Petri nets for control and monitoring: specification, verification, implementation

R. Valette

LAAS-CNRS, 7 Av. du Col. Roche, F-31077 Toulouse Cedex
e-mail robert@laas.fr

Abstract

The purpose of this paper is to illustrate the benefits of a Petri net based approach for control and monitoring of event-driven operations in process systems. A Petri net based specification contains structural information which is absent from a simple state transition graph. The verification scheme is not necessarily based on a state enumeration and many results can be obtained by analysing particular subnets. In particular the use of redundant places and siphons is illustrated. Finally the implementation by means of the token player approach can be easily complemented to guarantee monitoring functions.

1 Introduction

The control issue of discrete-event chemical processes is complex because of its hybrid nature. Various approaches have been proposed or are under development [6, 9, 14, 15, 20, 22], many of them are Petri net based.

The purpose of this paper is not to present a survey of Petri net theory. A very good introduction can be found in [16], for more detail the reader can refer to [8] and for the application to discrete manufacturing systems to [21]. The objective is to illustrate the benefit and feasibility of a Petri net based approach for the specification, the verification and the implementation of the discrete control and monitoring of process systems.

The organisation of the paper is the following. After a brief introduction to Petri nets, the advantages of specifying a process system under the form of a collection of sequential processes is illustrated by a simple example. Actually it is a good thing to have structural and behavioural information encapsulated in the same model. In particular, it is possible to consider the number of batches as a parameter although the behaviour of the system may significantly differ after an increase of this number.

The next section addresses the issue of verification. Before employing a model for performance evaluation or for production plan generation it is necessary to verify it in

1This paper will be presented at the workshop “Analysis and Design of Event-Driven Operations in Process Systems, Imperial College, Centre for Process Systems Engineering, London, 10-11 April 1995
order to be confident that the model actually depicts the physical system. When a Petri
net based description is used, the designer meets the following difficulty: there is no simple
necessary and sufficient condition to prove the correctness of a Petri net. However, a lot
of results of Petri net theory can be used to search for inconsistencies in the specification
and in consequence to detect a lot of errors. By means of a simple example, the fact that
some generic results (for instance results which do not depend on the batch number) can
be obtained for Petri nets representing batch systems.

The last section concerns the implementation of supervisory control and monitoring
functions. It is shown that the Petri net approach favours the real-time detection of many
abnormal behaviours. Indeed it is very easy to check, each time the discrete state is
updated, that the new state is consistent with the past one i.e. that this state change is
legal (corresponding to an enabled transition).

2 A brief introduction to Petri nets

Typically, discrete event systems are represented by automata or any kind of state transi-
tion graphs. When the system consists of a large number of sequential processes (a
sequential process can be viewed as an entity evolving along a sequence of events and
activities), the automata generation and analysis becomes unmanageable because of the
state explosion problem (too many states). The concept of Petri nets was first developed
by Carl Adam Petri in his Doctoral Dissertation [18]. The breakthrough is to consider that
a good model of a discrete event system has to integrate a structural view as a collection
of sequential processes and a state transition one. As a consequence, the primitives used
to depict the communications (interactions) between the processes have to be the same as
the ones used to depict the inner events and activities of the processes.

Informally, a Petri net is a graph consisting of two kinds of nodes. The transitions which
represent events are represented by bars (or small rectangles). The places which represent
activities are represented by circles. The state of a Petri net (its current marking) is
denoted by a token distribution in the places. Each token represents an entity instance
and the place where it is located is its current state. A place containing \( n \) tokens describes
the fact that \( n \) instances of the same class of entity are in the same state.

In the context of event-driven chemical processes, the entities may be batches (the
places are recipe steps or phases), reactors (places describe their states: idle, occupied for
operation \( op_i \), occupied for operation \( op_j \)...), etc.

A Petri net representing a physical system has to verify the so-called "good properties"
i.e. bounded, live and reversible [8, 16]. The first one implies that the number of states is
finite, the second one that there is no deadlock (batches remain infinitely blocked waiting
for idle vessels or pipes) and the third that a recipe can be repeated infinitely (an execution
does not entail an irreversible degradation of the process).

Over a Petri net, it is also possible to compute p-invariants and t-invariants. These
invariants can be seen as subnets having specific properties. For instance a positive p-
invariant containing a unique token is a sequential process and can correspond to the
various states of an equipment. In consequence, each decomposition of a net into a col-

2
lection of positive p-invariants results in a view of the modelled system as a collection of communicating sequential processes. This notion is therefore very important for the specification, as well as for the verification and the implementation. Siphons and traps [16] are other types of subnets which are important to prove that a net is live or not.

Let us now illustrate the benefits of using Petri net based models in the context of event-driven operations.

3 Petri net for specification

As it has already been mentioned, almost all approaches used to represent discrete event systems are based on a state transition graph. This section illustrates the fact that, from a practical point of view, the main drawback is not the state explosion but the fact that the natural structure of the system as a collection of concurrent processes is lost each time the entire system is represented by a unique graph.

3.1 Other approaches

In this section some more original approaches based on logic are briefly discussed.

The principle of these approaches is that the states are not individually enumerated. In contrast, groups of states are characterised by means of logical propositions. When a proposition is true, it is known that the system state belongs to the corresponding group. A first difficulty is to define the meaningful groups and the corresponding propositions. There is however a more essential difficulty. Classical logic is inadequate to reason about time and state evolution. Once a proposition has been proved true, it has to remain true otherwise an inconsistency would be detected (monotonicity issue). This inadequacy of classical logic has resulted in the well known frame and ramification problems in planning and there is no reason why any attempt to capture the event structure of batch systems in this way would be more successful.

In order to reason about time, it is necessary to work with some kind of temporal logic (based on modal logic) even if no explicit timeliness constraints are involved. However, it is often necessary to explicitly refer to a state transition graph which is the Kripke model defining the worlds (states) where the propositions are consistent within the classical logic framework. As a matter of fact, all reasoning about state changes has to be based on the state transition graph and the typical search for invariants (propositions true for all the states) although useful for formal proof of programme correctness is useless when the concern is the specification of a system evolution.

The second solution is to use Linear logic [12] in which propositions are considered as resources which are produced and consumed during the derivations. It seems very promising to use a Linear logic sequent to characterise a sequence of state changes. Actually, the existence of a sequence transforming an initial state into a final one can be considered as a proposition which is always true because it derives from the structure of the system (permanent knowledge, not a contingent one). As a matter of fact, Petri nets can be used
as a Kripke model defining the states of a set of propositions in Linear logic \cite{5, 19} and the combined use of Linear logic and Petri net theory seems seminal.

### 3.2 Petri nets based specification

Let us now, in a very practical way, point out the advantages of a Petri net modelling with respect to a state transition graph representation.

Let us consider the example represented by the flowsheet in figure 1. Two products are concurrently produced by means of two reactors. The recipe of *Product_1* authorises two alternate sequences: either *Buffer_1* and *Reactor_1* or *Buffer_2* and *Reactor_2*. On the other hand, *Product_2* is directly loaded in *Reactor_2* without intermediate storage.

If one batch of *Product_1* and one of *Product_2* are considered, the dynamics of this system is described by the Petri net in figure 1 (it is considered that the size of the buffers is sufficient). This Petri net is covered by 4 p-invariants $I_1$, $I_2$, $I_3$ and $I_4$ with the following interpretation:

- $I_1$: places $p_1$, $p_2$, $p_3$, $p_4$, $p_5$ with transitions $t_a$, $t_b$, $t_c$, $t_d$, $t_e$ and $t_f$, it represents *Product_1* recipe with its two options,
- $I_2$: places $p_6$, $p_7$ with transitions $t_g$ and $t_h$, it represents *Product_2* recipe,
- $I_3$: places $p_8$, $p_3$ with transitions $t_b$ and $t_c$, it represents the state evolutions of *Reactor_1* (place $p_3$ denotes the allocation of *Reactor_1* to *Product_1* as well as the fact that *Product_1* undergoes an operation in *Reactor_1*),
- $I_4$: places $p_5$, $p_9$, $p_7$ with transitions $t_e$, $t_f$, $t_g$ and $t_h$, it represents the state evolutions of *Reactor_2* and its possible allocation to the two products (it is a shared equipment).

This Petri net describes the system as a collection of communicating processes (the four p-invariants $I_1$ to $I_4$) and therefore its structure derives from the flowsheet. In a way,
the Petri net does not only describe the system functioning (behavioural model) it also captures the system structure (structural model).

Let us now present the state transition graph specifying the same behaviour of the process system.

3.3 State transition graph specification

Computing the reachable marking graph is the simplest way of deriving a state transition graph from a Petri net. Each node is a marking (i.e. a state) and an arc connecting two nodes denotes an event which possibly changes the system state from the input node to the output one.

The marking graph of the net in figure 2 is represented in figure 3. The inscriptions in the nodes denote the marking (we focus on the processes describing the recipe and dropped the places denoting the reactors in idle states). For instance 16 denotes a marking where places $p_1$ and $p_6$ contain one token (note that as place $p_8$ is only empty when place $p_3$ contains one token and as place $p_9$ is only empty when either place $p_5$ or place $p_7$ contain one token, they can be omitted and 16 denotes a marking such that places $p_1$, $p_6$, $p_8$ and $p_9$ contain one token. In order to explicit the relationship with the Petri net, the arc labels are represented under the form of a transition. This state transition graph looks like a Petri net but it is also a typical state transition graph because each transition exactly has an input state and an output state (i.e. any state transition graph can be considered as a Petri net belonging to a special class).

Let us first point that there is no state explosion because the number of states is exactly the number of places in the Petri net (9 places and 9 states).

The second important point is that the structure of the system has been completely lost. Two arcs outputs of the same node may represent a decision between two options within a recipe or two independent events (independent products).

In the example in figure 3, although the two products only share one common resource,
all the state nodes (except 56 and 47) have output arcs labelled with events belonging to the two recipes. For instance state 26 outputs are labelled by \( t_b \) (allocation of \( \text{Reactor}_1 \) to \( \text{Product}_1 \)) and \( t_g \) (allocation of \( \text{Reactor}_2 \) to \( \text{Product}_2 \)) although these two events are totally independent. Indeed, once the decision has been made to use the first option of the recipe to produce \( \text{Product}_1 \) (\( t_a \) fired and state 16 changed into state 26) there is no competition for the resources between the two products.

As a consequence of this second point, the decision structure is totally unclear in the state transition graph specification. The choice between the two options in the recipe of \( \text{Product}_1 \) is clearly stated in the Petri net representation: place \( p_1 \) has two output transitions \( t_a \) and \( t_d \). These two transitions are in conflict \( i.e. \) they share a common input place and only one of them can be fired. It is necessary to select one of them by making the decision \( d_a \) (\( t_a \) is fired) or the decision \( d_d \) (\( t_d \) is fired). These decisions are (in this specification) not related to the current state of the process corresponding to \( \text{Product}_2 \). They are not specified in the Petri net and could result from a management level operating with aggregated data optimising the system functioning on a particular time horizon. In contrast the Petri net clearly specifies when these decisions are used (have an influence on the system behaviour). In a similar way, the allocation of \( \text{Reactor}_2 \) to \( \text{Product}_1 \) or \( \text{Product}_2 \) is attached to the conflict resolution at the output of place \( p_9 \) (decision \( d_e \) for \( t_e \) or \( d_g \) for \( t_g \)).

In contrast, in the state transition graph in figure 3, the fact that three arcs are issued from state 16 suggests that the choices between \( t_a \), \( t_d \) and \( t_g \) are exclusive and it is not the
case. Indeed if \( t_a \) is selected (decision \( d_a \)), the next state is 26 and \( t_a \) can be fired. In place of having the decision points clearly specified, the decision structure is distributed among a set of nodes which have to be analysed as a whole.

It must be underlined that the main purpose of a clear specification of the discrete part of event-driven process is to explicit the interactions between the process states and the decision making system (including management levels for planning and scheduling) in charge of controlling it. In the above example, it is important to specify when each decision is made (in relation to which system states) and what are the conflicting decisions. The fact that the Petri net specification clearly offers a view of the system as a collection of communicating sequential processes is very important in this context.

### 3.4 The initial marking as a parameter

Let us now focus on another issue: the state transition graph approach does not allow considering the initial marking as a parameter. Let us consider the process system in figure 1 but with two batches of \( \text{Product}_1 \) and one of \( \text{Product}_2 \). In figure 2 it is sufficient to add a second token in place \( p_1 \). This new token denotes a second instance of the entity \( \text{Product}_1 \) recipe which will evolve concurrently with the first instance. In contrast, in the case of the state transition graph a new graph has to be generated.

By adding new tokens, the number of states is exponentially increased and very quickly the designer faces the state explosion problem. However, the main drawback is the fact that the structure of the model is totally modified when the number of entity instances of the same type (product recipes, reactors etc) increases.

### 3.5 Concluding remark

Indeed, it is strongly recommended to build a discrete event model as a set of communicating sequential processes. A Petri net based modelling authorises such an approach. Modelling an entire complex system by a unique state transition graph is unmanageable. It is the reason why they are typically specified by a collection of local state transition graphs rather than by a unique global one.

However, this scheme suffers various limitations. A first drawback is that the decomposition has to be done previously. This decomposition is rigidly set and cannot be modified for verification or implementation because the interactions between the state transition graphs corresponding to the various active components are either unspecified or specified with specific primitives which differ from the inner state transitions.

In contrast, with a Petri net based approach it is possible to focus on various possible decompositions (various p-invariants covering the net). It is also possible to automatically derive the unique state transition graph describing the entire system (the reachable marking graph of the Petri net). These points are essential for verification and will be illustrated now.
4 Petri net for verification

4.1 The purpose of verification

Is verification really important? What are its benefits? In industry, it is often considered that the important thing is to have a detailed representation of the system and to simulate it. However when the results of the simulation do not correspond to the expected ones (when they contradict the implicit cognitive model of the designer) three findings can be derived:

- the system actually behaves so (the cognitive model is incorrect and/or incomplete),
- there is an error in the specification model (it has to be corrected to fit to the cognitive model),
- there is an error in the simulator (contact the software consultant).

Without a powerful analysis and verification, it is very difficult to select one of these three options. But the worst situation occurs when the results although consistent with the intuition are incorrect because the specification model is inconsistent.

In a nutshell, the purpose of verification is to get confident about the system specification. It is unfortunately impossible to formally prove that a formal model exactly describes the informal problem present in the mind of the designer. However it is rare to be totally consistent when a specification error is done. By checking the good properties, most inconsistency will be detected and by deriving p-invariants and t-invariants specific behaviours will be pointed out. It must be underlined that discrete event systems are complex and, as they deal with integers, non linear. It is sometimes dangerous to rely on intuition to decide if the specification is correct or not. Let us now illustrate the importance of verification by focusing on a major issue: is the system deadlock free (i.e. is the Petri net live)?

4.2 Example of deadlock
This section discusses the issue of deadlock in batch systems. A toy example is considered to introduce the main issues. For more complex examples of deadlock in batch systems see [11].

Let us consider the flowsheet in figure 4. The process system is made up of two reactors (Re₁ and Re₂). We assume that the master recipe suggests two options: either a first operation (Op₁₁) is done in Re₁ and then a second operation (Op₁₃) is done in Re₂ or Op₂₁ is done first in Re₂ and then Op₂₃ is done in Re₁. This functioning is depicted by the Petri net in figure 5. The three sequential processes (the p-invariants) are:

- \( I₁ \): places \( p₁, p₁₁ \) (Op₁₁), \( p₁₂ \) (transfer from Re₁ to Re₂), \( p₁₃ \) (Op₁₃), \( p₂₁ \) (Op₂₁), \( p₂₂ \) (transfer from Re₂ to Re₁) and \( p₂₃ \) (Op₂₃), it denotes the master recipe with its two options,

- \( I₂ \): places \( p₂, p₁₁, p₁₂, p₂₂ \) and \( p₂₃ \), it represents the states of reactor Re₁,

- \( I₃ \): places \( p₃, p₁₂, p₁₃, p₂₁ \) and \( p₂₂ \), it represents the states of reactor Re₂.

Let us suppose that only one batch of product is handled at each time, the initial marking of the Petri net is that represented in figure 5 and for this initial marking the net has all the good properties. In particular, the net is live and thus deadlock free (no deadly embrace, no infinite wait of resource). This analysis can be done by state enumeration (only 7 states).
Let us now consider that two batches of product are simultaneously handled. This is represented by two tokens in place $p_1$ at the initial marking. From this state it is possible to fire transition $t_a$ and, immediately after, transition $t_e$ (the first option of the recipe for the first batch and the second one for the second batch). The reached marking is $M$ such that $M(p_1) = 1$, $M(p_2) = 1$ and $M(p) = 0$ for all other places. From this marking no transition can be fired: it is a deadlock. None of the batches can proceed and the process system is blocked.

### 4.3 Some intermediate conclusions

From this example, four findings can be pointed out:

- event-driven systems are complex and may have strongly undesired behaviours (deadlocks),
- these behaviours may appear after a tiny alteration of the system (adding one batch of a product),
- requesting a new resource before releasing the one currently used is dangerous (unfortunately this corresponds to any transfer operation in batch systems),
- it is very important to verify the specification and this verification has to be based on a global view of all the sequential processes and their interaction (it is not sufficient to analyse the behaviour of one batch of product in the system),

In a nutshell, if it is possible to use the state transition graph approach for specifying event-driven operations by independently describing the various sequential processes, this approach is inadequate for analysis and verification because the problems precisely result from the interaction between the processes. It is why C.A. Petri's intuition of using the same primitives for describing the inner dynamics of the processes and their interaction has been a substantial breakthrough.

It is sometimes considered that the Petri net approach is just a way of generating the state transition graph automatically (generating the reachable marking graph). It is true that when the good properties analysis is based on marking enumeration, the state explosion problem is not avoided. However, p-invariants, t-invariants and reduction rules [2, 3, 16] are not based on marking enumeration and they are very useful in practice although they do not offer any simple necessary and sufficient liveness conditions. Let us illustrate this point now.

### 4.4 Initial marking as a parameter, reduction rules

As the absence of deadlock may depend on the initial marking, it is important to be capable of deriving the liveness property for some classes of initial markings when it is possible, i.e. to consider that the token counts of some places are parameters. This is totally impossible when the analysis is based on the marking enumeration, but it is possible if it is based on the use of reduction rules [2, 3, 16]. The principle of these reduction rules is that each rule
transforms the net been analysed into a simpler one (the reduced net) which is equivalent in the sense that a necessary and sufficient condition for the initial net to have the good properties is that the reduced one has them.

Among all the possible reduction rules, one is of particular importance: the redundant place elimination sometimes called the implicit place elimination. Indeed the marking condition to eliminate such places is a minimal one i.e. the rule can be applied in the same way if the initial token count is increased. Let us consider a simple case of redundant place: the trivial redundant place which is represented in figure 6. Place $p_1$ is only connected to the remaining part of the Petri net by means of self loops. Its token count is not affected when transitions $t_a$ and $t_b$ are fired. If its initial token count is $n$ such that:

$$n \leq k_1 \text{ or } n \leq k_2$$

(1) then transition $t_a$ or transition $t_b$ can never be fired (the net cannot be live). If $n$ is such that:

$$n \geq \max(k_1, k_2)$$

(2) then $t_a$ and $t_b$ are fired when they are enabled by the other places of the net and place $p_1$ is redundant and can be deleted without any change about the Petri net properties (it does not play any role in the marking evolution).

Indeed, inequation 2 shows that $\max(k_1, k_2)$ is a threshold for the initial token count of place $p_1$. Under this threshold the net is not live (occurrence of deadlocks or existence of dead parts in the Petri net). Over this threshold, the place is redundant and the corresponding item (resource, product or pool of resources or of products) cannot be responsible for a bad behaviour.

For instance, if the net in figure 2 is analysed by reduction, after the application of other reduction rules (fusion of series transition [16]) the places $p_1$, $p_6$, $p_8$ and $p_9$ becomes trivial redundant places iff they initially contain at least one token. This means that in this example, batches of the two products or reactors of the two types can be added without any risk.

This kind of event-driven operations is thus deadlock free and it is not necessary to analyse the behaviour of the system (by means of state enumeration) for 1, 2, 3, etc batches of Product_1.
4.5 Emptying a siphon

Unfortunately, the net in figure 5 cannot be analysed by means of these reduction rules because of its more complex structure which is precisely responsible for the fact that the good properties are lost by an increase of the initial token count in some place. The particular template responsible for this is (in this case) a siphon which is a group of places in which it is impossible to re-introduce a token if they are all empty [16].

The particular siphon, present in the net in figure 5 (typical of the fact that two resources are requested with different orders [10]), is represented in figure 7. It is a p-subnet i.e. it is a subnet defined by a set of places \((p_2, p_3, p_{12}, p_{13}, p_{22}, p_{23})\) with all their input or output transitions. If all the places are empty, no transition can be fired because all of them has at least one input place belonging to the subnet. In consequence, no sequence can re-introduce a token into the siphon. There is a deadlock situation.

This situation is reachable if it is possible to empty the siphon by firing transitions \(t_a\) and \(t_e\) repetitively. The initial marking condition for the net in figure 5 to be live is that:

\[
M_0(p_1) < M_0(p_2) + M_0(p_3)
\]  

which means that the total number of reactors of types \(Re_{-1}\) and \(Re_{-2}\) has to be greater than the total number of batches.

For this class of process systems the analysis is more complicated but nevertheless a general result can still be obtained. In conclusion, this discussion has illustrated the fact that with a Petri net based approach it is possible to specify, analyse and validate particular classes of discrete-event systems without enumerating all the states and only operating on the system structure under the form of a group of interacting sequential processes.
4.6 Temporal analysis

Once it has been shown that the Petri net model is consistent (it has the good properties) and that it actually describes the discrete event part of the considered process system, it is possible to associate explicit timing consideration with it. Various approaches exist for introducing time and timeliness constraints in a Petri net. For a stochastic analysis the Generalised Stochastic Petri net [16, 21] allows obtaining a Markovian process from a Petri net model after the association of firing rates to some transitions. This approach is particularly interesting for reliability analysis when failure rates are attached to the apparatus. For a formal analysis of a set of timeliness constraints, the Time Petri net model [4] can be used and for the particular case of batch systems the arc timed model [13] has been developed. Finally, models integrating the continuous and the discrete temporal aspects within the framework of Petri nets have been proposed for event-driven process systems [7, 8, 9].

However the analytical approach is complex and the designer frequently faces the state explosion problem. In industry, simulation is used for intermediate buffer dimensioning, for performance evaluation or for ensuring that a particular manufacturing policy, or manufacturing plan or schedule is feasible. In this context a very interesting approach is to combine a Petri net based discrete event simulation with a continuous one. The operation durations are not associated as explicit times to the transitions but, on the contrary, they result from threshold crossings in the continuous simulation [6, 17]. Each time a significant threshold is crossed, the corresponding transition is fired. This firing results in changes in the process system (gates opening or closing) and therefore in the differential algebraic equations which are representing the continuous part being simulated. This kind of simulation is very close to the process system behaviour, duration are computed and depends of the actual state of the products. They have not to be rigidly predefined.

5 Petri net for implementation and monitoring

5.1 Some principles

Basically, there are two main approaches for implementing on a computer a discrete-event supervisory control which has been specified and designed by means of a Petri net. For the first one, by means of a procedural language such as ADA, a collection of tasks are coded in such a way that their overall behaviour emulates the dynamics of the Petri net. For the second one, the Petri net is considered as a set of declarative rules (each transition is a rule describing a possible state change). Then, an inference engine which does not depend on the particular Petri net to be implemented, operates on a data structure representing the net. This inference engine is called the token player [21] and this approach is very interesting because it automatically encapsulates a detection mechanism. Actually, at supervisory control level, monitoring functions including detection, diagnosis and reconfiguration are of the utmost importance in order to ensure safety and reactivity.
5.2 The token player

A supervisory control interacts with the physical process system by means of messages which frequently correspond to threshold crossings detected in local controllers \([1]\). In consequence, in the Petri net model some transitions denote message receivings and others message emissions.

The first task of the token player is to update the state representation of the physical system in the supervisory controller when a message is received from a local controller. This is represented in the left part of figure 8. When the token player wait for a message, it is in the stable state. When an event occurs (a message signaling a threshold crossing), the token player searches the data structure representing the Petri net for the transitions attached to it. These transitions are the state evolution rules explaining the event. The current state representation (current marking of the net) has to be so that one and only one of these transitions is enabled.

If it is not the case, then the system state representation at the supervisory level is inconsistent with the actual state. Either the physical system is faulty, or the control system is faulty or there has been a design error at the specification phase of the supervisory control. The fact that it is impossible to fire a transition which is not enabled (a marking is always positive) ensures that any failure will automatically be detected very rapidly.

The right part of figure 8 depicts the inner cycle of the token player. When the state representation has been successfully updated by means of a transition firing, a new marking is obtained. It is then necessary to look for enabled transitions which are not associated with message receivings. These transitions are fired and if message emissions are attached
to them they are executed. A new marking is obtained and a new search for the enabled transitions is done. This inner cycle terminates when all the transitions, enabled by the current marking, are associated with message receivings; it is the stable state.

5.3 Monitoring

We have seen in the precedent section that the implementation of the supervisory control by means of a token player, necessarily contains a detection mechanism. Any attempt to fire a transition which is not enabled reveals an inconsistency between the model of the system and its actual behaviour. In other words, this means that the physical system does not respect a precedence constraint (event $e_a$ has always to occur before event $e_b$). This mechanism is traditionally complemented by another one for which the durations are explicit. It is the classical use of watch dogs (timeouts). The timeliness constraints are of the form: event $e_b$ cannot occur after event $e_a$ before a duration $\theta_{\text{min}}$ and cannot occur after a duration $\theta_{\text{max}}$.

This kind of mechanism is easily integrated in the token player. It is just necessary to check that the corresponding transition has been enabled at least during $\theta_{\text{min}}$ when $e_b$ occurs and that the transition never remains enabled more than $\theta_{\text{max}}$. It is sufficient to attach to each enabled transition in the stable state its earliest and latest firing dates.

When an abnormal behaviour is detected, it is often necessary to reason about sequences of events in order to elaborate a diagnosis. It is for example the case when there is an attempt to fire a transition which is not enabled. The diagnosis has to calculate all the possible sequences leading from the last current state to a marking enabling the transition. It is necessary to backwardly search the Petri net for tokens. Linear logic is an adequate formal tool for this [19].

The detection mechanisms and diagnoses offered by the token player approach are a kind of model based diagnosis with a model of the correct behaviour. The model (the Petri net) is a behavioural model. However, when the transitions are considered as rules explaining the state changes, the Petri net can be considered as a causal model. In addition, we have stressed the fact that a Petri net describes a system under the form of a collection of interacting sequential processes and thus some structural information can also be derived from the Petri net model. It is the reason why we think that the Petri net based approach is seminal for developing monitoring systems.

6 Conclusion

There are three possible approaches for modelling a process system with event-driven operations:

- introducing discrete aspects within a continuous model by means of boolean or integer variables,
- introducing continuous aspects within a discrete model by means of time or of some predefined variables (e.g. batch size),
• employing two interacting models: a Petri net for the discrete part and differential algebraic equations for the continuous one.

The adequacy of these three approaches depends on the importance of the discrete aspect in relation to the continuous one. Petri nets play an important role in the second and third approaches.

The notions of state and transition have to be employed anyway. With classical state transition graphs, the designer faces a contradiction between the necessity of a system break-down for an easy specification and that of having a unique graph representing the entire system for a validation of the interactions between the components.

With the Petri net approach, as the interactions between the sequential processes are described by the same primitives as the state changes within the processes, it is possible to combine a system break-down at the specification stage with a unified view at the validation one. In addition it has been shown that the Petri net model captures the structure of the system and that it is possible to consider that the number of batches, the number of reactors etc are parameters: a new model has not to be reconstructed each time one of these numbers is changed.

References


[10] J. Ezpeleta: S3PR, a class of well structured Petri nets, 16th International Conference on Application and Theory of Petri nets, Torino Italy, June 1995


