Petri Nets and Production Systems

Manuel Silva, Enrique Teruel[1] *, Robert Valette[2], Hervé Pingaud[3]

[1] DIIS-CPS, María de Luna 3, 50015 Zaragoza, Spain
[2] LAAS-CNRS, 7 Av. Colonel Roche, 31077 Toulouse Cédex 4, France
[3] LGC-ENSIGC-CNRS, Chemin de la Loge, 31078 Toulouse Cédex, France

Abstract. Modern production systems pose a diversity of problems all along their life cycle which are often treated with particular independent formalisms and techniques. Production systems can be viewed as discrete event, continuous, or hybrid systems. Petri nets are a family of formalisms which can be used for the modelling, analysis, implementation, and control of these systems, with the benefit of improving the communication between stages of the life cycle.
The utilisation of Petri nets in several of these stages is illustrated in this tutorial paper through a selected set of examples.

1 Introduction

The behaviour of *production systems (PS)* is often extremely complex, and subtle or even paradoxical phenomena appear. By PS we refer to systems in both the *manufacturing* or the *process* industries, either to *discrete* or *continuous* systems, and also to *hybrid systems*, where these two "extreme" views of dynamic systems are applied to different subsystems.

Among the many diverse problems concerning PS we consider here some issues in the *design* and *operation* from a *systems theory* perspective (i.e., we disregard technological aspects that depend on the nature of the production process, and concentrate in the interaction of subsystems in an integrated system):

- The *design* phase (for a new plant or for extensive modifications), starts at a preliminary stage, when both the basic structure and dimension are determined, and then comes into more and more details until the implementation level is reached.
- The *operation* concerns aspects such as allocation of resources, scheduling, control, supervision, etc. of a working plant.

The range of problems and solutions encountered is extremely broad. As an example, in [13], where the performance evaluation (a kind of analysis) of flow lines (a kind of manufacturing system) is surveyed, the authors devote a section to review the reviews!

* The work of the first two authors was partially supported by Project TIC-94-0242 of the Spanish CICYT and Contract CHRX-CT94-0452 (MATCH) within the HCM Programme of the EU.

Although formal methods may be initially time consuming and difficult to apply, they usually improve the understanding of systems, allowing to identify key parameters and influences, lead to more efficient reasoning, help in the implementation, etc. Moreover, some formal methods facilitate the dialogue between the different people involved in the design and operation, specially when graphical/intuitive representations are provided. The diversity, specificity, and difficulty of the problems leads to the development of particular formalisms and techniques for each problem or class of problems, e.g.: Markov chains, queueing networks, or discrete simulation for performance evaluation; mathematical programming, or MRP for planning; PERT/CPM, or artificial intelligence techniques for scheduling; relay ladder logic diagrams, state diagrams, or algorithmic state machines for local controllers implementation, etc. Of course each kind of problems requires an adequate solution but this may appear as a Babel Tower where the different people at each stage of the design and operation of the PS can seldom communicate, at least formally.

Petri nets (PN) are a family of formalisms which provides a framework or working paradigm for PS design and operation. Broadly speaking, a major contribution of PN to the field is to provide a *family* of formalisms sharing basic principles in a consistent way. Although for each purpose or degree of detail the adequate formalism would be chosen from the family, the transformation from one formalism to another could be sound, if not formal or even automatic. Some appealing characteristics of PN are:

- *Generality*, or descriptive power, both w.r.t. PS types and problems.
- *Adequacy* for dealing with real systems. There is an inevitable modelling trade-off between *fidelity* to reality and *tractability*. If one wants to perform some analysis to gain insight into the basic structure and relevant parameters of a system, their influences, the cause of problems, etc. this will sometimes be paid for by the necessity of making strong assumptions.
- *Ease of use*. (This is naturally a matter of taste to some extent.)

The application of PN to PS is an active research field where many questions still remain open. In this tutorial we illustrate the usability of PN models along the life cycle of PS. By no means we intend to be exhaustive, but just give an impression on the contribution of PN to some classes of problems. We concentrate on some concepts and give illustrative examples, rather than trying to survey the contributions reported in the literature. The reader is referred to existing survey papers or books to gather links to these contributions. Books on or containing chapters about PN in (diverse aspects of) PS are [54] on performance modelling, [16] on PN and Grafcet, [20] on modelling, validation, and performance evaluation, [57] on control modelling, [19] on performance evaluation and control implementation, and [35] on modelling and analysis. Some survey papers are [43,59,42]. A short survey of the utilisation of PN to represent hybrid systems (systems having a discrete and continuous aspects) can be found in [10].

In Section 2 different types of PS and associated problems are briefly overviewed to provide a context for the application domain considered here. The adequacy of PN along the life cycle is discussed in Section 3, making special emphasis on the abstraction levels and different interpretations to cope with the diversity of systems to be modeled or purposes of the models. In Section 4 the modelling of PS of different kinds is illustrated through several examples. Sections 5–8 discuss and illustrate several applications of the PN modelling of PS, from analysing the logic and temporal behaviour, to controlling and monitoring the operation.

2 Production Systems Classification, and their Control

Production systems perform material transformation processes from raw materials to finished goods (where raw and finished are relative terms; for instance, a factory may provide subassemblies to another one, so the "finished" goods of the former are "raw" materials for the latter). The transformation processes consume material, energy, equipment, and labour. The control of these processes involves complex flows of information through the whole system, from the plant to the highest organisational levels.

2.1 Types of Production Systems, and Associated Problems

The classification of production systems can be done using two key criteria. The first one is the "intrinsic" nature of the material being transformed. The second one is the dynamic of the product trajectories through the plant.

Discrete versus continuous systems When the product is composed of discrete parts, it can be quantified in a discrete manner, by an integer number. Typically, manufacturing systems, as in car or aeronautics industry, process such kind of products. Assembly lines, for example, are typical systems where various parts of some device are assembled. In contrast, if the product is a fluid as it is mainly the case in process systems, like chemical or food industry, the characterisation becomes more difficult. Usually, two categories of such processes are distinguished: the continuous processes and the batch ones.

In a *batch process*, the material is operated by finite quantities (the batches). At any time, an integer number of batches are in operation at many different locations in the plant. In this configuration, the process looks like the manufacturing system introduced above, i.e., the batch of fluid in a vessel can be considered as a part. But the complete characterisation of a batch implies real numbers referring to the amount of material and its operating conditions (temperature, pressure, quality, etc.). During the transport of material between two equipments, the discrete nature of the product temporarily disappears. The fluid is continuously transferred through a pipe network from an upstream vessel to a downstream one, naturally by gravity or using energy given by special device (pump, compressor, etc.).

In a *continuous process*, the material continuously flows along the plant without disruption w.r.t. time. Each equipment is dedicated to the same and unique function during the life cycle of the process. Raw materials are fed and products are delivered at known flowrates, the plant is generally operated in steady state which means that all the variables describing the units are controlled to be kept at a constant value during production. There is no discrete representation of the dynamics of the product in a continuous process. Large capacity production plants such as petroleum refineries, heavy chemicals, or natural gas processing units, belong to this kind of processes.

Nevertheless, the distinction between discrete and continuous systems should not be considered as absolute. For instance, in the production of paper, the initial stages are rather continuous (e.g., from pulp preparation to obtaining paper rolls), while others are typically discrete (e.g., cutting and packing the paper products: sheets, envelopes, etc.). Actually, in any kind of PS the continuous and the discrete views are present, for instance, in a car factory, many continuous processes are found, like paint preparation and at a very detailed level a machining operation is indeed a continuous process. On the contrary, it is often possible to extract a discrete view in continuous processes such as chemical ones [55]. Sometimes the continuous and the discrete operations correspond to two different stages of the production system, for example a continuous production of fluid and the discrete packaging operations [48]. They are thus located in two different shops or even in two different plants. In contrast, in other cases, very frequent in fine chemical processes and in food industry, the continuous operations and the discrete ones alternate along the production process. Typically, sterilisation and cooling are continuous operations, done when the fluid is being transferred through pipes from one vessel to another and they take place between reactions and fermentation operations which are discrete ones executed in the reactors.

Classification with respect to the shop architecture Production plants are configured depending mainly on the product variety and the production volume. While uniformity of products and mass production require high efficiency (which can be achieved by a rigid automation) to reach an economy of scale, greater variety and lower production volumes require high flexibility to be able to change the product scenario to react to market changes. In order to conciliate the need of flexibility and efficiency, the concept of *flexible manufacturing systems (FMS)*, involving automated flexible machines, handling, transport, and storage systems, has arisen. The distinction depending on the product variety and the production volume is as valid for discrete goods as for continuous ones: for instance, considering continuous goods, the configuration of an oil refinery — a typical continuous plant — is quite different from that of a pharmaceutical laboratory — a typical batch production plant.

Concentrating for the moment in manufacturing systems, we find several different prototypical kinds of plants:

- *Transfer lines*: The production process is designed as a sequence of operations that take similar amounts of time to complete. The material is moved *synchronously* from one workstation to the next, and each workstation performs repetitively the same task. The main optimisation problem in transfer lines is *balancing*, i.e., making all the operation times in different workstations as similar as possible to reduce idle times and to balance the workload of the stations.

- *Production lines*: When some variability is required in the workstations, e.g., some are operated manually, parts move between them asynchronously. In this case the system does not need to be fully balanced, and possibly intermediate storage, or *buffers*, are introduced to filter out the variations. The efficiency of these systems is greatly affected by the *blocking* and *starvation* phenomena they exhibit. The *buffer allocation* problem becomes crucial: large buffers aim to neglect blocking and starvation, but increase the work in progress, with the consequent economical problems.
- *Flow shop*: When some products may be processed differently in, or even by-pass, some stations, or follow alternative paths, e.g., when producing a unique family of products that differ slightly from one another, the layout of the plant still reflects clearly the material flow, but not as strictly as in transfer or production lines. The *sequencing problem* tries to minimise inventory and production costs, due to machine and tool changeovers, idle times, etc., by finding an appropriate sequence of parts (or lots), naturally constrained to satisfy the production demand.
- *Job shop*: When the variety of products is greater, the layout does not reflect the material flow, because it may differ from one product to another. For each product a *production route* is defined, describing a sequence of machine operations. In principle, the greater versatility is paid by a lesser efficiency: flexibility of the stations make them not so efficient, particularly frequent machine and tool changeovers are required, machines may remain idle a significant part of the time, in-process inventory tends to increase, the transportation between stations appears as a new problem to be solved, etc. The integrated automation of the stations, storage, transportation, and handling becomes highly demanding, leading to FMS. In order to optimise the system performance while satisfying the production demands, complex *decision problems* must be addressed from the long to the short term.

Actually, manufacturing systems exist along a continuous from mass production to job shop, and even different types can be found in the same factory. For instance, in modern car factories, which produce a high variety of versions of one or more models, transfer lines or highly automated production lines are found in body making (presses and assembly), trimming is usually organised in production lines and flow shops, often with manually operated stations, and some complex subassemblies (engines, wiring, etc.) can be produced in flow or job shops within the same factory or in a supplier's one.

For the case of process systems, flexibility can be exploited by the management levels only in the case of batch processes. This is not only due to the possibility of dynamically changing the resource allocations and the trajectories of products as in the case of FMS. It also comes from the possibility of choosing the size of the batches in a continuous range of values. Splitting and mixing batches are classical operations that require to deal with material balance equations in order to correctly model the dynamics of the batch processes.

2.2 Hierarchical Architecture of the Control

The complexity and variety of problems encountered is reflected in a time horizon driven *hierarchy* of control problems, which covers a whole set of levels from the strategic decision one, to the real-time operation one and through the tactical decision one. In the long term strategic level, problems appear, such as the selection of the products, the equipment, its configuration, etc. The solution of these problems affects the design or extensive modification of the plant, under some assumptions on the long-term evolution of the market and taking into account the particular strategy of the company. In the medium term, the part types for immediate processing and their relative ratio, lot sizing, grouping of machines, allocation of resources, etc. must be fixed to respond to market changes. In the short term, scheduling of parts and tools, dispatching of the parts, reaction to disruption, etc. are decisions to be taken in order to satisfy the production plan issued at a higher level to satisfy the precise demand.

It is quite natural that control architectures reflect this hierarchy. Each control level takes as input the outcome of its upper level(s), and produces a reference for its lower level(s). This hierarchical structure implements a "divide and conquer" approach, where the combinatorial number of possible decisions is constrained from the upper to the lower level in order to find a solution. On the other hand, sometimes the upper levels respond to demands issued from lower levels (e.g., the coordination level asks the scheduler how to solve a conflict found in the operation). The separation between levels is not uniquely defined, partially depending on the nature of the PS (mass production, jobbing shop) and on design criteria. Broadly speaking, typical control levels are:

- *Planning.* The whole plant is considered with an estimated demand (unexecuted demands and an estimation of future ones). Further hierarchical refinements into shorter and shorter time horizons can lead to different planning levels, the lowest ones taking into account availability of raw materials, due dates, and available resources.
- *Scheduling.* Each operation (or group of operations) on each part or product is considered individually. The problem is to decide at which *date* a given operation will be performed. If scheduling operates on an estimated state of the PS, it must consider some *slack times*, or precise only the (partial) order of operations to be carried out when possible (the date is only implicitly and relatively given).
- *Global coordination and real-time monitoring.* Its function is to update the state representation of the PS in real- time, to *supervise* it and to *make real-time decisions*, following the schedule, or fixing it when it was implicit.
- *Subsystems coordination* and *local control* perform the actual control of the physical plant in real-time, measuring its state through *sensors* and influencing it through *actuators*.

Some books on PS from a *systems theory* perspective, are [18,54,33,23].

3 The Formal Framework Provided by Petri Nets

In *operational formalisms* (i.e., describing *how* the system is meant to work rather than *what* the system is meant to do), a *system* can be viewed as a collection of *objects* or entities with some attached attributes, some of them *fixed* and some of them *variable* (that define the *state*), and the relations between them. In PN, the state of a system is represented in a *distributed* fashion by the *marking*: places are local state variables whose value (or marking), depending on the chosen abstraction level, ranges from boolean (which is adequate for local control modelling, for instance), to typed and structured (which may be adequate for concise system descriptions). The possible *state changes* are locally defined by way of transitions and the firing or occurrence rule. In *autonomous* PN models, the occurrence rule considers the marking only, so the marking is properly a description of the state of the system. Nevertheless, since transitions are associated to events in the system, and these events may occur depending on "external" considerations, such as the timing, or some signals, implicitly (or explicitly), the state changes involve more information than the marking, so the marking is no longer a sufficient description of the state of the system in many *interpreted* PN models.

PN models of systems are not as different as they may seem at first sight to other models that are familiar to control engineers. For instance, in [41] the PN formalisms are introduced in a way that is close to the state-based description of sampled continuous systems.

We assume the reader is familiar with basic PN concepts [8,16,20,32,39]. Appealing characteristics of PN include:

- Ability to represent in a natural way concurrency, causality, synchronisation, resource sharing, conflicts, bulk or lot movements, etc.
- Locality of states and actions, allowing both top-down and bottom- up model construction, i.e., refinement, modularity, reusability, etc.
- Compactness due to the distributed state representation compared to a centralised sequential representation.
- Adequacy to represent the essential features of a given system by way of the selection of an appropriate abstraction level.
- Interpretability. It is possible to associate a wide range of different meanings and/or connections to the external world to the model objects; different interpretations are appropriate for the different purposes of the model: validation, performance evaluation, scheduling, control implementation, etc.
- Graphic representation facilitating their use in the documentation and monitoring of the system.
- Formal/precise semantics allowing to undertake rigorous analysis or to automate the implementation or code generation either for control or simulation.

The different interrelated *abstraction levels* of PN models, and the different *interpretations* that can be associated to a PN model, make them specially usable along the life cycle (design, preliminary and detailed, implementation, and operation). They define a space of formalisms adequate for different purposes (see

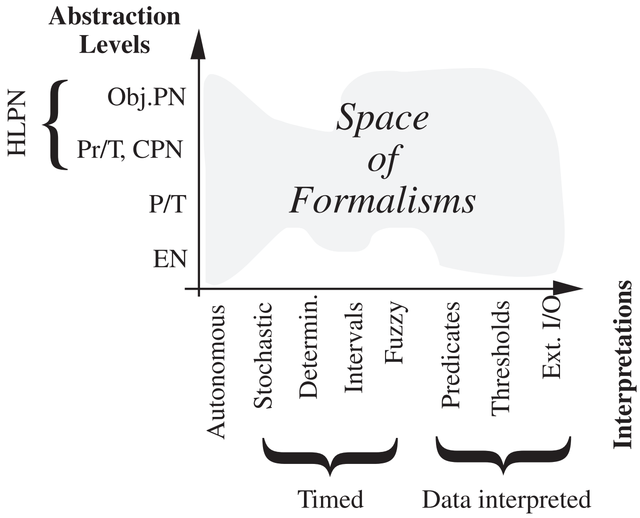

Fig. 1. A pictorial view of the space of PN formalisms.

Figure 1). Broadly speaking, in the abstraction levels axis we have *elementary net systems* where local states are boolean (conditions), *place/transition net systems* where local state variables are counters and bulk processing is allowed, and diverse *high level* formalisms where tokens are distinguishable typed items (i.e., attributes are attached to them) and transition enabling in a given mode requires specific tokens present in the input places. In addition one should consider in this axis some *extensions* such as *inhibitor arcs* or *priorities* which not only affect the conciseness but also the expressive power under some circumstances, or some *subclasses* which aim at improving the tractability at the price of losing modelling power. In the interpretation axis we have the *non-interpreted* or *autonomous* or basic PN model, different *timed interpretations* (stochastic, deterministic, firing times specified by intervals or fuzzy sets, etc., each one with several possible corresponding modifications of the firing rule), and *data interpreted* PN which add extra firing conditions to the transitions (enabled transitions are fired when some *predicates* are true, or when some variables reach some *thresholds* [51] or at the occurrence of *external signals*). Timed interpreted PN are adequate for performance modelling and scheduling, even for hybrid or batch systems [22,25]. Data interpreted PN models incorporate aspects of the systems that are not captured by the autonomous or basic PN model, for example, continuous state variables and differential algebraic equations, or the interaction of the controller with the plant. They are adequate for real-time control, coordination, diagnosis, etc.

These interpretations can be seen as different abstractions of the environment of the model, ranging from total abstraction in the autonomous model (which is totally non-deterministic in the sense that only logical pre-requisites for firings are given, without information on what/when to fire) to total explicitation in the models considering the actual signals (which should be totally deterministic). In between diverse assumptions are made (e.g., describing the time of occurrence of an enabled transition as a random variable or constraining it to be in some interval, associating probabilities or fairness constraints to conflicting transitions, or conditioning occurrences by the result of solving an implicit set of differential algebraic equations, etc.).

These formalisms can be related to other non PN formalisms which are used in common application domains. For instance, PN performance models add synchronisations to classical queueing networks, PN scheduling models add resource constraints and cyclic behaviours to PERT, PN controllers allow to deal with concurrent systems for which state diagrams are impractical, PN in artificial intelligence, compared to classical rule systems, add the ability to handle resources, etc.

PN models — selecting the appropriate formalism — have been used for diverse purposes within the field of PS, including:

- *Modelling*, and *documentation* of a design, to be used in the communication between the different people involved (in the design and also in the subsequent operation).
- *Analysis*, involving, for instance, correctness verification of the logical behaviour of a controller, performance or performability evaluation of a (part of the) plant, etc. Optimisation problems can be addressed to improve the design.
- *Simulation* when the model is too complex (e.g., high level non autonomous PN) for formal analysis. Typically after an analysis and optimisation based on an abstract model (autonomous PN), detailed simulation runs are necessary in order to verify that all the constraints captured by the interpreted PN are met.
- *Control design* aiming at, for instance, guaranteeing the desired logic behaviour (e.g., imposing a mutual exclusion constraint or avoiding a deadlock), or achieving optimum or sub optimum performance during operation (e.g., real-time scheduling).
- *Control implementation.* PN models can be regarded as executable specifications. Diverse approaches to the modelling of the control and techniques for its programmed implementation, including fault-tolerance issues, are available.
- *Monitoring and supervision.* PN models are used as deep models (as opposed to shallow models) for model based diagnosis of the correct operation of the system for fault detection and localisation. For instance, watch dogs are attached to each transition firing in order to detect any deviation between the actual system behaviour and the model.

The use of a single family of formalisms for such a diverse range of problems is not only beneficial from the point of view of communication and reutilisation of results. It has proven to lead also to a synergy, where the concepts and techniques developed in one area help in the solution of open problems in another one [40].

4 Modelling Manufacturing Systems

Model building is mainly a creative, thus difficult to automate, task. Nevertheless for a given kind of systems and problems it may prove useful to somehow limit creativity, either to automatically generate the model from a (graphical high level) description of the plant layout, machines, material handling, storage and retrieval systems, work plan, etc. [53,2,58], or to facilitate the subsequent analysis.

The PN formalisms (low or high level, discrete or continuous, etc.) are adequate to build models of PS. We shall show some examples where some of the appealing characteristics of PN enumerated in Section 3 are illustrated.

In the first example, the model for a simple manufacturing cell the model is built by composition (transition merging and removal of implicit places) of the submodels of the machines, buffer, and the robot. The autonomous model is a non-ambiguous description of the logical behaviour of the system, and can be used for correctness analysis. Incorporating the relation with the environment in terms of signals, this model can be sought as the specification for a logic controller. On the other hand, if appropriate timing is incorporated, performance can be evaluated, or scheduling policies can be investigated.

In the second example another simple manufacturing cell is used to illustrate that a system may exhibit undesired behaviours (namely, deadlock situations), and how PN theory can be used to analyse such problems and synthesise a control policy that avoids them.

In some applications within PS, e.g., coordination, monitoring, diagnosis, information systems modelling, etc., the major issue is expressive power and user-friendliness, rather than tractability. In such cases high level formalisms may be preferred, integrating modern software engineering concepts both regarding methodological and data/knowledge representation aspects, like algebraic specification and object-orientation. The third example shows a coloured PN model of a realistic PS (part of a flexible workshop of a car factory), taken from a case study described, for instance, in [29].

The fourth example illustrates the fact that PN are suitable to capture the discrete part of hybrid systems such as batch processes. The model is built by composition, as in the first example, and it can also be considered as the specification of a controller. This example will also be used to illustrate deadlock detection, the possibility of attaching data to the tokens, to introduce issues induced by the hybrid nature of batch systems, and also to give some hints about the differences between supervisory control and local control.

The fifth example details the issue of hybrid modelling, where the PN model captures the discrete evolution of the system, while differential and algebraic equations associated to it describe the continuous aspects, allowing for the integrated simulation of the whole system.

Example 1 A manufacturing cell — see the layout in Figure 2.a — is composed by three machines (*M1*, *M2*, and *M3*). The work plan is as follows: Raw parts

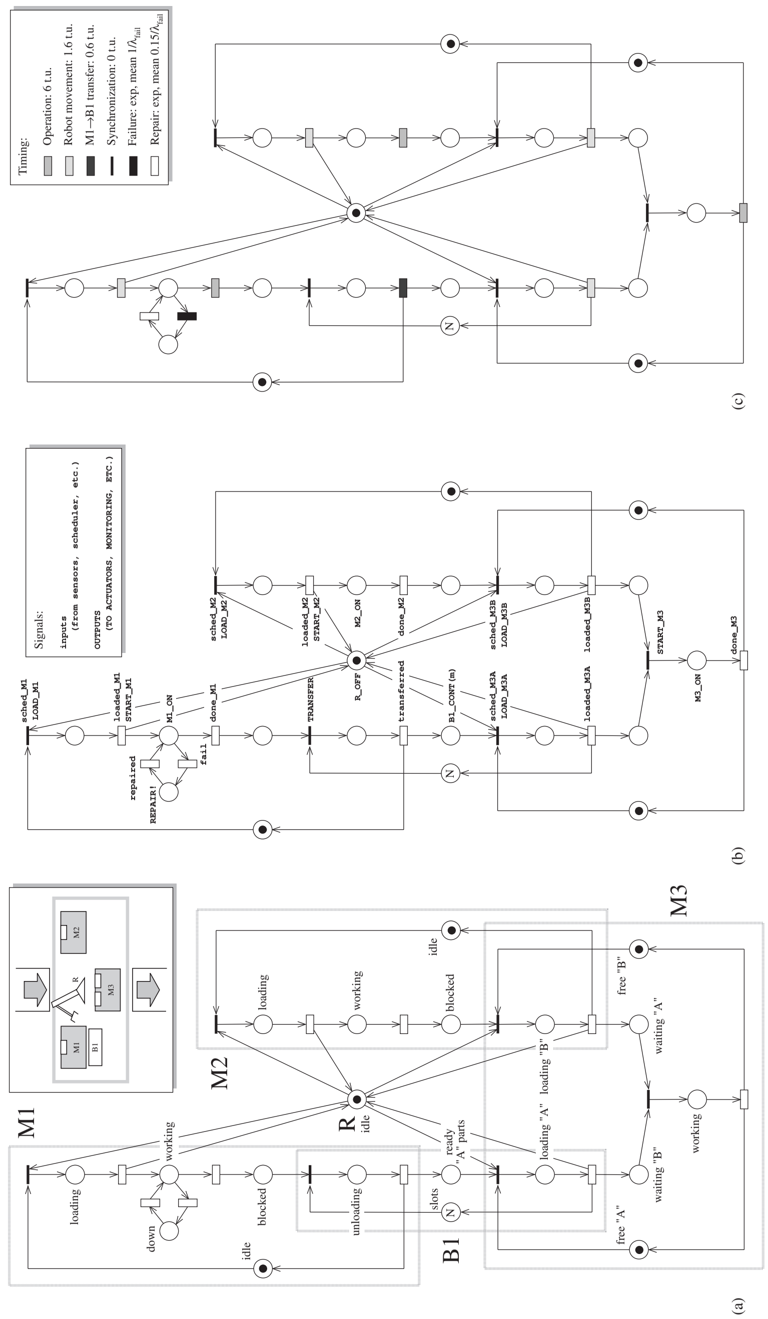

Fig. 2. Different interpretations of a PN model of a manufacturing cell, for diverse purposes.

arrive through a conveyor; A raw part is processed by *M1* to obtain a part of type "A", or by *M2* to obtain a part of type "B"; In *M3* two parts, one of each type, are assembled to obtain a final product, that leaves the cell; We assume saturation (i.e., the cell is never starved or blocked); Parts are handled by a robot (R). We assume that only *M1* may fail (operation dependent failures). To reduce the effect of *M1* failures, it deposits the "A" parts in a temporary buffer (*B1*, capacity N), without using R for this movement.

The PN model in Figure 2.a is self-explanatory. It models both the plant and the work plan, from a local coordination viewpoint. (It goes without saying that operation places could be refined to show the detailed sequence of operations in each machine, etc.) Notice the correspondence of subnets and subsystems (*M1*, *M3*, *M3*, *B1*, and R), and the natural representation of their mutual synchronisations. We have depicted as bars those transitions that represent control events, while transitions depicted as boxes represent the end of an operation, or the occurrence of a failure.

While the autonomous model is useful for some analytical purposes, it does not specify when enabled transitions do occur, or which one is selected in the case of conflict. If the model is meant as a specification for a logic controller, these matters need to be fixed, in addition to the outputs that must be emitted. The inputs, that condition the evolution of the controller, may come from plant sensors (e.g., when R finishes loading *M2* it emits a signal `loaded_M2`) or from other levels in the control hierarchy (e.g., when the scheduler decides — in view of the state of the system and the production requirements – - that *M1* should be loaded, it sends `sched_M1`). The outputs may command the actuators (e.g., `START_M3` initiates the assembly sequence in *M3*) or send information to other levels in the control hierarchy (e.g., `REPAIR!` raises an alarm to call the attention of maintenance staff, or an interrupt that activates automatic recovery; `B1_CONT(m)` updates the number of ready "A" parts in the production database, etc.). The PN model in Figure 2.b captures this information. Following appropriate conventions in the specification (e.g., those imposed in the definition of Grafcet [16]), a model similar to this one could be used directly as a logic controller program.

If the purpose of the model is to evaluate the performance of the manufacturing cell, or to investigate different scheduling policies, then timing information (e.g., duration of operations, mean time between failures, etc.) can be incorporated to the model, for instance specifying the delay in the firing of transitions. Diverse timing specifications are possible (e.g., stochastic, deterministic, time intervals, etc.), each one best suited for a particular purpose or degree of detail required. In Figure 2.c the delays are specified by their mean times. (For performance evaluation we assume later that the distribution of time delays corresponding to operations and movements is *phase-type*, namely *Erlang-3*, while for scheduling we regard it as deterministic.)

Example 2 The cell in Figure 3 may reach a deadlock situation. In this simple cell, two machines (*M1* and *M2*), belonging to two production lines, share two

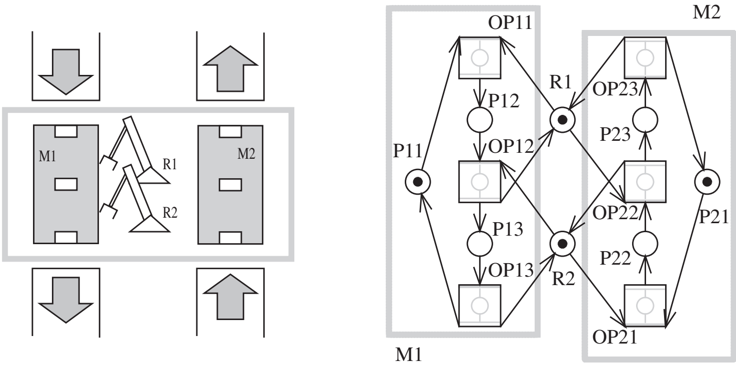

Fig. 3. A multirobot cell that possibly deadlocks, and its PN model.

handling robots (*R1* and *R2*). In each machine, three operations are performed in sequence (*OPij*). We can model each operation as a single transition, whose firing would take some time, or by a path *OPijs* $\longrightarrow$ *OPij* $\longrightarrow$ *OPije*, where the transitions *OPijs* and *OPije* represent the instantaneous events of starting or ending an operation. Machine *M1* requires *R1* to load the parts and assist during *OP11* and *OP12*, and requires *R2* to assist during *OP12* and *OP13* and unload the parts, while *M2* requires *R2* to load the parts and assist during *OP21* and *OP22*, and requires *R1* to assist during *OP22* and *OP23* and unload the parts. A deadlock is possible if, from the initial state shown in the PN model, *OP11* and *OP21* are performed: after completion, *M1* waits for *R2* while *M2* waits for *R1*.

Example 3 Coloured PN models exploit the symmetries of a system. The FMS shown in Figure 4 consists of:

– Several workstations (*S1* to *Sn*). All workstations behave in a similar way: car bodies to be processed are loaded in table L (input buffer of capacity one), then transferred to table P (actual processing), and then transferred to table U for unloading (output buffer of capacity one). For simplicity, we disregard the nature of the precise operations performed in the station, and therefore, we represent a model of a generic workstation. A station behaves as a pipeline with three stages: L, P, and U, which can be active simultaneously, represented by the corresponding places. The complementary places *FL*, *FP*, and *FU* represent, when marked, that the respective stage is free. The colour domain of all these places is $\{1, \ldots, n\}$ for the stations. A token of colour i in P represents that *Si* is processing. Transferring a processed part from table P to table U in *Si* requires i-tokens in P and *FU* and puts i- tokens in U and *FP*.
– An unidirectional transport system, consisting of several roller tables (*T1* to *Tn*). Car bodies input the system in table *T1* and leave it from *Tn*, after

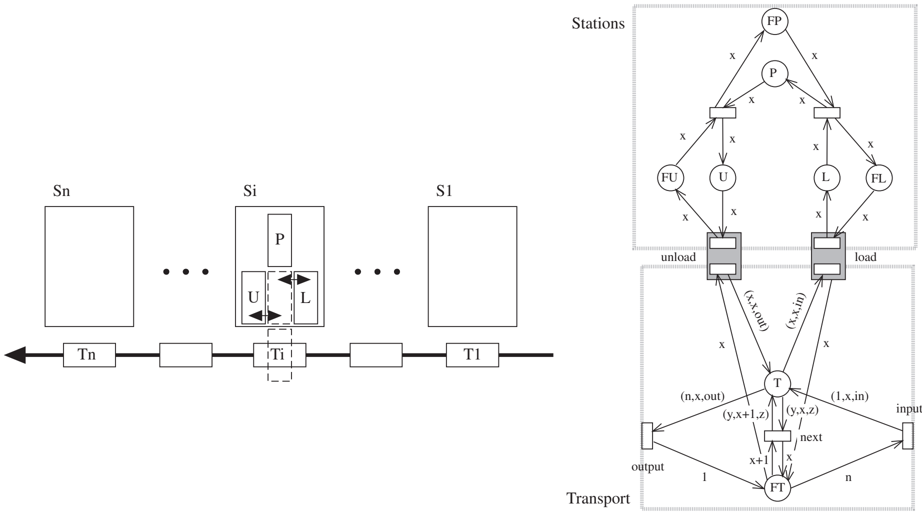

Fig. 4. A flexible workshop that processes car bodies in several stations, and its coloured PN model.

being processed in one station (the one decided by the scheduler). The model for this transport system consists of two places, T and FT, for the occupied and free tables, and transitions to represent the input or output of a car body, a movement to the next table, and the load or unload of a station. The colour domain of FT is $\{1, \ldots, n\}$ for the tables, and the colour domain of T is $(\{1, \ldots, n\}, \{1, \ldots, n\}, \{in, out\})$, where the first field identifies the table, the second the destination station of the car body, and the third the status of the car body (*in* when not yet processed and *out* when ready to leave the cell). Notice that, at the firing of transition *input*, a destination station is assigned to the incoming car body. In net terms, this means solving a conflict between the different firing modes of the input transition. This destination is determined by the scheduler, possibly taking into account the state of the system and production requirements. That is, the scheduler (placed at a higher level) controls the behaviour of the coordination model represented by the coloured PN.

The complete net model is obtained merging the *load* and *unload* transitions of the submodels for the workstations and the transport system. The loading of Si from Ti is represented by the firing of transition *load* in mode i: it consumes a token (i, i, in) from T and an i-token from FL and puts i- tokens in L and FT. Similarly for the unloading, where the "status" colour of the token deposited in T is *out* indicating that the car body in the corresponding table has been processed.

Example 4 Let us consider a simple batch production system. The recipe of the desired product (the recipes in batch systems are similar to part routes in discrete systems) is the following one. Firstly, a batch of raw material (fluid) is charged into a reactor. Then a reaction is executed. Then the batch is transferred to a buffer, and during this transfer, a cooling is done. It has to be pointed out that the reaction is a discrete operation and its duration does not depend on the batch size. In contrast, the transfer from the reactor to the buffer and the cooling is a continuous operation and its duration is proportional to the batch size because the flowrate in the cooling device is a constant which cannot be modified. After a certain stay in the buffer, the batch of product is transferred (another continuous operation) to a reactor for a second reaction and when this operation terminates, the batch is transferred out of the plant.

It is assumed that the production system comprises two reactors named *R1* and *R2* and one buffer. A specific cooling device is available at the input of the buffer. This plant layout is represented in Figure 5.

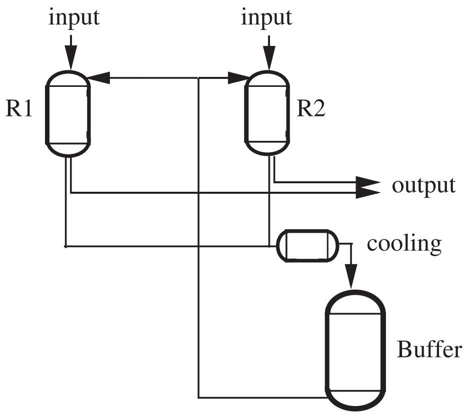

Fig. 5. A simple batch system layout.

The *master recipe* describes the sequence of all the operations which have to be executed in order to obtain the final product from the raw material. It is therefore similar to a kind of abstract production route [22]. The master recipe can easily be represented by a PN. This is illustrated by Figure 6.a. The devices remain unspecified and are just called *vessel_i*, *vessel_j* and *vessel_k*.

Then the *shop recipe* establishes a mapping between the master recipe and the available physical devices. Let us assume that the two reactors *R1* and *R2* are identical and that they can play the roles of *vessel_i* and of *vessel_k*. The *Buffer* and its attached cooling device will play the role of *vessel_j*. The various states of reactors *R1* and *R2* are represented in Figure 6.b and those of the device *Buffer* are represented in Figure 6.c. The model of the shop recipe is

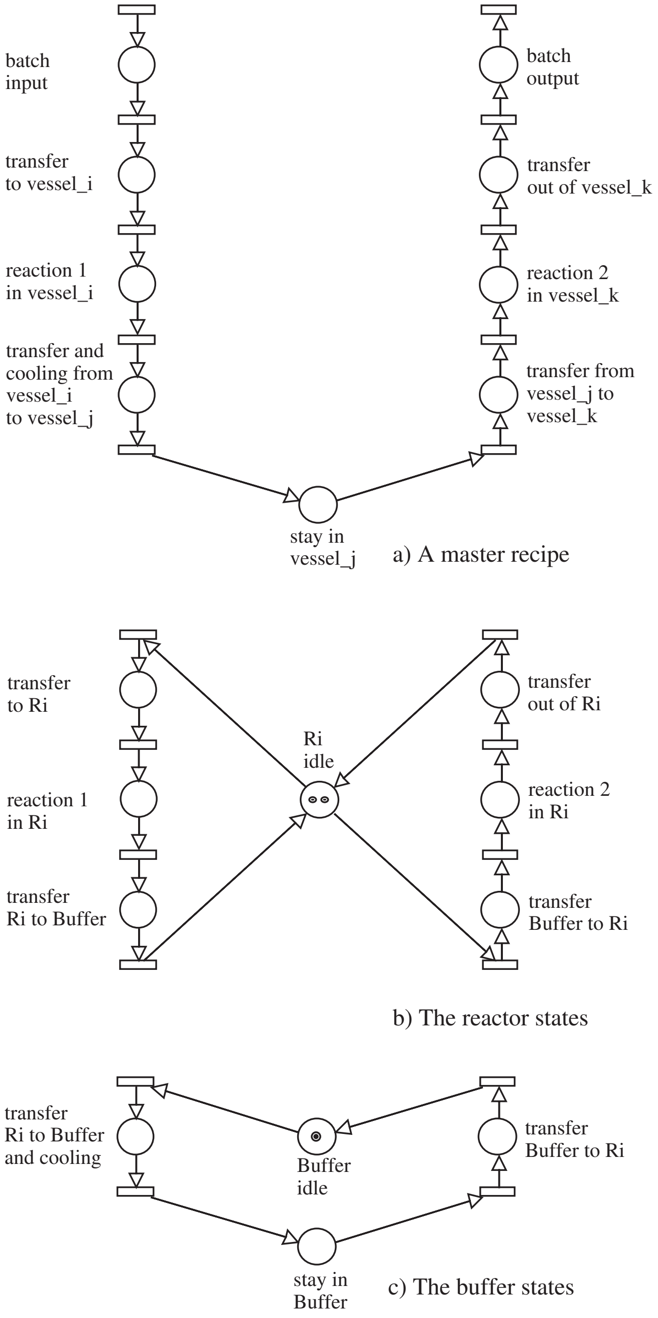

Fig. 6. Building the model.

obtained by the composition (transition merging and removal of implicit places) of the models in Figures 6.a, 6.b, and 6.c. It is represented in Figure 7.

Basically, the modelling process is the same as in the Example 1. The autonomous PN is useful for analytical purposes, but it is too abstract to capture essential features of this system, and associating sensors or durations to the transitions in order to perform simulation or to derive a controller is not so straightforward. For example, transition *input* is fired when the decision of loading a new batch in the process is made. It has therefore to be controlled by some management or supervisory level system. Transition *t1* is fired when enabled. If there is only one reactor (only one token in place *Ri idle*) then it is a mere control event. If there are two idle reactors, then a conflict resolution mechanism has to be implemented, and it is necessary to store the name of the chosen reactor.

Let us now consider the case of transition *t2*. For simulation purposes it is required to attach a duration to it. However, if the batch size is not always the same, the duration is not a constant, and the simple timed PN are not sufficient to capture this. The use of some high level PN is therefore required. The kind of events represented by this transition are generally called *state events* because in a simulation their firing date can only be computed if some state variables of the continuous view are known (the size of the batch in this case).

Transition *t3* represents *at the same time* the end of the reaction and the initiation of the cooling and transfer operation when the buffer is available. Notice that both events must be modelled by a single transition. Typically, in the case of a purely discrete system it would be broken down into two more elementary transitions: one representing the end of operation and the other one the control event (resource allocation and begining of the next operation). However a batch system is a dynamic system in which the continuous part evolves spontaneously. Indeed, it is not possible to wait for the buffer in the reactor because the reaction would go on and the product quality would decrease. Only the cooling operation can stop the reaction. It is very different from a machining operation which will stop anyway, even if the robot necessary to remove the part is not available.

Let us now illustrate the fact that for batch systems the availability of some resource may be of continuous nature. Indeed, the transformation of the *master recipe* into a *shop recipe* is not always as straightforward as in the preceding case. For example, the buffer may contain more than one batch. In this case, as the batches are made up of a fluid, the batches will mix in the buffer. In addition, due to the continuous nature of the two transfer operations (in and out the buffer), they can be done simultaneously. In a first approximation, the condition for starting a transfer from *Ri* to *Buffer* is that the remaining capacity of the buffer is sufficient to store a batch. The condition for initiating a transfer from *Buffer* to *Ri* is that the buffer contains at least one batch of product. If we assume that the buffer capacity is just sufficient for two batches, then the shop recipe is now represented by the model in Figure 8.a. A place denoting the fact that the cooling device is free has to be added.

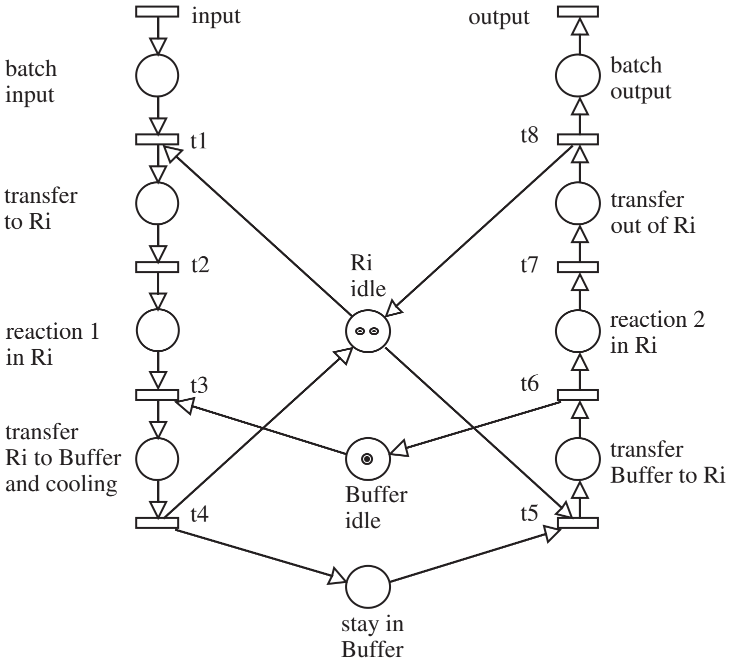

Fig. 7. The shop recipe.

However, we have already pointed out that the batches were not all of the same size. Indeed what has to be produced has to meet exactly the demand, and this not necessarily corresponds to a multiple of the maximal reactor capacity. As the batches mix in the buffer, the exact constraint for allowing a new batch of size s with a discharge d_i to be loaded in the buffer is that, knowing its current level V_0 and its future possible discharge d_o to the next reactor, its future level $V(t)$ will always verify the inequality

$$\forall t \quad V(t) \leq V_{max} \quad V(t) = V_0 + (d_i - d_o).t$$

This issue will be detailed in the next example. We just point out the fact that if we restrict to a pure discrete model, the set of constraints will be delimited by two PN models. The net in Figure 8.a describes a set of sufficient constraints (possibly too restrictive) and the net (without any continuous variables) in Figure 8.b a set of necessary constraints (possibly too permissive). Indeed, if the continuous constraint about $V(t)$ is taken into account, it is not possible to denote it by a place and a token count. A possible solution is to use high level PN and to attach $V(t)$, d_i and d_o as attibutes of the token in place *Buffer*. The value of $V(t)$ will be recomputed each time a transition connected to this place is fired and the values of the parameters d_i and d_o will be updated. This is why double arcs (denoting a self loop) are connecting the place *Buffer* to the transitions *t3*, *t4*, *t5* and *t6*.

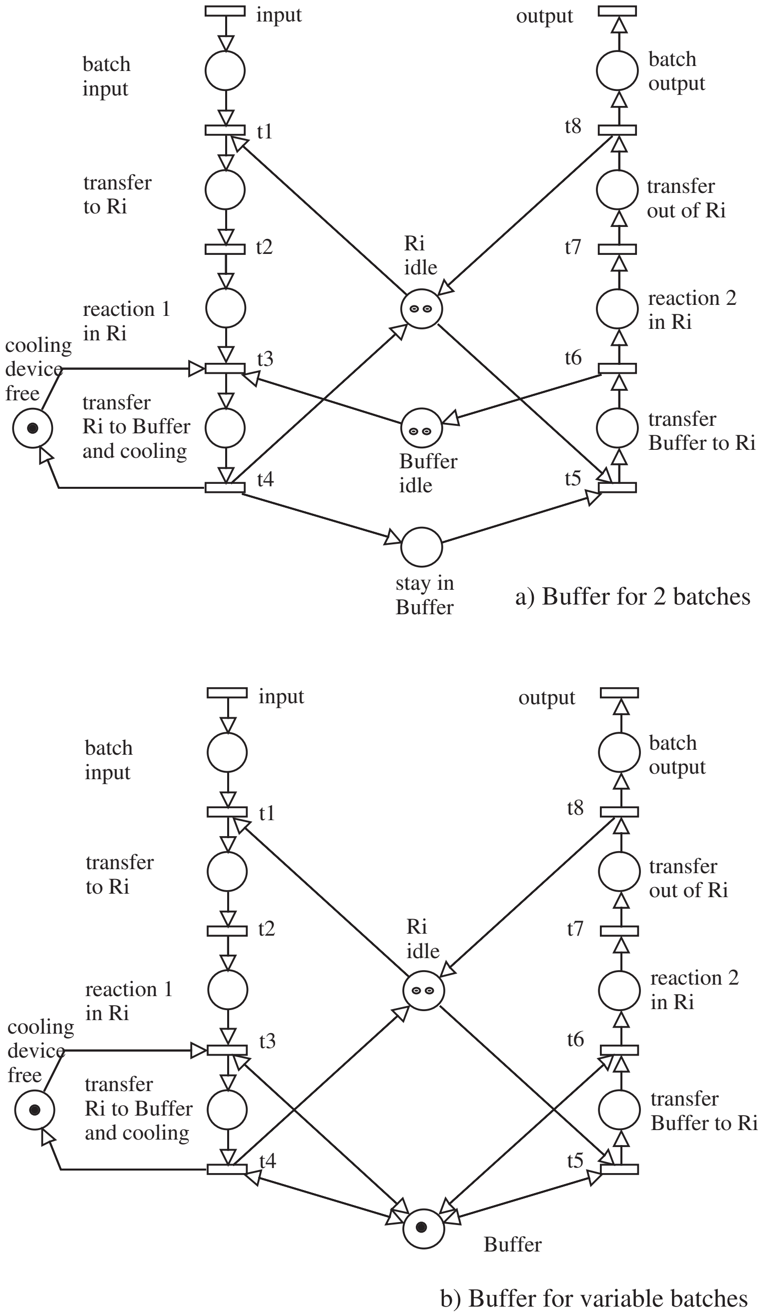

Fig. 8. Another shop recipe.

Let b_1 be the size of the batch of product for *reaction 1*. The time required to transfer it to the buffer is $\Delta t_1 = b_1/d_i$ as the discharge (flowrate) is constant and equal to d_i. The token in place *reaction 1* has b_1 as an attribute and the token in place *Buffer* has V as an attribute. The predicate attached to transition *t3* is the following one (d_i and d_o are assumed constant):

$$(V \leq V_{max}) \wedge (V + b_1 - d_o.\Delta t_1 \leq V_{max}) \tag{1}$$

Indeed, as it is linear, it is sufficient to test the capacity constraint at the beginning and at the end of the transfer.

No predicate is attached to transition *t4*. The arcs connecting it to place *Buffer* are just required in order to update the token attribute V. The predicate attached to transition *t5* is (b_2 is the size of the batch for *reaction 2*, Δt_2 the duration of the transfer out of the *Buffer*):

$$(V \geq 0) \wedge (V - b_2 + d_i.\Delta t_2 \geq 0) \tag{2}$$

Transition *t6* just updates V on the token in place *Buffer*.

If only the discrete constraints (this means the PN structure without the predicates) are considered, it seems that transition *t5* can be fired at any time. Indeed the continuous nature of the batches and of the volume of *Buffer* (batch size b_1 is not necessarily a simple multiple of batch size b_2) breaks down the recipe into two sub-recipes, one for *reaction 1* and one for *reaction 2*. Transition *t5* has to be controlled by the management decision level in the same way as transition *input*.

It is also possible to use extended PN in which the marking of some places is a real (and not an integer) to simultaneously represent the discrete part and the continuous part of the process [15,17,27]. However, restrictions have to be made on the continuous part of the model, so part of the PN theory is no longer valid, and new theoretical developments are needed.

Example 5 Let us consider an example detailing the issue of hybrid modelling. Continuous models of process unit operations in transient state are sets of differential and algebraic equations of the following form:

$$f(X', X, U, t) = 0 \tag{3}$$

with

- X' is the derivative of the state vector X wrt. time,
- X is the state vector of process variables (real numbers),
- U is the parameters of the process model,
- t is time

f is often a non linear and implicit real function of X. Given initial conditions on X, and the values of the parameters U for the corresponding configuration, the system (3) can be solved using appropriate numerical methods [14]. The

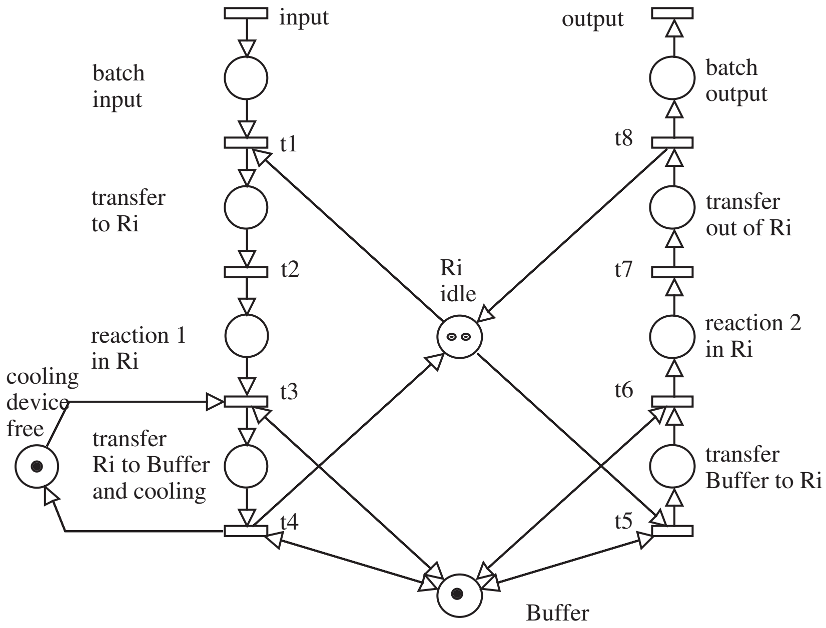

Fig. 9. Less restrictive shop recipe.

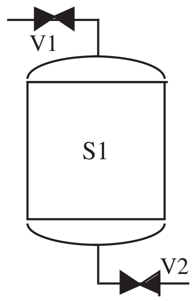

Fig. 10. Physical vessel layout.

parameters U correspond to physical variables which are constant and whose values are determined by the current configuration of the system.

As an example, let us consider the simple case of the process shown in Figure 10. The vessel *S1* is fed with a process fluid at a constant flowrate (F_{in}) when the On/Off valve *V1* is opened. Similarly, the material can leave the tank at constant flowrate (F_{out}) when valve *V2* is opened.

The mass balance on *S1* quantifies the mass hold up (m) of material depending on the state of the valves *V1* and *V2*.

The balance equation could take one of the three different expressions:

- state 1, *V1* is open and *V2* is closed, the mass hold up increases: $f_1(m', m, F_{in}, F_{out}) = m' - F_{in} = 0$
- state 2, *V1* and *V2* are closed, the mass hold up is constant wrt time: $f_2(m', m, F_{in}, F_{out}) = m' = 0$
- state 3, *V1* is closed and *V2* is open, the mass hold up decreases: $f_3(m', m, F_{in}, F_{out}) = m' + F_{out} = 0$

In the equations f_1, f_2 and f_3, m is the only component of the state vector X and F_{in} and F_{out} are the parameters U.

The state where the tank is loaded and unloaded simultaneously is assumed forbidden. Processing one batch of material is done by a sequence of three steps. Initially, the tank is empty. In the first step, the discrete state of the valves is set to (*V1* is open and *V2* is closed) so that the hold up increases until it reaches a pre-specified maximal threshold value (max). It means that function f_1 has to be integrated during this period until the state event E_1 is detected. This occurs when the condition of relation 4 becomes true.

$$E_1 \;:\; m(t) - max = 0 \tag{4}$$

Then, a batch time of 1 unit is needed for the process transformation to be completed. The function f_2 is substituted to function f_1 during this second step and the mass hold up remains at a constant value. In the third and last step, valve *V1* is closed and *V2* is open. Once again, the process model is changed, function f_3 replaces function f_2. The mass hold up decreases until a minimum threshold value (min) is obtained. The time at which this happens is fixed by a state event E_2, as for the first step.

$$E_2 \;:\; m(t) - min = 0 \tag{5}$$

The PN model in Figure 11 is the discrete part of the process model. The different configurations of the two valves are represented by the places *P1* to *P3* corresponding respectively to state 1 to 3. When one of the three states is active, the token in the place can be interpreted as the selection of the right expression of the mass balance. As a consequence, those places monitor the continuous part of the process model, they are called I-place, for Interpreted place. The link between the I-place and the function to be solved is represented in the figure. In this type of hybrid model, output transitions of I-places are all

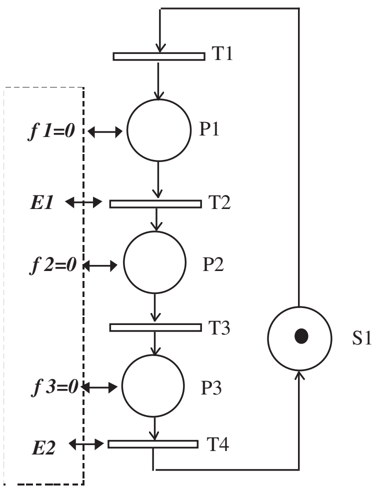

Fig. 11. Petri net repesentation of the vessel configurations.

interpreted. The firing rule is extended to take into account the occurrence of time or state events. *T1* is a control transition that decides the beginning of a batch processing, it cannot be fired as long as the vessel *S1* is allocated to the processing of a preceding batch. Transition *T2* is fired at the occurrence of event E_1, whose occurrence is computed during the continuous simulation. Transition *T4* behaves similarly; it is subject to the occurrence of event E_2. *T3* is only a timed interpreted transition.

5 Qualitative and Quantitative Analysis

One of the purposes of formal modelling is to enable the analysis of the logical and temporal behaviour. Correctness analysis aims to assert on logical properties of the behaviour of a system, while performance analysis aims to estimate with sufficient accuracy relevant indices on its temporal behaviour. PN models, with the appropriate interpretations, can be used for both kinds of analysis. Therefore, a system can be analysed from different perspectives using essentially the same model, even a synergy arises from the interleaving of both views and techniques [40].

Independently of their purpose, PN analysis techniques [8,39,32,1] can be classified as:

- Enumerative: the complete state space (reachability graph, embedded Markov chain, earliest state graph, etc.) is generated.
- Net driven: the net structure is taken into account to assist or facilitate an enumerative analysis. Some examples are: reduction and decomposition techniques, exploitation of symmetries, etc.

– Net based: only the (interpreted) net structure and the initial state, possibly as a parameter, are used in the reasoning, avoiding enumeration.

Additionally, PN models are useful to perform simulations to gain insight on the system behaviour or estimate its performance. Simulation is particularly useful in the case of batch systems, and in general in the case of hybrid systems [14]. Indeed, when modelling the batch system in Example 4 we have pointed out the fact that the discrete view represented by the autonomous PN model was not an accurate description of the actual constraints resulting from shared resources. An hybrid or mixed simulation combining the PN model and differential algebraic equations representing continuous constraints is frequently necessary in order to have an accurate knowledge of the behaviour of an hybrid system under a known control policy [51].

In what follows, we do not overview existing analysis or hybrid simulation techniques, but simply use some of them to analyse the examples from Section 4.

Example 1 Basic reduction rules [20,32] allow to transform the model in Figure 2.a into a marked graph:

1. Every path *start loading* $\longrightarrow$ *loading* $\longrightarrow$ *end loading* is a macrotransition. Therefore it can be reduced to a single *load* transition, preserving the (projected) language, hence liveness, boundedness, reversibility, etc.
2. After the previous step, place *R idle* self-loops around the four *load* transitions, and can be removed preserving the language (i.e., it was an implicit place).
3. The places *working* and *down* in *M1* and their connecting transitions form a macroplace.

The resulting marked graph is strongly connected, so it is bounded, and it does not contain unmarked circuits, so it is live and reversible. By reversibility, the reachability graph is strongly connected, and this allows to deduce ergodicity of the stochastic process with the interpretation given in the example (Figure 2.c), and the irreducibility of the underlying Markov chain.

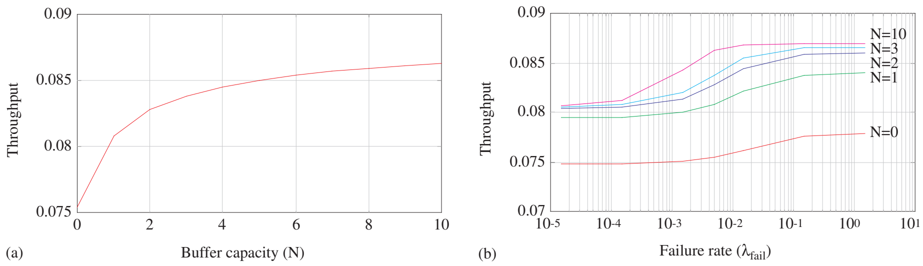

Fig. 12. Performance evaluation of the cell in Figure 2 with respect to buffer capacity and failure rate.

Markovian performance analysis can be used to assist in the dimensioning of *B1*. With given failure and repair rates for *M1*, throughput is plotted versus buffer size in Figure 12.a. Economic considerations (in terms of throughput, required investment, and work in progress) would allow to optimise the buffer size. The plots in Figure 12.b show how the effect of the buffer varies depending on the nature of the failures. Keeping the failure/repair ratio constant:

– Unfrequent failures with long repair times (left side of the plot) make the throughput insensible w.r.t. the buffer size, because the repair time exceeds largely the time to empty the buffer.
– On the other extreme, in the case of very frequent slight failures, a relatively small buffer is able to filter out the high frequency perturbations represented by the failures.
– When the order of magnitude of repair times are similar to the time required to empty the buffer, its size is most critical in order to increase the throughput.

Notice that for the case $N = 0$ the model in Figure 2 is changed, removing *B1* (*M1* becomes essentially identical to *M2*, except for the presence of failures), resulting in a more tight coupling of the machines that leads to a significantly lower throughput.

Example 2 The net system in this example follows a pattern that is frequent in PS and other domains: several sequential processes share some resources. The subclass of PN known as S^3PR [21] has been investigated to deal with this kind of systems. Among other results, when the processes are cycles, as in the example, the existence of deadlock situations is characterised in terms of the existence of net *siphons* (also known as *structural deadlocks*) that are not traps. In this case, the places *P13*, *P23*, *R1*, and *R2* are one such siphon, thus reaching a deadlock is possible. Later we shall explain how a deadlock avoidance control policy can be deduced in this case from the above siphon.

Example 3 It is current practice to build simulation models of PS in order to gain some confidence in the absence of problems. Nevertheless simulation does not guarantee finding such problems. The availability of a formal model allows to prove properties in a definite way. In this case, enumeration analysis proves existence of deadlocks: when all the tables in a given station are occupied and a car body is waiting in the corresponding table of the transport system to enter this station, a deadlock is reached.

Example 4 As we have pointed it out, the set of constraints resulting from the shared resources is not accurately represented by the PN models. The net in Figure 8.a is too restrictive as it only allows to fill the buffer with a new batch if at the current time the remaining volume is sufficient. It does not take into account the fact that, possibly, the buffer is simultaneously being emptied. On the contrary, the net in Figure 8.b is too permissive because it does not take into account the limitation of the size of the buffer. In addition, some operation durations depend on the batch size. When the complexity of the model required for an accurate representation of the system does not allow for a quantitative analysis by stochastic PN, it is possible to execute simulation runs by means of high level PN simulators [26,34].

The batch size can be implemented as a token attribute. Operation durations can be functions involving this attribute and the effective firing of some transition can depend on extra firing conditions based on predicates involving some continuous variables representing some continuous state variables. In this example, the continuous volume of stuff in the buffer $V(t)$ would be such a variable and it will be used in some extra firing conditions for transitions t_3 and t_5 in the net in figure 8.b. The continuous variables are updated when some transitions are fired. For example, variable $V(t)$ will be updated when transitions t_4 or t_6 are fired. In this way it is possible to check whether some specific control policy may provoke an overflow of the buffer or not.

Of course, it is possible to use some stochastic variables in order to take into account the occurrence of failures and abnormal behaviours of the devices. By a Monte-Carlo simulation, the buffer overflow probability under some control and monitoring policy can be derived. It is of the utmost importance for security analysis or to verify the satisfaction of environmental protection constraints, for example.

Example 5 The result of a simulation for one batch of material is reported in Figure 13. The mass hold up is a piecewise linear function defined by three intervals, one for each step. The token load of the places is also reported in the lower part of the figure. It is obvious from an analysis of this figure that the events are the common time points between the two parts of the model. The simulation is based on a synchronous evolution of the continuous and the discrete models, that only meet when an event occurs. This event is detected during the integration of the continuous model (end of the integration horizon), and its consequence on the continuous function is depicted by the firing of the corresponding transition.

6 On Manufacturing Systems Control

Controlling an existing PS means constraining its evolution in order to guarantee the desired logic behaviour or/and to optimise its performance at operation. If the plant to be controlled is modelled as a PN, the control decides the firing or not of enabled transitions. Usually, not every transition can be disabled (e.g., a failure, the completion of an operation, etc.), so transitions can be classified as *controllable* or *uncontrollable*. Controllable points are those at which the decision maker (e.g., a scheduler) influences the behaviour of the system.

Typically, concerning the logic behaviour, it is important to avoid undesirable or forbidden states, such as deadlocks, or to guarantee certain mutual exclusions,

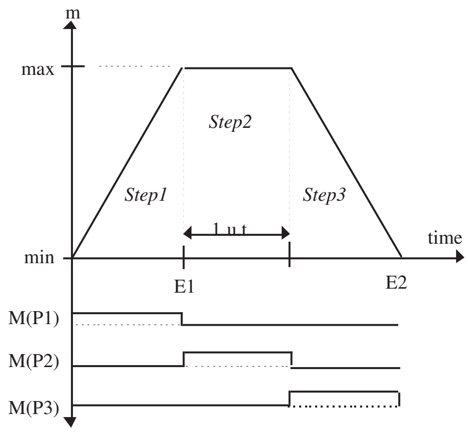

Fig. 13. Mass hold up as a function of time.

while performance control aims to maximise throughput or a more general cost function (e.g., involving also work in progress, machine utilisations, etc.), by determining the firing date for transitions (scheduling). PN with an appropriate timed interpretation are very well suited to the modelling of scheduling problems in parallel and distributed systems. PN allow to model within a single formalism the *functional*, *temporal*, and *resource* constraints. These determine the enabled transitions, and then the scheduling problem is reducing the undeterminism by deciding *when* to fire *which* transitions among the enabled ones. In scheduling theory [11] it is conventionally assumed that tasks are to be executed *only once*. Periodic or cyclic schedules [24] are seldom treated by the theory despite they abound in practice. PN scheduling techniques allow to face these problems. The same as for the analysis, enumerative, net-driven, and net-based approaches can be found in the literature. The computational complexity of scheduling problems leads in practice to sub-optimal solutions obtained using heuristics, artificial intelligence techniques, etc.

Usually, the control receives inputs from the plant, besides of emitting signals to it, so it operates in closed loop (the plant and the control are composed in parallel, in discrete event systems terminology). The same as PN can be used to model and analyse a PS, its control can often be represented within the PN formalism, perhaps incorporating an appropriate interpretation.

In Examples 2 and 4 we give a control policy avoiding the deadlock. In Example 3, besides avoiding deadlocks we consider a control policy to improve the performance. In Example 1 we concentrate on performance control (scheduling) in a simple case where only the allocation of resources is considered (in general, other decisions such as product-mix [9], lot size, etc., need to be taken also into account).

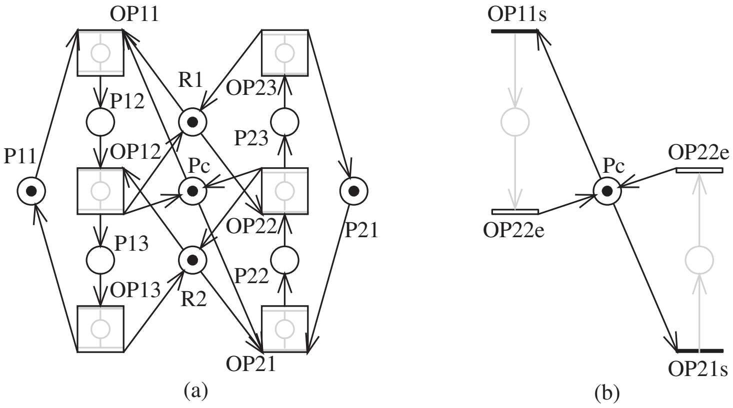

Fig. 14. Adding place Pc to the net model in Figure 3 is a deadlock avoidance control policy.

Example 2 Emptying the siphon $\{P13, P23, R1, R2\}$ shall be avoided in order to avoid the deadlock. In [47] a technique is presented to achieve this goal, which in this case consists in adding a place to the original net, see Pc in Figure 14.a. This place is obtained by addition of the places in the siphon, thus it is structurally implicit. With two tokens it would be implicit, but with just one it forbids the firing of *OP11* or *OP21* when only a token remains in the siphon, thus avoiding emptying it.

Notice that the net in Figure 14.a is the parallel composition of the model of the plant (Figure 3) and the model of the control given in Figure 14.b, where the grey places are not needed, or they can be removed after the composition since they become implicit. From this latter model, it is apparent that we need only controlling the firing of *OP11s* and *OP21s*, and we only observe the occurrence of *OP12e* and *OP22e*.

Example 3 In this case, the deadlock can be avoided by making sure that no more than three car bodies scheduled for the same station are present in the system at any time. This can be enforced by limiting the number of firings of *input* in a given mode w.r.t. the number of firings of *output* in that mode. This is implemented by place O (for orders) in Figure 15.a, whose colour domain is $\{1, \ldots, n\}$ for the destination stations, marked with three tokens of each colour.

Notice that, if O is marked with two tokens of each colour instead of three, unnecessary stoppages in the transport system, that would reduce the throughput, are avoided. These stoppages appear when a car body waits in front of its destination station because this station is processing and the load table is occupied. We cannot proceed to load the third car body until processing is completed, the processed car body is transferred to the table U, and the car body in table L is transferred to table P. In the meanwhile, other car bodies may be prevented from advancing to their destination beyond that station.

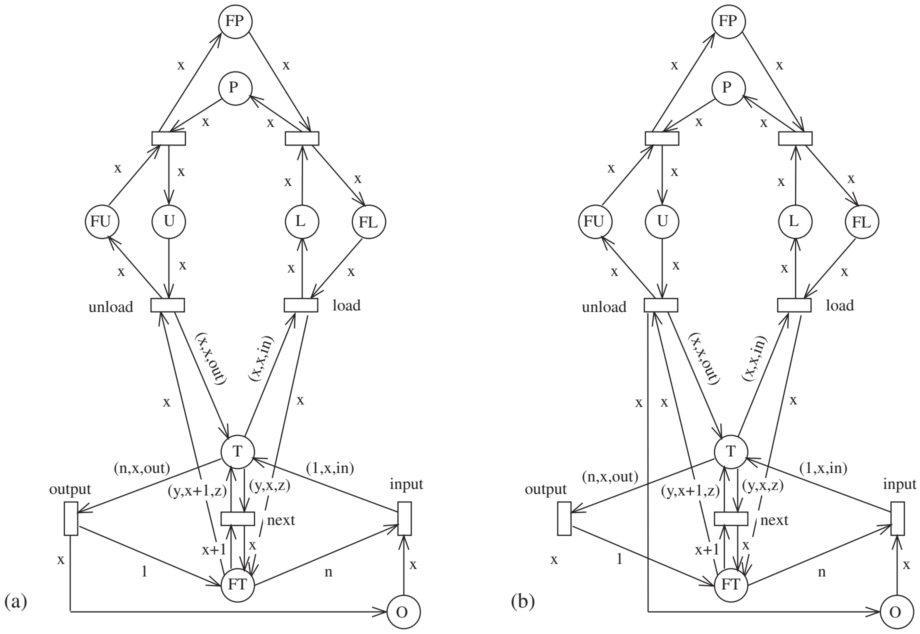

Fig. 15. Adding place O to the net model in Figure 4, with a suitable marking, avoids deadlocks and stoppages.

Finally, in the above control it was assumed that the scheduler controls transition *input* and observes transition *output*. If it observed the occurrences of transition *unload* it would be possible to improve the performance of the control policy by allowing a limited number of *unprocessed* orders in the system (see Figure 15.b).

Example 4 Deadlock avoidance is very important in batch systems. Actually, a deadlock generally results in the loss of one or several batches of product. A batch of fluid cannot be removed by hand as a part. In addition the loss of the batches can have severe consequences for the environment (take for instance the case of nuclear industry).

Let us consider the simple case of a unique size for the batches and of a buffer size equal to one batch as represented in Figure 7. Starting from this model, the problem is now to add a new place (or a new set of places) in order to build a control avoiding deadlocks. Although the production system is very different from that of Example 2, the issue of deadlock avoidance is very similar.

Indeed, places *Ri idle*, *Buffer idle*, *transfer Ro to Buffer*, *transfer Buffer to Ri*, *reaction 2* and *transfer out of Ri* form a siphon. Therefore, if it is emptied it will remain empty, as it can occur if three batches are simultaneously introduced (the first one is processed until the buffer and the two others until *reaction 1*).

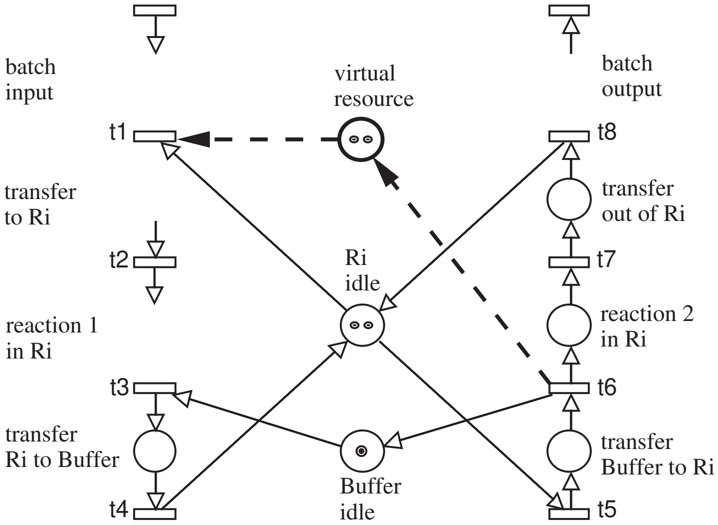

Fig. 16. The batch process siphon.

Among the transitions connected to the places of the siphon, transition t_1 is the only one consuming a token, and t_6 is the only one increasing the global token load of the siphon (one token is consumed in *transfer Buffer to Ri*, one is produced in *reaction 2* and one is produced in *Buffer idle*).

In order to avoid emptying the siphon, transition t_1 must be controlled. Let us introduce a new place *virtual resource* and the dotted arcs as represented in Figure 16. The variations of its token load will exactly be the same as that of the siphon. Initially, the global token load of the siphon is three. If the initial token load of place *virtual resource* is two, then it can be shown, either using siphon results [7] or net reduction techniques [20], that the deadlock is avoided.

For planning and scheduling place *virtual resource* is a new constraint which prevents the generation of any production plan exhibiting the possibility of a deadlock.

Example 1 Assume that, after the optimisation of the design that involved performance evaluation as discussed in Section 5, the capacity of the buffer is fixed to two. Although the plant parameters are fixed, the actual performance of the system may vary depending on how it is controlled.

As it was shown in Figure 2.b, which represents the control at a coordination level, the scheduler controls the evolution by enabling/disabling the transitions that initiate robot load operations (i.e., these are the controllable transitions).

Figure 17 shows the Gantt charts of two possible scheduling policies assuming deterministic timing and disregarding failures. In Figure 17.a operations are scheduled as soon as possible, solving eventual conflicts in the allocation of the

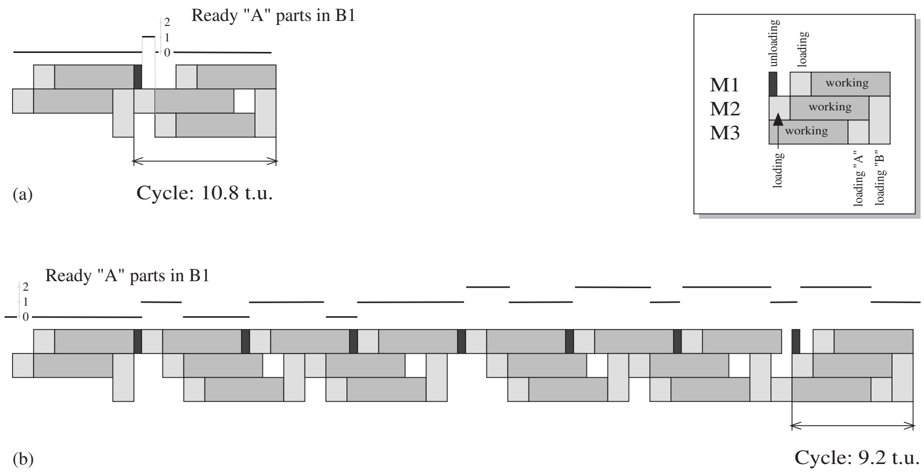

Fig. 17. Effect of different scheduling policies in the manufacturing cell of Figure 2.

robot by fixed priorities (*M2* is prioritary over *M1*). A periodic regime is quickly reached, in which:

- The cycle time is 10.8 (i.e., throughput 0.0926 *without failures*).
- The buffer contains at most one part, so parts are not accumulated to be used in the event of a failure.

The Gantt chart in Figure 17.b shows the evolution when the scheduler prevents interrupting *M1* until it gets blocked, and interrupting *M2* and *M3* from then on. This policy fills up the buffer to be prepared for eventual failures and achieves a cycle time of 9.2 (i.e., throughput 0.1087) in normal operation.

Observe that, in normal operation, the behaviour is cyclic, so the control can be represented or implemented by a regulation circuit net synchronised with the control transitions.

7 Implementation Issues

Once a suitable PN model for a controller has been obtained it has to be *implemented.* Basically an implementation is a physical device which emulates the behaviour expressed by the model. One advantage of using PN as a specification formalism is their independence w.r.t. the precise technology (pneumatic, electronic, etc.) and techniques (hardwired, microprogrammed, etc.) of the final implementation. Presently, in PS control, programmed implementations are the most usual, running on a wide range of computer systems (e.g., industrial PC's, programmable logic controllers, etc.).

The (programmed) implementation is affected by the selected PN *formalism* (low or high level, different interpretations of the firing rule), the *algorithmic approach* (interpreted, where the PN model is a data structure, or compiled, where a program is obtained from the given PN; centralised or parallel/distributed schemes), and the *computer architecture* (high or low level programming language; single or multi processor).

For the case of local controllers specified by low level PN with input and output signals (like that shown in Figure 2.b), a usual choice are "token players" [50,39]: the basic schema is a cyclic program that reads the inputs, computes the evolution of the marking, and generates the outputs once and again. A major issue is the efficient computation of enabled transitions. An example of an efficient technique for this purpose are *representing places* (see, for instance, [12]). The idea is to appropriately select one input place per transition (its *representing place*). It is always possible (perhaps after some net transformations) to classify places as either representing or *synchronisation places*, where each of the former is the representing place of all its output transitions. The marked representing places are kept in a list (we assume safeness for simplicity), that is updated at each transition firing. In each cycle, only the output transitions of marked representing places are tested for enabledness, eventually checking the marking of some synchronisation places. A possible selection of representing places for the net in Figure 2 are all but *R idle*, *slots*, *ready* "A"*parts*, *waiting* "A", and *free* "B" (thus, these would be the synchronisation places).

The inherent parallelism captured by a PN model is somehow dismissed in centralised implementations. Diverse parallel and distributed implementations have been proposed (see, for instance, [12]). The structure theory of PN allows to identify certain components in a given net that are useful for distributing or parallelizing the implementation. Particularly, live and safe state machine components lead to cyclic sequential processes that can be directly implemented, for instance, as Ada tasks. In such case, other places can be represented as global variables, semaphores, etc. Coming back to the example, we easily identify *M1* and *M2* as sequential tasks, *M3* can be decomposed into two synchronised sequential tasks, *slots* and *ready* "A" *parts* are semaphores, and *R idle* is a mutual exclusion semaphore.

In the implementation of higher control levels, some convergence has appeared between the fields of PN and artificial intelligence (see, for instance, [30], [49]). In this sense, transitions play the role of *rules* while the *working memory* can be split into several nodes corresponding to the respective input places. With respect to classical PN implementations the search for enabled transitions is carried out by the *matching phase* in the rule system, which can take advantage from the partition into local working memories. For the selection phase transitions can be grouped into *conflict sets* by inspecting the net structure, and each one can be provided with a particular resolution strategy.

An important issue when designing a control system is that of *safety.* Formal modelling and analysis tools are needed to engineer safe computer-controlled systems. For this task it is necessary to consider both the control system and its environment, for which PN are a suitable formalism [28]. When faults can happen the controller should be able to detect them and even react appropriately degrading system's performance as little as possible.

Let us concentrate here on the detection and recovery of faults in the controller itself, while detection and recovery of faults in the process is covered in Section 8. Several techniques have been proposed to produce safe and/or *fault-tolerant* PN based controllers. We illustrate next two of these techniques which are supported by PN theory: the *spy/observer* schema and application of *coding* theory.

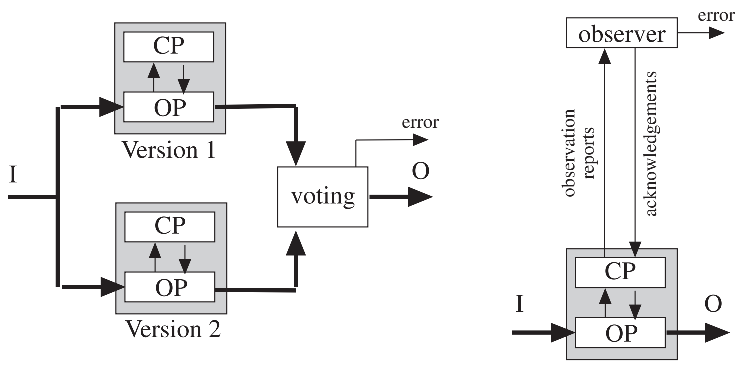

Fig. 18. Duplication versus observation.

In *N-version* programming techniques, several versions of the controller are implemented, and a voting mechanism is introduced [5]. A less expensive schema is based on the idea of an *observer* [6] or *spy* [52], which accepts "normal" behaviours seen through some *observable*, or *check*, points. Figure 18 duplication and observation schemes are compared. The observable points are transitions whose firing is reported to the spy/observer (transitions are classified as observable or non- observable, dually to the classification into controllable and uncontrollable). The spy/observer can be modelled as a PN equivalent to the original one w.r.t. observable transitions (non observable transitions are considered silent and can be reduced). In the final implementation, the code corresponding to the spy is merged with the code of the proper controller.

Coming back to Example 1, considering as observable all the synchronisation transitions in the net (i.e., those corresponding to the initiation of robot operations, initiation of a transfer from *M1* to *M2*, and initiation of an assembly in *M3*) the corresponding spy is shown in Figure 19. (Notice that this spy is obtained applying the same reduction rules that were applied for the analysis.)

For the application of coding theory [31] concepts to fault detection/recovery we can consider the marking as a *word* and the set of reachable markings as a *code*. Possibly, in a given PN model, the *Hamming distance* between words (markings) in the code is not large enough to allow the desired detection/correction. This distance can be increased by adding appropriate redundancies, so it naturally comes to mind the addition of *implicit places* [38,45] because they preserve the behaviour.

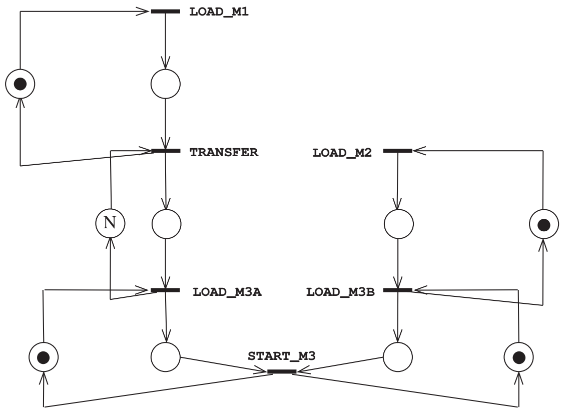

Fig. 19. A spy for the net in Figure 2.

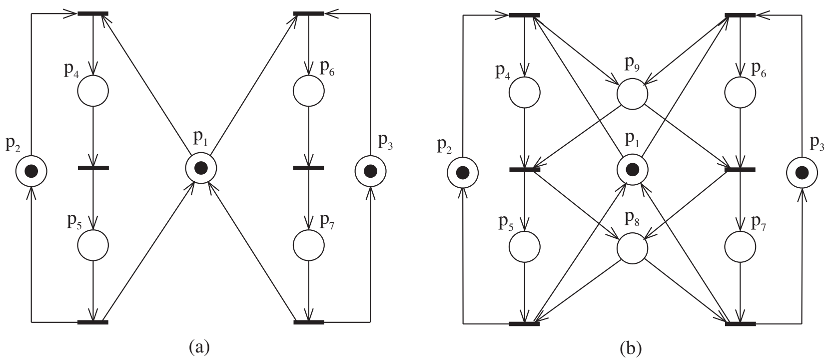

Fig. 20. Adding implicit places to increase the Hamming distance.

Take for instance the PN model in Figure 20.a. The Hamming distance in this case is two, which is the distance between the reachable markings $p_3 + p_4$ and $p_3 + p_5$ — the marking of p_4 and p_5 differ — or between $p_2 + p_6$ and $p_2 + p_7$ — differring in p_6 and p_7. By adding the implicit places p_8 and p_9, as shown in Figure 20.b, we increase the Hamming distance to four. Notice that, in the new net, the former markings $p_3 + p_4$ and $p_3 + p_5$ become $p_3 + p_4 + p_9$ and $p_3 + p_5 + p_8$, respectively — now they differ in the marking of p_4, p_5, p_8, and p_9 — and the former markings $p_2 + p_6$ and $p_2 + p_7$ become $p_2 + p_6 + p_9$ and $p_2 + p_7 + p_8$, respectively — now differring in p_6, p_7, p_8, and p_9. While a Hamming distance of two only guarantees detecting all single errors, a Hamming distance of four guarantees detecting all triple errors, and correcting all single ones. For example, we are able to detect that $p_3 + p_4 + p_5 + p_9$ is not reachable. Assuming that it suffers from a single error, we would correct it leading to $p_3 + p_4 + p_9$, which is reachable (notice that it is the only reachable marking at distance one from the given non reachable one).

A generalisation of this schema, disregarding the enabling constraints of the added places leads to the notion of *test places* [45], with improved applicability.

8 Supervision and Monitoring

8.1 Principles

The supervisory control and monitoring function has three main objectives:

- Implementing the provisional production plan which has been generated at upper management levels. This means that it has to detect the end of the operations and to start the execution of new operations when the resources are available and the planned date reached.
- Monitoring the plan execution. This means that if the latest starting date is reached for some operation (e.g., because the previous operation has not terminated or because the required resources are still allocated for another operation) then the violation of the provisional plan is detected. A new plan has to be elaborated, that is, new starting and ending dates are calculated taken into consideration the actual state of the production system (constraint propagation).
- Monitoring the behaviour of the physical system. This means that any failure in the various devices has to be detected and that recovery procedures have to be executed in order to avoid a failure of the whole production system and to guarantee the security and environmental constraints.

A typical architecture of a control system is represented in Figure 21. At the lower level, continuous controllers, programmable logic controllers (PLCs) and other control devices are found. Then there is the supervisory and monitoring level and then the management levels in charge of planning and scheduling. The supervisory level interacts with the management levels (it receives the provisional plan and sends reports on the actual production system behaviour) and with the

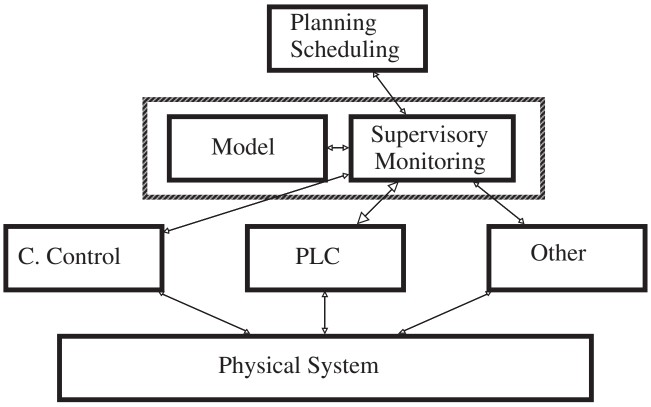

Fig. 21. Architecture of a control and monitoring system.

local control (it sends the commands for allocating the resources and initiating the operation executions, and it receives end of operation signals).

Typically, the supervisory level is based on a model of the production system which is emulated in real-time [3,4]. This model is used

- to store the current state of the physical system,
- to check that the received signals are consistent with the current state,
- to evaluate (by simulation) the future behaviour of the system after a decision.

8.2 Model for supervision and monitoring

When the model is PN based, time intervals are attached to each transition [36,37,46,56]. If the transition corresponds to the beginning of an operation, this time interval is the interval between the earliest starting date and the latest starting date as determined by the scheduling. If it corresponds to an end of operation, then the time interval corresponds to the normal duration of the activity. Transitions have to be fired within these time interval. If the signal of an end of operation is received when the corresponding transition is not enabled, or before the earliest date of the time interval, this means that the physical system behaviour differs from that of the model. A fault is detected and a diagnosis has to be done. An efficient diagnosis requires more information than that included in the model. Indeed, the model only includes correct (or normal, or nominal) behaviour whereas a diagnosis requires a model of the behaviour of the system when some equipment have failed (a model of the possible failures). In addition, the PN models represent the behaviour of the production system and it is often important to know the architecture of the system (the fact that some device is physically next to another one may be important to analyse the way a fault may propagate).

In the case of batch processes, a purely discrete model may be insufficient. It may be necessary to use PN models in which thresholds are attached to the transitions [3,4]. These thresholds involve continuous state variables. For instance in the Example 4, if the model in Figure 8.b is used, it is absolutely necessary to keep track of the current volume of product in the buffer, $V(t)$, in order to check if there is no overflow. It is also necessary to keep track of the batch sizes, for example as an attribute of the corresponding token, in order to compute the normal firing dates of transitions such as *t2* or *t4* as the durations of the operations associated with their input places are proportional to the batch sizes. (See [4] for more details.)

9 Concluding Remarks

The adequacy of PN to deal with a diversity of problems in the design and operation of PS (discrete, continuous, or hybrid), including modelling, analysis, control, implementation, and monitoring, has been discussed and illustrated through several examples.

The formal framework provided by PN, and particularly their representation of the structure of systems, has proved helpful in the treatment of hard problems in this and other domains. Moreover, the notion of a *family of formalisms* allows to adapt to particular problems and domains without losing the possibility of mutual communication and reutilisation of results. A number of PN computer tools have been developed to assist in the modelling and qualitative or quantitative analysis. Several tool descriptions can be accessed through `http://www.daimi.aau.dk/PetriNets/`, the Web page on PN maintained by DAIMI, Aarhus University.

Despite the great amount of work and achievements, much work remains to be done to meet industrial requirements.

References

1. M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets.* Wiley, 1995.
2. S. Amar, E. Craye, and J. C. Gentina. A method for hierarchical specification and prototyping of flexible manufacturing systems. In *Proc. IEEE Workshop on Emerging Technologies and Factory Automation*, pages 44–59, Melbourne, Australia, 1992.
3. D. Andreu, J.C. Pascal, H. Pingaud, R. Valette. Batch process modelling using Petri nets. In *1994 IEEE International Conference on Systems, Man and Cybernetics*, pages 314–319, San Antonio, USA, October 1994.
4. D. Andreu, J.C. Pascal, R. Valette. Events as a Key of a Batch Process Control System. In *CESA'96 IMACS Multiconference, Symposium on Discrete Events and Manufacturing Systems*, pages 297-302, Lille, July 1996.
5. A. Avizenis and J. P. Kelly. Fault tolerance by design diversity: Concepts and experiments. *Computer*, 17(8):67–80, 1984.

6. J. M. Ayache, P. Azema, and M. Diaz. Observer, a concept for on line detection for control errors in concurrent systems. In *Proc.* 9^{th} *IEEE Int. Sypm. Fault-Tolerant Computing*, pages 79–86, Madison, WI, USA, June 1992.
7. K. Barkaoui, J. F. Pradat-Peyre. On liveness and controlled siphons in Petri nets. In *Application and Theory of Petri Nets*, Lecture Notes in Computer Science 1091, pages 57–72, Springer, 1996.
8. G. W. BRAMS. *Réseaux de Petri: Théorie et Pratique.* Masson, 1983.
9. H. Camus, H. Ohl, O. Korbaa, and J. C. Gentina. Cyclic schedules in FMS with flexibilities in operating sequences. In Silva et al. [44], pages 97–116.
10. R. Champagnat, P. Esteban, H. Pingaud, R. Valette. Petri net based modelling of hybrid systems. In *ICIMS-NOE ASI'96 Conference, Life cycle approaches to production systems, management, control, supervision*, pages 53–60, Toulouse, France, 1996.
11. P. Chretienne, E. G. Coffman, J. K. Lengstra, and Z. Liu, editors. *Scheduling Theory and its Applications.* Wiley, 1995.
12. J. M. Colom, M. Silva, and J. L. Villarroel. On software implementation of Petri nets and colored Petri nets using high-level concurrent languages. In *Proc.* 7^{th} *European Workshop on Application and Theory of Petri Nets*, pages 207–241, Oxford, England, July 1986.
13. Y. Dallery and S. B. Gershwin. Manufacturing flow line systems: A review of models and analytical results. *Queueing Systems: Theory and Applications*, 12:3–94, 1992.
14. B. Daubas, A. Pages, H. Pingaud. Combined simulation of hybrid processes. In *1994 IEEE International Conference on Systems, Man and Cybernetics*, pages 320-325, San Antonio, USA, October 1994.
15. R. David and H. Alla. Continuous Petri Nets. In 8^{th} *European Workshop on Application and Theory of Petri Nets*, pages 275–294, Zaragoza, June 1987.
16. R. David and H. Alla. *Petri Nets and Grafcet.* Prentice-Hall, 1992.
17. I. Demongodin, N. Audry, F. Prunet. Batches Petri Nets. In *IEEE International Conference on Systems, Man and Cybernetics*, pages 607–617, Le Touquet, France, October 1993.
18. A. Desrochers, editor. *Modeling and Control of Automated Manufacturing Systems.* IEEE Computer Society Press, 1989.
19. A. Desrochers and R. Y. Al-Jaar. *Applications of Petri Nets in Manufacturing Systems.* IEEE Press, 1994.
20. F. Dicesare, G. Harhalakis, J. M. Proth, M. Silva, and F. B. Vernadat. *Practice of Petri Nets in Manufacturing.* Chapman & Hall, 1993.
21. J. Ezpeleta, J. M. Colom, and J. Martínez. A Petri net based deadlock prevention policy for flexible manufacturing systems. *IEEE Trans. on Robotics and Automation*, 11(2):173–184, 1995.
22. H. Genrich, H-M. Hanisch, K. Woellhaf. Verification of recipe-based control procedures by means of predicate/transition nets. In *Application and Theory of Petri Nets*, Lecture Notes in Computer Science 815, pages 278–297, Springer, 1994.
23. S. B. Gershwin. *Manufacturing Systems Engineering.* Prentice-Hall, 1994.
24. C. Hanen and A. Munier. Cyclic scheduling problems: An overview. In Chretienne et al. [11].
25. H. M. Hanisch. On the use of Petri nets for design, verification and optimization of control procedures for batch processes. In *1994 IEEE International Conference on Systems, Man and Cybernetics*, pages 326-330, San Antonio, USA, October 1994.

26. K. Jensen, et al. Design/CPN, Reference manual, Meta Software and Computer Science Department, University of Aarhus, Denmark 1997. On-line version: http://www.daimi.aau.dk/designCPN/
27. J. Le Bail, H. Alla, and R. David. Hybrid Petri Net. In *European Control Conference*, p.1472-1477, Grenoble, France, July 1991.
28. N. G. Leveson and J. L. Stolzy. Safety analysis using Petri nets. *IEEE Trans. on Software Engineering*, 13(3):386–397, 1987.
29. J. Martínez, P. Muro, and M. Silva. Modeling, validation and software implementation of production systems using high level Petri nets. In M. Silva and T. Murata, editors, *Invited Sessions: Petri Nets and Flexible Manufacturing. IEEE Int. Conf. on Robotics and Automation*, pages 1180–1185, Raleigh, NC, USA, April 1987.
30. J. Martínez, P. Muro, M. Silva, S. F. Smith, and J. L. Villarroel. Merging artificial intelligence techniques and Petri nets for real time scheduling and control of production systems. In R. Huber et al., editors, *Artificial Intelligence in Scientific Computation*, pages 307–313. Scientific Publishing Co., 1989.
31. F. J. McWilliams and N. J. A. Sloan. *The Theory of Error-Correcting Codes.* Handbooks in Operations Research and Management Science. North-Holland, 1981.
32. T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.
33. H. T. Papadopoulos, C. Heavey, and J. Browne. *Queueing Theory in Manufacturing Systems Analysis and Design.* Chapman & Hall, 1993.
34. C. Proix, C. Lansade. MissRdP5 Manuel, de reference, IXI, Toulouse, January 1997, (IXI/TLS/mrdp/MdR/D63.E).
35. J. M. Proth and X. Xie. *Petri Nets. A Tool for Design and Management of Manufacturing Systems.* Wiley, 1996.
36. A. Sahraoui, M. Courvoisier, R. Valette. Some considerations on monitoring in distributed real-time control of flexible manufacturing systems. In *International Conference on Industrial Electronics, Control and Instrumentation, IECON 86*, p.805-810, Milwaukee, USA, Sept. 1986.
37. A. Sahraoui, H. Atabakhche, M. Courvoisier, R. Valette. Joining Petri nets and knowledge based systems for monitoring puposes. In *IEEE International Conference on Robotics and Automation*, p.1160-1165, Raleigh, USA, April 1987.
38. J. Sifakis. Realization of fault-tolerant systems by coding Petri nets. *Design Automation and Fault-Tolerant Computing*, 3(2):93–107, 1979.
39. M. Silva. *Las Redes de Petri: en la Automática y la Informática.* AC, 1985.
40. M. Silva. Interleaving functional and performance structural analysis of net models. In Ajmone Marsan (ed.), Application and Theory of Petri Nets 1993, volume 691 of *Lecture Notes in Computer Science.* Springer, 1993, pages 17–23.
41. M. Silva and E. Teruel. A systems theory perspective of discrete event dynamic systems: The Petri net paradigm. In P. Borne, J. C. Gentina, E. Craye, and S. El Khattabi, editors, *Symposium on Discrete Events and Manufacturing Systems. CESA '96 IMACS Multiconference*, pages 1–12, Lille, France, July 1996.
42. M. Silva and E. Teruel. Petri nets for the design and operation of manufacturing systems. *European Journal of Control*, 3(3), 1997.
43. M. Silva and R. Valette. Petri nets and flexible manufacturing. In G. Rozenberg, editor, *Advances in Petri Nets 1989*, volume 424 of *Lecture Notes in Computer Science*, pages 374–417. Springer, 1989.
44. M. Silva, R. Valette, and K. Takahashi, editors. *Procs.* 1^{st} *Int. Workshop on Manufacturing and Petri Nets*, Osaka, Japan, June 1996.
45. M. Silva and S. Velilla. Error detection and correction on Petri net models of discrete event control systems. In *Proc. ISCAS 85*, pages 921–924, 1985.

46. A.K.A. Toguyeni, S. El Khattabi, E. Craye. Functional and/or structural approach for the supervision of flexible manufacturing systems, In *IEEE-SMC CESA'96 Multiconference, Symposium on Discrete Events and Manufacturing Systems*, p.716-721, Lille, France, July 1996.
47. F. Tricas and J. Martínez. An extension of the liveness theory for concurrent sequential processes competing for shared resources. In *IEEE Int. Conf. on Systems, Man, and Cybernetics*, pages 4119–4124, Vancouver, Canada, October 1995.
48. C. Valentin, P. Ladet. Flow modelling in a class of hybrid (continuous-discrete) systems. In *IEEE International Conference on Systems, Man and Cybernetics*, Le Touquet, France, October 1993.
49. R. Valette and M. Courvoisier. Petri nets and artificial intelligence. In R. Zurawski and T. Dillon, editors, *Modern Tools for Manufacturing Systems*, pages 385–405. Elsevier, 1993.
50. R. Valette, M. Courvoisier, J. M. Bigou, and J. Albukerque. A Petri nets based programmable logic controller. In *IFIP* 1^{st} *Int. Conf. on Computer Applications in Production and Engineering*, Amsterdam, Holland, April 1983.
51. R. Valette, H. Pingaud, A. Pagès, D. Andreu, J.C. Pascal. Modelling, simulation and control of event-driven operations in process systems. In *INRIA/IEEE Conference on Emerging Technologies and Factory Automation ETFA'95*, p. 119-128 (Vol. 3), Paris, France, Oct. 1995.
52. S. Velilla and M. Silva. The spy: A mechanism for safe implementation of highly concurrent systems. In *Real Time Programming 1988,* 15^{th} *IFAC/IFIP Workshop*, pages 95–102, Valencia, Spain, May 1988. Pergamon.
53. J. L. Villarroel, J. Martínez, and M. Silva. GRAMAN: A graphic system for manufacturing system design. In S. Tzafestas, A. Eisinberg, and L. Carotenuto, editors, *IMACS Symp. on System Modelling and Simulation*, pages 311–316. Elsevier, 1988.
54. N. Viswanadham and Y. Narahari. *Performance Modeling of Automated Manufacturing Systems*. Prentice-Hall, 1992.
55. E.C. Yamalidou, J.C. Kantor. Modeling and optimal control of discrete-event chemical processes using Petri nets. *Computers Chem. Engng*, Vol.15, No 7, p.503-519, 1991.
56. E. Zamai, A. Chaillet-Subias, M. Combacau, A. de Bonneval. A hierarchical structure for control of discrete event systems and monitoring of process failures. *Studies in Informatics and Control*, Vol.6, N 1, p. 7-15, 1997.
57. M. C. Zhou and F. DiCesare. *Petri Net Synthesis for Discrete Event Control of Manufacturing Systems*. Kluwer Academic Publishers, 1993.
58. A. Zimmermann, S. Bode, and G. Hommel. Performance and dependability evaluation of manufacturing systems using Petri nets. In Silva et al. [44], pages 235–250.
59. R. Zurawski and M. C. Zhou. Special issue on Petri nets in manufacturing. *IEEE Trans. on Industrial Electronics*, 41(6), 1994.