period of 0.25 s for the control process greater than the theoretical one; and (b) with a period of 0.19 s smaller than the theoretical one.

7 Conclusions

This paper has focused on using Time Petri Net formalisms for specification, validation and code generation in robot control applications. To illustrate this approach we have accomplished the development of the control system of a mobile robot with a rotating rangefinder laser sensor with two degrees of freedom to be used in navigation tasks with obstacle avoidance based on artificial potential field techniques.

The use of formal methods in the development cycle can fulfil the essential reliability requirement of real time systems allowing the verification of functional and temporal requirements and the error detection in early stages of the development cycle. Moreover, the use of a unique formalism in the whole devel-

opment cycle simplifies it. In this paper, Time Petri Nets have been proposed as the formalism for the development of real time systems and robotic systems in particular. Petri Nets have been widely used for modelling and analyzing discrete event systems because of features like the possibility of modelling concurrency, resource sharing, synchronizations, ... Moreover Petri Nets have a strong mathematical foundation which allows the validation and verification of a wide set of correctness and liveness properties. The Petri Nets extended with time, like Time Petri Nets, allow modelling and analyzing real-time systems. The advantages stated in this paper can be summarized as follows:

- This paper has been mainly focused on the implementation stages, however Time Petri Nets are suitable in all phases of the life cycle. The use of TPNs does not suppose the assumption of any methodologies and there is considerable work in the literature dealing with the integration of Petri Net formalisms and modelling methodologies. They have been used successfully in specification, analysis, design and code generation. Moreover, since they are an executable formalism they can drive system prototyping and test phase.

- It allows the verification of functional and temporal requirements. Moreover, the use of the TPN formalism allows us also not to restrict the structure of systems in order to analyze their temporal constraints. The design flexibility is increased with respect to the use of classical and most extended analytic techniques such as Rate or Deadline Monotonic Analysis. In these approaches, in order to allow the analysis, the communications be-

tween the periodical tasks must take place through an intermediate server with no guarded entry. The use of TPNs for the analysis of real-time systems eliminates this kind of restriction. It has been shown how formal model analysis can assess the design process. As an example, the minimum sample period the system needs to ensure the completion of all the tasks or processes involved in the control has been computed. Using such a formalism we can take advantage of all the theory and tools developed around it, making the whole development easier.

- The automatic code generation from formal model avoid coding mistakes. The special features of centralized implementations make it especially suitable for the system prototyping and testing phase. One process encapsulates the whole state of the system and performs all the control actions, making the system debugging easier. Moreover, the coordinator is built as an interpreter of a data structure representing the net structure. Therefore, this technique is especially suitable for the prototyping and the simulation of systems (just the way it was used during the mobile robot application development), since changes in the system involves only changing the data structure interpreted by the coordinator. Nevertheless, despite the simplicity in the implementation, the technique presents several problems that have been stated in the literature. The presence of the coordinator introduces an overload into the implemented systems which reduces the maximum schedulable utilization. Moreover, the whole evolution of the system depends on the coordinator and thus, the implementation is sen-

sitive to faults since if the coordinator fails, the whole system fails too. And, finally, the number of concurrent tasks in the implementation may be greater than the actual concurrency of the modelled system. To avoid all these problems, the final implementation of the system is carried out by decentralized techniques where the control of the net is split into several sequential subnets, each of which is implemented in a separate process, concurrent with the others.

All these characteristics tend to simplify the system development and, therefore, reduce its cost and the time needed significantly. We can estimate we have spent three months-man from system specification to former code generation and implementation, including partial tests of the system. Only one week-man was needed to test the whole system and to make the final system implementation and tuning. We would like to highlight that it was thanks to the formalism we have used that we have been able to save time in the final tuning phase. By using it, many of the errors in the system design and implementation can be avoided in the early stages, and those detected during the tuning stage can be easily localized and corrected.

However, the techniques based on Time Petri Nets still need a considerable research effort, mainly as regards real time planification. There is not much work on planification and Time Petri Nets, and the little there is, has focused on static planification of models with fixed duration semantics. In this sense, new planification techniques must be developed to assign priorities to transitions.

Acknowledgements

This work has been supported in part by project TAP97-0992-C02-01 from the Comisión Interministerial de Ciencia y Tecnología of Spain.

References

Aalst, W. v. d. 1993, Interval timed coloured petri nets and their analysis, *in* M. A. Marsan, ed., 'Application and Theory of Petri Nets 1993', number 691 *in* 'Lecture Notes in Computer Science', Springer Verlag, Berlin, pp. 453–472.

Berthomieu, B. and Diaz, M. 1991, 'Modeling and verification of time dependent systems using time petri nets', *IEEE transactions on Software Engineering* **17**(3), 259–273.

Bruno, G., Castella, A., Macario, G. and Pescarmona, M. 1992, Scheduling hard real time systems using high-level petri nets, *in* K. Jensen, ed., 'Application and Theory of Petri Nets 1992', Springer Verlag, Berlin, pp. 93–112.

Caloini, A., Magnani, G. and Pezze, M. 1998, 'A technique for designing robotic control systems based on petri nets', *IEEE transactions on Control Systems Technology* **6**(1), 72–87.

Carlier, J. and Chretienne, P. 1988, Timed petri net schedules, *in* G. Rozenberg, ed., 'Advances in Petri Nets 1988', number 340 *in* 'Lecture Notes in Computer Science', Springer Verlag, Berlin, pp. 62–84.

Colom, J., Silva, M. and Villarroel, J. 1986, On software implementation of petri nets and colored petri nets using high-level concurrent languages, *in* 'Proc of 7th European Workshop on Application and Theory of Petri Nets', Oxford, pp. 207–241.

Coste-Manière, E. and Turro, N. 1997, The maestro language and its environment: Specification, validation and control of robotic missions, *in* 'Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)', Grenoble, France, pp. 836–841.

Crowley, J. 1995, Mathematical foundations of navigation and perception for an autonomous mobile robot, *in* 'Tutorial at Int.Workshop on Reasoning with Uncertainty in Robotics', Amsterdam, The Netherlands.

Felder, M., Ghezzi, C. and Pezze, M. 1993, 'High-level timed petri nets as a kernel for executable specifications', *Real-Time Systems* **5**(2), 235–248.

García, F. and Villarroel, J. 1996, Modelling and ada implementation of real-time systems using time petri nets, *in* 'Proc. of the 21st IFAC/IFIP Workshop on Real-Time Programming', Gramado - RS, Brazil.

García, F. and Villarroel, J. 1998, Decentralized implementation of real-time systems using time petri nets. application to mobile robot control., *in* 'Proc. of the 5th IFAC Workshop on Algorithms and Architectures for Real-Time Control, AARTC'98', Cancun, Mexico.

García, F. and Villarroel, J. 1999, Translating time petri net structures into ada 95 statements, *in* 'Reliable Software Technologies - Ada-Europe'99',

number 1622 *in* 'Lecture Notes in Computer Science', Springer Verlag, Berlin, pp. 158–169.

Giovanni, R. D. 1990, Petri nets and software engineering: Hood nets, *in* 'Proc. of 11th International Conference on Application and Theory of Petri Nets', Paris, France, pp. 123–138.

Hanisch, H. 1993, Analysis of place/transition nets with timed arcs and its application to batch process control, *in* M. A. Marsan, ed., 'Application and Theory of Petri Nets 1993', number 691 *in* 'Lecture Notes in Computer Science', Springer Verlag, Berlin, pp. 282–299.

Hu, H., Brady, J., Du, F. and Probert, P. 1995, 'Distributed real-time control of a mobile robot', *Intelligent Automation and Soft Computing* **1**, 63–68.

Khatib, O. 1986, 'Real-time obstacle avoidance for manipulators and mobile robots', *International Journal of Robotics Research* **5**, 90–98.

Merlin, P. and Faber, D. 1976, 'Recoverability of communication protocols', *IEEE transactions on Communication* **24**(9).

Montano, L. and Asensio, J. 1997, Real-time robot navigation in unstructured environments using a 3d laser rangefinder, *in* 'IEEE-RSJ International Conference on Intelligent Robots and Systems', Vol. 3, Grenoble, France, pp. 526–532.

Murata, T. 1989, 'Petri nets: Properties, analysis and applications', *Proc. of the IEEE* **77**(4), 541–580.

Muro, P., Banares, J. and Villarroel, J. 1998, 'Kron: Knowlege representation oriented nets for discrete event systems applications', *IEEE transactions on Systems Man and Cibernetics* **28**(2).

Oliveira, P., Pascoal, A., Silva, V. and Silvestre, C. 1998, 'Mission control of the marius autonomous underwater vehicle: system design, implementation and sea trials', *International Journal of Systems Science* **29**(4), 1065–1080.

Popova, L. 1991, 'On time petri nets', *J. Inform. Process. Cybern.* **EIK** **27**(4), 227–244.

Ramachandani, C. 1974, Analysis of Asynchronous Concurrent Systems by Timed Petri Nets, PhD dissertation, Massachussets Inst. of Technology.

Simon, D., Espiau, B., Kapellos, K. and Pissard-Gibollet, R. 1997, 'Orccad: software engineering for real-time robotics. a technical insight', *Robotica* **15**, 111–115.

Sloan, R. and Buy, U. 1996, 'Reduction rules for time petri net', *Acta Informatica* **33**(7), 687–706.

Villarroel, J. 1990, Integración Informática del Control de Sistemas Flexibles de Fabricación, PhD thesis, Dpto. de Ingeniería Eléctrica e Informática, University of Zaragoza.

A Code generated for the mobile robot application

In this appendix several pieces of code corresponding to the decentralized implementation presented in section 5.2 are shown. The result of net partition is:

- Four processes: Supervisor, Control, Laser and the periodical activator of the Control process, the Control_Activation process.
- Eight buffers of private destination implemented as relays.
- Three places of no private destination implemented as protected objects: *Goal*, *State* and *Scans*.

To illustrate the generated code, this section includes the code of the four processes, an example of relay place and another example of protected object.

```
-----------------------------------------------
--  Periodical activation of Control process --
-----------------------------------------------

task body Control_Activation is
  last_update : time;
  period: time_span:= to_time_span (0.25);
begin
  accept start;
  last_update := clock;
CT1:
  loop
    select
      CC1.Demark ;
      exit CT1;
    then abort
```

```
      delay until last_update + period;
      last_update := last_update + period;
      CT2.Mark ;
    end select;
  end loop CT1;
end;

-----------------------------------------------
--           Control process                 --
-----------------------------------------------

task body Control is
 Read_Location_TO: constant duration:=0.1;
 Send_Set_Point_TO:constant duration:=0.1;
begin
C1 :
  loop
    select
      accept Control_Star_Period;
      select
        delay Read_Location_TO;
        CC3.Mark;
        exit C1;
      then abort
        Read_Location;
      end select;
      State.Demark;  Store_State;
      State.Mark;    CC4.Mark;
      Scans.Demark;  Read_Scans;
      Scans.Mark;    Scans_Integration;
      Goal.Demark;   Read_Goal;
      Goal.Mark;     Compute_Setpoint;
      select
        delay Send_Setpoint_TO;
        CC3.Mark;
        exit C1;
      then abort
        Send_Setpoint;
      end select;
    or
      accept Stop_Control;
      exit C1;
    or
      accept End_Of_Trajectory;
      Move_To;
      exit C1;
```

```
    end select;
  end loop C1;
  Robot_Stop;
end;

-----------------------------------------------
--             Laser process                 --
-----------------------------------------------

task body Laser is
  Scan_Read_TO: constant duration := 0.1;
begin
L1:
  loop
    select
      LC1.Demark;
      exit L1;
    then abort
      select
        delay Scan_Read_TO;
        exit L1;
      then abort
        Scan_Read;
      end select;
    end select;
    State.Demark;  Laser_Read_State;
    State.Mark;    Correct_Scan;
    Scans.Demark;  Store_Scan;
    Scans.Mark;
  end loop L1;
  Stop_Scan;
end;

-----------------------------------------------
--    No private destination place State     --
-----------------------------------------------

protected body State is
  procedure Mark is
  begin
    M:= M + 1;
  end;
  entry Demark when M > 0 is
  begin
    M:= M - 1;
  end;
```

```
end;

-----------------------------------------------
--             Relay place CC2               --
-----------------------------------------------

task body CC2 is
begin
  loop
    select
      accept Mark;
    or
      terminate;
    end select;
    Control.Stop_Control;
  end loop;
end;
```

List of Figures