IEEE International Workshop on Intelligent Motion Control
Boğaziçi University, Istanbul, 20-22 August 1990.
Edited by Okyay KAYNAK. IEEE Cat. No: 90TH0272-5

An Expert System for Designing and Supervising a Discrete Event Model

A. Cois, A. Fanni * A. Giua **

* Istituto di Elettrotecnica — Università di Cagliari
Piazza D'Armi, 2 — 09100 Cagliari (Italy)
email: lb03@vaxca2.infn.it
** Electrical, Computer, and Systems Engineering Department
Rensselaer Polytechnic Institute — Troy, New York 12180 (USA)
email: giua@ecse.rpi.edu

Abstract

We present a model based expert system for designing and supervising a traffic-light process. The model of the process is based on place/transition Petri nets. The supervision is implemented through a production rule system. The architecture of the system may be applied to other control problems that require to introduce in the supervisor a distributed intelligence, not fitted to an algorithmic formalism.

1 Introduction

Traffic-light processes are an example of discrete (in time and state-space), asynchronous (not necessarily clock-driven) and non-deterministic systems. They belong to the class of *Discrete Event Systems* (DES's), that is dynamical systems with a discrete state space and whose evolution depends on the occurrence – at irregular and unforeseen intervals – of discrete events.

The purpose of this paper is twofold: we show how a model of a traffic-light process, capable of capturing its logical behavior, may be automatically designed using place/transition Petri Nets (PN); we present an architecture for a mixed-mode expert-system in which the knowledge of the model, combined with additional heuristic knowledge, is used to control the system.

A modelling formalism, based on linguistic operators, has been developed. The model is designed in a bottom-up fashion, combining the modules that represent the interacting subsystems in a way that reflects how they are connected. This approach, based on the supervisor design techniques of Ramadge and Wonham [5], enables the model to capture the essential features of the process and gives us the means to ensure that the model has the desired properties (liveness) and that its behavior is correct (safeness). The PN model is created by an off-line module of the system given the knowledge of the traffic-lights present in the process and of the mutual constraints between the flows they control. The model contains all the features of the process but does not contain the knowledge necessary to ensure an optimal supervision since important issues – such as the timing of the process – depend, as will be shown, on external additional knowledge.

The design and the analysis of the model is only the first step in the study of a DES. The final goal is that of modifying, by control actions, the behavior of the system so that it satisfies given criteria. The control actions often depend on random or statistical patterns and on empirical data. Thus trying to express them in a mathematical framework is a hard task. We use a knowledge-based approach that offers the possibility of introducing heuristic knowledge in the control system [3]. The heuristics are expressed as production rules; they act as a supervisor capable of driving the trajectories of the controlled system in response to different demands. The supervisor will be able to resolve, for instance, timing problems compiling the production rules, the knowledge contained in a database (where information on the traffic flows of the particular process during different hours of the day has been collected) and the knowledge coming from sensors.

The system has been implemented in Prolog. Logic programming is a versatile tool that allows with little effort the formalization of both procedural and declarative knowledge.

Although this example has solely didactic and explicative nature, we also present an architecture for a mixed mode expert systems, i.e., a model based system but controlled by heuristics, that can be used in a wide range of problems. In fact the same approach may be fruitful in all those control problems that require to introduce in the control system a distributed intelligence, not fitted to an algorithmic formalism. Mathematical techniques such as the solution of differential equations, probabilistic analysis, procedural programming, etc., are of remarkable importance, but not all forms of knowledge may be represented making use of them. Thus the main emphasis of this work is on the need in control theory of new representations for human-oriented systems.

2 The Proposed Architecture

The architecture we propose is depicted in Figure 1. The dashed part is solely used during the implementation of the system: it generates the model of the process to be controlled; this model does not change during the normal execution. The part drawn with a continuous line contains the modules that work on-line and controls the evolution of the process.

The off-line module (i.e., the model generator) accesses to two Knowledge bases: the one containing information on the particular process (topology of the crossroad, number of flows, etc.) and the one containing general knowledge for designing a model of a traffic-light process (model of a traffic-light, model of the specifications, operators to combine them, etc.). From these information this module produces as output a PN structure that will be the model of the process studied. It can be show [1] that the model has the desired property of being controllable and that it is non-blocking (we will not consider these issues in this paper).

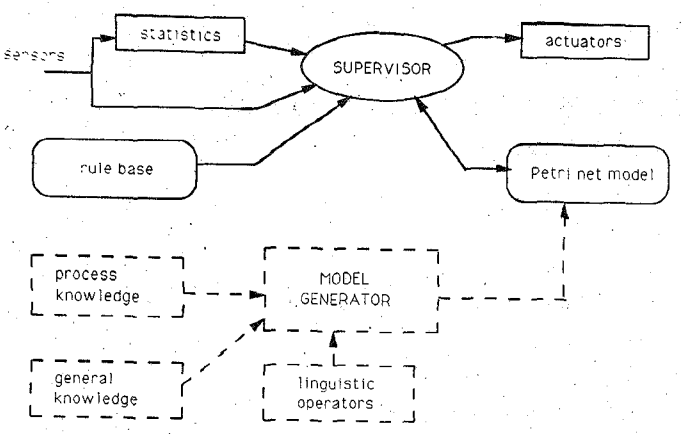

Figure 1: Architecture of the system.

It may be worth noting that the generation of the model is a completely automatic procedure, computationally easy and that, due to the use of a PN formalism, the complexity of the model is greatly reduced with respect to, say, the models that may be obtained using finite state automata for the representation of DES's.

The supervisor has in input:

1. The traffic statistics, collected in a preliminary phase. The statistics could be updated by the system itself, making use of the sensors to determine at a given instant the traffic flows.

2. The data coming from sensors. The data contain information on the traffic flows and on the occurrence of anomalous situations such as *traffic jam* or *presence of high-priority vehicles* (ambulances, police cars, etc.).

3. A set of *if-then* rules that describe the possible control actions.

The supervisor communicates with the PN model. Here the information flows in both ways, as in a closed-loop control system. The model informs the supervisor on its state and on the actions that may be legally performed. The supervisor determines the optimal action and updates the model.

Finally the supervisor drives the actuators that physically make the process change of state signaling or halting the traffic-lights. The actuators also manage simple and non-flexible operations such as the intermediate *yellow-red* transition and its timing.

3 The Model

3.1 Discrete Event Systems and Models

A Discrete Event System is a dynamic system with a discrete state space and piecewise constant state trajectories; the time instant at which state transitions occur, as well as the actual transitions, will in general be unpredictable.

The state transitions of a DES are called *events* and may be labeled with the elements of some alphabet Σ. These labels usually indicate the physical phenomenon that caused the change in state. For example, in a manufacturing environment typical event labels are *"machine 1 starts working on part A"*, *"machine 1 breaks down"*, etc.

In modeling a DES a common simplifying assumption is to ignore the times of occurrence and consider only the order in which they occur [2]. This simplification is justified when the model is to be used to study properties of the event dynamics that are independent of specific timing assumptions. This kind of models are called *logical models*. Logical models have been successfully used to study the qualitative properties of DES's in a variety of applications. The behavior of these systems is described by the sequences of discrete events or *traces* that they generate. Let Σ be the set of discrete events labels and let Σ^* be the set of all finite sequences of events in Σ (including the empty trace λ). Thus a possible behavior of the system may be represented by

$$\sigma = s_1 s_2 s_3 \dots$$

where $\sigma \in \Sigma^*$ and $s_i \in \Sigma$, $i = 1, 2, \dots$

There are two main assumptions here:

1. we are interested in the *order* in which the events occur but not in the *real time* at which they occur;

2. an event is *atomic*, i.e., it is considered to occur in a single step, whereas in the real system it may be implemented as a sequence of instructions.

In a logical model, the behavior of a given system is thus given by a subset language $L \subset \Sigma^*$, consisting of all the traces (or *strings*) that the system can generate. In most systems of interest, L will be an infinite set so that it cannot be given simply by listing all the traces. We will use a Petri net structure to model DES's.

3.2 Petri Nets and Petri Net Languages

A Petri net (PN) structure is a directed graph $S = (P, T, I, O)$ where P and T are disjoint sets of nodes. More precisely:

$P = \{p_1, \dots, p_n\}$ is a finite set of *places*;

$T = \{t_1, \dots, t_m\}$ is a finite set of *transitions*;

$I : P \times T \to \mathcal{N}$ is the *input function* that defines the set of directed arcs from P to T;

$O : T \times P \to \mathcal{N}$ is the *output function* that defines the set of directed arcs from T to P;

The state of the net is specified by the *marking*, i.e., a function assigning to each place a non negative integer number of tokens. A marking is denoted by μ, an n-vector where n is the total number of places. The p-th component of μ, denoted by $\mu(p)$, is the number of tokens in the place p. A transition t is *enabled* if each input place p of t is marked with at least $I(p,t)$ tokens, where $I(p,t)$ is the number of arcs from p to t. An enabled transition t may fire, yielding a new marking μ' with

$$\mu'(p) = \mu(p) + I(p,t) - O(t,p)$$

i.e., the firing of t removes $I(p,t)$ tokens from the each input place p of T and adds $O(t,p)$ tokens to each output place p of t. It is often the case that a net offers *choice* as to which transition should fire, i.e., more than one transition may be enabled by a given marking. In the following we will show how this conflict may be solved with additional knowledge external to the net.

A PN may be considered as a language generator. Let us consider a PN structure on which the following have been defined:

1. A *labeling function* $\sigma : T \to \Sigma$, assigning to each transition a symbol of the alphabet Σ.

2. An *initial marking* μ_0.

3. A finite set of *final markings* F.

The set of strings generated by all possible firing sequences starting from the initial marking μ_0 and terminating in a marking

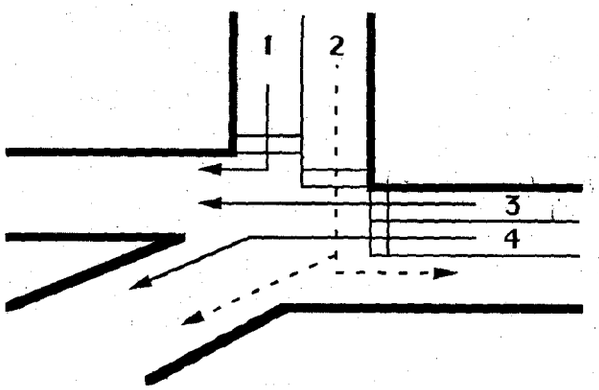

Figure 2: An example of a traffic-light process.

$\mu \in F$ defines a formal language. Hence a DES may be represented by a 4-tuple:

$$\mathbf{G} = (S, \sigma, \mu_0, F)$$

whose language $L(\mathbf{G})$ consists of all the traces (or *strings*) that express the system's behavior.

Without any loss of generality, we could consider DES in standard form. Here the initial marking μ_0 consists of single token in a place p_s called *starting place* and F contains only one marking μ_f that consists of one single token in a place p_f called *final place*. There will also be an *initial transition* and a finite set of *final transitions* labeled on the empty string λ (see [1,4]) as in the examples we will show.

3.3 Synchronization Operator

Let $\Sigma_1, \ldots, \Sigma_n$ be alphabets, not necessarily disjoint. Consider the alphabet $\Sigma = \Sigma_1 \cup \ldots \cup \Sigma_n$ and let $w \in \Sigma^*$. We define $P_i(w)$, the **projection** of w over Σ_i, as the string obtained by w deleting all the symbols not belonging to Σ_i. Formally the projection operator is a mapping

$$P_i : \Sigma^* \rightarrow \Sigma_i^*$$

Let us now consider the languages $L_1, \ldots, L_n$ over alphabets respectively $\Sigma_1, \ldots, \Sigma_n$. Let $\Sigma = \Sigma_1 \cup \ldots \cup \Sigma_n$. We define

$$L = L_1 \parallel \ldots \parallel L_n$$

the **synchronization** of $L_1, \ldots, L_n$, i.e., the language over Σ defined as

$$L = \{w \in \Sigma^* \mid P_i(w) = w_i \in L_i, \;\; i = 1, \ldots, n\}$$

Given two (or more) PN modules $\mathbf{G}_1$ and $\mathbf{G}_2$, a simple algorithm (see [1]) constructs the net $\mathbf{G}$ generating the language $L(\mathbf{G}) = L(\mathbf{G}_1) \parallel L(\mathbf{G}_2)$. To gain a better understanding of the meaning of the system $\mathbf{G}$ obtained by synchronization let us consider two examples (see also [5]).

1. $\mathbf{G}_1$ and $\mathbf{G}_2$ are two machines generating the languages $L(\mathbf{G}_1) = (a_1(b_1^* \cup c_1^*))^*$ and $L(\mathbf{G}_2) = (a_2(b_2^* \cup c_2^*))^*$. The synchronized system will generate all the possible interleaving of strings in $L(\mathbf{G}_1)$ and in $L(\mathbf{G}_2)$, i.e., it represents the concurrent execution of the two systems. There is no interaction between the two systems, since their alphabets are disjoint.

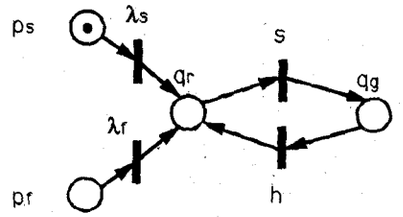

Figure 3: Petri net model of a traffic-light.

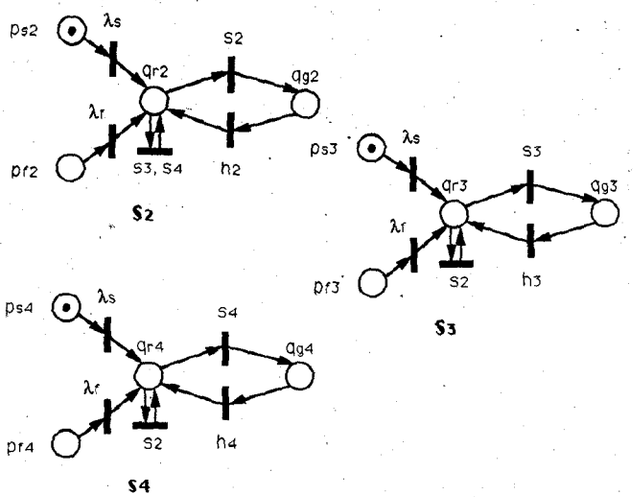

Figure 4: Specifications for the example in Figure 2.

2. $\mathbf{G}_1$ is a machine generating $L(\mathbf{G}_1) = (a(b^* \cup c^*))^*$. $\mathbf{G}_2$ represents a specification we want to enforce, stating that the transitions b and c should occur alternatively. In this case $L(\mathbf{G}_2) = (bc)^*$. $\mathbf{G}$ will generate the language $L(\mathbf{G}) = (abac)^*$, i.e. the set of all strings belonging to $L(\mathbf{G}_1)$ and such that the transitions b and c occur alternatively.

It is clear that given m DES's $\mathbf{G}_1, \ldots, \mathbf{G}_m$ working concurrently and n legal specifications represented by their language generators $\mathbf{S}_1, \ldots, \mathbf{S}_n$, the synchronization of the $m+n$ nets represents the system with the minimally restricted behavior that at the same models the joint execution of the m sub-systems and satisfies all the n constraints.

3.4 The Traffic-Light Application

In our domain of application we have two kinds of DES's. As an example, let us consider the crossroad in Figure 2.

1. The traffic-lights will be modelled as in Figure 3. Here the place q_{ri} (q_{gi}) of $\mathbf{G}_i$ contains a token when the light i is *red* (*green*). The two transitions of $\mathbf{G}_i$ (a part from initial and final transition) are labeled s_i (signal) and h_i (halt). We have one light for each flow.

2. The specifications, shown in Figure 4, capture the mutual exclusion constraints between different flows. In the example, $\mathbf{S}_2$ means that the lights $\mathbf{G}_3$, $\mathbf{G}_4$ can switch from red to green only if $\mathbf{G}_2$ is red. Here again we have one constraint for each flow (a part the flows which have exclusion with no other).

Following the general technique previously discussed, the overall model of the crossroad may be obtained by synchronization

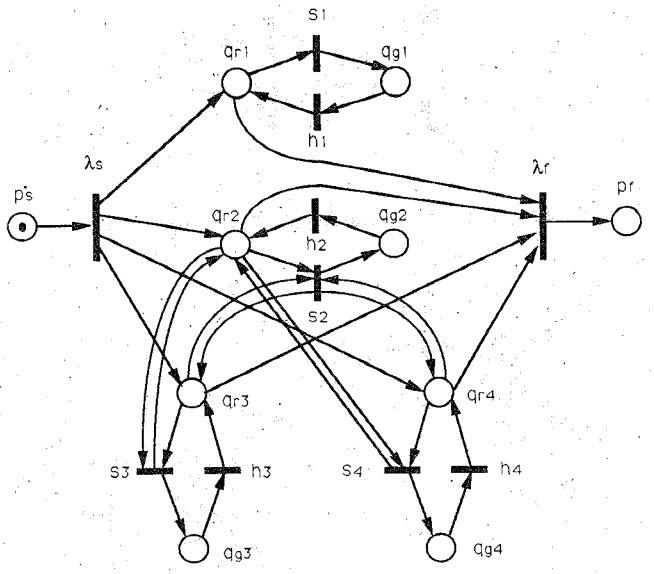

Figure 5: Model of the process in Figure 2.

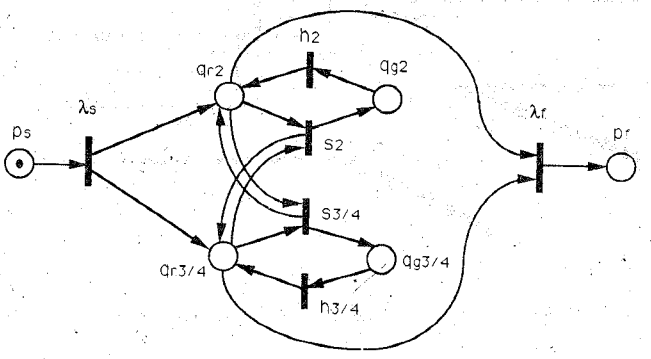

Figure 6: Reduced model of the process in Figure 2.

of the G's and S's. We note, however, that each specification $\mathbf{S}_i$ contains already all the information about the behavior of the i-th light and it is not necessary to include $\mathbf{G}_i$ if $\mathbf{S}_i$ is present in the model. Formally let Σ_i be the alphabet labeling $\mathbf{G}_i$. Since

$$P_i(L(\mathbf{S}_i)) = L(\mathbf{G}_i)$$

we have that

$$L(\mathbf{S}_i) \parallel L(\mathbf{G}_i) = L(\mathbf{S}_i)$$

Then the synchronization will be performed solely with the specification nets (and eventually with the nets of the unconstrained flows).

For the example of Figure 2 the synchronized net is shown in Figure 5.

3.5 Model Reduction

A module of the expert system is concerned with the design of the model, starting with the information on flows and mutual exclusions among them. The model may be subsequently refined as follows:

1. Any unconstrained flow can be eliminated, i.e., there is no need of a light for it.
2. When two or more flows have the same mutual exclusions and do not intersect each other they can be assigned to the same light.

The reduced net for the example of Figure 2 is shown in Figure 6. Nevertheless the supervisor is capable of handling in a optimal way even non-reduced models, as we will show in the next section.

4 The Distributed Supervisor

Through the PN model, the system monitors the process, i.e., it can retrieve a description of the present state and of the legal next states. Nevertheless the model gives the most ample behavior of the process satisfying the specifications; it has no knowledge of which transition (should more than one be enabled at the same time) should fire in order to optimize the process. This additional knowledge has been implemented in a set of *if-then* rules. The use of a rule based system offers two main advantages with respect to a rigid algorithmic decision formalism: *flexibility*, i.e., the system may easily adapt itself to different situations; *modularity*, i.e., the possibility of changing the overall behavior of the system by simply introducing new rules.

Once the model has been designed, the rule based system allows to supervise the process in the normal operating mode and in case particular situations occur. In the normal operating mode a priority value n_i is assigned to each light; this value is proportional to the flow the light controls. The supervisor reads at discrete intervals ΔT the state of the net, i.e., the lists S and H of all the *signal* and *halt* transitions that may fire. Among all of them it chooses the optimal one(s) that should fire, according to rules such as:

1. s_i should fire if its priority is greater among the other s transitions that may fire.
2. h_i should fire if a traffic jam is detected.
3. h_i should fire if a time greater than $n_i \Delta T$ has elapsed since the firing of the corresponding s_i transition.
4. h_i should fire if a high-priority vehicle is approaching on a lane controlled by light different from $\mathbf{G}_i$.

5. s_i should fire if a high-priority vehicle is approaching on a lane controlled by the light G_i.

6. s_i should fire if the the flow controlled by the light G_i does not intersect any other flow already enabled.

and so on. Once the supervisor has determined the list of transitions that should fire, it passes this information to the actuators, so that they may be executed, and to the PN model, so that its state may be updated. The chaining of the rules is executed by the inferential engine of Prolog.

5 An Applicative Example

As a concrete case we may consider an example developed for the process in Figure 2 for a normal operation mode. Here the model is not reduced in order to show better how the supervisor works.

Let us assume that the priorities n_i of the four flows are:

flow	n_i
1	0
2	2
3	3
4	1

and that all the lights are presently in the red state. The supervisor retrieves from the model the lists of the transitions that may fire in accordance to the constraints:

$$S = [1,2,3,4] \qquad H = [\,]$$

Transition s_3 should fire in accordance to rule 1. The state of the model is updated but no command is yet sent to the actuators. The transitions that may fire now are:

$$S = [1,4] \qquad H = [3]$$

Transition s_4 should fire (rule 1). The updated situation gives:

$$S = [1] \qquad H = [3,4]$$

Transition s_1 should fire in accordance to rule 6. We have:

$$S = [\,] \qquad H = [1,3,4]$$

No other transition should fire, since the preconditions of rule 3 are not satisfied.

The supervisor passes the information to the actuators that execute the required actions. After a time interval ΔT the cycle is repeated. The behavior of the process will be:

interval	*lights on*
1	1,3,4
2	1,3,4
3	1,3,4
4	1,2
5	1,2
6	1,3,4
...	...

that continues until the situation changes (emergency situation, change of priorities at a different time of the day, etc.).

6 Conclusion and Future Research

We have presented a model based expert system for designing and supervising a traffic-light process.

The model of the process is based on place/transition Petri nets. We have developed a modelling formalism, based on linguistic operators. The model is designed in a bottom-up fashion, combining the modules that represent the interacting subsystems.

The supervisor, implemented through a production rule system, is capable of driving the trajectories of the process. It ensures a flexible mode of operation, i.e., it is capable of optimizing the behavior of the process in response to different configurations of the traffic flows that may occur during the day.

Future developments include the study of the interactions of different processes, such as a sequence of cross-roads on a street. The modelling formalism should be extended to a network of inter-related processes and different heuristics for a global optimization of the network should be considered.

References

[1] A. Giua "Petri Net Languages for the Control of Discrete Event Systems", Troy, New York: RPI, Dept. Electrical, Computer, and Systems Engineering, MS Thesis (1990).

[2] K. Inan, P. Varaiya, "Finitely Recursive Process Models for Discrete Event Systems", *IEEE Trans. on Automatic Control*, Vol. AC-33, No. 7, pp. 626–639, July, 1988.

[3] J. Martinez, P.R. Muro, M. Silva, S.F. Smith, J.L. Villaroel, "Merging Artificial Intelligence Techniques and Petri Nets for Real Time Scheduling and Control of Production Systems", Technical Report GISI-1/88, Dpto. Ingenieria de Eléctrica e Informática, Universidad de Zaragoza (Spain), January, 1988.

[4] J.L. Peterson, "Petri Net Theory and the Modeling of Systems", Prentice-Hall, 1981.

[5] P.J. Ramadge, W.M. Wonham, "The Control of Discrete Event Systems", *Proceedings IEEE*, Vol. PROC-77, No. 1, pp. 81–98, January, 1989.