Control and Cybernetics
vol. **24** (1995) No. 1

Petri net model of a mobile robot

by

Grzegorz J. Blinowski

Institute of Computer Science,
Warsaw University of Technology
ul. Nowowiejska 15/19,
00-665 Warsaw, Poland
e-mail: gjb@ii.pw.edu.pl

This paper addresses the problems of modelling and evaluation in distributed parallel systems using *Petri nets with time*. A net model incorporating time, control over nondeterministic behaviour and resource pools is presented. The model guarantees that if the set of processes is correct (i.e. non-deadlocking) then corresponding net, even after resources are added, will be also non-deadlocking.

It is shown how basic programme constructs such as choice and loop statements can be implemented in nets. A single programme consists of *events* which can be *internal* or *external*. Internal events represent computational steps, while external – instances of inter-process communication. The events will map to Petri net transitions.

Resource sharing can lead to deadlocked systems. *Simple Extended* nets can be safely used in time modelling. Such nets will be used in the presented case study.

Probability introduced into the Petri net formalism allows to model real world events more adequately; while *priority* assigned to certain transitions enables construction of a process scheduler.

Our main goal is performance estimation. *Case study* concerns a communicating sensor system in an Autonomously Guided Vehicle. We construct a net model of sensor framework of a mobile robot. Starting from a quasi-subsumption paradigm we obtain a model of the net of intelligent sensors operating independently, sharing information about the surrounding world. It is assumed that some sensors operate faster than others, while providing less information per unit time. The basic events are instances of executions of data processing algorithms and of data communication. It is shown how overall performance is influenced by such factors as the number of available processing units and speed of data processing algorithms. *A Figure of Merit* for estimating global system performance is proposed.

1. Introduction

During the past decade considerable effort has been devoted to research in fields related to mobile, intelligent robots. While such problems as multi-sensor fusion, path planning, obstacle avoidance, navigation, etc. have been heavily studied, only recently the need of integration of the above areas was realised, Wang (1991). One aspect of such an integration is an important, and still unsolved problem of the estimation of systems performance. The measure of performance is needed for both evaluation of systems throughput vs. complexity (i.e. cost) and unbiased comparison of different architectural models.

Performance estimation requires a high-level model which will integrate mentioned areas, and will be based on formal specification of the systems' architectures. Such an approach will help to clarify routes of the information flow, process dependences between all levels and modules of the system; in the consequence a meaningful estimation of the systems quality can be obtained.

This paper describes an effort to create such an integrating model. It was partly carried out in the Oxford University Robotics Research Group (Dep. of Engineering Science) as a contribution to the Oxford AGV project Hu, et al. (1989).

Petri nets with time and some additional enhancements were chosen as a primary modelling tool. Petri net theory has come to be a recognised modelling methodology in the robotics literature, especially in the field of Flexible Manufacturing Systems, Freedman (1990). Recently it has been applied to other areas such as individual machine modelling, Scimachen, Grotzinger, Nemec (1990) and mobile robot control, Wang (1991).

The developed model is based on the formal specification of communicating, intelligent sensor architecture (logical sensors) described in Dijan, Probert (1990). The transition from the specification (written in *CSP*) to the net model was described in Blinowski, Probert (1992). A whole sensor network equivalent to a mobile vehicle was implemented in Petri nets. Since the obtained net system was far too complex to investigate analyticaly a series of computer experiments was carried out in order to obtain the following:

- Verification of the formal model.
- Finding of time-critical dependencies between various data-processing algorithm times and vehicle reaction times.
- Estimation of the processing power needed to support critical time parameters.
- Choice of an optimal process scheduling method.

This paper is divided into three parts. In the first one the basics of Petri nets are covered. The issue of process modelling is tacled in the second, while the last one contains the case study.

2. Place/transition systems – theoretical overview

We will begin with presenting the outline of the P/T-net theory – the definition of a net, followed by definitions of a P/T system and its behaviour. In subsequent section the problem of fairness of the token game will be discussed, which will lead to a definition of an extended "Probability/Priority" P/T system.

Certain useful properties i.e. liveness and boundedness can be proved only for some subclasses of nets, time performance can be evaluated analytically only for nets of particular topologies: Section 2.3. defines classes of nets in the order of increasing complexity.

Time is the central topic in our considerations. Various methods of incorporating time into presented net theory will be discussed in section 2.4. The most popular approaches are going to be presented, with the stress on models which assign a fixed time duration or a well defined time interval to an event.

2.1. Fundamental concepts of P/T systems

The overview of P/T-systems can be found in Reisig (1987), Lautenbach (1987), Reisig (1985), definitions and theorems in this and the following sections are based on Reisig (1987), Lautenbach (1987).

Definition 2.1 *A triple* $(S;T;F)$ *is called a net iff:*

1. S *and* T *are disjoint sets (of places and transitions respectively).*
2. $F \subseteq (S \times T) \cup (T \times S)$ *and* $\forall t \in T$ *there exists* s, s' *such that* sFt *and* tFs'.

Let us furthermore define the sets of predecessor and successor elements:

Definition 2.2 *For* $x \subseteq S \cup T$ *let* ${}^\bullet x := \{y \mid yFx\}$ *and* $x^\bullet := \{y \mid xFy\}$

Definition 2.3 *A net marking is a function* $M : S \to \mathbf{N}$

If $M(s) = 0$ the place s is said to be empty;
if $M(s) > 0$ we say that a token(s) is (are) present in place s.

In order to introduce token dynamics the net definition must be augmented to that of the Place/Transition system:

Definition 2.4 *A six-tuple* $(S;T;F;K;W;M_0)$ *is called a place/transition system if:*

1. (S,F,T) *is a net where S-elements are called places and the T-elements are called transitions*
2. $K : S \to N^+ \cup \{\infty\}$ *is a capacity function*
3. $W : F \to N^+$ *is a weight function*

4. $M_0 : S \to N$ *is an initial marking function, which satisfies* $M_0(s) \leq K(s)$ *for all* $s \in S$

The W function is often canonically extended to cover the whole sum of Cartesian products by defining:
$W(x, y) = 0$ iff $\neg(x, y) \in F$.

The next definition describes dynamic aspects of P/T systems:

Definition 2.5 *1. Transition t is enabled at marking M if:*

$$\forall s \in S : W(s,t) \leq M(s) \leq K(s) - W(t,s)$$

2. *if t is enabled at M then t may occur (fire), yielding a new marking M' given by the equation:*
$$M'(s) = M(s) - W(s,t) + W(t,s) \text{ for all } s \in S$$
3. *The occurrence of t changes the marking M into the new marking M'; this fact being denoted by $M[t > M'$, MtM' or $M \xrightarrow{t} M'$.*
4. *By $[M_0 >$ we denote the smallest set of markings of Σ such that:*
   (a) $M_0 \in [M_0 >$
   (b) *if* $M_1 \in [M_0 >$ *and* $M_1[t > M_2$ *for some* $t \in T$ *then* $M_2 \in [M_0 >$

The previous definition states simply that:

- transition is enabled when there is sufficient (defined by the W function) amount of tokens in all places being its inputs, and that there must be enough 'free space' in places being its outputs.
- Enabled transition fires, removing tokens from its input places and putting tokens in its output places. Note that not every enabled transition must fire: tokens from its input places can be removed by another transition.

2.2. Fairness & Probability/Priority P/T system.

When performance is being studied, fairness of the token game becomes the key issue; also, when modelling real-world systems some control over nondeterministic behaviour of P/T nets is desirable. Up to this point it has been assumed that the choice of transition-to-execute is random and generally fair. Note that the liveness of the net does not imply fairness.

Definition 2.6 *The net is live if for any transition and all reachable markings there exists such a sequence of reachable markings that will eventually lead to marking in which the given transition is enabled.*

There is no guarantee that the enabled transition will be fired. In fact it is possible that some transitions will "conspire" against the other ones to

prevent their firing. These issues are extensively discussed in Merceron (1987). The problem of fairness is going to be solved here by the introduction of firing rules which are located on a different level of abstraction than the basic P/T formalism. Such an approach is purely practical: these new rules simply try to enforce the "real world" behaviour on abstract nets.

Now, an Extended Probability/Priority P/T system (**PPP/T**) in which the procedure of choosing among enabled transitions is completely clarified is going to be introduced:

Definition 2.7 *Let P be a firing probability function defined for certain transitions only:*

$$if \quad \forall s \in S, \forall t \in s^{\bullet}, \forall s' \in {}^{\bullet}t \quad |s'^{\bullet}| > 1 \;\Rightarrow\; {}^{\bullet}(s'^{\bullet}) = \{s'\}\}$$

then P is defined:

$$P : t \rightarrow < 0, 1 > \quad such\ that : \quad \sum_{t' \in s^{\bullet}} P(t') = 1$$

($|A|$ denoting cardinality of A)

Definition 2.8 *Let T_P be the set of transitions over which P is defined:*

$$T_P = \{t \in T | \forall s' \in {}^{\bullet}t, |s'^{\bullet}| > 1 \;\Rightarrow\; {}^{\bullet}(s'^{\bullet}) = \{s'\}\}$$

Example 2.1 *Consider Figure 1, P is defined for t_0, t_1 and t_2; $t_0, t_1, t_2 \in T_P$ but $t_3, t_4 \notin T_P$*

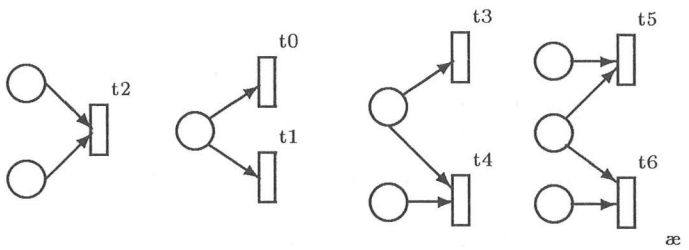

Figure 1. Probabilistic and non-probabilistic transitions.

We assign probability to these transitions which we know to be always simultaneously enabled with some fixed set of other transitions (or enabled independently). In case of ambiguity i.e. transition t_3,t_4,t_5,t_6 in Figure 1 it is difficult to come up with any reasonable, **marking independent** probability assignment, in such case we will define priority instead. Priority will be assigned to all transitions which do not belong to T_P set. Furthermore the $T \setminus T_P$ set will be partitioned into subsets: priority groups.

Definition 2.9 $\forall t \in T\ ,\ t \notin T_P$ *let:*
$P_{group} : t \rightarrow \mathbf{N}$ *be a partitioning function.*
$P_{prio} : t \rightarrow \mathbf{N}$ *be a priority function.*

In other words: The set of all transitions which cannot be assigned to the set of probabilistic transitions is divided into disjoint sub-sets. As the next definition will formally show only the transition with the highest priority in a given priority group can be fired, thus transitions belonging to the same group and having lower priority will be able to fire only if transitions with larger priority will not be enabled. This mechanism will prove to be useful in modelling schedulers.

Now a P/T system with a firing mechanism strictly defined can be introduced. The enabling conditions and firing rule remain the same for extended **PPP/T** system. The probability of execution of a particular transition under given marking is now stated to be $PR(t)$.

Definition 2.10 *For all transitions in enabled set a firing probability function is defined:*

$$PR(t) := \begin{cases} \frac{|T'_p|P(t)}{|T_E|} & \text{if } t \in T_P \\ \frac{|T_s|}{|T_E||T_h|} & \text{if } t \notin T_P \wedge t \in T_h \\ 0 & \text{for other transitions} \end{cases}$$

Where:

$T'_p(t) := \{t' | {}^\bullet t' = {}^\bullet\}$

$T_s(t) := \{t' | P_{group}(t') = P_{group}(t)\ \wedge t' \in T_E\}$

$T_h = \{t' | t' \in T_s\ \wedge\ P_{prio}(t') = max(P_{prio}(T_s))\}$

And T_E *is a set of enabled transitions.*

A firing probability of zero is assigned to priority transitions with priority lower than the maximum priority in a given group. The probability of firing the transition with highest priority in a particular group is the same for all transitions with maximum P_{prio}.

The above definition does not imply that for any run of an arbitrary net the actual firing rates will match those defined above, this will be the case only in a *live* net.

To summarise:

Probability: As was previously stated firing probability can be defined only for transitions for which it would be meaningful (i.e. marking independent). For a group of transitions having only one and common input place

(such transitions are always simultaneously enabled), execution probability will be assumed to be evenly distributed, unless otherwise indicated in the graph description (see also the section on the simulator's graph language).

Priority: It was stated that all transitions for which probability could not be defined belong to some priority group in which conflicts are resolved in favour of the transition with highest priority. By default, however, it will be assumed that non-probabilistic transitions belong to one element priority groups. Thus, only when a priority group and priority function are explicitly written down is this mechanism in effect in practice. One must take notice that creating real priority group will always lead to nets which are not live. Time delays must be chosen carefully in such a net.

2.3. Classes of P/T nets

There is a lot of confusion in the literature regarding P/T net "taxonomy". Generally, "Place/Transition Net" is a term used when referring to the widest possible class of nets. A "Petri net" is usually a P/T system devoid of "pathologies'/.

By introducing certain limitations on functions describing a P/T net, particularly by constraining the F function various net classes are obtained. It was shown how such essential properties as *liveness* and *boundness* can be proved only for some types of nets with simplified topologies, and how increasing the generality of a net makes analysis more difficult. The same issue of universality vs. applicability of analytical methods will be brought forward in the section on time/timed P/T nets.

Let us constrain the P/T model by introducing:

Definition 2.11 *A* Petri net *is a P/T net for which:*

$$K : S \rightarrow \infty$$
$$\textit{and}$$
$$\forall s \in S\{t|sFt \ \wedge \ tFs\} = \emptyset$$

Thus in *Petri nets* the issue of contact will be dropped all together, and one transition "loops" will not be permitted. Notice that the enable criterion for a transition is now simpler, namely transition in a Petri net is enabled iff:

$$\forall s \in S : W(s,t) \leq M(s)$$

Now two very limited, but still important sub-classes of Petri nets are going to be defined:

Definition 2.12 Σ *is said to be* a marked graph *(or* a decision free net*) iff:* Σ *is a Petri net and:*

$$\forall s \in S \ |{}^\bullet s| = |s^\bullet| = 1$$

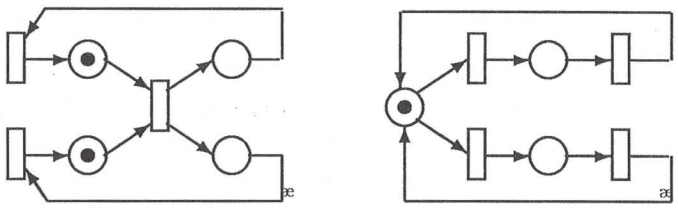

Figure 2. Marked graph & state machine.

Definition 2.13 Σ *is said to be a state machine iff:*
Σ *is a Petri net and:*

$$\forall t \in T \; |{}^\bullet t| = |t^\bullet| = 1$$

Definition 2.14 Σ *is* a free choice net (FC) *iff:*
Σ *is a Petri net and:*

$$\forall s \in S : |s^\bullet| > 1 \Rightarrow {}^\bullet(s^\bullet) = \{s\}$$

Definition 2.15 Σ *is an* extended free choice (EFC) *net iff:*

$$\forall s_1, s_2 \in S \qquad s_1^\bullet \cap s_2^\bullet \neq \emptyset \quad \Rightarrow \quad s_1^\bullet = s_2^\bullet$$

An EFC permits a place to 'share' more than one transition with other places under the condition that if some transitions are shared *all* of them have to be shared.

Definition 2.16 Σ *is an* extended simple *net iff:*

$$\forall s_1, s_2 \in S \; s_1^\bullet \cap s_2^\bullet \neq \emptyset \Rightarrow s_1^\bullet \subseteq s_2^\bullet \vee s_2^\bullet \subseteq s_1^\bullet$$

This class is the widest one which preserves liveness, after transition time delays are introduced.

Example 2.2 *Figure 2 shows the examples of* marked graph *and* state machine nets.

2.4. Modelling time in P/T-nets.

Petri net models with time assign a fixed duration to either:

- places – a condition is true for, or after a certain amount of time – see Sifajis (1980).

- transitions – firing is not instantaneous, rather it takes some time to complete – more precisely: a firing event is delayed by some amount of time from the moment of enabling – see Merlin (1976).

It was demonstrated that these two approaches are, in fact, equivalent: it is possible to transform the net with timed conditions into the one with timed transitions while retaining the same behaviour. The second case associating **time** with **events** seems more 'natural' and promising both in applicability and in possibilities of extension. There are also various approaches to assigning time to transitions: A transition can consume a token as soon as it becomes enabled, but produces the output token after some delay of time, alternatively we can assume that tokens remain in their input places whilst the transition is in the process of 'delayed firing'. First model is more limited, but makes theoretical reasoning and analysis easier. We are going to adopt the latter approach, since our primary method of analysis is simulation. Note that the first type of a transition can be modeled by a simple path of zero-time transition, followed by a place, followed by a timed output transition. It will be also assumed that when a token enters an input place of the already enabled transition it has to wait until the firing completes and then re-enable the transition. This approach is realistic – it assumes that events are atomic.

Generally, the nature of any timed transition model can be expressed by a function : $\Theta : T \rightarrow \mathbf{R}^+$ or $\Theta : T \rightarrow \mathbf{R}^+ \times \mathbf{R}^+$. Θ assigns a time duration or a time interval to a transition, it can also depend on other variables: e.g. marking, time or token value (in a P/T system with variable/value environments tied to tokens). In a simplest possible model $\Theta : T' \rightarrow \mathbf{R}^+$ a fixed duration is assigned to each transition from a subset T' of T, firing is delayed by this amount of time. This approach was developed by Ramchandani (1974) to study the performance of supercomputer processor structures.

Merlin's Time Petri net (TPN), Merlin (1976), is a more general model:

$$\Theta : T \rightarrow [\tau_1, \tau_2] \quad \tau_1, \tau_2 \in \mathbf{R}^+ \quad \wedge \quad \tau_1 \leq \tau_2$$

An event will always occur between times τ_1 and τ_2 from the moment of enabling (of course like in Ramchandani's model a transition can be disabled before firing if the net is nondeterministic). This model together with a subclass of PN consisting of combination of marked-graph and state-machine classes was successfully used by many researchers to analyse systems performance.

3. Modelling processes with Petri nets

3.1. The modelling tools

Let us bring together the modelling tools that have been presented so far: Petri nets with time intervals assigned to transitions are going to be used. Further, probability and priority functions have been introduced to influence the choice of an event (i.e. transition) to execute.

The model itself will be built from *processes*. A process will be regarded as a black box, with most of the internal mechanisms concealed. A process behaviour is given by describing its interactions with the environment. At the process level of abstraction we are not concerned with any details of the actual algorithm used inside the 'black box'. A given process will be described by its interactions with the outside world, namely, by the input and output *events*. Events are the fundamental, indivisible entities – quanta – of the model. Two types of events are going to be distinguished: *internal* and *common* ones. An instance of the *internal event* can appear only in one process, thus a process can engage in its internal events whenever it is ready to do so, without any interaction with the environment (other processes' events). Internal events either model the execution of some computational procedure – to account for its execution time, or represent a strictly nondeterministic choice of a process. In reality such a choice reflects the decision taken by some data evaluation programme (for example sensor integration module deciding that its local sensing device is faulty). *Common events* are shared between (usually) two processes. They model synchronisation of exchange of data. Again we are not concerned here with the nature of this data. We might, however, be interested in such factors as data packet length and communication channel speed to evaluate the time duration of the communication event.

3.1.1. Time, resources and the underlying architecture

Our goal is to model execution of concurrent processes running on a certain type of a parallel architecture. This 'target' architecture should provide the following (a transputer system was intended as a platform for this particular project):

- Multiple processors.
- Message passing communication system.
- Some form of support for process scheduling.
- Full Scalability i.e. the number of processing elements in the system should be transparent to application software. System's performance will of course be influenced by the number of processing elements.

It will not be assumed that the number of processing units is unbounded. On the contrary, we are going to constrain the model in such a way that limited number of processors, a 'processor pool', will have to cope with many parallel tasks. To introduce this resource sharing mechanism, it must be assumed that all temporal events fall into two basic classes: *communication* or *computation* oriented, and further, that only the events from the latter class actually require processor power.

Our resource scheduler is going to work on *First Come, First Served* basis, possibly using priorities to choose from a number of tasks waiting for a processor. By FCFS we understand that the scheduler is not preemptive.

Example 3.1 *Figure 3 shows two processes P1 and P2 both performing some processor-demanding computation – compute. Each one is able to engage in this activity only if a processor – a token in R – is available. When the computation is finished the resource is released, which is modeled by returning arcs from t_{12}, t_{22} to R.*

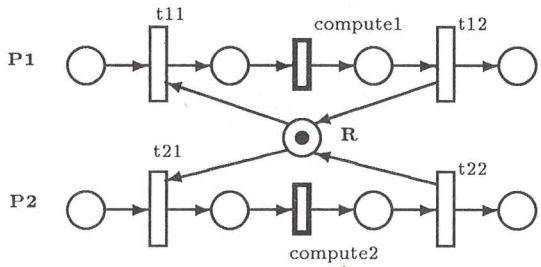

Figure 3. Processor sharing.

If several computation modelling events are locked like this, the level of 'serialisation' can be regulated by the number of tokens in the resource pool – thus it is possible to study the behaviour of processor bottlenecks.

3.1.2. Semaphores

A mechanism identical to the one used in resource sharing is used in our model to 'lock' certain sections of the net in order to simulate *a critical section* guarded by a *semaphore*. This is used in our case study, for example, in sensor integration module: the operation of *World Model Updating* is performed on a database which must be kept consistent.

3.2. Time, liveness and net classes

Remark 3.1 *Timed Petri nets can have dead transitions, even when their non-timed equivalent are live. This is a consequence of the fact that the Reachability Graph of Timed Petri net can be smaller than RG of its non-timed equivalent.*

A question arises: What conditions must a net meet in order to preserve liveness after time is added? In the intuitive process–to–net translation procedure described above *Free Choice* nets will be obtained if resource sharing is not taken into account. If resource sharing is permitted, and further, multiple resources per transition are permitted *Simple Extended* nets will be the result of process–to–net translation.

Resource sharing involves adding extra places which model the resource pool, and connecting them to a group of transitions modelling events whose execution

requires resource acquisition. The same procedure is used to incorporate binary semaphores, which are simply one token resource pools.

An important restriction that must be imposed on the whole class of transitions involved in resource sharing is that groups of transitions sharing a particular resource have to be either disjoint, or be contained in some other resource sharing group. In other words: it is prohibited for a resource group to "partially share" another resource – see Figure 4.

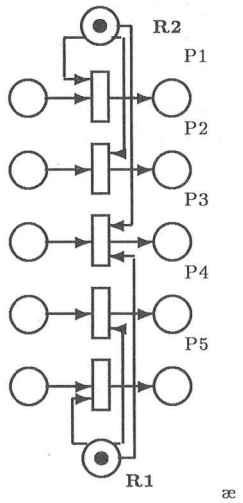

Figure 4. Illegal resource sharing. Process $P3$ cannot share **R1** and **R2** in this configuration.

Now time can be introduced into Extended Simple nets – simply by assigning time intervals/time delays of events to their respective transitions. The remaining transitions will have a time delay approaching zero.

Theorem 3.1 (After Starke, 1989) *Let $\Sigma = (S;T;F;W;M)$ be a live Petri net with the following properties:*

- $\forall s \in S\ s^\bullet \neq \emptyset$ *(every place has a post-transition).*
- *The net (S;T;F) is* Extended simple.
- $\forall s \in S\ \forall t,t' \in s^\bullet\ W((s,t)) = W((s,t'))$ *(All arcs emerging from one place have the same weight)*

Then, the timed net, with token holding transitions $(S;T;F;W;M;\Theta)$ is live for every $\Theta : T \rightarrow \mathbf{N}^+$.

4. A Petri net model of sensor communication in an AGV – a case study

Introduction

A Petri net model of an intelligent mobile robot will be presented in this section. The principles of modelling were explained in the previous section. Performance of the model is going to be evaluated via computer simulation in order to obtain:

- Time-critical dependencies between various data-processing algorithm times and vehicle reaction times.
- Estimation of the processing power needed to support critical time parameters.
- Choice of an optimal process scheduling method.

The presented AGV is based on the model of the *layered* architecture, which is derived from the **subsumption** paradigm proposed by Brooks (1985). A classical subsumption architecture consists of independently and concurrently operating *layers* with access to robot's actuators. Lower layers can subsume the operation of the higher ones when necessary, only one layer at a time actually controlling the robot. For example the path planner will be located at a higher level than the moving-obstacle avoidance layer, the latter being able to intervene when an obstacle is detected, and return control to the path planner when the AGV is safe. The main difference between standard subsumption architecture and the one used here is the presence of inter-sensor communication in the latter. This difference will be further underlined in next sections.

To fulfil all the 'universality' requirements we have chosen to represent an AGV as a network of logical sensors of the type previously described in Dijan, Probert (1990). The sensor model had to be translated from *CSP* to Petri net representation; some nonessential features have been removed and the integration module has been redesigned.

4.1. The logical sensor and the subsumption architecture

The main characteristics of a Logical Sensor (LS) are:

- Unified interface – Different physical sensors present a unified interface, a LS can be used at various layers of the robot architecture without the need of its redesign.
- Integration through communication – sensors can communicate with each other to establish a unified environmental model, furthermore such a model can be *integrated* from incomplete or partial information.

The LS is also expected to be able to:

- Assess its own condition (for example to disconnect itself from the network in case of malfunction).
- Employ a parallel design — thus it can be implemented on several processors.

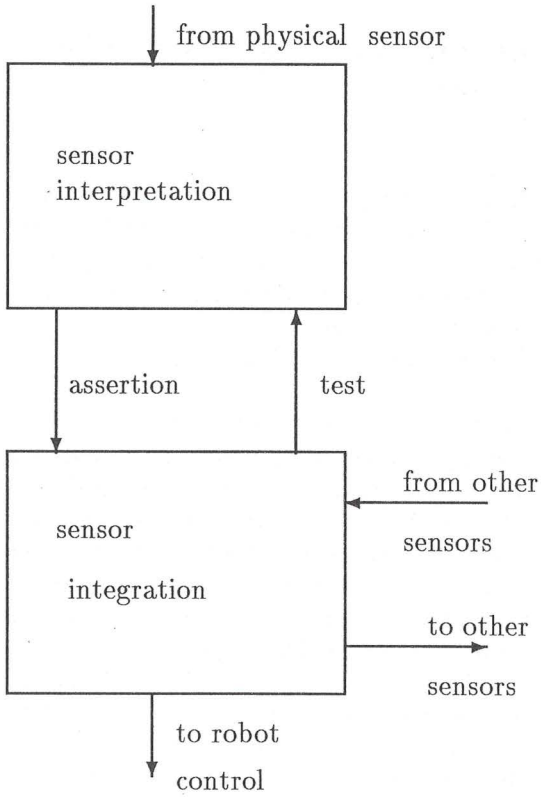

Figure 5. The logical sensor

4.1.1. The operation of a logical sensor

An LS produces an interpretation of the state of the its environment, it is also able to integrate its own interpretation with the information received from other sensors. Thus a high level implementation of a LS consists of two modules: sensor interpretation and sensor integration – Figure 5.

Sensor interpretation: this module contains the actual physical sensor device, it processes the signal received from environment and sends an assertion of its interpretation of the surroundings to the integration module.

Sensor integration: updates an internal world model according to input received from the sensor interpretation module and from the other sensors, informs other sensors about its new assertions, produces feedback to the interpretation part, interfaces directly to the robot control hardware.

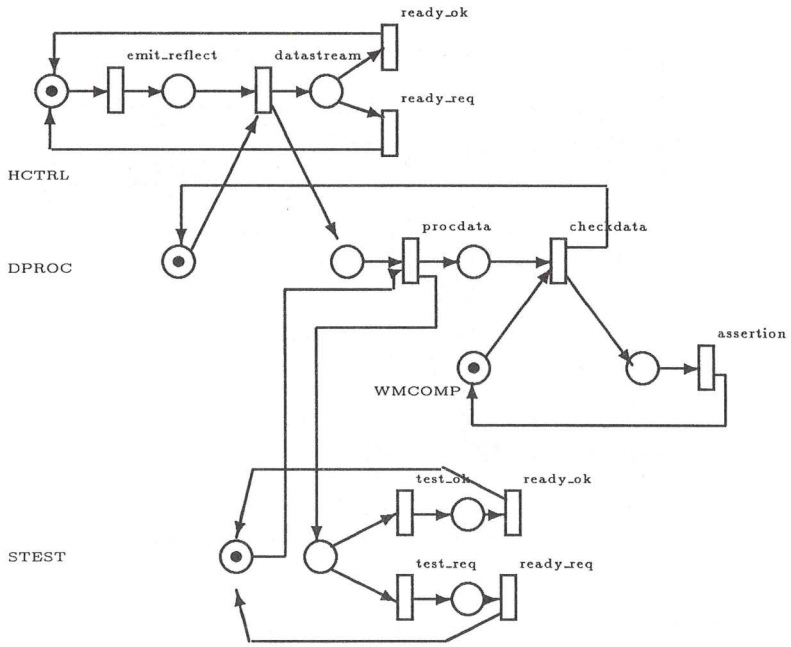

Figure 6. The logical sensor: interpretation module. Transitions *ready_ok* and *ready_req* appear twice: they are duplicated for the sake of clarity of the figure and represent the same objects.

The operation of sensor interpretation module. The interpretation net – Figure 6 consists of four circuits each representing a single, independent process. Each such circuit is PN invariant containing always exactly one token. The position of the token represents process' current state. The *HCTRL* – Hardware Control, *DPROC* – Data Processing, *STEST* – Self Test and *WMCOMP* – World Model Comparison circuits synchronise the transitions modelling message and data passing between processes.

HCTRL process performs the actual measurement – *emit_reflect* transition, passes raw data to *DPROC* through *datastream*, and checks whether it should make a new measurement or perform a self-test – *ready_ok, ready_req* transition choice.

DPROC receives raw data from *HCTRL* and passes it to *STEST* and *WMCOMP* processes by *procdata* and *checkdata* transitions respectively.

STEST synchronises with *DPROC* on transition *procdata* and waits for integration module to enable *test_req* or *test_ok*, subsequently enabling either *ready_req* or *ready_ok* for *HCTRL*. This circuit is responsible for propagating self-test requests from integrating part of the sensor, it will ensure that a faulty

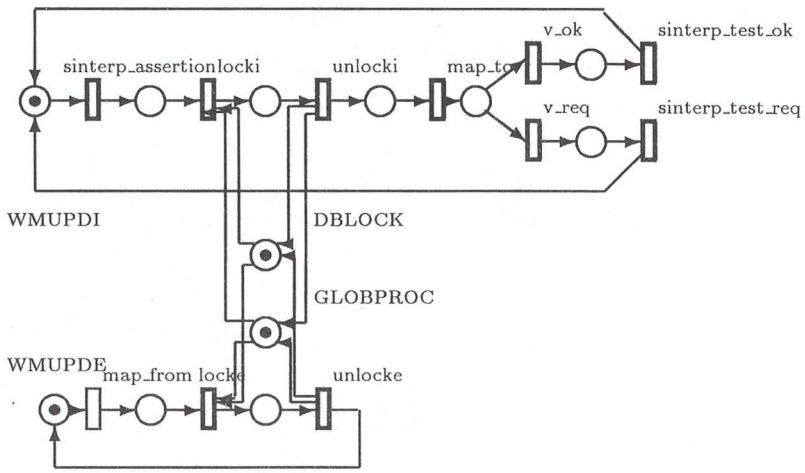

Figure 7. The logical sensor: integration module.

sensor will not pass data to integration module.

WMCOMP outputs a hypothesis *assertion* based on physical sensor's data *checkdata* to sensor integration module. We assume that *assertion* is a probabilistic function of *checkdata*, the actual description of this function is not relevant here.

The operation of sensor integration module. This module – Figure 7 – maintains the world model, WM. It services the requests from sensor interpretation and other sensors to access this data base. The requests will not necessarily involve updating of the WM – for example sonar data can be matched against other sensors map to perform a triangulation. They will nevertheless always require consistent data to operate upon, thus a database access cannot be shared.

Two circuits: WMUPI (World Model Update – Internal) and WMUPE (World Model Update – External) model the database operations. WMUPI synchronises the transition *assertion* with the integration module, then it locks the world model database – acquiring this resource through *locki* transition, performs the update operation which takes T_{locki} time units, releases the database – *unlocki* and communicates the update to other sensors *map_to*. Finally a verdict concerning integration module proper functioning is issued – this is modeled by non-deterministic transition choice: *v_ok, v_req* (currently it is assumed that the sensor always provides correct readouts). WMUPE circuit synchronises with the external sensors' *map* transitions and updates the internal db – *locke*, *unlocke* sequence is identical to the one in WMUPI circuit.

Updating the world model uses considerable computing power – thus a processor has to be allocated from the common pool in order to continue with the

update operation – *locki, unlocki* and *locke, unlocke* transition sequences handle this global resource.

4.1.2. The layered architecture

The logical sensor model described above was designed to operate in a specific type of a *Layered* or *Subsumption* architecture.

The architectures we propose to study differ considerably from the 'classical' layered model – they provide the means for inter-layer communication (in an ordinary subsumption architecture integration takes place only in the actuators). In fact the operation of this communication path is probably the most important factor influencing the performance. Presence of a local world model in each sensor is a second feature which a subsumption architectures lack (Brooks explicitly denies the need for maintaining a world model, claiming that the world is a best model of itself). Having underlined this difference of approaches it is suitable to mention that the trade offs between heavy inter-sensor communication and sensor autonomy will be one of the most interesting aspects of performance evaluation.

4.2. The AGV

Consider a simple mobile robot – Figure 8, being able to navigate in a known, indoors environment. It is equipped with four clusters of sonars each with a 90^o view angle, and a 360^o IR scanner. Sonars are mainly responsible for moving and avoidance of unexpected obstacles, serving possibly a secondary purpose of navigating. An IR scanner keeps track of the vehicle surroundings – in effect locating the AGV on the map.

4.2.1. The sensor network

The corresponding LS network is shown in Figure 9 . Following communication paths exist in the sensor network:

- Sonar to sonars – Since scan areas of two neighbouring sonar sensors overlap distance measurement data for common sections is exchanged. This is a low bandwidth link.
- Sonars to IR – Distance data is passed to IR sensor, possibly to perform additional triangulation computations, message length is insignificant, but there is some computation involved in processing this data.
- IR to all sonars – Map segments are communicated via this link, the bandwidth and necessary computations in sonar sensors are considerable.

A part of the network in larger detail is shown in Figure 10. The *forksonar* – Figure 11 and *forkir* – Figure 11 processes are responsible for distributing messages, thus reducing the load on sensors' integration modules. Each fork process synchronises on sender's *map_to* transition and closes recipient's *world model update – external* circuit with *map_to_a, map_to_b, map_to_c* transitions.

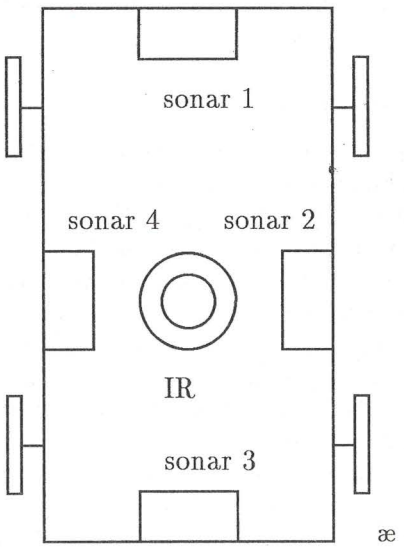

Figure 8. The AGV: Physical model.

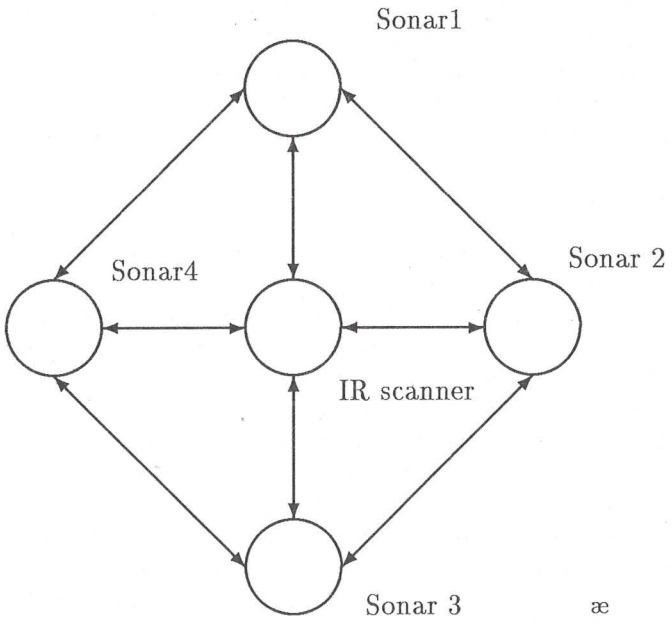

Figure 9. The AGV: Inter-sensor communication.

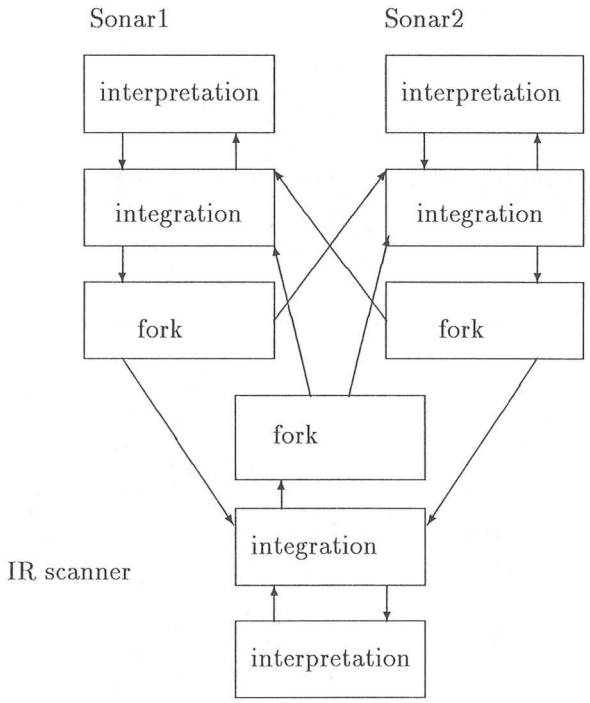

Figure 10. The AGV: Part of the network showing sonar-to-sonar and sonar-to-IR communication paths.

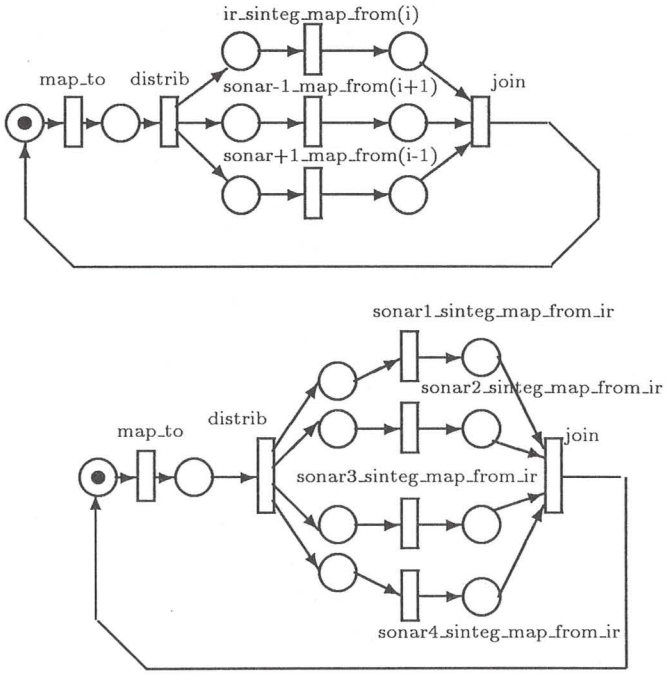

Figure 11. Sonar (up) and IR (down) sensors' fork processes.

With four sensors trying to access IR sensors' database, each such operation taking T_{ir_locke}, communication bottleneck can occur. Therefore, some modifications of this sensors' integration module have been introduced in order to find whether an optimal communication scheme exists and the cycle tradeoffs between IR and sonar sensors. Following variations of IR sensor have been implemented:

- first–come–first–served database access
- priority in database access – IR sensor is privileged

All sensors share one common processor resource – *GLOBPROC*. Initial number of tokens in this place regulates the amount of simultaneous world model update operations.

4.2.2. Sample simulation results

Through the simulation of the token-game in the model described above we try to:

- Extract timing information, i.e. find the average cycle times for integration circuits – in effect finding the rate at which commands are sent to actuators by different layers.
- Find the optimal value for common resources, here, the number of processors.
- Investigate the speed tradeoffs between different sensors working in varied communication schemes (with/without message priorities).
- Find the optimal process scheduling method.

The following values of time parameters were used in this model:

trans	*sonar*	*IR*
measure	5,15,30	100
datastream	0.01	0.1
procdata	0.01	0.1
checkdata	0.01	0.1
test_ok	0.0	0.0
test_req	0.0	0.0
assertion	1	1
map_to	1	20/5
external WM update	10	20
internal WM update	2–30	5–100

The firing time of the *map_to* transition is calculated as follows: we assume that IR sensor holds a map in the form of a 150 x 150 occupancy grid; this map is propagated to *forkir* process – 150 x 150 x 8 bits through 10Mb/s datalink

take 20ms to transmit. The map is divided into four segments in fork module and propagated to sonars. It takes 5ms to transmit a quarter of the map.

Architecture with message priorities. In this configuration IR sensor has a higher priority when updating its world model, which can be accessed by sensors interpretation module whenever a free processor exists in a common pool.

First the cycle time of sonar module – that is, an average time between firings of *emit_reflect* transition, was measured as a function of the global number of processors allocated and the time of updating IR sensor's database – $T_{ir\_sinteg\_locki}$. The results are shown in Figure 12. For the number of processors greater than one the cycle time of IR sensor is a linear function of $T_{ir\_sinteg\_locki}$ in this configuration, regardless of the number of processors – see Figure 12. This is a direct consequence of IR sensor's priority in updating its world model.

The cycle time of IR sensor is roughly equal to

$$T_{ir\_sinterp\_emit\_reflect} + T_{ir\_sinteg\_locki} + T_{sinteg\_map\_to}$$

in the priority configuration. The shortest possible cycle is :

$$max(T_{ir\_sinterp\_emit\_reflect}, T_{ir\_sinteg\_locki} + T_{sinteg\_map\_to})$$

because interpretation and integration modules can operate concurrently, however, with forced priorities this two sub–modules will rather operate in serial: *sinterp_emit_reflect* $\rightarrow$ *sinterp_assertion* $\rightarrow$ *sinteg_locki* $\rightarrow$ *sinteg_unlocki* $\rightarrow$ *sinteg_map_to* $\rightarrow$ *sinterp_ready* ... with no events overlapping in time.

In the second series of simulations the sonar cycle time was measured as a function of the number of processors and the time of external update of IR sensor's database – $T_{ir\_sinteg\_locke}$. The results are shown in Figure 13. The cycle time of IR sensor was also measured – Figure 13, if more than 2 processors are used it is almost constant – due to higher priority of internal world model updates. For 1 and 2 processors a considerable increase of this interval can be noticed. It can be concluded that with the present system configuration the optimal number of processors is 3.

The simulations were repeated for different sonar *emit_reflect* times: for fast sonar measurement – 5ms and slow sonar measurement – 30ms, the results, very similar to those obtained for 15ms measurement, proved that event times where resource sharing occurs have the greatest influence on system's performance, while the timing of other events is of lesser importance.

Architecture without message priorities. In this version the IR sensor has no priority over sonar sensors in updating its own world model data base.

Comparing Figures 12 with 14 we can conclude that, generally both IR and sonar sensors operate faster when no priorities are used. This is more apparent for the sonar because it has now a fairer chance of accessing IR sensors' database. The latter operates faster because its modules "desynchronise" –

various operations start to overlap in time.

Figure 15 shows that abandoning priorities is not always a correct choice: for short $T_{ir\_sinteg\_locke}$ times IR sensor considerably slowed down external *lock* events coming from sonars.

4.3. Figure of Merit

To provide an universal judgment of a multisensor systems' performance we propose to calculate a *Figure of Merit* based on individual sensor's throughput and response time. Sensor's throughput can be expressed as:

$$Th = \frac{nI}{\Delta t}$$

where n is the number of data points collected and processed in time Δt and I is the information quantity in one data point. Throughput is scaled with figure expressing the importance of minimising the response time. *Figure of Merit* – FM for an individual sensor is:

$$FM = \frac{t_0}{\Delta t} Th = \frac{t_0 nI}{\Delta t^2}$$

where t_0 is some desired maximum response time. FM for k sensors is therefore:

$$FM = \sum_{i=1}^{k} \frac{t_0 n_k I_k}{\Delta t_k^2}$$

It must be noted that this is an approximation of system's performance, since neither inner/inter sensor integration of data nor the quality of individual sensor's decision is taken into account.

For the example system evaluated here FM is:

$$FM = 4\frac{n_S I_S}{\Delta t_S^2} + \frac{n_{IR} I_{IR}}{\Delta t_{IR}^2}$$

Sonar and IR parameters follow:

	sonar	*IR*
n	3	100
$I[bits]$	6	8

The application of the Figure of Merit will be discussed in the following section.

5. Discussion

System verification was the first of the goals of the modelling process. The following conclusions can be drawn from the model design and computer experiment phases:

- The very design stage provides valuable insights into system functioning, deadlock/livelock and performance related issues.
- The simulation provides unlimited degree of insight into the dynamics of the system. Statistical parameters such as average process time, average synchronisation/resource wait time, average token flow are available on demand.
- Detailed step-by-step analysis of the token game can reveal an unexpected and/or undesired behaviour of the system.
- A Petri net graph is easier to understand than a corresponding programme written, for example, in *CSP*. Serious design flaws can be easily found at the early stage of the simulation.

The main goal of the simulation stage was to obtain dependencies between data-processing algorithms execution times and vehicles reaction times. As was explained in section 3., sensor cycle time is equal to vehicle reaction time to an event detected by a given sensor (as each logical sensor module tries to drive the vehicle independently). Reaction times were determined – for both sonar and IR sensors – as a function of time taken by sensors world model updating algorithm. Each simulation was performed for 1 to 5 processor pool, and for two types of process scheduling method. World model update event can be triggered internally (i.e. by information coming from the interpretation module of a particular sensor) or externally (by information coming from other sensors), thus two types of updating events, with different execution times, were modeled.

Computer experiments showed that the most critical processing times were associated with IR sensors database updating algorithms. These processing times influence not only the speed of operation of the IR sensor, but also the speeds of all other sensors in the system - see Figures 12 to 14. This effect is attributed to the central role that the IR sensor plays in the communication structure. The influence of the scheduling method on reaction times is significant: higher priority given to IR sensor (the most computation intensive one) may result in heavy slowdown of other sensors.

It can be also concluded that the optimal number of processors in the system is 3, larger processor pool gives only small improvement in the reaction times. In certain cases a larger number of processors can cause a communication bottleneck which slows down the system, see Figure 15.

Reaction times of individual sensors do not provide a full picture of systems performance, that is why a *Figure of Merit* has been introduced. Figure of Merit has been calculated for both scheduling models described above, and is plotted in Figure 16 as a function of IR sensors internal world model update time (the most time consuming algorithm in the system), and as a function of the number

of available processors. A systematic, but not very significant improvement of performance can be observed for the number of processors greater than two.

FM for a three – processor system for both scheduling methods has been plotted on the same graph: Figure 17. We can conclude that the choice of a scheduling method should depend on the speed of the algorithm used to process and integrate data from IR sensor's measurements. For a slow (that is significantly slower than equivalent sonar sensors algorithm) IR sensor database updating algorithm, no priority scheduling should be chosen.

Topics which are worth further interest and research:

- The net model, and of course the simulator, can be augmented by variable/expression mechanism by assigning expressions to transitions, boolean guards to inhibit firings and vectors of variables to tokens, as was shown in Golz, Reisig (1985).
- The current model can be significantly expanded, for example a self-test procedure was introduced but was not fully exploited in the simulation stage. An implementation of a second processor pool servicing interpretation modules and a model of a vision module is worth consideration.
- The *Figure of Merit* should take into account "behavioural quality", not only data throughput and reaction times.
- The influence of environment on systems behaviour could be taken into account.

6. Summary

A modelling strategy for parallel systems has been proposed. It was based on the idea of processes able to exchange data and synchronise with other processes. Petri nets with time, probability and priority were used as a modelling tool. It was shown how process exclusion can be combined with timed events and still preserve the liveness of the net.

A 'case study' application was presented. It was shown how a whole AGV architecture can be implemented and simulated with a Petri Net model. The design was based on a set of universal sensors able to exchange information with each other and share common resources. Internally a sensor was represented as a group of sequential, cyclic processes synchronising with each other the information exchange events.

In the course of the simulation, system modules' reaction times were estimated, most time-critical module was found, and its influence on the rest of the system evaluated. Optimal number of processors was also determined. A Figure of Merit, giving objective judgment about systems performance was proposed, and evaluated as a function of critical algorithm time, available processing power and the scheduling method used.

Acknowledgment

I would like to express my gratitude to Dr. P. Probert from Univerity of Oxford Robotics Research Group for guidance in this work.

References

BLINOWSKI G., PROBERT P.J. (1992) Petri Nets and Performance Evaluation in Robotics. University of Oxford, Department of Engineering Science Reports; Report No. OUEL 1843/92.

BROOKS R.A. (1985) A layered intelligent system for a mobile robot. In: *Third Intl. Symp. Robotics Research.*

DIJAN D., PROBERT P.J. (1990) Case Study in Formal Specification for Distributed Sensor Integration. University of Oxford, Department of Engineering Science Reports; Report No. OUEL 1844/90.

FREEDMAN P. (1990) Time, Petri Nets, and Robotics. *IEEE Transactions on Robotics and Automotion*, Vol. **7**, No. **4**, August 1990.

GOLTZ U., REISIG W. (1985) CSP-Programmes as Nets with Individual Tokens. *LNCS* 188. Springer, Heidelberg, pp. 169-196.

HOARE C. (1985) *Communicating Sequential Processes.* Prentice Hall.

HU H., LEONARD J., PROBERT P., BRADY M., DURRANT-WHYTE H., RAO B.S.Y. (1989) Sensor-based control of AGVs. *IEE Computing and Control.*

LAUTENBACH K. (1987) Linear Algebraic Techniques for Place/Transition Nets. *LNCS* 254. Springer, Heidelberg, pp. 142-167.

MERCERON A. (1987) Fair Processes. *LNCS* 266, pp. 180–195.

MERLIN P. (1976) A methodology for design and implementation of communication protocols. *Proc. IEEE Trans Commun.*, pp. 614-621.

RAMCHANDANI C. (1974) Analysis of asynchronous concurrent systems by Petri nets. Project MAC, TR-120, M.I.T., Cambridge, MA.

REISIG W. (1985) *Petri Nets. An Introduction.* Springer.

REISIG W. (1987) Place/Transition Systems. *LNCS* 254. Springer, Heidelberg, pp. 142-167.

SCIOMACHEN A., GROTZINGER S., NEMEC B., (1990) Petri Net–Based Emulation for a Highly Concurrent Pick–and–Place Machine. *IEEE Trans. on Robotics and Automation.* Vol. **6**, No. **2**, Apr 1990.

SIFAKIS J. (1980) Performance Evaluation of Systems using Nets, *LNCS* v. 84, pp. 307–319.

STARKE P.H. (1989) Some properties of Timed Nets under the earliest firing rule. *LNCS* 424, pp. 418–432. Distributed Computing, Paris 1981, *IEEE*, No. 81 CH 1591–7.

WANG F.Y. ET. AL (1991) A Petri-Net Coordination Model for an Intelligent Mobile Robot. *IEEE Trans. on Systems, Man and Cybernetics*, Vol. **21**, No. **4**.

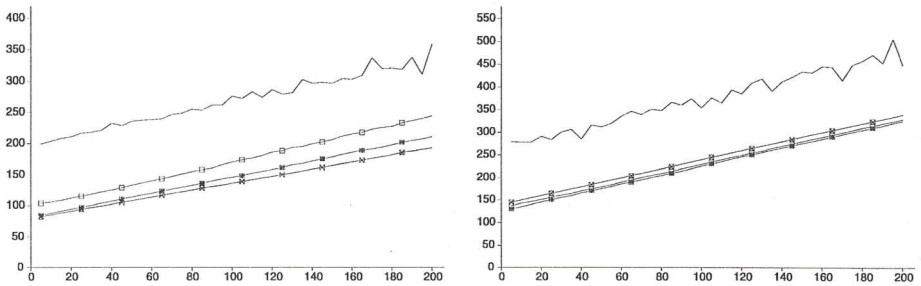

Figure 12. Left – Sonar sensor's cycle time as a function of $T_{ir\_sinteg\_locki}$. Right – IR sensor's cycle time as a function of $T_{ir\_sinteg\_locki}$. In this, and in all the subsequent diagrams in this chapter, smooth line refers to a system with one processor, line with hollow squares to 2 processor system, filled squares – 3, two triangles – 4, hourglasses – 5.

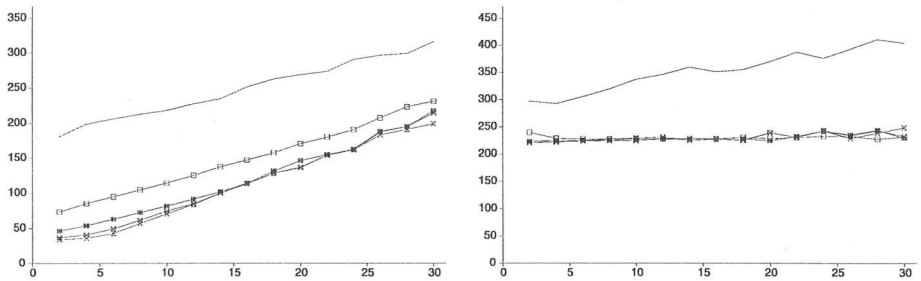

Figure 13. Left – Sonar sensor's cycle time as a function of $T_{ir\_sinteg\_locke}$. Right – IR sensor's cycle time as a function of $T_{ir\_sinteg\_locke}$.

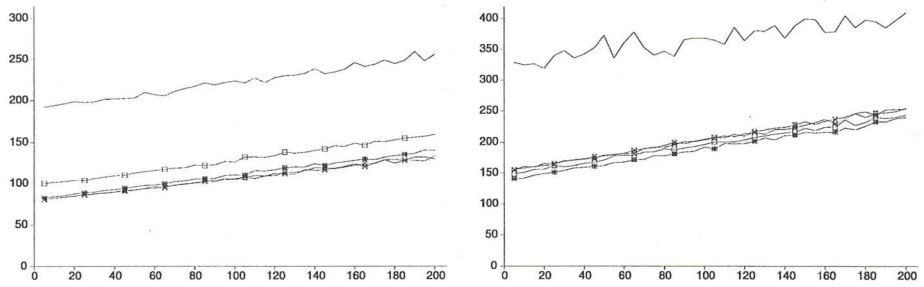

Figure 14. Left – No priority configuration: Sonar sensor's cycle time as a function of $T_{ir\_sinteg\_locki}$. Right – No priority configuration: IR sensor's cycle time as a function of $T_{ir\_sinteg\_locki}$.

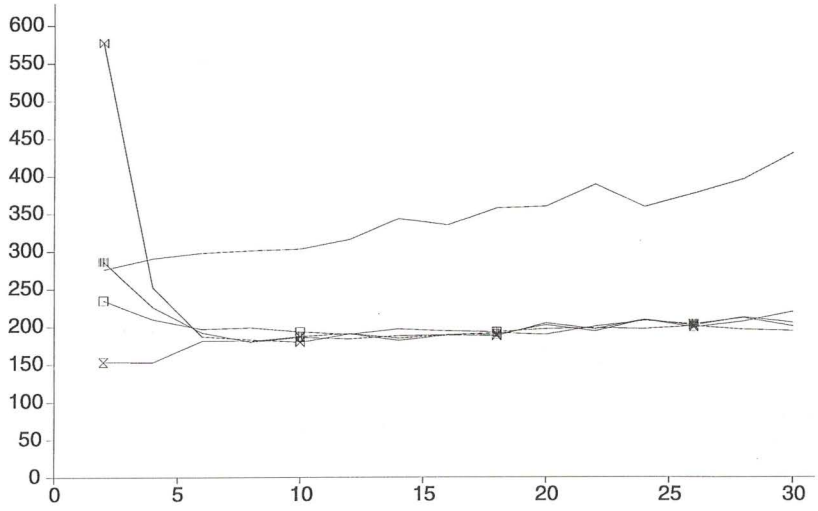

Figure 15. No priority configuration: IR sensor's cycle time as a function of $T_{ir\_sinteg\_locke}$.

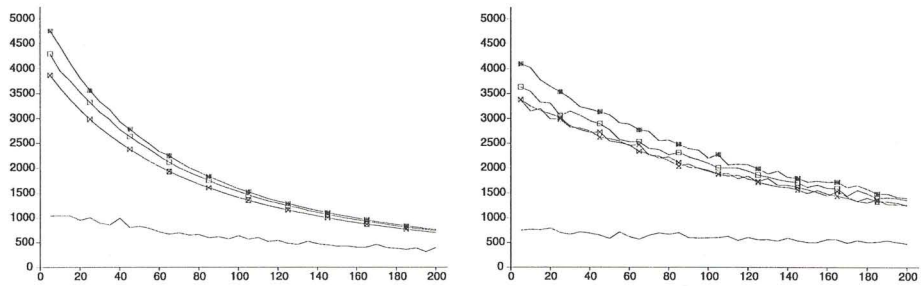

Figure 16. *Figure of Merit* for priority (left) and no priority (right) architectures as a function of $T_{ir\_sinteg\_locki}$.

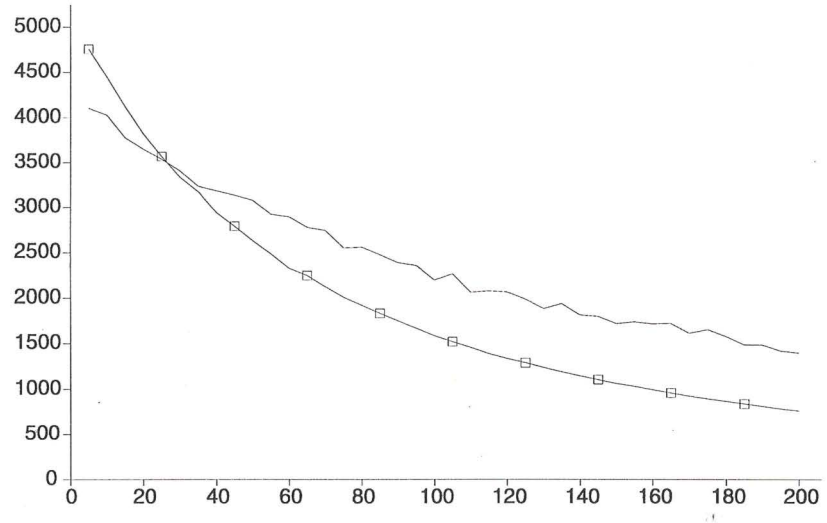

Figure 17. *Figure of Merit* for 3 processor priority (dotted line) and no priority architectures as a function of $T_{ir\_sinteg\_locki}$.