Petri nets and manufacturing systems: An examples-driven tour

L. Recalde*, M. Silva, J. Ezpeleta*, and E. Teruel*

Dep. Informática e Ingeniería de Sistemas, Centro Politécnico Superior de Ingenieros, Universidad de Zaragoza, María de Luna 3, E-50012 Zaragoza, Spain
{lrecalde,silva,ezpeleta,eteruel}@unizar.es

Abstract. There exists ample literature on Petri nets and its potential in the modelling, analysis, synthesis and implementation of systems in the manufacturing applications domain (see for example [54, 15, 18]; moreover, in [65] an important bibliography is presented). This paper provides an examples-driven perspective. Nevertheless, not only complete examples from the application domain are considered. Manufacturing systems are frequently large systems, and conceptual complexity often appears because of some particular "local" constructions.
The examples considered in this selected tour try to introduce in a progressive way some applied concepts and techniques. The starting point is an assembly cell, for which models concerning several phases of the design life-cycle are presented. Afterwards, some pull control and kanban management strategies are modelled. Then, two coloured models of production lines are presented. After that, a manufacturing system with two cells is modelled, and the difficulty of the practical analysis is shown. For very populated manufacturing systems or systems with high cadence, relaxation of discrete event models leads to hybrid and continuous approximations, an example of which will be introduced.

1 Motivation and objectives

Petri Nets (PNs) constitute a well known *paradigm* for the design and operation of many systems allowing a discrete event view [53]. The purpose of this work is to present, in a tutorial style, some examples in which manufacturing systems are modelled, and analysed. Several books about PNs and the design and operation of manufacturing systems have been published at the end of the last century [17, 15, 64, 17, 44, 65]. In the sequel, the reader is assumed to be introduced to the main concepts in Petri Nets [50, 42].

Basically a case study driven perspective is provided in this work. Nevertheless, not only full examples from the application domain are considered. Manufacturing systems are frequently large systems, and conceptual complexity appears because of some particular constructions that appear in part of the system. The examples considered in this selected tour try to progressively present some applied concepts and techniques.

* Partially supported by project CICYT and FEDER TIC2001-1819

The starting point (Section 2) is a manufacturing cell in which some conveyors move parts that, processed into two different machines (*M1* and *M2*), are assembled and evacuated. The internal movements of the parts in the cell are executed by an industrial robot. Moreover, due to a relatively high rate of failures of a machine (*M1*), a buffer allows a partial decoupling with respect to the assembly machine (hence, also with respect to *M2*). This store (or buffer) is a capacity, and act like condensers in RC circuits: filtering high frequency perturbations (i.e., attenuating the effect of frequent short failures that usually lead to many small unavailability periods). From an abstract perspective, this introductory example shows some interesting interleaving among *cooperation* (here, the assembly of two different kinds of parts) and *competition* (for the shared resource: the robot) relationships. In general terms, the intricate interleaving of these two kinds of relationships leads to the kernel of the conceptual complexity to master the behaviour of discrete event systems (DES). The presentation of this introductory example is focused on the advantages of using different models of the same PN modelling paradigm in order to deal with the different phases of the design and operation that appear during the life cycle of the process.

In general terms, the control of manufacturing systems uses frequently some pre-established strategies. Among them the *push* strategy (from the input to the output: from the raw parts to the finished products), *pull* (from the output backwards to the input: from the demand to the input of raw parts) and *kanban*, that may represent many different kinds of tradeoffs between the above mentioned basic strategies, are specially relevant. The purpose of Section 3 is to show that this kind of control mechanisms (or management strategies) can be appropriately modelled by means of PNs (see, for example [11]). Analysis and optimisation of the obtained models can be done, but this topic is not considered in detail in this section, since the main purpose is to show the practical modelling power of the PN formalisms. This paper is mainly devoted to aspects related to modelling, analysis and control design, and not on other topics, like simulation or implementation issues, that although interesting and useful are not developed here. However, simulation will be used in this particular section to illustrate the comparison of different control techniques.

In many manufacturing systems a significant part of the apparent complexity may derive from the existence of several subsystems having identical (similar) behaviours, or from many parts having similar processing plans. Under these conditions (i.e., having significant symmetries among components), the use of high level PNs may be of interest. For this purpose two different examples are presented. The first one (Section 4) concerns a manufacturing line for car assembly in France. The basic model is constructed in a very systematic way, by merging a coloured PN model of the stations where manufacturing operations are performed and a coloured PN model for the transportation system. The problem with this basic model is that deadlocks may appear. A quite simple solution is presented, being directly implementable in PN terms, just by adding a place (i.e., a constraint) appropriately marked. A step further is done through

the presentation of a closed line corresponding to an ovens production factory sited in Zaragoza (Section 5).

In order to approach the limits of the actual knowledge in the theory and application of PNs to manufacturing examples, two additional cases are introduced in Section 6. In the first one (Section 6.1), the model of a Flexible Manufacturing System (FMS) (held in the Department of Computer Science and Systems Engineering of the University of Zaragoza) is established [25]. Even if modelling can be done in this case in a "straightforward" way, analysis "requires", in the actual state of the art, some manipulations allowing the computation of sequentialised views for the different process plans. In other words, it is not a direct application of theory that brings some solutions, but an indirect-pragmatically oriented engineering approach. Going in the same direction, in Section 6.2 modelling with object nets is done: this leads to a powerful modelling approach [61]. Unfortunately, it usually happens that the higher the abstraction level the formalism allows, the more complicated its analysis becomes. However, it is always possible to apply simulation techniques, which can give insight of some system behaviours.

Discrete event "views" may be very convenient in many cases for manufacturing systems. Nevertheless, in some other cases, either because of computational complexity problems (due to state explosion) or because the system presents a "regular" high cadence behaviour or is highly populated, fluidification or continuisation may be of interest [3, 51, 52]. A hybrid (partially continuised) model of this category is presented in Section 7. For systems in which some parts are "naturally perceived as continuous", a different PN interpretation leads to hybrid modelling (PrTr-DAE). In the present state of knowledge, this last approach uses simulation as the main analysis technique (besides the application of standard analysis techniques for the study of the underlying discrete model). Hybrid models analysis techniques should much improve in the future. Finally, some concluding remarks close this work.

2 Life cycle and an introductory example: An assembly manufacturing cell

This introductory example deals with a system in which the process plan is quite easy: Parts "*A*" and "*B*" should be produced (at machines *M1* and *M2*, respectively) and later assembled (a *rendez-vous*) in machine *M3* to obtain a final product that leaves the manufacturing cell. In this trivial cooperative system, two additional elements are introduced. First, relatively important failures and repairs are taken into account for *M1*. With the idea in mind of partially decoupling these accidents with respect to the operation of downstream machines (here *M3*), a *buffer* (inventory place, deposit) is introduced. If *M1* fails, the downstream machine, *M3*, may continue working for a while consuming the parts already in the buffer. If the upstream machine *M1* is repaired before the buffer is emptied, the failure will not affect the downstream line (here *M3*, only). Since *M3* is an assembly machine, its stopping condition will propagate to the

upstream line (here *M2*). The buffer is a passive element. At this point, the full systems only exhibit cooperative activities. A typical competition relationship is introduced by means of the movement of parts inside the system. In this case a robot, working in mutual exclusion (*mutex*), feeds *M1* and *M2* (from the conveyor belt), feeds the buffer (from *M1*), and moves parts *A* (from the buffer) and *B* (from *M2*) to *M3*. Thus this introductory example (Figure 1, that will be explained more in detail in Section 2.1) has *cooperation* and *competition* relationships. If the competition for the use of the robot is ignored, the cooperative parts can be described by a *free-choice* net system [56]. The addition of the robot-idle place transforms the net into a *simple* or *asymmetric choice*.

2.1 Basic autonomous model: dealing with basic relationships at the net level

The net in Figure 1 models both the *plant* and the *work plan*, from a coordination viewpoint. In the initial state, all the machines and the robot are idle, and the buffer is empty. The only enabled transitions are those that represent the start of the loading operation of either *M1* or *M2*, but only one of them can occur (i.e., there is a *conflict* situation). The autonomous model leaves undetermined which one will occur, it only states that these are the possibilities. Assume *M1* is to be loaded, what is represented by the occurrence of transition *t1*. Then the marking changes: one token is removed from each input place of the transition (*R idle* and *M1 idle*) and one token is put into the output place (*M1 loading*). Notice that tokens were required from two input places, meaning that the loading operation requires that both the machine and the robot are ready: it is a *synchronisation* of both. Now the only enabled transition is the one representing the end of the loading operation, but the autonomous model leaves undetermined when will this happen, it only states that it can only happen whenever loading is in course (which allows to represent *sequencing*). At the firing, the token is removed from *M1 loading* and tokens are put in *M1 working* and *R idle*. In this new marking, both output transitions of *M1 working* are enabled in conflict (it may either complete the work or fail), and also the start of the loading of *M2* is enabled. This latter transition and a transition from *M1* can occur simultaneously, or in any order (their enabling is independent), what allows to faithfully model *concurrency*. Notice the correspondence of subnets and subsystems (*M1*, *M2*, *M3*, *B1*, and *R*), and the natural representation of their mutual interactions. (It goes without saying that operation places could be refined to show the detailed sequence of operations in each machine, etc.)

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*. At the present stage of autonomous systems, these drawing conventions, and also the various labels, are literature: the dynamics of the model is not affected by these details, which are intended to make clearer the "physical" meaning of the model.

This autonomous model can be used for documentation/understanding purposes, and also to formally analyse the indeterministic possible behaviours. Clas-

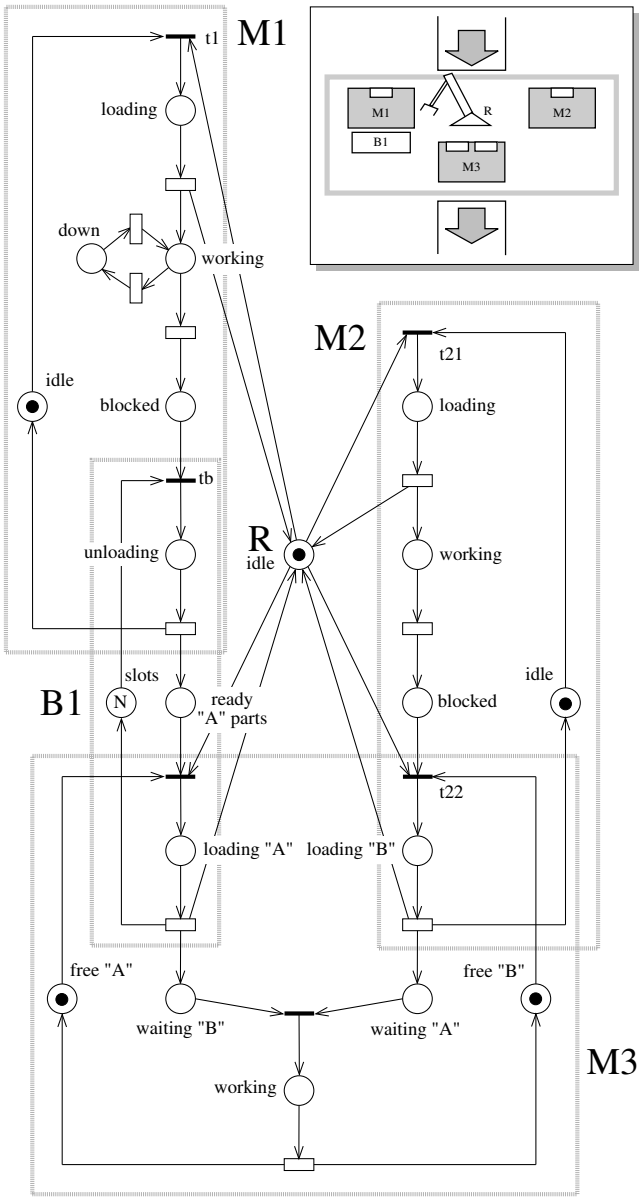


Fig. 1. An autonomous place/transition system that formally describes the logic behaviour of a manufacturing cell.

sical PN analysis techniques allow to efficiently decide that this system model is bounded (i.e., finite state space), live (i.e., no action can become unattainable), and reversible (i.e., from any state the system can evolve to its initial state).

Classical (and basic) reduction rules [49] allow to transform the model 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. Therefore, it is structurally bounded (i.e., it is bounded for any initial marking, not just for the one that is shown here), and it does not contain unmarked circuits, so it is live and reversible.

2.2 The performance evaluation model: stochastic T-timed interpretation and analysis

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 the firing delays are specified by their mean times.

In a preliminary design stage, where the issue is machine selection and dimensioning of the system, a stochastic timing specification, such as that of *generalised stochastic PNs* [1], is best suited. In the example we assume that the distribution of time delays corresponding to operations and movements is *phase-type*, namely *Erlang-3*, while failures and repairs follow *exponential* distributions. All other transitions are *immediate*, they fire as soon as they are enabled (so they are *prioritary* w.r.t. timed transitions). Conflicts between timed transitions are solved by *race* policy, while conflicts between immediate ones are solved in a probabilistic fashion).

It was seen in Section 2.1 that this system is reversible. Therefore, the reachability graph is strongly connected, and this allows to deduce ergodicity of the stochastic process and irreducibility of the underlying Markov chain.

Markovian performance analysis can be used to assist in the dimensioning of *B1*, or to analyse its impact. With given failure and repair rates for *M1*, throughput is plotted versus buffer size in Figure 3.

Economic considerations (in terms of throughput, required investment, and work in progress) would allow to optimise the buffer size. The plots in Figure 4 show how the effect of the buffer varies depending on the nature of the failures. Keeping the failure/repair ratio constant (i.e., the % of unavailability of the machine due to a failure is constant), different situations can be observed:

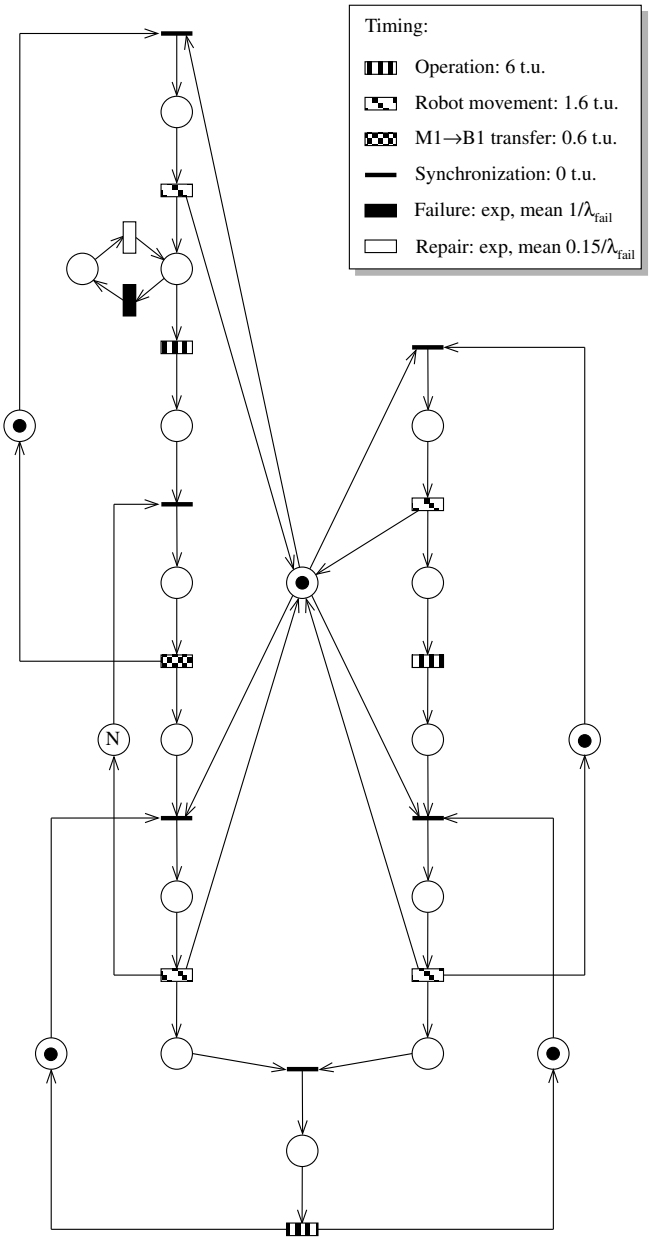


Fig. 2. A timed place/transition system that allows performance evaluation and optimisation of a manufacturing cell.

– Very unfrequent failures with very long repair times (left side of the plot). The throughput is reduced, and is insensible to the buffer size, because the repair time exceeds largely the time to empty the buffer.

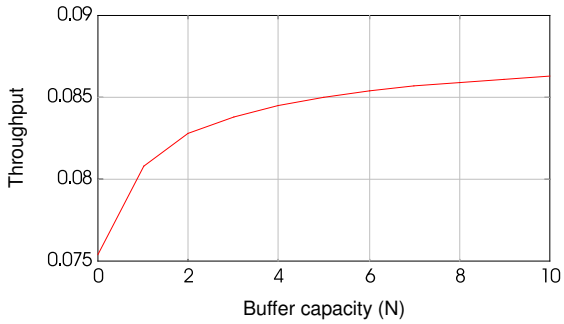


Fig. 3. Performance evaluation of the cell in Figure 1 with respect to buffer capacity.

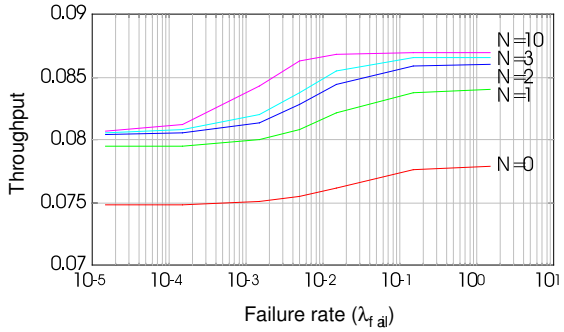


Fig. 4. Performance evaluation of the cell in Figure 1 with respect to failure rate.

– 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, and the throughput is equal to the throughput in the case of no 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 1 should be changed, removing *B1*. That is, the "*unloading*" operation should be merged with the "*loadingA*" and place *slots* removed since it becomes implicit. Then, *M1* becomes essentially identical to *M2*, except for the presence of failures. It results in a more tight coupling of the machines that leads to a significantly lower throughput.

2.3 On the optimal scheduling: Performance control

Assume that, after the optimisation of the design that involved performance evaluation, 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. The scheduler is in charge of controlling the evolution by enabling/disabling the transitions that initiate robot load operations (i.e., these are the *controllable* transitions here).

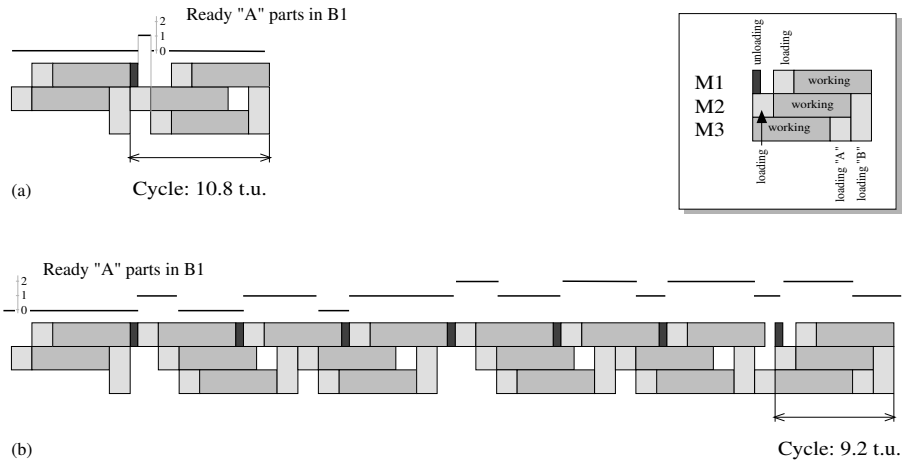


Fig. 5. Effect of different scheduling policies in the manufacturing cell of Figure 1.

Figure 5 shows the Gantt charts of two possible scheduling policies assuming deterministic timing and disregarding failures. In Figure 5(a) operations are scheduled as soon as possible, solving eventual conflicts in the allocation of the 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 5(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, thus the buffer allows to increase productivity in more than 11%. Let us check that this policy can be proved to be optimal.

As already mentioned, let us consider the system without failures (i.e., removing the failure-repair loop). One way of reasoning to obtain the optimal schedule for this system is as follows: the skeleton of the system is clearly a strongly connected marked graph provided with a monitor place (idle state for the robot). Thus the unique T-semiflow is $\mathbf{x} = \mathbf{1}$ (i.e., a vector of 1s). This means that all the transitions, in particular the four immediate in which the robot starts to work, should be fired in the same proportion in any "long enough" sequence.

Even more, the steady state should be defined by repeating sequences in which *t1*, *tb*, *t21* and *t22* (i.e., all the transitions before the "*loading*" places) appear once. Since those transitions are the only ones that may be in conflict, the scheduling problem reduces to choosing the relative order in which they should be fired. Given the repetitive behaviour of the steady state, in principle any transition can be taken as the first, thus there exist at most $3! = 6$ possibilities to explore. Assume *t22* is fired first. In this case nothing opposes to take *t21* as the second one to fire, because there is a marked place (*M2idle*) connecting the end of the first loading operation with the start of the second one (in other words, by choosing $t21$ as the second one no constraint is added). Therefore, the question now is to choose between *t1* and *tb*. Before going to that question, let us observe that firing an appropriate transient sequence the buffer can be filled, at least partially. In doing that, the firing of *t1* and *tb* are "decoupled" by a finite sequence, i.e., both can be fired in any order, while keeping the goal of computing an optimal schedule. If, after *t21*, transition *t1* is fired, the cycle of use of the shared resource (the robot) is finished by firing *tb* (and later *t22* for a new cycle).

A general upper bound of the throughput (lower for the cycle time) of the original system can be computed by means of a linear programming problem [9]. For this particular case, the lower bound for the cycle time is 9.2 time units. Looking at Figure 5(b) it is clear that this lower bound can be reached with the previous ordering. However, an alternative procedure can be used to prove it.

Introducing places $\{p2, p3, p4\}$ to put an order in the use of the robot: *t21-p2-t1*, *t1-p3-tb*, *tb-p4-t22* (observe that *p1*, for *t22-p1-t21*, is equal to *M2idle*, and so it is already present and marked), the place representing the idle state of the robot becomes concurrently implicit [55], thus can be removed and a marked graph is found. Under deterministic timing the exact cycle time for any marked graph can be computed by means of the same linear programming problem mentioned above [8]. The obtained value for this case is once again 9.2, thus under deterministic timing and no failures, the set of added constraints, places $\{p2, p3, p4\}$, constitute an optimal scheduler. The reason is that adding that constraints (places *p2*, *p3* and *p4*) the lower bound for the cycle time is now known to be reachable.

2.4 The controller: The marking diagram interpretation and fault-tolerant implementation

Controlling an existing manufacturing system (MS) means constraining its evolution in order to guarantee the desired logic behaviour or/and to optimise its performances 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, 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 epoch for transitions (scheduling). PNs with an appropriate timed interpretation are very well suited to the modelling of scheduling problems in parallel and distributed systems. PNs 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 indeterminism by deciding *when* to fire *which* transitions among the enabled ones. In scheduling theory [12] it is conventionally assumed that tasks are to be executed *only once*. Periodic or cyclic schedules [34] 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 an MS, its control can often be represented within the PN formalism, perhaps incorporating an appropriate interpretation.

Coming back to the manufacturing example, if the model is meant as a specification for a logic controller, the firing of transitions must be related to the corresponding external events or inputs, and the outputs that must be emitted have to be specified. The inputs, which 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 6 captures this information. Following appropriate conventions in the specification (e.g., those imposed in the definition of Grafcet [15]), a model similar to this one could be used directly as a logic controller program.

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 PNs 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 MS control, programmed implementations are the

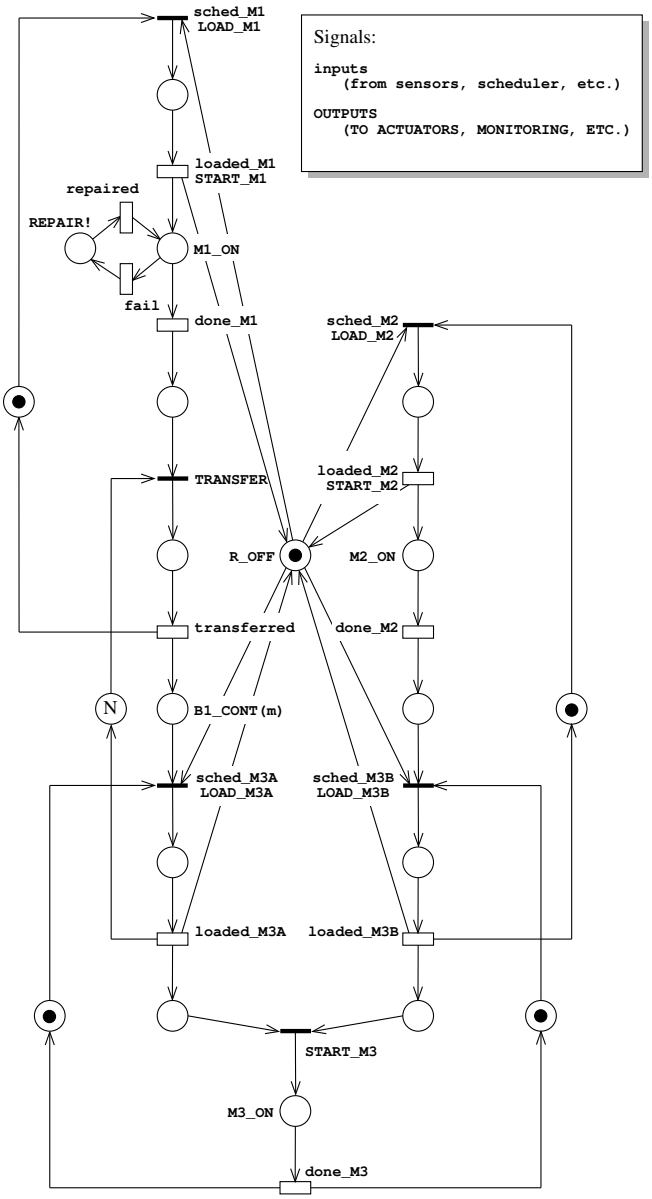


Fig. 6. A marking diagram that specifies the behaviour of the logic controller of a manufacturing cell.

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 schemas), and the *computer architecture* (high or low level programming language; single or multi processor).

For the case of local controllers specified by low level PNs with input and output signals (like that shown in Figure 6), a usual choice are interpreted implementations ("token players") [60, 48]. 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, [13]). 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 6 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, [13]). The structure theory of PNs allows to identify certain components in a given net that are useful for distributing or parallelising 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 PNs and artificial intelligence (see, for instance, [40], [59]). 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 PNs 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 PNs are a suitable formalism [37]. 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 briefly concentrate here on the detection and recovery of faults in the controller itself. Several techniques have been proposed to produce safe and/or *fault-tolerant* PNs based controllers. We illustrate next one of these techniques which are supported by PNs theory: the *spy/observer* schema.

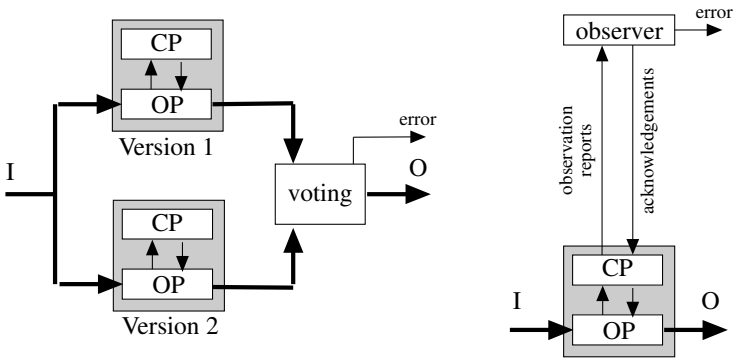


Fig. 7. Duplication versus observation.

In general, *N-version* programming techniques, that is, the controller is replicated and a voting mechanism is introduced [4] can be used. A less expensive schema is based on the idea of an *observer* [5] or *spy* [62], which accepts "normal" behaviours seen through some *observable*, or *check*, points. In Figure 7 duplication and observation schemas 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. An observer is also employed in [19] for formal validation.

Coming back to the example, 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 8. (Notice that this spy is obtained applying the same reduction rules that were applied for the analysis.)

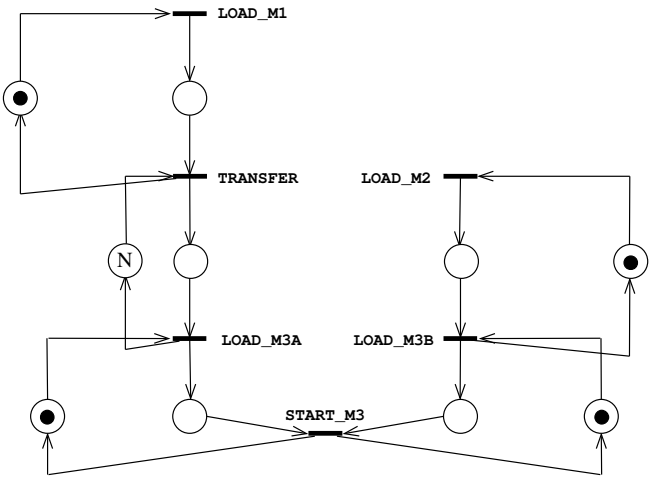


Fig. 8. A spy for the net in Figure 1.

3 Modelling some classical management strategies in manufacturing: pull control and kanban

The primary goal of many manufacturing systems can be expressed in terms of the maximisation of the production rate, the minimisation of the *work-in-process* (WIP) inventory, and minimisation of the delivering delay (difference between the date of a demand and the date of serving it). The above criteria usually leads to some contradictory situations. For example, minimising WIP usually lead to higher delivering delays, what may even represent losing some selling opportunities (impatient clients).

Among the many imaginable strategies for the management of production systems, *push control* is based on the idea of "advancing" tasks relative to production as much as possible. Thus the behaviour of the production plant is "externally" constrained by the raw materials available, and by the capacity of buffers for storing finished products. Under this strategy, raw materials "push the production", and delivering delays are minimised at the expense of, eventually, important WIP costs. In many cases push-type behaviours use demand forecasts to generate the production plans. On the contrary, under the basic *pull control* strategy, the customers demands trigger the production, i.e., "pull the production". Thus the WIP cost is reduced to a minimum, at the expense of more important delays for delivering, i.e., at the expense of decreasing the quality of customer service.

In the manufacturing arena, it is well known that *just in time* (JIT) approaches lead to low WIP costs. In order to conciliate the above mentioned contradictory performances, many hybrid push/pull control algorithms have been proposed in the literature. *Kanban systems* allow to deal with different kinds of those strategies, trying to smooth and balance material flows by using several

appropriately controlled intermediate inventories. In essence kanbans are cards that circulate between a machine (or sequence of machines) and a downstream buffer. When a withdrawal operation liberates a position of an intermediate buffer, a card is recirculated in order to allow the production of a new part to compensate "the previous loss" in the inventory site. The number of kanbans around a machine(s)-buffer subsystem determines the buffer size. In a kanban controlled system, production of parts is triggered in response to "intermediate demands". As already mentioned in the cell manufacturing example of Section 2, the parts in any intermediate buffer try to "protect" the operation of downstream machines from possible interruptions of upstream machines. If the repairing time of the machine under failure is "not too big", the buffer will not empty and the failure will not affect the downstream machine. Therefore intermediate buffers "can be perceived" as condensers in electrical circuits or resorts in mechanical systems, allowing relatively uncoupled behaviours on production lines subsystems. A certain number of questions arise in order to optimise the production: Where to put the intermediate buffers?, How large?, Which strategies should be used for control?, etc.

The point here is that at a general level, Petri nets –with some timed interpretation, for example, Generalised Stochastic Petri Nets [1]– can be used to model different designs and control strategies. By using appropriate performance evaluation models, the optimisation of the strategy used to control the material flow (i.e., making the more appropriate decisions), even the tuning of its parameters, can be formally studied.

Single-output assembly manufacturing systems have usually, from the output point of view, a tree-like topology. In the manufacturing domain, it is usual to represent machines as circles and buffers as triangles (Figure 9). The (output) root of the tree represents the finished goods buffer. In order to simplify the presentation, let us assume a single level assembly stage and two previous manufacturing stages (Figure 10).

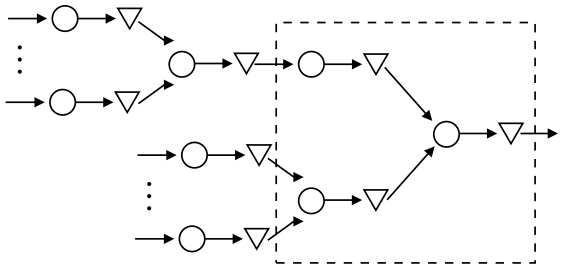


Fig. 9. Topology of an assembly manufacturing system: machines are shown as circles and buffers as triangles.

The basic schema of a production stage can be easily described in PNs terms by means of the connected marked graph in Figure 11(a). According to that,

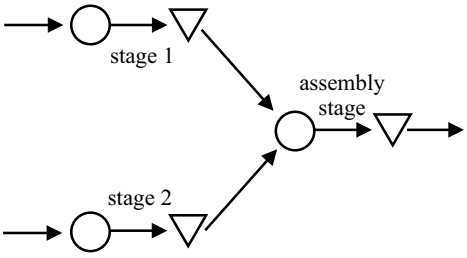


Fig. 10. Two manufacturing stages (with their buffers) followed by an assembly stage (with the finished products buffer).

production stages are composed of a raw parts container (*raw*) synchronised with a demand for production (*demand*), followed by the waiting queue and machine working place (*dr*), and the place representing the single machine (*machine*); and finally its output buffer of finished parts (*f*). The transition in the self-loop of the machine is timed (processing time of a part). Thus the utilisation rate of the machine is given by the probability of non null marking in place *dr* (at least one part needs to be processed).

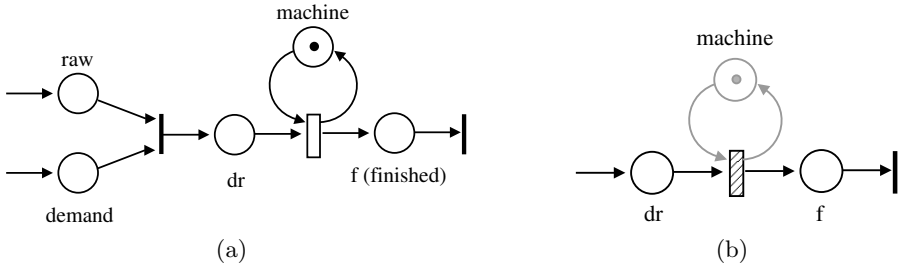


Fig. 11. Basic schema of a production stage.

It is common in certain cases to assume that there are always enough raw parts. This means that place *raw* can be removed because it is not a constraint any more (it is never the unique that forbids the firing of its output transition). In doing so, because the transition between places *demand* and *dr* is immediate, both places can be merged into a single one (we keep the name *dr*). In Figure 11(b), the simplified model is presented. It will be a basic building block for the models of this section. In order to simplify the drawing of nets, in the sequel place *machine* will be removed, while it is assumed that the firing semantics of the corresponding transitions is *single server* [8]. Transitions with single servers semantics will be graphically denoted here as dashed timed transitions. Observe that at this level it is assumed that the machines do not fail.

A basic pull control system (*base stock control system*, BSCS [11]) is presented in Figure 12. It consists of two production stages (with $k1$ and $k2$ parts finished

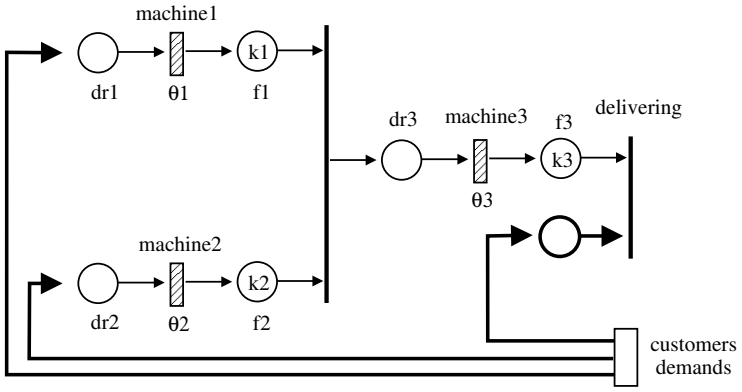


Fig. 12. Production of parts A and B (stages 1 and 2) and final assembly (stage 3), with a basic stock (pull) control system (BSCS) and assuming single server semantics.

in stage 1 and stage 2, respectively), feeding an assembly stage (initially with $k3$ finished parts). When a customer's demand appears, places *dr1* and *dr2* receive a (new) token, in order to produce another part for each stage. Customers demand allows to serve finished parts, represented by tokens in place *f3*, initially marked with $k3$ tokens. A main problem in this basic schema is that the limitation of the WIP is not assured in any of the three stages (two for production and one for assembly, in the present case). It is not difficult to see that *under saturation of customers demands* (i.e., under the hypothesis that there exists an infinite number of customer demands), the production cycle time (the inverse of the throughput) is bounded by the slower of the three machines:

$$\theta = max\{\theta 1, \theta 2, \theta 3\}$$

Simultaneous kanban control system (SKCS) and *independent kanban control system* (IKCS) are modelled in Figures 13 and 14. As happened before, in both cases two production stages are followed by an assembly stage. Even under saturation of customers demands, the capacity of the stages are $k1, k2$ and $k3$, respectively, while the production cycle time under deterministic timing is once again θ, i.e., defined by the slower machine (because all ki are greater than zero). Under stochastic timing, θ is a lower bound for the cycle time (i.e., $1/\theta$ is an upper bound for the throughput).

The difference among SKCS and IKCS is that the first one feeds simultaneously the assembly stage and the new production order for the (two) previous stages. In the second case, separate kanbans feed stages 1 and 2, while feeding the assembly stage is automatic, when appropriate parts exists (in *b1* and *b2*).

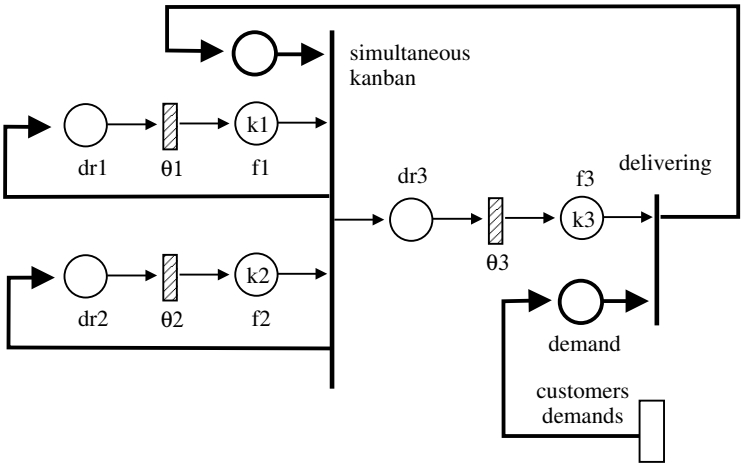


Fig. 13. Simultaneous kanban control system (SKCS).

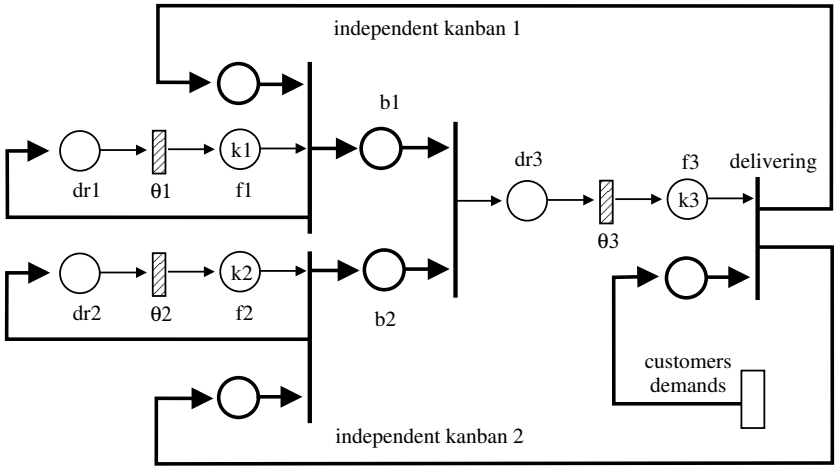


Fig. 14. Independent kanban control system (IKCS): Kanbans are independently generated for machine 1 and machine 2.

Obviously, in transient behaviours, the independent case can be better than the simultaneous one.

A more elaborated kanban system is presented in Figure 15. It is the so called *independent extended kanban control system* (IEKCS) [11]. Under saturation of customers demands it behaves exactly like the above schemes (SKCS and IKCS). Nevertheless, in this case different kanbans send simultaneously requests for the production of primary parts (in stage 1 and stage 2), for an assembly to be done, and for the delivery of a finished part. This may lead to some interesting

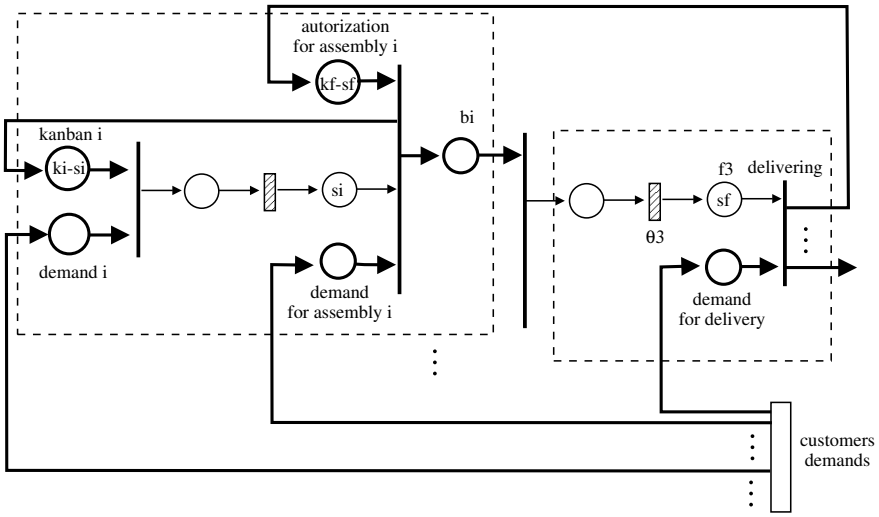
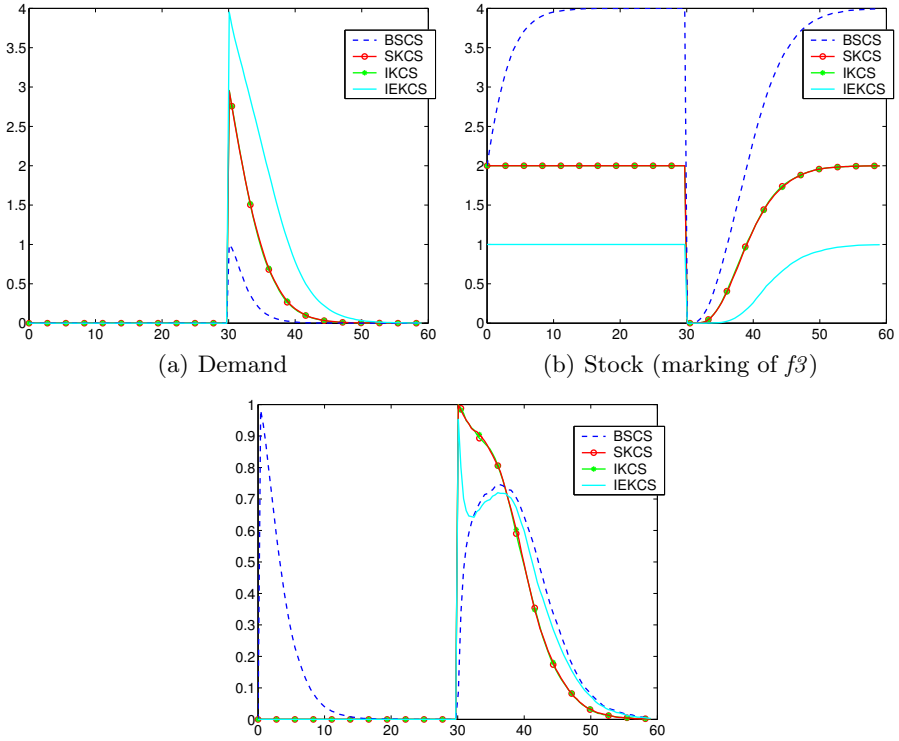


Fig. 15. Independent extended kanban control system (IEKCS).

behaviours, potentially reducing the WIP, while keeping a good reactivity to demands.

These control policies have been simulated assuming in all cases that $\theta 1 = 1$, $\theta 2 = \theta 3 = 2$, $k1 = 2$, $k2 = 3$, $k3 = 2$, and for IEKCS, $s1 = s2 = s3 = 1$. A burst of 5 simultaneous demands is simulated at 30 t.u. The results for the different control systems in Figures 12-15 are represented in Figure 16, where (a) shows the marking of place *demand* (unsatisfied demand), (b) shows the marking of place *f3* (complete products in stock), and (c) shows the throughput of the assembly station. Because the "delivering" transition is immediate, the unsatisfied demand at 30 t.u. is equal to 5 minus the products in stock: 1 for BSCS, 3 for SKCS and IKCS, and 4 for IEKCS. It can be seen that SKCS and IKCS are in this case equivalent. BSCS is the first to "satisfy the demand" (the marking of the place *demand* returns to zero), while IEKCS is the last one. However, the stock of complete products in absence of demand is much larger under BSCS (4), than under IEKCS (1). With respect to the throughput, SCKS, IKCS and IEKCS, work on demand, so the throughput is zero before the demand. Under BSCS, a first outburst of the production appears, since the intermediate stocks *f1* and *f2* are used to produce the final assembly. In other words, the system tries to complete as much products as it can, instead of keeping stocks of intermediate elements. That is the reason why although the stock under BSCS is 4 and under IEKCS is only 1, it does not take four more times to satisfy the demand in the latter case, but only about twice.

Many other schemes of this type can be imagined. The important point at this level is that modelling with PNs is frequently quite straightforward (if control strategies do not depend too much on particular data), and analysis can provide useful information about the behaviour of the intended control strategy.



(a) Demand

(b) Stock (marking of *f3*)

(c) Throughput of the assembling station (labelled with $\theta 3$)

Fig. 16. Simulation of the different control policies in Figures 12-15.

4 A coloured model for a car manufacturing line

A relatively frequent characteristic of production systems is the existence of *symmetries* due to the presence of subsystems that behave "in a similar way". Coloured PNs allow to exploit these symmetries and generate a more compact model. Coloured Petri nets can also be extended, as in [30, 31], or abstraction on the formalism (i.e., the underlying PN model) can be done in application oriented interfaces, as in [63]. Here just basic coloured Petri nets will be used to model some examples.

4.1 A car manufacturing system

The following example shows a coloured PN model of a realistic MS (part of a flexible workshop of a car factory), taken from a case study [39].

The FMS shown in Figure 17 consists of:

– Several workstations (*S1* to *Sn*). All the workstations behave in a similar way: car bodies to be processed are loaded in table L (input buffer of capacity

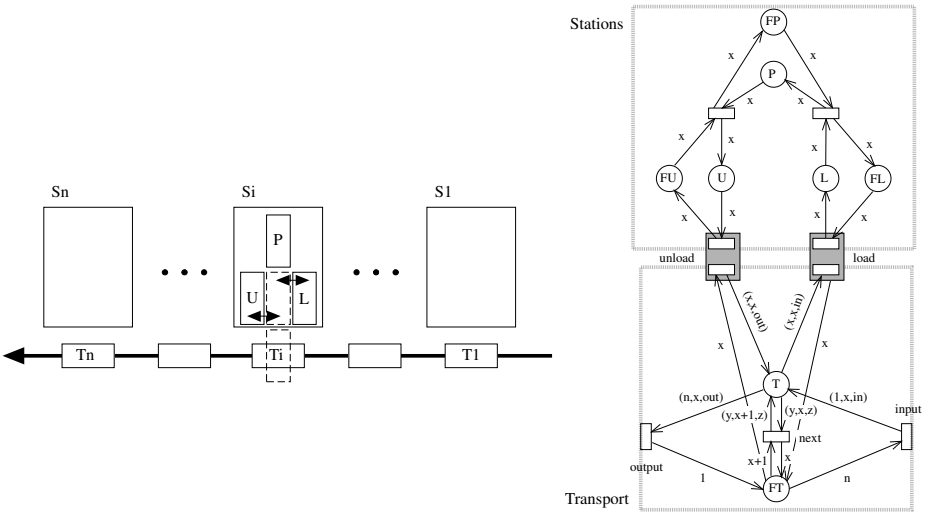


Fig. 17. A flexible workshop that processes car bodies in several stations, and its coloured PN model

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, represented by the corresponding places, which can be active simultaneously. 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 place P represents that workstation Si is processing. Transferring a processed part from table P to table U in workstation Si requires one i-token in P and FU, and puts one i-token in U and FP.

– An unidirectional transport system, consisting of several roller tables (*T1* to *Tn*). Car bodies enter the system in table *T1* and leave it from *Tn*, after 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 one the destination station of the car body, and the third one 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 among the different firing modes of the input transition. The destination is determined by the scheduler, possibly

taking into account the state of the system and the 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.

4.2 On the control of the production line

Besides avoiding deadlocks, let us consider a control policy to improve the performance.

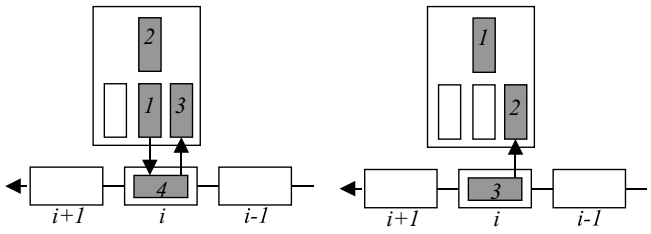


Fig. 18. (a) Complete deadlock (b) Temporary deadlock.

Analysis of this system proves the 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, see Figure 18(a). 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 19(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 could 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, see Figure 18(b). We cannot proceed to load the third car body until processing is completed, the processed car body is transferred to the table U,

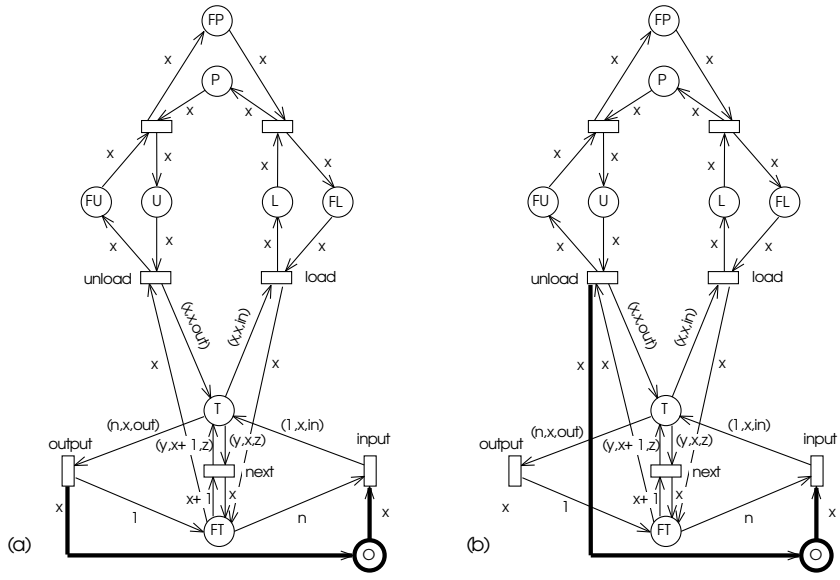


Fig. 19. Adding place O to the net model in Figure 17, with a suitable marking, avoids deadlocks and stoppages.

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.

The first columns in Table 1 (observe the output) compare the steady state throughput of these two control policies for different processing times in a three cells workshop. All the cells are assumed to be equal, and the car bodies are sent to all of them with the same probability. The transitions are assumed to follow exponential distributions, of mean one for all the transport operations (both inside and outside the cells). It can be seen that, if the processing is fast with respect to the transport, the two policies are more or less equivalent. However, if the processing takes "much time", the throughput is better under the most restrictive policy. Intuitively, since the processing needs more time than the transportation, it is better to be sure that the parts can advance till the processing station.

Finally, in the above control it was assumed that the scheduler controls transition *input* and observes just transition *output*. If it observed also the occurrences of transition *unload* it might be possible to improve the performance of the control policy by allowing a limited number of *unprocessed* orders in the system (see Figure 19(b)).

Table 1 compares the results of both control policies for the previous example. It shows that if the number of orders allowed in the system for each machine is 2, the throughput increases slightly when the *unload* transition is observed. However, if three orders are allowed, the throughput decreases. Intuitively, with

Mean processing time	Observe the output			Observe the unload		
	Throughput			Throughput		
	Three orders	Two orders	Increase	Three orders	Two orders	Increase
1	0.2971	0.2984	0.45 %	0.2969	0.3002	1.11 %
5	0.2434	0.2763	13.54 %	0.2378	0.2809	18.14%
10	0.1669	0.2173	30.24 %	0.1617	0.2210	36.66%
15	0.1227	0.1671	36.17 %	0.1189	0.1690	42.12%
20	0.0964	0.1331	38.07 %	0.0935	0.1341	43.45%
50	0.0418	0.0578	38.51 %	0.0406	0.0579	42.70%

Table 1. Throughput comparison for the system in Figure 19(a), if place O is marked with two or three tokens of each colour.

at most three orders for each machine the system was already saturated, and allowing a greater number of car bodies only makes it worse.

5 On a production line for ovens

This section describes a new manufacturing system where the set of production orders compete for a set of physical resources. The system is quite similar to the one in the previous section. Here, the attention is focused on how to obtain the coloured Petri net model, by first modelling the plant layout taking into account the possible ways parts can flow through the system and the imposing to each flowing part the execution of its associated process plan, which needs of model refinement. Finally, it will be shown how to prevent deadlocks and how the deadlock related control approach can be improved taking a more abstract point of view.

5.1 System description

Figure 20(a) depicts the structure of a flexible manufacturing cell for the production of microwave ovens (a more detailed description can be found in [24]). The cell has an entry station, `EntryStation`, an exit station, `ExitStation` and n workstations, $w_0, w_1, \ldots, w_{n-1}$. These workstations are loaded and unloaded by a circular conveyor belt with a continuous movement in a unique direction. The manufacturing of each oven is made according to its process plan. There are several scales and models of ovens with their respective process plan. The components of an oven arrive at `EntryStation` after having been previously pre-assembled; once an oven reaches that point, it is fixed to a pallet that will be inserted into the transport system when possible. One of such loaded pallets must visit a set of workstations, according to the process plan of the part it contains, and then leave the system through the `ExitStation`. The pallet goes then to the pallet store, to be reused. The system has a total of K pallets.

As detailed in Figure 20(b), each workstation w_i has an input buffer I_i and an output buffer O_i. Both consist of two roller tables, each with capacity for one

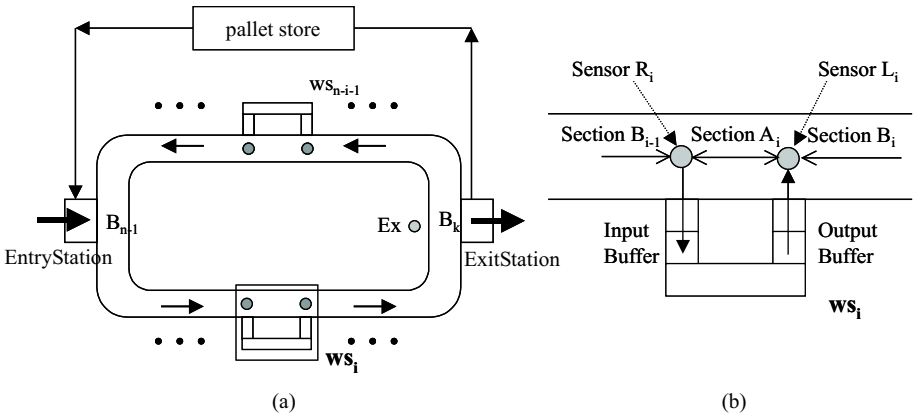


Fig. 20. a) General plant representation of a cell for the manufacturing of microwave ovens. b) Detailed view of the structure of a workstation and its related sections.

pallet. The pallets in each buffer follow a FIFO policy. A workstation can operate with one pallet at a time. In order to control the system, the conveyor belt has a set of sensors distributed as shown in Figure 20(b): $R_0, L_0, \ldots, R_{n-1}, L_{n-1}$ and Ex. Associated to these detection points there are mechanisms that, under the control of the workshop coordination system, allow to carry out the following transfer operations, schematised by means of arrows in Figure 20(b): introduction of a pallet from `EntryStation`, exit of a pallet from the `Ex` point towards `ExitStation`, loading of a pallet in workstation w_i by transferring it from position R_i to the input buffer of w_i, I_i, unloading of a pallet from the output buffer of w_i, O_i, to point L_i of the conveyor belt. Each A_i or B_i section will have its own capacity, which corresponds to the number of pallets the section can hold.

5.2 A coloured Petri net model of the coordination system

A first approach to the modelling of material flow is shown in Figure 21. Let us explain the main elements in the model.

The transport system: The set of states a pallet can be in the transport system is modelled by means of places B, R, A, L. Place B models the set of B sections. Place A models the set of A sections, while places R and L model sensor points between sections B_{i-1} and section A_i and between sections A_i and B_i, respectively. The colour domain of all these places is $WS = \{w_0, \ldots, w_{n-1}\}$, the set of workstations. The initial marking of each one of these places is the multi-set 0, which means that, at the initial state, no pallet is inside the system. Transitions t_{in} and t_{out} model the actions by which a pallet with a new oven enters the system and a pallet with a terminated oven leaves the system, respectively. Ordinary (non-coloured) place AP models the set of free pallets, whose initial marking is K, the

number of available pallets. In the system, it is assumed that `EntryStation` loads pallets into section B_{n-1} and that `ExitStation` unloads pallets from section B_k.

Places BC and AC, whose colour domain is also WS, model the capacities of B_i and A_i sections, respectively. The initial marking of BC is the multi-set $\sum_{i=0}^{n-1} b_i \cdot w_i$, being b_i the capacity of section B_i. Analogously, the initial marking of AC is the multi-set $\sum_{i=0}^{n-1} a_i \cdot w_i$, being a_i the capacity of section A_i. Places CR and CL represent that only one pallet can be in sensor points R_i and L_i, respectively. The initial marking of both places is $\sum_{i=0}^{n-1} 1 \cdot w_i$.

Transition t_{br} models a pallet reaching an R_i sensor (the function labelling the arc (t_{br}, R), $w@1$, represents the addition of 1, modulo the number of sections, n). Transition t_{ra} models a pallet entering an A_i section, transition t_{al} models a pallet reaching an L_i sensor. Finally, transition t_{lb} models that a pallet reaches a B_i section.

Transition t_{ls} (t_{us}) models a pallet being loaded into (unloaded from) a workstation.

The set of workstations: A pallet loaded into a workstation, by means of the firing of transition t_{ls}, must, successively, visit the two input buffer positions (places $IP1$ and $IP2$), to be processed in the workstation (place W), and visit the two output buffer positions (places $OP1$ and $OP2$). The initial marking of any of these places is the multi-set 0: there is no pallet in any workstation.

Places $IC1, IC2, WC, OC1$ and $OC2$ impose the capacity constraints of being able to have at most one pallet in each one of the components of a workstation. The initial marking of any of these places is $AW = \sum_{i=0}^{n-1} 1 \cdot w_i$.

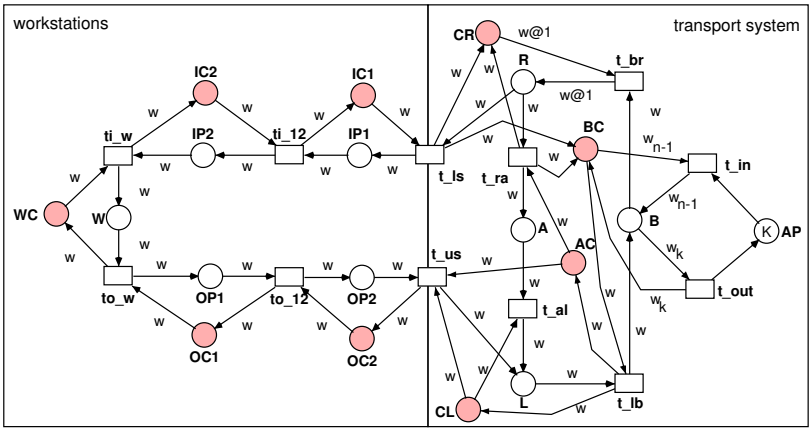


Fig. 21. A coloured Petri net model of flow of pallets in the system in Figure 20.

It is important to notice that, even if all the transitions in the model represent system actions that change the system state, from the control point of view two kinds of transitions are considered:

- Transitions whose firing is observable but not controllable. This is the case of $\{t_{br}, t_{ra}, t_{al}, t_{lb}, ti_{12}, to_{12}\}$. Since the conveyor has a continuous movement the firing of one of such transitions will be realised when a pallet reaches or leaves the corresponding sensor. The events can be noticed and thus the system state can be updated in the model.
- Transitions whose firing is decided and executed by the control system (controllable transitions). These are the transitions that can be controlled in order to ensure that every incoming part will be processed according to its associated process plan, and also to impose some control policy in order to ensure some desired properties, as deadlock freeness or to impose some scheduling policies. This set of transitions is composed of $\{t_{ls}, t_{us}, t_{in}, t_{out}, ti_w, to_w\}$.

5.3 Inclusion of the process plans

Each oven that enters the system must execute its associated process plan, which consist of a sequence of operations to be executed in the system workstations. This sequence is described by means of a sequence of pairs (o, w), where o defines the operation to be executed, and w the workstation where such operation must be done. The sequence of operations for an oven has been pre-established by the system controller before loading the oven into the system. In the specification level considered here, which concentrates on the material flow control, it is possible to make abstraction of the operations to be executed, describing the process plan as the ordered sequence of workstations to be visited by the oven. Therefore, a process plan will have the following form: $p = (w_p^1; w_p^2; \ldots; w_p^{n_p})$, where each $w_p^i,\ i \in \{1 \ldots n_p\}$, belongs to WS.

There exists a set of predefined process plans $PP \subset WS^+$. Each part that enters the system has an associated process plan belonging to PP. The first element in the ordered sequence of workstations in the process plan corresponds to the first workstation to be visited. In order to identify the state in the processing of a part in the system, tuples of the form $(p, i) \in PP \times \mathbb{N}$ will be used: p identifies the process plan, while i identifies the position in the process plan sequence of the next workstation to be visited. For instance, when an oven whose associated process plan is $p = (w_p^1; w_p^2; \ldots; w_p^{n_p})$ enters the system, it will be identified by means of the token $(p, 1)$, meaning that w_p^1 is the next workstation to be visited. When the oven is processed in w_p^1, the tuple identifying the oven will be $(p, 2)$; when terminated, it will be identified by means of $(p, n_p + 1)$.

According to this codification of the processing state of an oven in the system, the model in Figure 21 must be transformed. Since the system layout is still the same, only colour domains and functions in the arcs have to be changed. If in the initial model a token in place A, for instance, was of the form w, just indicating the concrete A-section where the pallet was, now a token in such place will be of the form (p, i, w) indicating that there is a pallet in w A-section, containing an

oven whose associated process plan is p and that has to next visit workstation w_p^i. Accordingly, the colour domain of places modelling physical locations that can contain pallets with ovens is $PP \times \mathbb{N} \times WS$.

Notice that, in order to forbid a pallet to enter a workstation that is not its next destination, predicate $[w_i^p = w]$ has been associated to transition t_{ls}. Also, predicate $[i = n_p + 1]$ has been associated to transition t_{out} so that only pallets containing ovens whose process plan has been completely executed can be unloaded from the system. Notice also that the firing of transition to_w transforms a token of the form (p, i, w) into $(p, i+1, w)$, which corresponds to changing the next destination workstation for the considered oven.

The resulting model is shown in Figure 22, *where places modelling resource capacity constraints have not been represented*, for the sake of clarity. In any case, they are exactly the same as in Figure 21.

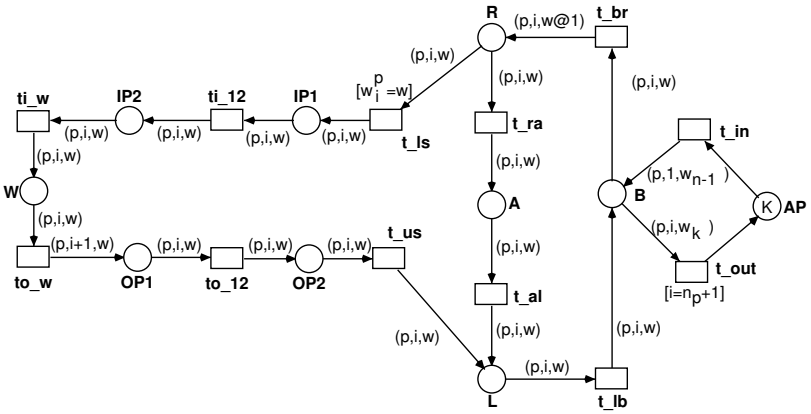


Fig. 22. A coloured Petri net model of the system in Figure 20 once the process plans are considered (capacity constraints have not been represented, for the sake of clarity).

5.4 Preventing deadlocks. A first solution

If the control model in Figure 22 is directly implemented, the system can reach deadlock situations. Let us consider, for instance, a reachable state in which a workstation w_i is full (input and output buffers are full and the workstation is also processing an oven) and also the transport system is full of pallets that must enter workstation w_i. In this situation, no new pallet can enter the system, no pallet in the conveyor can be loaded into workstation w_i and no pallet can leave it since the conveyor is full. All the deadlock situations are related to states in which full stations require to unload pallets to the transport system, which is full of pallets that must enter a full workstation.

An easy way of preventing such situations consists in ensuring that no more than five pallets inside the system need to visit a given workstation. This is the deadlock control implemented in the following. The implementation is based on the following function, called *workstation requirements*, and defined as follows. Let $p = (w_p^1; w_p^2; \ldots; w_p^{n_p})$ be a process plan, and let $i \in \{1, \ldots, n_p + 1\}$ be an index associated to p. For the tuple (p, i) the following multi-set of workstations is defined: $wr(p, i) = \sum_{j=0}^{n-1} \lambda_{p_i}^j \cdot w_j$, where $\lambda_{p_i}^j$ is 1 if $w_j \in \{w_p^i, w_p^{i+1}, \ldots, w_p^{n_p}\}$ (in the case of $i = n_p + 1$ the addition is made over an empty set of workstations, and it is assumed to be the empty multi-set). Notice that, in fact, $wr(p, i)$ is the characteristic function of the workstations to be visited by the oven from the index i until the associated production plan is terminated. Notice also that if $i_1 < i_2$, then $wr(p, i_1) \geq wr(p, i_2)$.

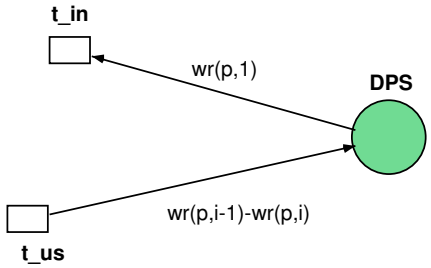


Fig. 23. The implementation of a deadlock prevention solution for the considered system.

In order to implement such control policy in the Petri net model place DPS (Deadlock Prevention Solution) is added, whose colour domain is WS and whose initial marking is the multi-set $\sum_{i=0}^{n-1} 5 \cdot w_i$ (Figure 23 shows the Petri net elements to be added to the model in Figure 22). For a pallet that enters the system (firing transition t_{in}) with an oven whose associated process plan is p, the set of possible workstations the pallet must visit is "reserved". This is implemented by means of the function $wr(p, 1)$ labelling the arc (DPS, t_{in}). Moreover, each time a pallet leaves a workstation, if this oven does not need to visit that workstation again in the future, the reservation must be released. This is implemented by means of the arc (t_{us}, DPS). As noticed previously, the label $wr(p, i-1) - wr(p, i)$ is properly defined since $i - 1 < i$. Notice also that the control is related to transitions t_{in} and t_{us}, which are both controllable.

5.5 Preventing deadlocks. A more accurate solution

The solution for deadlock prevention just proposed is of the same type as in Section 4. However, taking a detailed look at an abstract view of the underlying non-coloured model a more accurate solution can be adapted. Let us, for

instance, consider a process plan $p = (w_1; w_2)$. Taking into account that with an adequate control every pallet in the transport system can reach any workstation and also that every free position in the transport system can be used for the downloading of any workstation, the ordinary Petri net in Figure 24 is an abstract view of the processing of a part whose process plan is p.

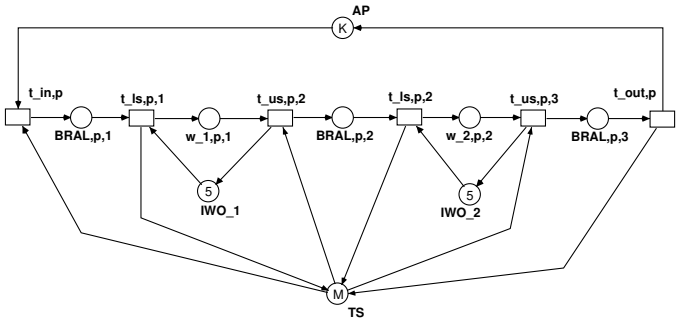


Fig. 24. An abstract point of the processing of a part whose associated process plan is $p = (w_1, w2)$.

The meaning of the different elements in the model are the following. Place TS in an abstraction of the whole transport system; its initial marking is $M = \sum_{i=0}^{n-1} a_i + b_i$, the total number of available locations for parts in the conveyor. Place IWO_1, whose initial marking is 5, models the total capacity of workstation w_1, considering in it the input buffer, the output buffer and the workstation itself. Places "$BRAL, p, *$" model the different states of a part of type p in the transport system. Transitions "$t_{ls}, p, *$" model the different firings of transition t_{ls} when the processing of a part of type p advances. Analogously for transitions "$t_{us}, p, *$". Transition t_{in}, p (t_{out}, p) models the loading (unloading) of part of type p into (from) the system. Considering the models of all the involved process plans, a final model will be obtained by means of the fusion of the places in the models of the process plans modelling the capacities of the resources they share.

The resulting Petri net belongs to a class of resource allocation systems (RAS) which have been intensively studied in the literature, and for which a wide set of different approaches for deadlock prevention and avoidance have been developed. [23, 57] use an structure-based approach to synthesise the deadlock-freeness related control. In both cases, the Petri net structure (siphons) is used to characterise deadlock problems and also to obtain generalised mutual exclusion solutions that forbid deadlock related stated. These mutual exclusion constraints are implemented by means of the addition to the former uncontrolled model of new places and arcs. Any of the solutions can be used to control the system here considered. The implementation can be done as in [22], in an analogous way as

in the previous subsection, by means of the addition of a control place (as is the case of place DPS previously used) and some related labelled arcs.

The use of any of these last approaches will yield, in general, more permissive solutions than using the approach in section 5.4 (the less states of the uncontrolled system a control policy allows, the less permissive it is). However, they have the drawback that since the control is based on a deep use of the abstract unfolded model and the competition relations among the involved process plan models, the addition of new process plans will require the re-computation of the necessary control, making the approach less adaptable to changes in the production than using the approach in section 5.4.

6 A pragmatically oriented approach

In some cases, analysing the "natural" model an engineer produces is not an easy task. This is due to the fact that the resulting model can be complex. Analysis techniques (mainly those techniques that do not use the reachability graph or simulation, such as structure-based techniques or transformation techniques, for instance) have some limitations for general Petri net models, becoming more difficult when using high level Petri nets. In this section two new practical cases are described. The first one uses ordinary Petri net models, but there are not techniques able to control the natural model (deadlock-freeness related control is once again the objective). This problem is then solved by the transformation of the initial model into one with an equivalent behaviour, and for which control techniques exist. The second case uses a different modelling approach, based on the *Nets-within-Nets* paradigm as used in [61]. This paradigm falls into the object-oriented modelling approach.

6.1 Modelling and deadlock avoidance for a two cells manufacturing system

The objective is to model and control, avoiding deadlock states, the manufacturing system in the Department of Computer Science and Systems Engineering of the University of Zaragoza. To do that, ordinary Petri nets have been selected as the modelling tool. It could have been modelled also using coloured PNs, as the previous examples. However, since the technique that is being used for the control needs a non-coloured model, it has been decided to use ordinary nets instead of building a coloured model and unfolding it afterwards.

The system and the modelling approach Figure 25 depicts the plant of the manufacturing cell, consisting of six machines (M1 to M6) that process the components, one buffer with place to store up to 16 intermediate products, and two robots (R1 and R2). The process is organised in two rings, with the buffer connecting them. A final product (Figure 26) is composed of a base on which three cylinders are set. The base may be black or white, and there are three types of cylinders: cylinders that are composed of a case, a piston, a spring,

and a cover (called "complete" cylinders), cylinders with just a case and a cover (called "hollow" cylinders), and cylinders in one piece (called "solid" cylinders). The cases and the solid cylinders may be red, black or metallic. Bases, pistons, springs, covers, cases, and solid cylinders are considered as the raw materials. An unbounded amount of raw material is assumed to feed the system. A set of 330 different products can be composed using these materials.

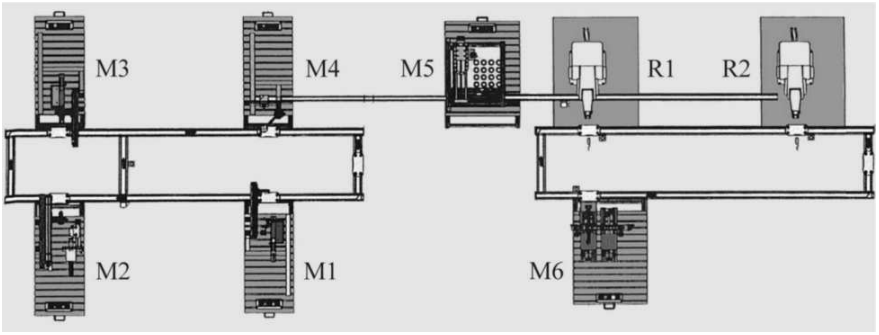


Fig. 25. A plan of the physical system.

The processing goes as follows: machine M1 takes a case from a feeder, and verifies that it corresponds to the order, that is, if the colour is correct and whether it is a case or a solid cylinder. If it is not correct, then it is discarded, otherwise, it is put on a pallet, and the kind of processing that the part needs is written on the pallet. If it is a solid cylinder, a switch is activated to carry it directly to M4. Otherwise it goes to M2. Machine M2 puts the piston and the spring, if the cylinder needs them, and then the part goes to M3, which adds the cover. In M4 the parts are verified, the pallets are released and the parts are put on a conveyor that moves them to the entrance of the buffer. Machine M5 can temporarily store the cylinders in the buffer. When needed to assemble the final product, M5 puts them in a conveyor that takes them to robot R1. Machine M6 puts a base of the right colour on a pallet, and it is carried to robot R1. The robot takes the three cylinders one by one and puts them on the base. The product is then complete, and goes to robot R2, which takes it out of the system.

The adopted modelling approach is as follows. Each possible *production order* (corresponding to a type of product) has been modelled by means of a Petri net. Then, a set of places, modelling the capacity constraints of the physical resources involved in the production process (robots, intermediate store, pallets, etc.), have been modelled.

Figure 27 shows the Petri net model of one of the products in the system here considered: a product made of three complete cylinders is shown. Place `IDLE` represents the state in which the production order has not been started, the rest of "tagged" places model the system resources (resource places), while

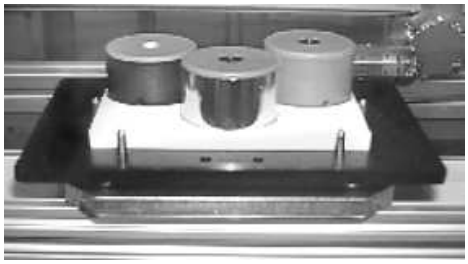


Fig. 26. The kind of products that the system in Figure 25 produces.

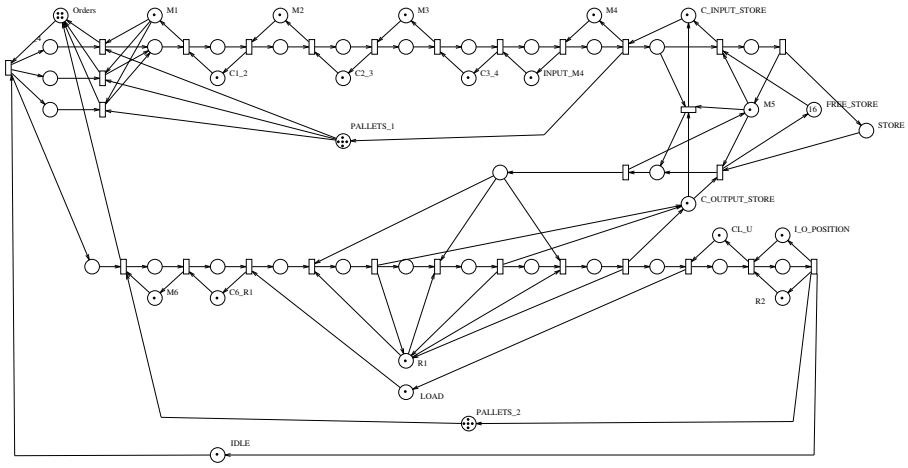


Fig. 27. A non-sequential RAS modelling the assembly of a product made of three complete cylinders and a base.

the "non-tagged" places model the different states of the component elements inside the system (state places). In the example the resources are of two kinds. On the one hand there are machines, robots, and space in the intermediate buffer (i.e, physical constraints). On the other, there are constraints that are not strictly necessary but are advisable for the correct evolution of the system, for example not to allow more than one pallet on each conveyor segment, that make the conveyor segment to be considered as a resource with capacity one. The final model will be obtained by means of the composition, by fusion of the common places modelling system resources, of the models corresponding to the whole set of products.

Deadlock avoidance control In order to have a completely automated system, the objective now is to synthesise the control necessary to ensure that no deadlocks can appear. As in Section 5.4, the system falls into the class of Resource Allocation Systems: it is composed of a set processes which in their execution

must compete for the set of system resources. The complexity of dealing with deadlocks strongly depends on the system structure. Different classes of RAS systems have been defined in the literature. The features that distinguish these classes refer to the process structure (wether the process is sequential or concurrent and whether routing flexibility is allowed or not, mainly) and the way in which resources are allowed to be used and allocated/released (one-by-one or as multi-sets). These characteristics define the class of Petri nets the model belongs to. In the case of a process with a sequential nature (sequential RAS), a state machine can be used to model it (places modelling constraints capacities imposed by the physical or logical resources have then to be added); in the case of non sequential processes, more sophisticated Petri net models are needed, including fork/joint transitions (non-sequential RAS). In systems where resources are allowed to be allocated/released as multi-sets, weights will appear in the arcs related to places modelling resources, which means that the model will belong to the class of generalised Petri nets. These elements will directly influence the analysis and synthesis capabilities of the Petri net model.

An "easy" way of applying deadlock related control is based on the computation of the reachability graph of the system model, to detect the deadlock states and then to forbid them somehow. However, computing the reachability graph of the whole system was not possible, because of its enormous size (for instance, the reachability graph of just one production order as the one in Figure 27 has 2442 states, while the reachability graph with two production orders being concurrently executed had 241951 states; computing the reachability graph in the case of three production orders was not possible). Therefore, some deadlock prevention/avoidance strategy based on the model structure instead of the reachability graph is needed.

In the case of sequential RAS many different solutions can be found in the literature, adopting different points of view. See, for instance, [23, 35, 43, 26] as a very short list of solutions. However, in our concrete case, there exist transitions with more than one input state place (see Figure 27), which make our system to belong to the non-sequential RAS class. Adopting a Petri net perspective [47, 28] propose deadlock avoidance solutions for sub-classes of assembly systems. However, the present system falls out of these classes.

In the sequel, a different engineering strategy is adopted: to transform the problem into one with known and applicable solutions. If a deadlock avoidance strategy is adopted, any resource-related state change in the system must be controlled in such a way that only if the reached state is proved to be *safe* (safe means that it can be ensured that all the active processes can be terminated) the change is allowed, otherwise it is forbidden. This means that the application of a deadlock avoidance method imposes a kind of "sequentialisation" in the system behaviour. Therefore, and concentrating on the execution of a production order, substituting its model by the state machine corresponding to the reachability graph of the production model itself is just a change in the model, but not in the behaviour. Notice that doing so a sequential RAS model for the system is obtained. Resource places of the initial model are added to this state machine

(they are implicit places and can be added without changing the behaviour) and the final system model is obtained by means of the composition by fusion of the places modelling system resources of the sequential models of the set of products. The considered model belongs to the class of systems for which a deadlock avoidance method is proposed in [26], which can be, then, applied to control the considered system.

The control algorithm is based on an adaptation of the Banker's algorithm [20, 33]. In order to consider a given state as safe, the Banker's algorithm looks for an ordering in the set of active processes such that the first process can terminate using the resources granted to it plus the free ones, the second process can terminate using the resources it holds plus the ones free upon the hypothetical termination of the first process, and so on. The basic step is to know if a given process is able to terminate using a given set of available resources. The solution in [26] is a two steps algorithm. First, mark those state places of the state machine modelling the considered process and that require no more resources than the free ones plus the ones in use by the process itself. Second, look for a path of marked state places joining the place corresponding to the state the process is in and the final state.

One important issue when applying deadlock avoidance approaches is the time used to decide whether a given state is safe, since the procedure must be called every time a state change engages new resources. Implementing the control method the following results have been obtained. In the case of the non-sequential RAS in Figure 27, the corresponding sequential model (the reachability graph of the net in that figure) has 2442 state places, 7814 transitions, using each state up to 22 types of resources. Checking if an active process was able to terminate using the free resources has been implemented. Its takes about 0.003 CPU seconds using a Pentium(4) processor at 1.7 GHz under Microsoft Windows 2000 operating system (this computation uses a Depth First Search algorithm, which is linear in the size of the unfolded system). If the whole system is considered, and given that no more than 26 components can stay at the same time in the system (considering the 10 pallets plus the 16 storage places in Figure 26) and that a direct implementation of the algorithm in [26] grows in a quadratic way with respect to the number of active production orders, the time to know if a system state is safe takes about 2 CPU–seconds in the worst case.

In order to obtain more efficient solutions some approaches are currently being studied trying to solve the problem for non-sequential RAS using directly the initial model structure. A solution for a class non-sequential RAS, where processes must have a tree-like structure can be found in [27].

6.2 Beyond the state of the art for the analysis: Modelling with object nets

The aim of this section is to show a different approach for the modelling of production systems. It is based on the clear and intuitive characteristic that in a production system, among other elements, there are two main components. On the one hand, the *system architecture*, which corresponds to the distribution of

the physical elements in the plant. Usually, this structure is rather static, and not easily changeable. On the other hand, the set of *process plans* corresponding to the different types of products to be produced in the system. These plans can be seen as logical constraints to be imposed to the free flow of parts in the system. In many cases the set of process plans can change (new process plans are required to face demands of new products, while others disappear, corresponding to products with very low demand). Therefore, doing a separated consideration of that elements when designing the system control software makes easier to adapt it to changes in the set of products the system is able to deal with.

A way of doing that was proposed in [22], where the final model was a coloured Petri net in which the system architecture provided the net skeleton (the set of places, transition and arcs) while the set of part flow restrictions imposed by the process plans were modelled by means of the colour domains of places and transitions and the functions labelling the arcs. This has also been the approach followed in the previous sections. In this section a different approach is going to be adopted. It is based on the *Nets-within-Nets* paradigm, as used, for instance in [61], which support a modelling of systems by Petri nets following the paradigm of Object Oriented Modelling. Applications of the paradigm to the case of manufacturing systems can be seen in [29, 41, 38].

Roughly speaking, one of such models is composed of a *System Net* and one or more *Object Nets* which can be seen as token objects of the system net. Both, the system net and the object nets are Petri nets. A token in the system net can be either a reference to an object net or a black token. Each object net state represents the state of the element it models. Changes in such state can be produced by its own internal dynamics (*autonomous* occurrences), but can also be due to some interactions with the system net. On the other hand, some transitions in the system net can influence the internal state of object nets, but others just move object nets between different locations of the system nets (*transport* occurrences).

Therefore, in the definition of an elementary object system, besides the system net, the set of object nets and the initial marking, a set of *interactions* must be considered. The interactions define how the system net and the object nets must synchronise their activities. These concepts directly apply for the modelling of manufacturing systems. The model of the physical system will correspond to the system net, while each part will be modelled by means of an object net.

The objective of this section is not the introduction of the Nets-within-Nets paradigm, but just to show that it is very well adapted to model production systems. To do that, let us apply it to the same example used in [61]. Figure 28 depicts a manufacturing cell composed of four machines, $M1, M2, M3$ and $M4$ (each one can process two products at a time) and three robots $R1$, $R2$ and $R3$ (each one can hold a product at a time). There are three loading buffers (named $I1, I2, I3$) and three unloading buffers (named $O1, O2, O3$) for loading and unloading the cell. The action area for robot $R1$ is $I1, O3, M1, M3$, for robot $R2$ is $I2, O2, M1, M2, M3, M4$ and for robot $R3$ is $M2, M4, I3, O1$.

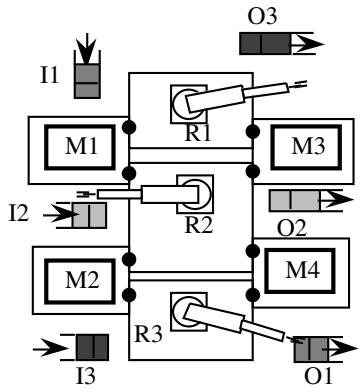


Fig. 28. A manufacturing cell composed of four machines and three robots. Black dots represent the possibility of part flow between two resources.

Every raw product arriving to the cell belongs to one of the three following types: $W1$, $W2$ and $W3$. The type of product characterises the process to be made in the cell as follows: 1) a raw product of type $W1$ is taken from $I1$ and, once it has been manufactured, is moved to $O1$. The sequences of operations for this type are either $(M1, op1); (M2, op2)$ (execute $op1$ in $M1$ and then $op2$ in $M2$) or $(M3, op1); (M4, op2)$ (execute $op1$ in $M3$ and then $op2$ in $M4$). 2) a raw product of type $W2$ is taken from $I2$, manufactured in $M2$ (operation $op5$) and then routed towards $O2$. 3) a raw product of type $W3$ is taken from $I3$, manufactured in $M4$ (operation $op4$) and then in $M3$ (operation $op3$) and, finally, routed towards $O3$. Figure 29 represents, by means of directed acyclic graphs, the possible operation sequences for such set of types of parts.

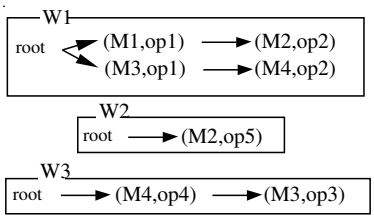


Fig. 29. Three directed acyclic graphs specifying three different types of parts to be processed in the cell depicted in Figure 28.

Analogously as in the example in 5.3, the (uncontrolled) Petri net in Figure 30 represents the possible flow of parts in the considered system. In order to be able to ensure that each part in the system will be produced according to its corresponding process plan, some control has to be added to this skeleton model,

which will correspond to the system net in the Nets-within-Nets model (the meaning of places named $W1r, W2r, W3r$ and $W1t, W2t, W3t$ will be explained later).

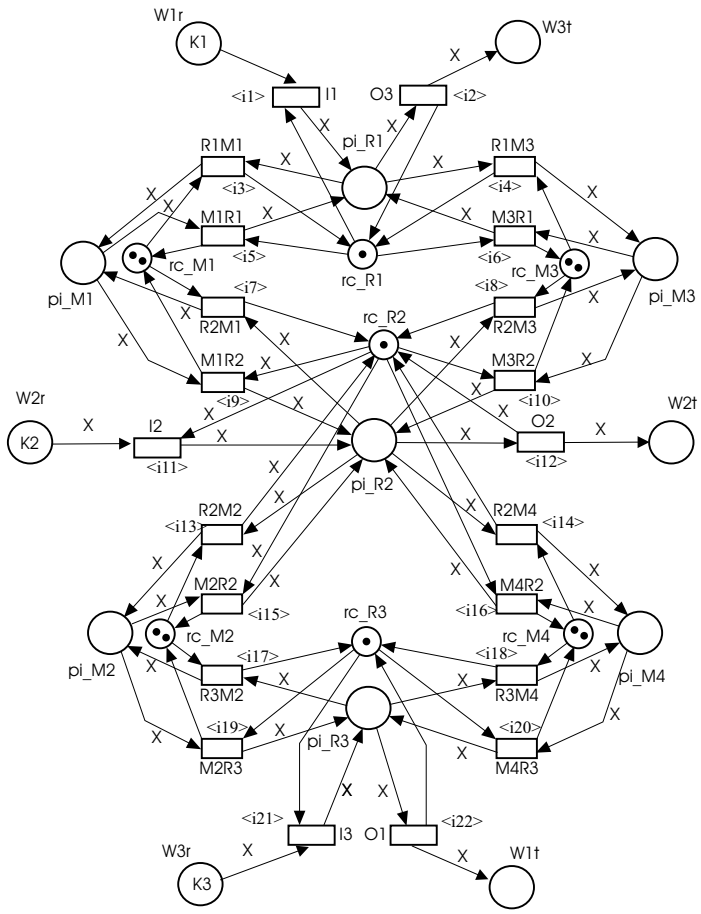


Fig. 30. Petri net model of the part flow in the cell depicted in Figure 28.

Figure 31 shows three object nets corresponding to the three types of parts to be produced in the considered system (since in this example all the transitions in the object nets must interact with the system, transition names in Figure 31 are not represented, just the interactions, for the sake of clarity). Let us explain one of these models. The Petri net labelled $W2$ in Figure 31 corresponds to a part type $W2$ (in fact, each $W2$-type part will be modelled by one instance of such net). The token in place $p2_1$ models the raw material for one of such products before being loaded into the system. This state is changed when that raw material enters the system. According to the system net in Figure 28, this

is done by the firing of transition $I2$. Therefore, firing such (system) transition must also make the token in $p2_1$ to move to place $p2_2$, which is imposed by the interaction $\langle i11 \rangle$. Place $p2_2$ models a part of type $W2$ inside the system and that must be processed in $M2$. Transition $t2_2$ is used to model the fact that such part enters $M2$, which in the system net corresponds to transition $R2M2$. Interaction $\langle i13 \rangle$ takes that into account. Interaction $\langle i15 \rangle$ is used to move the part from $M2$ to the robot $R2$. Finally, interaction $\langle i12 \rangle$ is needed to model the unloading of such part from the system.

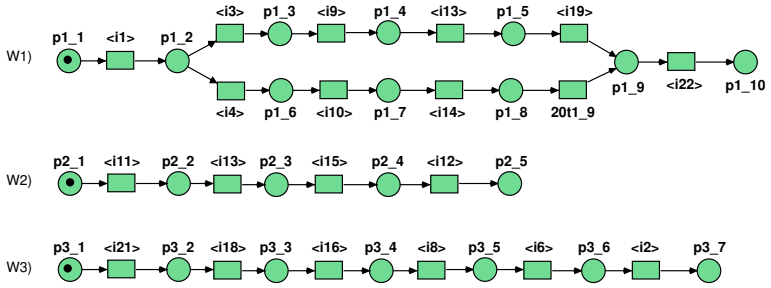


Fig. 31. Three object nets modelling the three types of parts to be processed in the system in Figure 28. Transition names are not presented, only the interactions with the system net.

In the system net in Figure 30 tokens in place $W1r$ are instances of object net $W2$ in Figure 31, and correspond to raw parts of type $W2$ (there are $K2$ of such net instances). Once terminated, these object nets will be in place $W1t$, which "collects" terminated products of type $W2$.

Any further refinement in the model is easy to be done. Let us suppose also the different operations each machine is able to do need to be considered. For instance, machine $M3$ is able to carry out operations $op1$ and $op3$. Figure 32(a) shows how place pi_M3 in the net in Figure 30 could be refined in order to consider the operations it is able to do (capacity of $M3$ is not represented for the sake of clarity). On the other hand, Figure 32(b) shows how the place $p3_5$ of the object net corresponding to the processing of parts of type $W3$ in Figure 31 could be refined so that the process plan it models takes into account that the operation $op3$ has to be done in $M3$ for such parts (notice that transitions $M3_1$ and $M3_2$ correspond to transport occurrences).

High level Petri net-based formalisms provide very useful tools for the modelling, analysis and control of complex concurrent systems. However, the higher the abstraction level the formalism allows, the more complicated its analysis becomes. This is the case of coloured Petri nets, for instance (structure-based techniques are not as general as in the case of ordinary Petri nets) and also the case for Nets-within-Nets models. It is always possible to apply simulation techniques, which can give insight of some system behaviours allowing the system

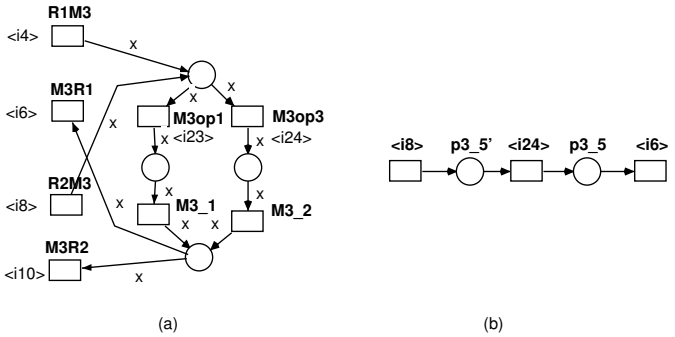


Fig. 32. A refined model for machine $M3$ and how it affects the object net modelling $W3$ parts.

designer to easily test different system configurations in order to have arguments to choose one or another. In the case of Nets-within-Nets, the tool Renew [36] is a good environment for modelling and simulation.

7 From discrete event models towards hybrid models

In the last years a new kind of models based on Petri nets has appeared. They differ from the previous ones in that they are not discrete event models, but hybrid models. That is, the state is not only represented by discrete variables, but it is partly relaxed into continuous variables (in the extreme case, even all the variables may be continuous in piecewise continuous systems).

These hybrid models have been defined in many different ways. For example, (discrete) Petri nets may be combined with differential algebraic equations associating them either to places (Pr/Tr Petri nets) [10] or to markings (DAE Petri nets) [58]. Another possibility is to partially relax the integrality condition in the firing of the transitions, i.e., continuise or fluidify the firing, as in Hybrid Petri nets [3]. This means that the marking of the places around these transitions is no longer guaranteed to be integer (with the possible exception of self-loop arcs). When a total fluidification is done the result is a Continuous Petri net [14, 51]. This kind of hybrid models can be used both to represent systems whose "more reasonable view" is hybrid, or as an approximation of discrete systems under high traffic conditions. The idea of continuisation of discrete models is not new and has been employed in many different fields, for example, population dynamics [46], manufacturing systems [16, 32], communication systems [21], etc. In the following we will concentrate on Hybrid Petri nets.

In timed models, in order to associate a time semantics to the fluidification of a transition, it should be taken into account that a transition is like an station in Queuing Networks, thus "the meeting point" of clients and servers. Assuming that there may be many or few of each one of them, fluidification can be

Table 2. The four cases for possible continuisation of a transition [52]

Clients	Servers	Semantics of the transition
few (D)	few (D)	Discrete transition
few (D)	many (C)	Discrete transition (servers become *implicit places*)
many (C)	few (D)	Continuous finite server semantics (bounds to firing speed)
many (C)	many (C)	Continuous infinite servers semantics (speed is enabling-driven)

considered for clients, for servers or for both. Table 2 represents the four theoretically possible cases. If there were few clients, the transition should be considered discrete.

Basically, the idea is to use a first order (or deterministic) approximation of the discrete case [45], assuming that the delays associated to the firing of transitions can be approximated by their mean values. A similar approach is used, for example, in [6]. This means that in continuous transitions the firing is approximated by a continuous flow, whose exact value depends on the semantics being used. The two basic semantics defined for continuous transitions (see Table 2) are *infinite servers* (or *variable speed*) and *finite servers* (or *constant speed*) [3, 45]. Under finite servers semantics, the flow of t_i has just an upper bound, $\boldsymbol{\lambda}[t_i]$ (the number of servers times the speed of a server). Then $\mathbf{f}(\tau)[t_i] \leq \boldsymbol{\lambda}[t_i]$ (knowing that at least one transition will be in saturation, that is, its utilisation will be equal to 1). Under infinite servers semantics, the flow through a timed transition t is the product of the speed, $\boldsymbol{\lambda}[t]$, and the enabling of the transition, i.e., $\mathbf{f}[t] = \boldsymbol{\lambda}[t] \cdot \text{enab}(t, \mathbf{m}) = \boldsymbol{\lambda}[t] \cdot \min_{p \in \bullet t}\{\mathbf{m}[p]/\mathbf{Pre}[p, t]\}$.

It should be pointed out that finite server semantics, equationally modelled by bounding the firing speed of continuised transitions, corresponds at conceptual level to a *hybrid* behaviour: fluidification is applied only to clients, while servers are kept as discrete, counted as a finite number (the firing speed is bounded by the product of the speed of a server and the number of servers in the station). On the other hand, infinite servers semantics really relax clients and servers, being the firing speed driven by the enabling degree of the transition. In this case, even if the fluidification is total, the model is hybrid in the sense that it is a piecewise linear system, in which switching among the embedded linear systems is not externally driven as in [7], but internally through the minimum operators.

The following example is taken from [2, 3]. It models a station in a Motorola production system. This station can produce two kinds of parts, *c1* and *c2*, whose processing corresponds to the left and right part of the figure, respectively. The parts arrive in batches of 30000 and 20000 parts at times 0 and 1000. After the arrival of a bach, parts are downloaded into a buffer at a speed of 1 part per time unit. The processing does not start immediately, but waits until at least 500 parts of type *c1* or 600 parts of type *c2*) have been downloaded. At that point some set up is done on the machine, which takes 300 time units for parts *c1* and 360 for *c2*, before the processing starts. When all the parts in the batch have been processed, the machine is liberated. Pieces are removed in batches of the input size.

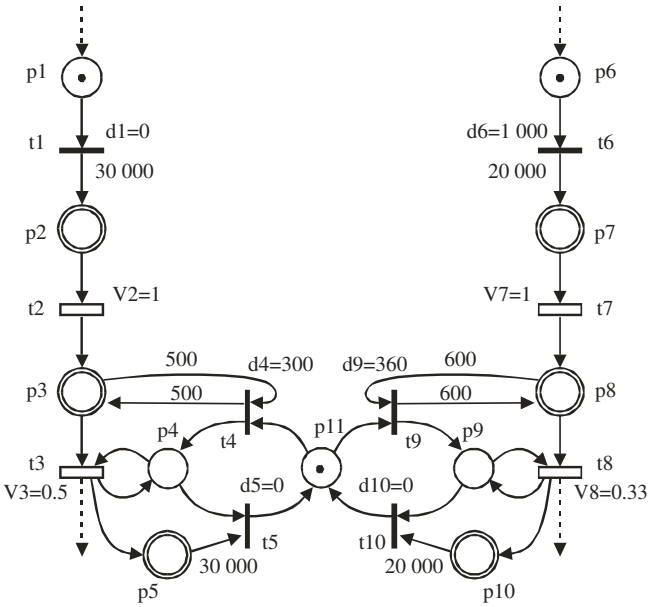


Fig. 33. Hybrid Petri net modelling the behaviour of a production system.

A model of this system can be seen in Figure 33. Although it is a discrete system, the model is not discrete, but hybrid. The transitions represented as bars in the figure are discrete (the usual transitions in Petri nets), while those represented as boxes are continuous. Analogously, the circles drawn with a simple line are discrete, while those with the double line are continuous.

In this example, since the size of the batches is quite large, the firing of transitions t_2, t_3, t_7 and t_8 can be approximated by a continuous flow. This kind of approximation (when applicable) may simplify the study of the system. For example, in [2] it is reported that for this system the simulation time reduces from 454 sec. to 0.15, that is, it is divided by 3000!

Basic understanding of hybrid systems, and analysis and synthesis techniques need much improvement before they can be effectively used [51, 52]. Moreover, it should be pointed out that there exist some "natural" limits to the properties that can be studied. For example, mutual exclusion (in the marking of places or in the firing of transitions), and the difference between home space and reversibility cannot be studied in general [51]. Additionally, basic properties like deadlock-freeness of the autonomous continuised model is neither necessary, nor sufficient for the discrete case [51]. However, the use of hybrid models as partial relaxations of discrete models is a quite new and promising approach.

References

1. M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets*. Wiley, 1995.
2. H. Alla, J.B. Cavaille, M. Le Bail, and G. Bel. Les systémes de production par lot: une approche discret-continu utilisant les réseaux de Petri Hybrides. In *Proc. of ADPM'92*, Paris, France, January 1992.
3. H. Alla and R. David. Continuous and hybrid Petri nets. *Journal of Circuits, Systems, and Computers*, 8(1):159–188, 1998.
4. A. Avizenis and J. P. Kelly. Fault tolerance by design diversity: Concepts and experiments. *Computer*, 17(8):67–80, 1984.
5. 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.
6. F. Balduzzi, A. Giua, and G. Menga. First-order hybrid Petri nets: A model for optimization and control. *IEEE Trans. on Robotics and Automation*, 16(4):382–399, 2000.
7. A. Bemporad, A. Giua, and C. Seatzu. An iterative algorithm for the optimal control of continuous-time switched linear systems. In M. Silva, A. Giua, and J.M. Colom, editors, *WODES 2002: 6th Workshop on Discrete Event Systems*, pages 335–340, Zaragoza, Spain, 2002. IEEE Computer Society.
8. J. Campos, G. Chiola, J. M. Colom, and M. Silva. Properties and performance bounds for timed marked graphs. *IEEE Trans. on Circuits and Systems-I: Fundamental Theory and Applications*, 39(5):386–401, 1992.
9. J. Campos, G. Chiola, and M. Silva. Ergodicity and throughput bounds of Petri net with unique consistent firing count vector. *IEEE Trans. on Software Engineering*, 17(2):117–125, 1991.
10. R. Champagnat, R. Valette, J.C. Hochon, and H. Pingaud. Modeling, simulation and analysis of batch production systems. *Discrete Event Dynamic Systems: Theory and Application*, 11(1/2):119–136, 2001.
11. C. Chaouiya and Y. Dallery. Petri net models of pull control systems for assembly manufacturing systems. In *Procs. of the* 2^{nd} *Int. Workshop on Manufacturing and Petri Nets, ICATPN*, pages 85–103, Toulouse, France, 1997.
12. P. Chretienne, E. G. Coffman, J. K. Lengstra, and Z. Liu, editors. Wiley, 1995.
13. 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.
14. R. David and H. Alla. Continuous Petri nets. In *Proc. of the 8th European Workshop on Application and Theory of Petri Nets*, pages 275–294, Zaragoza, Spain, 1987.
15. R. David and H. Alla. *Petri Nets and Grafcet*. Prentice-Hall, 1992.
16. R. David, X. Xie, and Y. Dallery. Properties of continuous models of transfer lines with unreliable machines and finite buffers. *IMA Journal of Mathematics Applied in Business and Industry*, 6:281–308, 1990.
17. A. Desrochers, editor. *Modeling and Control of Automated Manufacturing Systems*. IEEE Computer Society Press, 1989.
18. Alan A. Desrochers and Robert Y. Al-Jaar. *Applications Of Petri Nets In Manufacturing Systems. Modeling, Control, And Performance Analysis*. IEEE Press, 1995.

19. M. Diaz, G. Juanole, and J. P. Courtiat. Observer — a concept for formal on-line validation of distributes systems. *IEEE Trans. on Software Engineering*, 20(12):900–913, 1994.
20. E. W. Dijkstra. Cooperating sequential processes. In F. Genuys, editor, *Programming Languages*. Academic Press, 1968.
21. E.I. Elwadi and D. Mitra. Statistical multiplexing with loss priorities in rate-based congestion control of high-speed networks. *IEEE Transactions on Communications*, 42(11):2989–3002, 1994.
22. J. Ezpeleta and J.M. Colom. Automatic synthesis of colored Petri nets for the control of FMS. *IEEE Transactions on Robotics and Automation*, 13(3):327–337, June 1997.
23. 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, April 1995.
24. J. Ezpeleta and J. Martínez. Formal specification and validation in production plants. In *Proceedings of the 3th. International Conference on Computer Integrated Manufacturing*, pages 64–73, Rensselaer Polytechnic Institute, Troy (New York), May 1992. IMACS.
25. J. Ezpeleta and L. Recalde. A deadlock avoidance approach for non-sequential resource allocation systems. In *Proc. of the Int. Conference on Systems, Man and Cybernetics*, Hammamet, Tunisia, 0ctober 2002.
26. J. Ezpeleta, F. Tricas, F. García- Vallés, and J.M. Colom. A Banker's solution for deadlock avoidance in FMS with routing flexibility and multi–resource states. *IEEE Transactions on Robotics and Automation*, 18(4):621–625, August 2002.
27. J. Ezpeleta and R. Valk. A polynomial solution for deadlock avoidance in assembly systems modelled with petri nets. In *Proceedings of the Multiconference on Computational Engineering in Systems Applications (CESA2003)*, pages 1–8, Lille (France), July, 9–11 2003.
28. M.P. Fanti, B. Maione, and B. Turchiano. Design of supervisors to avoid deadlock in flexible assembly systems. *The International Journal of Flexible Manufacturing Systems*, 14:157–175, 2002.
29. B. Farwer, D. Moldt, and F. Garcí-Vallés. An approach to modelling fms with dynamic object petri nets. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, Hammamet (Tunisia), October 2002.
30. J. C. Gentina, J. P. Bourey, and M. Kapusta. Coloured adaptive structured Petri nets. *Computer-Integrated Manufacturing*, 1(1):39–47, 1988.
31. J. C. Gentina, J. P. Bourey, and M. Kapusta. Coloured adaptive structured Petri nets (II). *Computer-Integrated Manufacturing*, 1(2):103–109, 1988.
32. S. B. Gershwin. *Manufacturing Systems Engineering*. Prentice-Hall, 1994.
33. A. N. Habermann. Prevention of systems deadlocks. *Communications of the ACM*, 12(7):373–385, July 1969.
34. C. Hanen and A. Munier. Cyclic scheduling problems: An overview. In Chretienne et al. [12].
35. YiSheng Huang, MuDer Jeng, and Xiaolan Xie. A deadlock prevention policy for flexible manufacturing systems using siphons. In *Proc. of the 2001 IEEE International Conference on Robotics and Automation*, pages 541–546, Seoul (Korea), May 2001.
36. O. Kummer and F. Wienberg. Renew. the reference net workshop. *Petri Net Newsletter*, (56):12–16, 1999.
37. N. G. Leveson and J. L. Stolzy. Safety analysis using Petri nets. *IEEE Trans. on Software Engineering*, 13(3):386–397, 1987.

38. E. López-Mellado and J.G. Morales-Montelongo. Agent-based distributed controllers for discrete manufacturing systems. In *Proceedings of the Multiconference on Computational Engineering in Systems Applications (CESA2003)*, pages 1–7, Lille (France), July, 9–11 2003.
39. 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.
40. 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.
41. D. Moldt and J. Ezpeleta. A proposal for flexible testing of deadlock control strategies in resource allocation systems. In *Proceedings of the International Conference on Computational Intelligence for Modelling Control and Automation (CIMCA'03)*, pages 586–595, Vienna, Austria, February 2003.
42. T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.
43. J. Park and S. Reveliotis. Deadlock avoidance in sequential resource allocation systems with multiple resource acquisitions and flexible routings. *IEEE Transactions on Automatic Control*, 46(10):1572–1583, October 2001.
44. J. M. Proth and X. Xie. *Petri Nets. A Tool for Design and Management of Manufacturing Systems*. Wiley, 1996.
45. L. Recalde and M. Silva. Petri Nets fluidification revisited: Semantics and steady state. *APII-JESA*, 35(4):435–449, 2001.
46. E. Renshaw. A survey of stepping-stone models in population dynamics. *Adv. Appl. Prob.*, 18:581–627, 1986.
47. E. Roszkowska and R. Wojcik. Problems of process flow feasibility in FAS. In K. Leiviska, editor, *IFAC CIM in Process and manufacturing Industries*, pages 115–120, Espoo, Finland, 1992. Oxford: Pergamon Press.
48. M. Silva. *Las Redes de Petri: en la Automática y la Informática*. AC, 1985.
49. M. Silva. Interleaving functional and performance structural analysis of net models. In M. Ajmone Marsan, editor, *Application and Theory of Petri Nets 1993*, volume 691 of *Lecture Notes in Computer Science*, pages 17–23. Springer, 1993.
50. M. Silva. Introducing Petri nets. In *Practice of Petri Nets in Manufacturing*, pages 1–62. Chapman & Hall, 1993.
51. M. Silva and L. Recalde. Petri nets and integrality relaxations: A view of continuous Petri nets. *IEEE Trans. on Systems, Man, and Cybernetics*, 32(4):314–327, 2002.
52. M. Silva and L. Recalde. On fluidification of petri net models: from discrete to hybrid and continuous models. In *IFAC Conference on Analysis and Design of Hybrid Systems, ADHS03*, pages 9–20, Saint-Malo, France, June 2003.
53. 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.
54. M. Silva and E. Teruel. Petri nets for the design and operation of manufacturing systems. *European Journal of Control*, 3(3):182–199, 1997.
55. M. Silva, E. Teruel, and J. M. Colom. Linear algebraic and linear programming techniques for the analysis of net systems. In G. Rozenberg and W. Reisig, editors, *Lectures in Petri Nets. I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*, pages 309–373. Springer, 1998.

56. E. Teruel, J. M. Colom, and M. Silva. Choice-free Petri nets: A model for deterministic concurrent systems with bulk services and arrivals. *IEEE Trans. on Systems, Man, and Cybernetics*, 27(1):73–83, 1997.
57. F. Tricas, F. García-Vallés, J.M. Colom, and J. Ezpeleta. An iterative method for deadlock prevention in FMS. In R. Boel and G. Stremersch, editors, *Discrete Event Systems: Analysis and Control. Proc. of the Workshop On Discrete Event Systems 2000*, pages 139–148, Ghent, Belgium, Aug 2000. Kluwer Academic Publishers.
58. C. Valentin-Roubinet. Modeling of hybrid systems: DAE supervised by Petri nets. the example of a gas storage. In *Proc. of ADPM'98*, pages 142–149, Reims, France, March 1998.
59. 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.
60. 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.
61. R. Valk. Petri nets as token objects - an introduction to elementary object nets. *Lecture Notes in Computer Science: 19th Int. Conf. on Application and Theory of Petri Nets, ICATPN'98, Lisbon, Portugal, June 1998*, 1420:1–25, June 1998.
62. 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.
63. 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.
64. N. Viswanadham and Y. Narahari. *Performance Modeling of Automated Manufacturing Systems.* Prentice-Hall, 1992.
65. Mengchu Zhou and Kurapati Venkatesh. *Modeling, Simulation, and Control of Flexible Manufacturing Systems : A Petri Net Approach*, volume 6 of *Series in Intelligent Control and Intelligent Automation.* World Scientific, 1999.