Some issues about Petri net application to manufacturing and process supervisory control

Robert Valette

LAAS-CNRS
F-31077 Toulouse Cedex 4, France
http://www.laas.fr/~robert

Abstract. After some general considerations concerning modeling issues, the paper concentrates on the fact that places are generally interpreted as logical conditions. There are various kind of logical propositions, some (the resources) are consumed when they are used in the same way tokens are removed from places when transitions are fired. Linear logic has pointed out the fact that resources had to be handled with a restricted set of logical rules and that they could not easily be handled concurrently with classical logic propositions. This can be exploited for developing modeling methods based on Petri nets. Then two cases of combining Petri nets and other formal tools are examined: Petri nets and fuzzy sets and Petri nets and differential algebraic equations. It is shown that it is better to avoid a complex integration of the various approaches in order to keep the analytical capabilities of Petri nets.

1 Theory, tool and application

When an industrial application is addressed, the major issue concerns the trade-off between fidelity to reality and tractability. Typically, the theoretical results are only valid for specific cases which do not exactly map to the actual system. The major issue is therefore to obtain a model which can be analyzed and whose behavior is not too far from that of the physical system. It is also clear that in order to cope with the complexity, a tool is required. In a nutshell, dealing with a complex application implies solving three problems:

- find theoretical results which could aid the engineers,
- find a method in order to obtain a good model of the system,
- find a tool to support the approach.

Figure 1 illustrates the complexity of the relationships between theory, tool and application and allows to refine the issues arising from complex applications.

When an academic group or a commercial firm decides to develop a tool, a general application domain is generally considered. However, the natural trend is that the tool has to be as general as possible. Tools such as Design/CPN, MISS-RdP or AMI may be used for manufacturing systems as well as for protocols etc.

As a consequence, when a specific application is concerned, the issues concerning modeling are twofold:
- how obtaining a tractable model of the system,
- how obtaining a model supported by the selected tool.

These issues may not coincide because each tool has its way of considering modularity, its way of adding timing considerations, its way of adding data and data transformation etc.

As a matter of fact, each line in figure 1 corresponds to a specific kind of work:

- developing a generic tool for Petri net based simulation and analysis,
- dealing with industrial case studies with the help of an existing tool,
- analyzing classes of industrial problems and developing specific methods for these classes.

The third case differs from the second one. In the third case people try to find a trade-off between using simplifying hypotheses in order to apply theoretical results and building comprehensive models. In the second case people try to describe the system behavior by means of the primitives of the chosen tool. Although, we have had a lot of interactions with the developers and the users of MISS-RdP (a Petri net tool specially suited for Monte Carlo simulation) we have tried not to mix the necessity of a method and that of a tool and our research has essentially been of the third type.

Two ways of addressing this kind of work can be characterized. The first one, which could be called bottom-up, consists of observing how engineers (and students) use Petri nets to model their problems in order to analyze their difficulties. It is also important to observe which part of Petri net theory they use, or they do not use, and to understand why. The second way of addressing this kind of work, consist of starting from theoretical results and trying to develop a method for a practical use of them. We try to combine these two approaches.

In order to build comprehensive models of physical systems, it is generally necessary to take into account timing considerations, data transformations (on integers or reals), imprecision etc. This is done either by extending Petri nets, or by adding inscriptions to it, or by combining Petri net theory with other ones. How this can be done and what are the best choices are our major objective.
2 Some issues about the logical interpretation of places

Models of manufacturing systems generally involve “conditions” which are used to describe the control of sequences or resource allocations. When Petri net are used as a modeling tool [48], these conditions are represented by places. However, interpreting places as logical conditions poses some problems. Two issues frequently met in practice will be presented first. Then some studies about Petri net and logic will be briefly presented and in next section it will be shown how linear logic is an aid for understanding what are places from a logical point of view.

2.1 Representing control sequences

It is well known that control sequences can easily be represented by Petri nets. An industrial standard, the Grafcet, has been derived from this Petri net representation. The Petri nets which are used, are of a simple class: they are condition-event nets. This means that places are interpreted as conditions and transitions as events. All this seems so crystal clear that it is surprising to hear students saying that this representation is inconsistent. Let us detail their point of view.

In figure 2 an elementary fragment of a controller sequence is represented. The correct representation is in the left part of the figure. When the device is in the step \(\text{step}_i\), then the controller has to monitor the sensor \(v\) and when \(v\) is greater than \(k\), the device passes from \(\text{step}_i\) to \(\text{step}_{i+1}\). The conditions, from this point of view, are the steps and the event is the threshold of sensor \(v\). The standard specification is represented in figure 2.a. The threshold is specified by means of an inscription attached to a transition.

![Diagram](image)

**Fig. 2.** Two ways of modeling controller programs

However, \(\text{event}_i\) is the consequence of two conditions: “if the device state is \(\text{step}_i\)” and “if the sensor \(v\) value is greater than \(k\)”. Stated in this way, it appears that the correct representation should be symmetrical between the
two conditions and that the representation in figure 2.b is more consistent. Indeed in figure 2.a, the only condition mathematically expressed as a predicate \( \text{sensor}_v > k \) is attached to a transition, not to a place. Actually it is sensible to say that the transitions also denote conditions in the same way places do and that this representation is confusing.

In order to develop a method aiding a designer to model control sequences, it is necessary to clearly and formally explain the difference between the predicate \( \text{sensor}_v > k \) and the condition “device is in state step\(_1\)”.

2.2 Modeling shared resource

The management of a manufacturing system typically consists of allocating machines (or reactors or other devices) to products in order to perform operations. Resource allocation mechanisms are therefore essential. Modeling a resource shared between two processes (or tasks) seems simple and natural for a Petri net addict. The Petri net in figure 3 is considered as the good model. Place \( R \) denotes the resource in the idle state. From the place invariant

\[
M(1_{on}) + M(2_{on}) + M(R) = 1
\]

the following inequality can be derived

\[
M(1_{on}) + M(2_{on}) \leq 1
\]

which formally proves the mutual exclusion between task\(_1\) and task\(_2\) in state \( "on" \).

![Petri Net Diagram](image)

**Fig. 3.** A first Petri net of the mutual exclusion

However an untrained designer is likely to obtain the net in figure 4 and to find it more natural. The net comprises 4 places whereas the net in figure 3 is composed of 5 places. It is thus simpler. It perfectly describes the states of the two tasks (the active processes) and clearly expresses the logical conditions that have to be checked before entering the critical sections. The process task\(_1\) can only enter the state \( 1_{on} \) if the process task\(_2\) is in state \( 2_{off} \) (and therefore not in \( 2_{on} \)). Such a model seems closer to an implementation on a computer because it only comprises the two active entities task\(_1\) and task\(_2\). But, the incidence matrix
of the net in figure 4 is exactly the same as that of the Petri net in figure 5 because elementary loops are not represented. From the two place invariants:

\[
M(\text{task}_1 \text{off}) + M(\text{on}) = 1 \\
M(\text{task}_2 \text{off}) + M(\text{on}) = 1
\]

(3)
(4)

it is only possible to derive

\[
M(\text{on}) + M(\text{off}) \leq 2
\]

(5)

and the inequality 2 cannot be obtained (it is no use adding places in the elementary loops to enrich the incidence matrix). As a consequence the mutual exclusion can only be proved by a state enumeration procedure and not by linear programming.

For such a simple model it is not a problem. However if you want to develop a method to derive Petri net models of resource allocation mechanisms which can be proved correct by linear programming (or even by construction), it is essential to understand the differences between the nets in figures 3 and 4. As in the preceding case, the heart of the issue is the understanding of the nature of two different logical conditions: why is it richer (produces models easier to analyze) to use the condition “if the resource R is idle” than to use the condition “if the process task$_2$ is in state 2.off”?

2.3 Some results about Petri nets and classical logic

The relationships between Petri net theory and classical logic have been mainly analyzed in the context of logic programming [38, 42] and in that of diagnosis [5, 43].

In the first approach [38, 42], the interesting point is that the Petri nets modeling logic programs are not bounded. As a matter of fact, if a proposition $a$ is associated with a place $P_a$, putting a token in $P_a$ is interpreted as having constructed a proof, from the initial knowledge to $a$. If the proof scenario has been executed a first time, it can be executed again and again. As $a$ does not
belong to the initial knowledge, the tokens in \( P_a \) will not be consumed when the proof is executed. As a consequence, the token load in \( a \) can be increased infinitely and \( P_a \) is not bounded.

A consequence of this is that the net cannot contain positive \( p \)-invariants (the places comprised in a positive \( p \)-invariant are structurally bounded). This modeling technique, interesting for logic programs, is of no use for sequential control and resource allocation mechanisms.

In the second approach [5], Petri nets are used to describe the causality relations between the causes of failures and their observation. The Petri nets are safe, this means that putting a token in a place corresponds to the fact that the associated logical proposition is true. If there is no token the proposition is yet unknown. If the proposition has been proven false, then a white token is put in the corresponding place. The Petri net has to be cycle free because it is impossible to have a causality dependence between a proposition and itself. As the net is cycle free, no meaningful \( p \)-invariant can be deduced. Therefore this approach cannot be used for resource allocation mechanism.

It seems thus that any attempt to use Petri nets to model deduction mechanisms has two consequences:

- they have very specific structures (not bounded, or safe and cycle free),
- they are not adequate to depict resources.

This means that “good” Petri nets adequate for representing and analyzing (by means of positive \( p \)-invariants as in figure 3) resource allocation mechanisms and control sequences are not based on an interpretation of the places as logical conditions in the classical logic sense.

3 Petri nets and linear logic

3.1 What is a resource from a logic point of view

A resource is a proposition which is consumed when it is used to derive a new proposition. When we have modeled a control sequence and a shared resource we have met such propositions. For example when we state in section 2.1 that:

if “device is in \( step_5 \) and \( event_i \) then “device is in \( step_{i+1} \)”
the propositions “device is in step_i” and “device is in step_{i+1}” are typical resources. Indeed when you use “device is in step_i” to derive the next control state, you immediately transform the representation of the state of the device stored in the controller and the proposition is no longer available to derive other propositions.

In the same way, in section 2.2 the condition “the resource R is idle” is a resource whereas the condition “the process task_2 is in state 2 off” is not a resource because the state of task_2 is not altered when task_1 enters the critical section.

3.2 What is linear logic

Linear logic has been proposed by J.Y. Girard [21, 22, 23, 31] in order to deal with resources. In place of defining new connectives and rules, he started from the sequent calculus and discovered that two structural rules (the contraction and weakening rules) were inconsistent with the notion of resource. This means that the linear logic sequent calculus is a restriction of the sequent calculus in the context of classical logic.

The inconsistency of the contraction and weakening rules with respect to resources derives from the fact that if a resource a is not necessary to produce a resource b, it cannot be added to the left part of the sequent because it will not be consumed to produce b. In the same way, a resource cannot be added to the right part of a sequent. If a resource is added to the right part (left part respectively) of a sequent it has to be also added to its left part (right part respectively), and with the same connective. It is why this logic is called linear.

Another point which is very important is that a resource can be available for one instance, or two instances, or n instances. A resource is not simply true, it can be counted. Actually, if in classical logic a \land a reduces to a, it is because the contraction rule is valid. It must be pointed out that in a manufacturing system it is absolutely not the same thing to have one or two or n machines which are available. Its performance will be completely different.

The last consequence of the elimination of the contraction and weakening rules is that the linear logic negation cannot be considered as having the same interpretation as the classical one. Indeed it is possible to simultaneously have one machine which is available (a) and to need another one (a^⊥) which is the linear logic negation of a) because two machining operations should be initiated.

The connectives of linear logic will not be detailed here. Let us just recall that Linear logic has 3 sets of connectives:

- the multiplicative ones : \otimes (called “times”), \exists (called “par”) and \rightarrow (the “linear implication”),
  - \otimes expresses “AND” of resources,
  - \rightarrow expresses causal dependency between resources,
- the additive ones & (called “with”) and \oplus (called “plus”) which permit external and internal choices.
the exponential ones! (of course) and ? (why not). They permit to reintroduce contraction and weakening notions when required.

3.3 Places as linear logic resources

In recent years various Petri net interpretations of linear logic have been explored in the literature [8, 18, 26, 35, 36]. The approach of C. Gunter and V. Gehlot [26] offers a direct translation between Petri net formalism and linear logic formulas. The places of a Petri net are considered as resources, or rather resource classes. Producing/consuming instances of a resource corresponds to producing/consuming tokens in the associated place.

Our aim is not to solve some decidability problem in linear logic by means of Petri net theory, neither to prove some new property of Petri nets by means of linear logic. Initially, it has been to improve the understanding of what are places in our models. We have rapidly noticed that linear logic could be used to develop a formal notation for sequences (or scenarios) of transition firings [10, 11, 24, 44, 45]. This characterization is useful in at least two contexts:

- analyzing the possibility of executing some scenario starting from a marking,
- analyzing the scenarios producing some specific partial states.

The first point is important when recovery procedures have to be implemented in manufacturing systems and the second one is required for diagnosis and safety analysis.

The sequence characterization will be addressed in the next subsection and we concentrate first on the logical interpretation of the places. A practical consequence of the work of J.Y. Girard is that it is not sound to deal with those logical propositions which are in fact resources in the same way as with classical logic propositions. It even seems that it is difficult to simultaneously deal with the two kinds of logical propositions: even if the exponential connectives allow to re-introduce in a controlled way contraction and weakening, classical negation is not reintroduced in linear logic.

By observing Petri net theory, it can be noticed that indeed negation is absent (an arc cannot allow checking that a place is empty) and when it is re-introduced (inhibitor arcs) the classical Petri net properties are no longer decidable. In addition, p-invariants and t-invariants computation does not take into account inhibitor arcs.

At the present point, it seems that it is better to avoid combining the two notions in a unique integrated formalism. The designer should rather build a Petri net model in which places and transitions uniquely denote resources and their production and consuming. Predicates in classical logic should be attached to the transitions as inscriptions restricting the possible evolutions. Considering back the two examples in subsections 2.1 and 2.2 the difference between the two alternative models in each case is now simple.

In the first case (section 2.1) “device is in step_i” is a resource (consumed when used). It is therefore natural to code it as a place. In contrast, “if sensor
V is greater than k” is a predicate in classical logic. Actually, firing transition event \( i \) will not instantaneously change the value of \( V \). As a consequence, it is not consistent to associate it with a place as in figure 2.b.

The second case (section 2.2) is very similar. Indeed, when \( task_k \) enters the critical section, it consumes the resource \( task_{k\_off} \) to produce the resource \( 1_{on} \). In the model in figure 3, it also consumes \( R \), but in figure 4 it does not consumes \( task_{2\_off} \). In this case, the elementary loop is a trick to introduce a proposition in classical logic. Not surprisingly, the net in figure 3 is simpler to analyze because it only involves logical propositions which are resources. In contrast the net in figure 4 is more complex because it tries to simultaneously deal with resources and with classical logic propositions.

### 3.4 Sequence characterization

Analyzing how engineers developing large Petri net models for simulation proceeed, it can be noticed that they do not analyze the obtained Petri nets with respect to the “good” properties, and they do not compute invariants. They have the feeling that if their Petri net is not live, this will be detected during simulation. If it is not the case then probably it is because the association of timing considerations prevent the net from reaching the dead markings and the properties of the Petri net without time do not reflect the behavior of the actual system. As they deal with large Petri nets, they adopt a structured way of constructing the model and the \( p- \) invariants which have an important meaning (the set of states of the devices for example) are explicitly introduced in the model.

Their major difficulties result from the absence of a formalism dealing with scenarios. A t-invariant defines how many times the transitions are fired, but it does not give any information about the order of the transition firings (\( t_i \) fires before \( t_j \), \( t_i \) fires after \( t_j \) or \( t_i \) fires in parallel with \( t_j \)). In order to validate their model, they spend a long time playing the token game (by hand or by a step-by-step simulation with a tool) in order to check some predefined behavior.

![Fig. 6. A Petri net fragment](image-url)
Although, linear logic is commutative \((a \otimes b\) \text{ and } \(b \otimes a\) are not different), sequences of transition firings and scenarios can be differentiated by the way the logical deductions are made. The approach is the following one: we use the rules of the sequent calculus to build some proof schemes. From two valid sequents describing the firing of a transition (or of a sub-scenario), we derive a characterization of a larger scenario which is a composition of the two elementary ones.

Basically, two rules are used. The “cut rule” is used to characterize the fact that the two transitions (or the two sub-scenarios) are fired one after the other (in a total order). Actually, the cut rule eliminates what has been produced by the first sub-scenario because it is consumed by the second one; and it is not possible to consume what has not yet been produced. The second rule is a parallel composition of two sub-scenarios, the “\(\otimes R\)” rule is used. Let us illustrate this on a very elementary example.

Let us consider the net in figure 6. Its translation into linear logic formulas is:

\[
\begin{align*}
t_1 &: (a \otimes d) \rightarrow b \\
t_2 &: b \rightarrow (c \otimes d)
\end{align*}
\]

The firing of transition \(t_1\) one time, and from the minimal required marking is denoted by the sequent:

\[(a \otimes d), ((a \otimes d) \rightarrow b) \vdash b\]  \hspace{1cm} (6)

From the marking \((a \otimes d)\) and by means of \(t_1\) the marking \(b\) can be obtained (modus ponens).

If \(t_1\) is fired before \(t_2\), this means that the resources (tokens) produced by \(t_1\) are consumed by the firing of \(t_2\). This is expressed by the “cut rule” (completed by the right introduction of the linear implication). The sequent denoting the firing of \(t_1\) is on the left part and that denoting the firing of \(t_2\) on the right.

\[
\frac{(a \otimes d), ((a \otimes d) \rightarrow b) \vdash b \quad b \rightarrow (c \otimes d) \vdash (c \otimes d)}{((a \otimes d) \rightarrow b) \rightarrow ((c \otimes d) \rightarrow (c \otimes d))} \text{ cut} \hspace{1cm} (8)
\]

If \((t_1; t_2)\) denotes the scenario in which \(t_2\) is fired after \(t_1\) then we will have the following characterization:

\[(t_1; t_2) : (a \otimes d) \rightarrow (c \otimes d)\]  \hspace{1cm} (9)

If now the desired scenario consists of firing \(t_2\) first and then \(t_1\), the cut rule has to be applied in a different way. In effect, in the above proof, transition \(t_1\) exactly produces what will be consumed by \(t_2\). In contrast, for \((t_2; t_1)\) if the intersection between what is produced by \(t_2\) and what is consumed by \(t_1\) is not empty (one token in \(d\)), \(t_2\) produces \(c\) which will not be consumed and \(t_1\) requires \(a\) which has not been produced. In order to allow the cut, in place of firing \(t_1\) from the minimal possible marking, it is necessary to specify that it will be fired in a context where the resources \(a, c\) and \(d\) are all available. As in the left part of any sequent the comma is interpreted as a \(\otimes\) this can be written: \((c \otimes d), a\).
The sequent denoting the firing of \(t_2\) is on the left part and that denoting the firing of \(t_1\) on the right. Then:

\[
\frac{b, (b \rightarrow \circ(a \otimes d)) \vdash (c \otimes d), a, ((a \otimes d) \rightarrow b) \vdash (b \circ c)}{((a \otimes d) \rightarrow b), (b \circ (c \otimes d)) \vdash ((a \otimes b \circ d) \circ (c \otimes b \circ d))} \text{\hspace{1cm} cut (10)}
\]

The following characterization is derived:

\[
(t_2; t_1) : (a \otimes b) \rightarrow \circ(c \otimes b)
\]

(11)

If \((t_1 \parallel t_2)\) denotes the scenario in which \(t_1\) and \(t_2\) are fired concurrently (simultaneously) then the “\(?\)R” rule is used:

\[
\frac{(a \otimes d), ((a \otimes d) \rightarrow \circ b) \vdash b, (b \circ (c \otimes d)) \vdash (c \otimes d)}{((a \otimes d) \rightarrow \circ b), (b \circ (c \otimes d)) \vdash ((a \otimes b \circ d) \circ (c \otimes b \circ d))} \text{\hspace{1cm} \(\otimes\)R (12)}
\]

The following characterization is derived:

\[
(t_2 \parallel t_1) : (a \otimes b \circ d) \rightarrow (c \otimes b \circ d)
\]

(13)

It is very important to note that the three firing scenarios are formally defined and that in addition to the marking transformation which would have been obtained by the product of the incidence matrix by the firing vector (consuming \(a\) and producing \(c\)), the minimal marking required to execute the scenario is defined. The Pre and Post vectors defined for transitions are extended to scenarios which are not strictly sequential. Clearly for each scenario we get different Pre and Post (see equations 9, 11, 13). For \((t_1; t_2)\) a token has to be present in place \(d\) before the scenario and it will be present at the end of it. For \((t_2; t_1)\) it is the case of place \(b\) and for \((t_1 \parallel t_2)\) it is the case for both \(b\) and \(d\).

### 3.5 Conclusion and future work

Our work about linear logic and Petri nets has had as a first consequence a better understanding of the modeling of manufacturing systems and other complex discrete event systems. Each time a designer is about to introduce an elementary loop between a transition \(t\) and a place \(p\), he has to check that this elementary loop indeed is an aggregation of a sequence of two actions: consuming a token from \(p\) and then producing a token in \(p\). If the elementary loop corresponds to the classical logic predicate \(M(p) \geq 1\) then it is better to put this predicate as an inscription attached to \(t\). This inscription will be used as an extra firing condition for simulation, but it will be clear that it will not influence the formal analysis of the uninterpreted Petri net. It is better to have a formal consistent model (the uninterpreted Petri net) which can be used for mathematical analysis complemented by a comprehensive model (the interpreted Petri net i.e. with inscriptions) for simulation rather than to have a model in which different types of knowledge are coded in the same way, leading thus to confusions.

Future work will address the issue of scenario characterization with a computation of their durations. As a matter of fact, until now the analysis of time Petri...
nets has concentrated on the construction of the reachable marking class graph [37] which allows the detection of deadlocks and which allows to characterize all the reachable states. However this graph produces a very imprecise evaluation of the duration of the sequences of transition firings because the possibility of concurrent firing is not explicit.

Another interesting point is the possibility of building formal proofs by a concurrent use of place invariants (derived by linear programming) and firing sequences (obtained by linear logic proofs constructions). Finally, we will study the utilization of the additive connectives in order to characterize scenarios involving choices and/or imprecision about the availability of some resources.

4 Fuzzy Petri nets

Before studying linear logic, we have studied another formalism in order to take into account imprecise knowledge: fuzzy set theory; and we have combined it with Petri nets.

Various approaches can be used to combine Petri nets and fuzzy sets. It is possible to define a kind of fuzzy marking by denoting place token loads by means of fuzzy numbers. It is also possible to define fuzzy markings by attaching to each token its fuzzy location characterized by a fuzzy set of places (the places where it is likely to be located). A third alternative is to associate fuzzy variables with tokens i.e. to use the data structure attached to the tokens in high level nets. Finally, it is possible to define fuzzy firing sequences which are more or less likely to be fired. A review of the various kinds of fuzzy Petri nets which have been developed can be found in [12].

The combination of Petri nets and fuzzy sets will only be promising if it is consistent. This means that something of the Petri net theory has to be respected. The evolutions of the fuzzy markings by means of transition firings have to be consistent with the p-invariants and t-invariants of the underlying ordinary Petri net.

The starting point of most approaches has been to consider places as logical propositions and Petri nets as reasoning models. We have seen in section 2 that this assumption has complex consequences and can result in inconsistencies. By restricting our approach to the case of Petri nets representing physical systems, we have implicitly selected the case in which places were denoting resources (linear logic propositions) [9, 39, 52].

This means that places describe the various states of physical objects or devices. They are typically covered by p-invariants because an object instance (denoted by a token) is always in one and only one state. Within a Petri net model for which tokens are individuals [47], a fuzzy marking can then be defined by associating with each token a fuzzy set of places which delimit it possible location (which is ill-known but unique).

If we restrict to the case of imprecise knowledge (the membership function has either value 1 or 0), then this notion of fuzzy marking coincides with the delimitation of the set of possible states which can be reached by the firing of an
imprecise sequence characterized by a linear logic sequent containing additive connectives [10, 11].

In this approach we have combined two formalisms, Petri nets and fuzzy set theory. However we have tried to avoid any alteration of them. The ordinary Petri net remains the model of the physical system and the set of its reachable markings is not altered. A fuzzy marking is just a representation of an imprecise knowledge about the current marking of the Petri net.

5 Batch processes

5.1 What is a batch process?

The automation of batch processes poses difficult issues because it is necessary to concurrently deal with continuous and discrete models. Actually, this kind of system deals with continuous raw material (fluids) but the production and packaging is partially based on batches. In a discrete production system, a batch is a number of parts to be machined in sequence in order to avoid set-up times. It is characterized by an integer: the number of parts. In a batch process, a batch is the quantity of material which can be loaded in a reactor and its size is likely to vary in function of the reactors used in the recipe. Typically, it has to be characterized by a real number expressing the amount of material in mass, molar or volumetric units. In addition it is often necessary to define the batch quality in terms of a mixture of several pure chemical components. As a consequence, a comprehensive model of batch systems has to include discrete event aspects as well as continuous ones.

This kind of production systems, whose behavior is also called “event driven operations”, is becoming very common in food industry and fine chemicals plants. Modeling is required for two purposes which are not independent. The first one is to check, by simulation, the feasibility of an aggregated predefined schedule i.e. to evaluate the performance of the production system. The second one is to elaborate a monitoring and supervisory control in such a manner that any abnormal behavior could easily be detected and corrected by re-scheduling and re-validating the obtained schedule with a hot start simulation (i.e. the initial conditions of the simulation have to be consistent with the current state of the system). Real-time abnormal behavior detection as well as hot start simulation require real-time animation of a realistic model of the batch process which takes into account its discrete aspect as well as its continuous one.

5.2 Overview of existing approaches

Various approaches have been proposed or are under development in order to address the complexity of the concurrent execution of various recipes sharing an existing set of resources (equipments). It is important to underline that associating explicit timing considerations (with a continuous time) with places or transitions is the first way for taking the hybrid nature of batch processes into
account. Actually, if the input flow in a vessel is a constant, the volume of raw material in it will be proportional with time. Not surprisingly, interesting developments have been made in this direction with ordinary Petri nets and time [27, 28], or high-level nets [20, 29, 30].

Another trend is to employ Petri nets and try to integrate as much as possible continuous aspects in them. In continuous Petri nets, tokens are real numbers [15]. Then this model has been extended and hybrid Petri nets have been defined where there are discrete and continuous places [1, 33]. Negative markings have been introduced in the flow-graphs [19]. Finally places representing batches of product have been introduced in batch Petri nets [6, 16, 46].

Other authors have preferred to associate Petri nets with differential and algebraic equations without achieving a complete integration of the two mathematical tools [2, 3, 4, 13, 14, 40, 49, 50, 51, 53].

Strongly related to the problem of event driven operations in continuous processes, interesting work has been developed in supervisory control and in qualitative simulation for diagnosis in critical industrial processes. In supervisory control, the model in charge of supervision is entirely discrete but interacts with a continuous one which describes the process at a lower level. Petri nets have been used in this context [34, 41, 55]. In the context of qualitative simulation, the Petri net is in charge of defining the time intervals during which simple approximations are valid (for each interval the qualitative model defines the adequate approximation) [17, 25]. Very closely related but with a broader objective, differential equations have been introduced in high-level Petri nets [7, 54].

5.3 The co-operative approach

It is largely accepted that the difficulty of dealing with batch processes is a consequence of the fact that the world of discrete mathematics and the continuous one are very different. The dynamics of a continuous system is expressed by means of differential equations that remain always valid. In a sense, it is similar to logical propositions which remain true once they have been proven. Our feeling is that it is of no use to try to integrate the discrete view and the continuous one in a very close manner if there exists some incompatibility at the level of the available mathematical tools. Integration techniques for differential equations have little to do with state space exploration and sequence characterization.

It is why, when H. Pingaud [14, 40] presented his co-operative modeling technique, we decided to combine our efforts.

In a nutshell, the approach [2, 3, 4, 13, 14, 40, 53], can be seen as the utilization of a Petri net in order to monitor a set of differential algebraic equations. Actually, each time the industrial process changes from one configuration to another, its model, under the form of a set of equations, has to be modified accordingly. However, the configuration changes are driven by threshold crossings and the Petri net marking evolution has to be driven by these events and therefore by the equations.
Only the places denoting a continuous activity have an associated set of equations. This set is loaded into the integrator when a token is put into the place. The initial conditions of the variables and the values of the parameters are assumed to be attributes of the token. The influence of the discrete model on the continuous one is therefore implemented by means of the places. The reverse interaction is implemented by means of the transitions. Thresholds are attached to the transitions. The threshold attached to \( t \) is a predicate involving the variables and the parameters which are involved in the equations associated to the input places of \( t \). The firing date of \( t \) is the first time point for which the associated predicate turns true (the threshold is crossed). When a token is removed from a place (one of its output transition fires), the associated equations are removed from the integrator.

The uninterpreted Petri net (i.e. the places and transitions without the continuous part) precisely describes all the possible configurations of the batch process, and its dynamics from the discrete point of view i.e. the legal sequences of configuration changes. It can be analyzed and give a consistent important view of the process. This discrete dynamics is clearly separated from the continuous one.

This approach is been applied to various case studies, in particular within a joint project with IXI, Alliance Agro-Alimentaire, Elf Aquitaine and Turbomeca which is supported by ANVAR. The major problem is that for all the reachable markings, the set of equations present in the integrator (one subset for each token of the current marking) has to be homogeneous in order to be integrated. We are currently addressing this issue.

6 Conclusion

As we have already written it, when industrial applications are addressed, the major issue concerns the trade-off between fidelity to reality and tractability. Fidelity to reality implies that the model is very comprehensive and detailed. Tractability implies that the mathematical properties of a formal abstract model should not be lost.

In the context of Petri nets, whatever the extensions, it should be possible to be able to extract an ordinary Petri net offering a consistent view of the system. It is clear that this view will be partial and that complementary information should have been provided, for example by means of inscriptions.

Operating this way, a formal analysis is done on the fragment (or on the view) of the system for which it is possible (that depicted by the ordinary Petri net), and a representation of the behavior of the system at a fine grain level is available for simulation or implementation.

It is in this spirit that we have developed our approach for fuzzy Petri nets and for the models of batch processes. It is also in this way that the standard approach in figure 2.a can be formally justified.

It seems to us that, by trying to integrate too deeply various views and various mathematical formalisms, people generally completely lose the analytical
properties of Petri nets. The result is not only the impossibility of formally derive such properties as liveness, boundedness, etc - indeed it is true that this formal analysis is rarely done in practice - it is also a loss of legibility. A method leading to models difficult to understand favours human errors and is eventually not reliable. It will soon be abandoned. It is why, many effort has to be devoted to develop methods allowing to derive Petri net models which are simultaneously powerful and simple.

Acknowlegements

This work would not have been possible without the collaboration of a lot of colleagues. I am particularly grateful to all of them with which I have written my papers. They are too numerous to be listed but they appear in the references. I also want to express my thanks to C. Lansade and his colleagues from IXXI, and to M. Saleman for our fruitful collaboration.

References

32. C. Lansade, P. Tayrac: Dossier final phase 1 - Projet ANVAR Simulation Mixte, Ref. IXI/PAU/97/TLS89, January 1997


This article was processed using the \textsc{LATEX} macro package with LLNCS style