Contents

2 Petri Nets: a Graphical Tool for System Modelling and Analysis

Annie Choquet-Geniet and Pascal Richard

2.1 Overview of Petri Nets

Petri nets have been defined in 1962 by C.A. Petri [9,10] in order to model or to specify sequential and parallel systems including resources, data and events management, sequential evolution of the program counter. They are used either as a specification tool, or for analysing a previously designed system. In the first case, the system to design is modelled by a net, which is then implemented. In the second case, an existing system is modelled by a net, and its properties are then deduced, verifications are performed. One can verify that the system meets the requirements expressed in the specification, or can use the nets for performance analysis. Petri nets constitute a compromise between finite automata and the Turing machine .

There are two kinds of definitions concerning Petri nets:

- Definitions related to the structure of the nets, which are correlated to the static description of the system: what are the different parts of the system, what actions are performed by the system, what conditions are required for an action to be feasible, what effects does an action have on the different parts of the system ?
- Definitions related to the behaviour of the net, which describe the dynamic evolution of the system: what are the possible actions according to the current state, what happens when some of them are performed, what kinds of situations are then possible to reach, what further evolution can be considered...

A very nice aspect of Petri nets is that they support a graphical representation, which enables a good general view of the modelled system and a rather intuitive perception of its different components.

A Petri net (or Place/Transition net) is an oriented graph with two kinds of vertexes (see figure 2.1). It consists in a finite set of places (P), viewed as circles, a finite set of transitions (T), viewed as rectangles (corresponding to the different actions), a set of labelled arcs from places to transitions or from transitions to places, which express the conditions required for an action to be feasible and its consequences when it occurs. The valuation function W, is defined from $P \times T \cup T \times P$ in N, the set of natural numbers.

A marking function $M : P \to N$ is added to the previous description of the net, which represents the current state of the system, while the graph represents only the topology of the net, i.e. its different parts, and the connections between these parts. For each place p, $M(p)$ is interpreted as the number of tokens held by p.

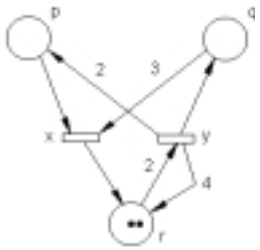


Fig. 2.1. There are three places (p, q, r) and two transitions (x and y). The firing of x requires one token in p and three in q, and it produces one token in r. The firing of y requires two tokens in place r, and it produces 2 tokens in p, one in q and 4 in r. In the initial state, p and q are empty, and r holds two tokens.

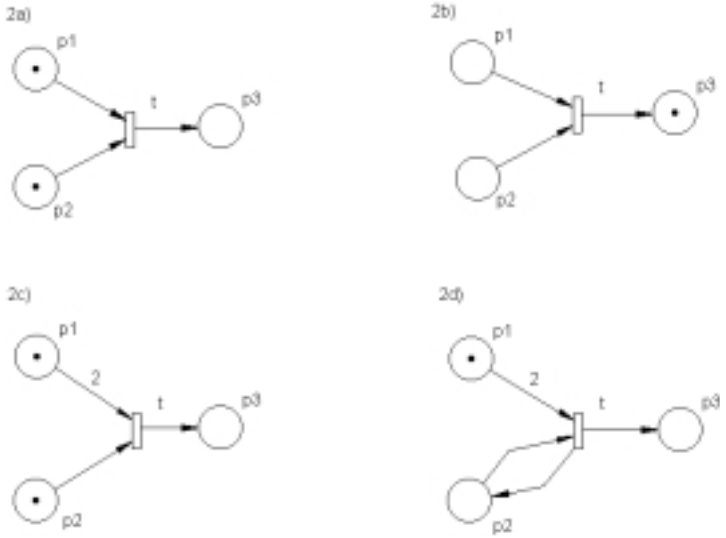


Fig. 2.2. The firing rule: on figure 2.2.a, the transition t can fire, and its firing produces the marking of Fig 2.2.b. On Fig. 2.2.c and 2.2.d, the transition t cannot fire because one token is missing, in place p1 (Fig. 2.2.c) or p2 (Fig. 2.2.d).

The dynamics of the system is described by means of the firing rule (figure 2.2): a transition can fire (or is enabled) if each of its input places p contains at least as many tokens as indicated by the valuation of the arc from p to t. These tokens are removed from the input places when the transition fires, and tokens are added to the output places, again according to the valuation of the arcs from t to these places.

The firing of transitions can straightforwardly be extended to the firing of sequences of transitions. For instance, if we consider the net of figure 2.1, the sequence $yyyx$ can fire and leads to the marking M so that $M(p) = 5, M(q) = 0$ and $M(r) = 9$.

Let us finally mention that there exist a large number of tools for editing and analysing Petri nets. As an example, we can mention Design/CPN [4], which provides both an editor of Petri nets, and an analysis environment. And let us also mention that an ISO norm 15909 for Petri nets is currently in preparation, and will be useful in the future.

2.2 Analysis and Specification of Case 1

We first present a brief overview of the different semantics which can be attached to the different components of a Petri net for the purpose of modelling. A place may be associated to: a class of resource, a counter, an event, a buffer (possibly with capacity), a condition. Transitions are generally connected to actions concerning: resources (allocation / desallocation), the evolution of a process (incrementation of the program counter), the processing induced by the occurrence of an event or by the verification of a condition, a buffer (production / consumption). Finally, tokens can represent: instances of a resource, contents of a buffer (in these two cases, tokens are associated to data), occurrences of an event (tokens are here associated to signals), the position of the program counter, the fact that a condition holds.

As we show next, there may be several approaches for modelling a system, according to the way the analysis of the system is approached. We present in the next section two approaches. In the first case, we treat a single instance of an order, but we can specify the ordered quantity. In the second case, we can specify the number of instances of an order, but not the ordered quantity.

2.2.1 One Order with a Data/Action Approach

We have assumed that quantities for each reference of a product are expressed as integer number. We deal with one single order, and we take into account neither arrival of new orders nor restocking. In subsection "One Order - One Reference", we consider that the stock contains only one reference of a product, and in subsection "One Order - Several Reference", we enlarge our hypothesis and consider several references.

One Order - One Reference

Question 1: *What are the basic data ?*

Answer: *We have defined four types of data:*
- *the stocked products, which constitute the stock;*
- *the ordered products, which define the pending order;*
- *the invoiced order;*
- *the non satisfied order.*

The last two data are semantically connected to the condition "the order can be invoiced", and correspond to the two possible values of this condition.

Question 2: *What actions are performed by the system ?*

Answer: *The processing of the order, which consists in invoicing it, either completely, or partially. An order may not be completely invoiced, due to the lack of the required amount of product.*

Question 3: *What does the invoicing of the order require ? What are its effects ?*

Answer: *There must be enough products in the stock, i.e. at least as many as mentioned in the order. If it is the case, the corresponding quantity of the reference is removed from the stock, the order is deleted, and an invoiced order is produced. Otherwise the stock is emptied, the order remains partially pending (there is a partial invoicing), and a "non satisfied order" is produced. Two pieces of information are thus represented: the status of the order when processed, and, if it has not been completely invoiced, the amount of products which could not be handed over (figure 2.3).*

Once the analysis of the system is achieved, we define the corresponding Petri net. From our analysis four places appear, each of them corresponding to a type of data: one place represents the stocked products (ST), one the ordered products (OP), one the invoiced order (IO) and one the non satisfied order (NSO). Initially, the marking of the place ST corresponds to the amount of products held in the stock, the marking of OP the quantity of products ordered, and both places IO and NSO are empty since the order has not still been processed.

Question 4: *Which functionality must be supported by the net ?*

Answer: *The net must be able to compare two markings: the marking of the place ST must be greater than or equal to this of the place OP for the order to be invoiced. If it is not the case (and only in this case) a non satisfied order must be produced.*

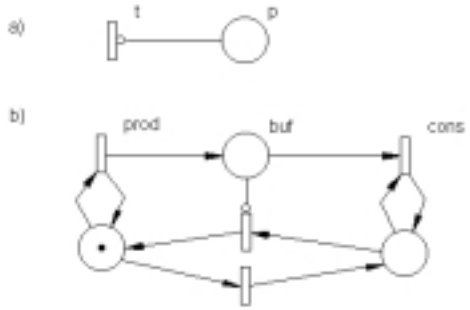


Fig. 2.3. (a) Representation of an inhibitor arc (b) this net models the activity of a producer which produces some objects and lays them down in a store. After a while, a consumer starts to consume the objects, and the producer goes on producing only when the store is empty.

Since the test to zero is required, we will use the extension of Petri net which integrates inhibitor arcs since they explicitly implement the test to zero.

Nets with inhibitor arcs (figure 2.4) are Petri nets where the firing rule has been modified in order to explicitly integrate the test to zero. An inhibitor arc enables the firing of a transition only when its input places are empty (i.e. it inhibits the firing when the input places hold tokens). The valuation function W is defined from $P \times T \cup T \times P \rightarrow N \cup \{\Theta\}$ (where Θ is a symbol, which does not belong to and model the inhibitor arc) and the firing rule becomes: t can fire if $M(p)$ is greater than or equal to $W(p,t)$ for each p so that $W(p,t) \in N$ and $M(p) = 0$ for each p so that $W(p,t) = \Theta$.

We introduce three transitions: a classical transition HO which corresponds to the handing over of products. Its firing requires one token in both places OP and ST. It means that each time a product is taken within the stock, it is removed from the order. This transition can fire until either the stock is empty or the ordered products have been completely handed over. Two further transitions (Error and OK), connected to inhibitor arcs enable the production of a token within either the place IO or the place NSO.

First case : $n < q$. After n firings of HO, Error will fire since the marking of ST is 0 and the marking of OP is still positive. In the final marking, ST is empty, OP contains still $q - n$ tokens (the ordered quantity which could not be invoiced), IO is empty since the order has not be completely invoiced and NSO holds one token.

Second case : $n \geq q$. After q firings of HO, the transition OK will fire since the marking of OP is 0. In the final marking, ST holds $n-q$ tokens (the quantity still present in stock), OP is empty, IO holds one token (the order has been completely invoiced) and NSO is empty. The net of figure 2.4 presents our solution.

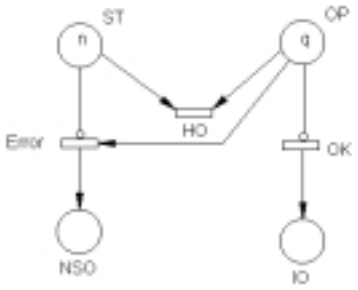


Fig. 2.4. A solution using inhibitor arcs. Legend: ST: stock; OP: ordered products; IO: invoiced order; NSO: non satisfied order; HO: handing over of product; Error: the order could not be satisfied; OK: the order is invoiced. n is the quantity present in the stock and q is the ordered quantity.

One Order - Several References

We now enlarge our hypothesis, and assume the existence of several references. The stock as well as the order are thus defined by the quantities of each of them, either stocked or ordered.

Question 5: *What is an order composed of ?*

Answer: *It consists of a list of pairs (reference, quantity), where a given reference appears at most once. The amount of each reference is still assumed to be known.*

The number of places representing either the stock or the order increases: if there are n possible references, n places ST1, ST2, ... , STn model the stock, and n places OP1, ... , OPn model the order. The transition HO is split up into n transitions HO1,... , HOn, each of them concerning one reference and having the same semantics as the transition HO of figure 2.4.

The order is then invoiced if and only if every place OPi is empty. Finally, the transition Error is also split up into Error1, ... , Errorn. After the complete treatment of the order, either IO contains one token if the order could be completely handed over, or the place NSO contains as many tokens as there are (partially) missing references. This extended solution is presented on figure 2.5, for two references.

2.2.2 One Order with a Structural Approach

In this section, we analyse the system differently, and consider that we deal with an unique kind of order: the ordered quantity cannot be chosen, i.e. it is a constant of the system, while the stocked quantity remains a parameter. In our first approach, we were able to consider any order, but only one instance.

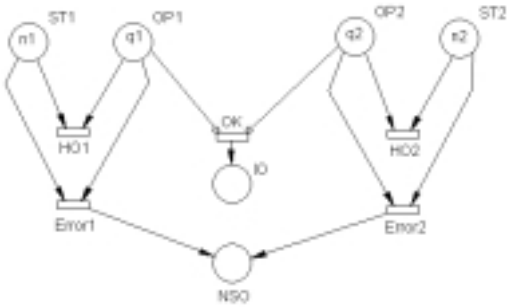


Fig. 2.5. A solution for the problem of one order with two references using inhibitor arcs.

In this second approach, we take into account several instances of a given order: the number of instances is a parameter (expressed through the initial marking), but not the ordered quantities.

If we assume the ordered quantity to be known before we proceed to the construction of the Petri net, we can avoid the use of inhibitor arcs. In this case, the comparison of integers is supported by the firing rule (which integrates it explicitly). The ordered quantity is included within the topology of the net.

One Order - One Reference

Here again, we first deal with one single reference, and afterwards, we will assume several ones.

Question 6: *What are now the data, and what does the net model do here?*

Answer: *The data in this approach are: the stocked products, the pending orders, the invoiced orders. An order is here considered in its whole, thus it is either invoiced, or it remains pending. The partial handing over of the ordered products cannot here be modelled.*

Using this approach, we define three places: one models the stocked products (ST), one the pending orders (PO) and one corresponds to the invoiced orders (IO). The marking of ST corresponds again to the amount of products held in the stock. The marking of PO corresponds to the number of pending orders. Finally, the initial marking of IO is 0. It will afterwards represent the number of already invoiced orders.

There is one single transition (HO), corresponding to the handing over of the order, considered in its whole. It can fire only if there are enough products within the stock. This condition is expressed by means of the valuation of the arc between ST and HO. It then removes the number of tokens corresponding

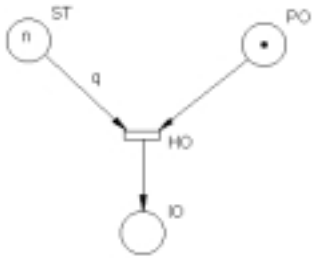


Fig. 2.6. A solution to the problem of one order for one reference, using a structural approach.

to the ordered quantity from the place ST, one token from PO, and adds one token to the place IO (figure 2.6).

In the first solution, the modification of the order does not affect the topology of the net, but is taken into account by a new marking. In the second approach, the modification of the order induces the modification of the net which has thus to be recompiled after any modification of order. Furthermore, in our second approach, we can express only the fact that an order can be invoiced, but if it can not, we have no information about the missing amount of products.

One Order - Several References

Here again, we enlarge the system by taking several references into account. Our assumption about the composition of an order is the same as in the previous section and we still suppose that the ordered quantity for each reference is known before the design of the net is carried over.

The only change brought to the net of figure 2.6 concerns the modelling of the stock. Again, the place ST is split up into n places (if there are n different references). The arcs between STi and HO are valuated by the amount of the i-th reference which is ordered (see figure 2.7).

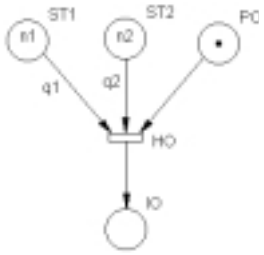


Fig. 2.7. A solution for the problem of one order, with two references.

In the next section, which deals with case 2, we will focus on this second approach. However, whatever the chosen approach, we will have to use a high level net. The structure of the net when using the first approach, would be more complex: we would have used fifos for modelling the file of pending orders; then, for each order, we would start to hand it over, but, in the case of non satisfaction, we would have to restore the stock, and then to delete the order from the fifo. This would have required the use of several inhibitor arcs. For the sake of simplicity, we have chosen to present only the solution corresponding to the second approach here.

2.2.3 Several Orders

In this section we complete the model in order to take into account several orders. As in the previous section, orders and products are completely defined at the modelling step.

Question 7: *How to model several orders ?*

Answer: *We consider different references of products, which differ by the ordered quantities and, for each kind, there may exist several instances. In the sequel, order will mean « kind of order ». Since an order is modelled through the structure of the net, the associated subnet must be repeated for every order. New places « pending » and « invoiced » must be created and also a new transition for changing the state of the order and consuming products in the stocks. Consider two orders: order 1 requires three products A and two products B, and order 2 requires one product B. As shown in figure 2.8, adding a new order increases the size of the net with two places, one transition and at most $n + 2$ arcs (where n is the number of different products). The size of the net is polynomially bounded in the number of orders.*

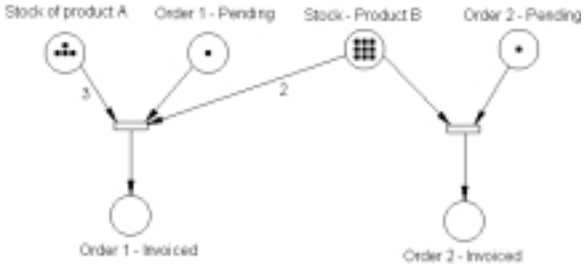


Fig. 2.8. Low-level net modelling the system with two orders.

This modelling approach leads to the problem of the growth of the size of the net. It cannot be avoided with classical low-level nets since every order is modelled by new places, transitions and arcs. The size can be decreased using high

level nets such as coloured nets [6]. Coloured nets have been defined in order to give a concise model even for complex systems, while keeping the same expressiveness (i.e. every coloured net can be unfolded into a classical low-level net). It allows to merge all identical parts of a low-level net into only one structure. We now only present basic coloured net concepts used hereafter.

A coloured net is defined by a finite set of colours, let us denote it C. A colour can also be considered as a data type (e.g. a type of order). Colours are totally ordered and every token is coloured. To every transition is associated C or a subset of C, and to every arc is associated a function which models colour changes when a transition is fired (in practice the arcs are only labelled by the name of the functions). As example we give three classical functions that are widely used (n is the number of colours in the set C, and the symbol | denotes the modulo operation):

Identity: $Id < Colour_i >=< Colour_i >$
Successor: $Succ < Colour_i >=< Colour_{i+1|n} >$
Predecessor:$Prec < Colour_i >=< Colour_{i-1|n} >$

Figure 2.9 gives an example of a coloured net with the set of colours:

$$C = \{< r >, < b >, < j >, < v >\}$$

Note that the ordering of colours is very important. For instance $< b >$ is the successor of $< r >$, and conversely $< r >$ is the predecessor of $< b >$. Furthermore the functions Successor and Predecessor behave cyclically: precisely $Succ < v >=< r >$, and $Prec < r >=< v >$.

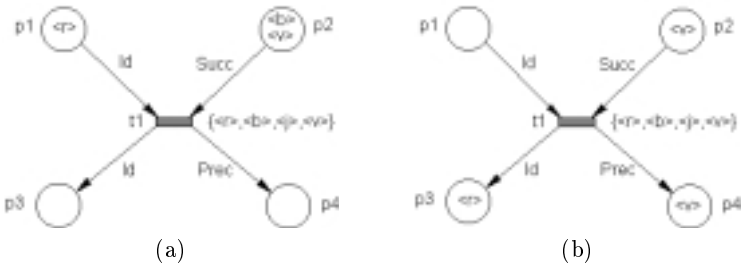


Fig. 2.9. (a) Structure of a coloured net with the set of colours C={<r>,<b>, <j>,<v>}; (b) The net after the fire of transition t1 with colour <r>.

We describe next the behaviour of a coloured net. We first define the enabling process of a transition and then the firing process . Let t be a transition and c be a colour belonging to the set associated to t, we check that the required coloured tokens (computed with functions using the colour c) are available in every input place of t. If it is not the case, we proceed in the same way with an

other colour associated to t until the previous condition holds for a colour a. We say that the transition is enabled for the colour a. The firing is processed in two steps. First tokens in the input places of t are removed using the functions on the arcs connecting the places to t, and then coloured tokens are added to the output places of t according to the functions labelling output arcs of t.

Consider figure 2.9, we first show that the transition $t1$ is enabled using the functions on arcs connected to the input places of $t1$. For $p1$ $Id < r >=< r >$, since $p1$ contains a token with colour $< r >$, the condition is satisfied. For $p2$, $Succ < r >=< b >$, since such a token is available in $p2$, the condition is also satisfied. As consequence $t1$ is enabled for the colour $< r >$. During the firing, $Id < r >$ is removed from $p1$, $Succ < r >$ is removed from $p2$, Id $< r >$ is added to $p3$ and $Prec < r >$ is added to $p4$.

Question 8: *How to model the products and their stocks ?*

Answer: *The places ST1,...,STn are merged into one single place ST. One colour is created per product. The colour can be viewed as the name of the product. The stocks are modelled by tokens of the corresponding colours and quantities.*

Question 9: *Question: How to model the orders ?*

Answer: *Here again places PO1,...,POq are merged into one single place PO and one colour is associated to each order. An order becomes a coloured token. All these tokens must have a different colour. The set of pending orders (coloured tokens) are stored in the place PO. The quantities of products required in the orders are defined by a function which gives the number of tokens in the colours associated to the products. The places modelling invoiced orders are also merged into a place IO. For instance, consider an order <1> of three units of product A and two units of product B, and an order <2> which only requires one unit of product B. The function (Qty) modelling these orders is defined as follow:*

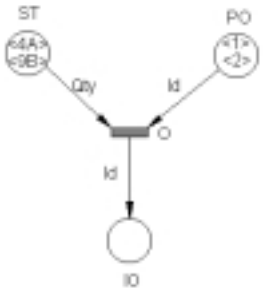
Qty <1> = <A><A><A><B><B>

Qty <2> = <B>

Functions, as the set of colours, are not directly integrated within the structure of the net (i.e. places, transitions and arcs). But they are rather stored within a table associated to the net. In the following O denotes the set of colours associated to the set of orders, and P denotes the set of colours associated to the set of products. Figure 2.10 gives the whole model of the system.

As said before every coloured net can be unfolded into an equivalent low-level Place/Transition net. The unfolded net of the net presented figure 2.10 is exactly the net of figure 2.8.

Question 10: *Can we schedule the orders with a given policy ?*



Qty	
<1>	<A><A><A><B><B>
<2>	<B>

Fig. 2.10. Coloured net modelling the order system, with its table of functions.

Answer: *In the net of figure 2.10, all orders that can be invoiced correspond to the selected colours that enable the transitions. Scheduling orders is equivalent to schedule enabled transitions. Moreover, if a given order cannot be satisfied because there are not enough products in stock, it must be delayed in order to pass to the next order with respect to the policy. So we need first to test that the selected order can be satisfied. If not, it is delayed, else it is invoiced.*

But as explained before, such tests cannot be done with low-level nets. And since coloured nets have the same modelling power as low-level nets, then it cannot be done with them. To implement a scheduling policy of orders with our approach, we need an extension of Petri nets including inhibitor arcs and fifos or priority nets. But these extensions increase the expressiveness power, and as consequence, most of the classical properties that can be verified on classical Petri nets fall indecidable (the model becomes as powerful as a Turing machine).

2.3 Analysis and Specification of Case 2

Up to now the system studied is fully static: all orders and products are known at the modelling step. We now extend the previous case to model the flow of orders and the entry flow of products in stocks. Without loss of generality, we only consider that new orders can arrive in the system, but not new references. For the other cases, the principles would be the same and are left to the reader.

2.3.1 Entry Flow in Stocks

Question 11: *How to model an entry flow in stocks ?*

Answer: *We need a special transition that can be enabled without any condition. Such transitions are called source transitions , since they do not have any input place. Source transitions are always enabled and can fire at any time. So for modelling entry flow of products in stocks, we add a source*

transition, noted later EFO, connected to the place ST. The set of colours that labelled the transition is the subset of colours dedicated to the products. The function associated to the corresponding arc is Identity. So that at any time every colour of P satisfies the enabling condition and the corresponding product is added to the stock.

Question 12: *How is the entry flow controlled ?*

Answer: *The entry flow is nondeterministic, in the sense that the reference introduced in the stock is not controlled. If the source transition fires three times with colour <A> then three coloured tokens are added to the Stock. But the selected colour is always chosen arbitrarily. With these nondeterministic choices, the analysis deals with all possible behaviours of the system. Figure 2.11 gives the model of the entry flow of products.*

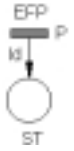


Fig. 2.11. Order systems with entry flow of products EFP.

2.3.2 Flows of orders

Question 13: *How to model new orders ?*

Answer: *As for the entry flow of products, we need a new source transition. It is connected to the place that holds pending orders. In practice the order system can deal with an infinite behaviour and so with an infinite set of orders. Distinguishing the orders requires to generate one new colour that must not exist in the system. This source transition, noted later EFO, is in fact a generator of new colours in order to deal with an infinite number of orders. Furthermore quantities of products indicated in the order are defined by extending the Quantity function (noted Qty in the net). The table containing the function Qty can become infinite.*

But the unfolding of a Coloured Net with an infinite set of colours produced an infinite set of places and transitions. So the graph of the unfolded net is infinite. It has been shown that the expressiveness of the model becomes in that case equivalent to a Turing machine. But in practice the number of orders in the system can be kept finite by destroying invoiced orders. As a consequence the set of colours and the size of the function Quantity are finite.

Question 14: *How to cancel pending orders or destroy old invoiced ones ?*

Answer: *In that case we have to consume tokens. The only way to proceed is to use a sink transition (i.e. a transition without output place). When a sink transition fires tokens are removed from the input places. Since no output place exists no new token is generated. Removed tokens from the entry places have been destroyed. So in the final model we need two sink transitions, DIO and CO, for modelling respectively the deleted invoiced orders and the cancelled orders.*

The choices of cancelled orders or destroyed invoiced orders are nondeterministic. We include all feasible behaviours of the order system. When an order is destroyed, the table of function Qty must be updated and the corresponding removed colours can be reused later. Figure 2.12 gives the final model of the case study 2.

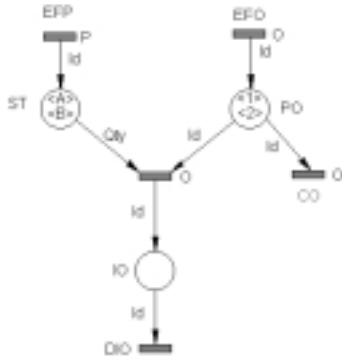


Fig. 2.12. Final model.

2.4 Validation of the Specification

Petri nets analysis consists in verifying two kinds of properties: behavioural properties and structural properties [1,8]. Behavioural properties depend on the initial marking (initial distribution of tokens in the places) and by opposition structural properties focus on the structure without any initial marking consideration. Generally solving these problems requires exponential time and space, even for simple net structures [3]. The main behavioural properties are reachability, liveness, boundedness, deadlock-freeness. And the main structural properties are invariants and structural liveness, structural boundedness and also structural deadlock-freeness.

Obviously when coloured Petri nets are used the objective is not to unfold the net because of it usually enormous size (even if it is always possible). Some

specific algorithms have been developed. But in order to be clear, we do not detail such specific solutions and we only present the analysis of a low-level net.

Behavioural properties can be checked by building the reachability graph from the Petri net. In this graph vertexes are markings and edges are labelled by transitions that change one marking into another. If the system has an infinite number of different states, then the reachability graph is infinite.

The reachability property consists in verifying that a marking (a given state of the system) is reachable from the initial marking. The problem is very complex to solve, the decidability proof has held for ten years. So deciding if a marking is reachable can be decided by searching the according vertex in a finite reachability graph (otherwise a complex algorithm is required). The path from the initial marking to the searched marking produces a feasible firing sequence that proves the reachability. Liveness consists in verifying that every transition can always fire in the system. More precisely, there is for every reachable marking a firable sequence, containing at least once each transition. This property ensures that every operation (modelled by transitions) can always be performed in the system. So there is no partial deadlock of the system. If the number of different markings is finite, then a live net has a strongly connected reachability graph. That graph property can be checked in $O(n^2)$ in the size of the reachability graph (which usually has itself an exponential size in comparison with the size of the net). For instance the net of figure 2.12 is live. The third property is boundedness: there is no place that can have an infinite number of tokens while playing the token game. In order to have an infinite number of tokens in a place, there must be an infinite firing sequence producing them. So an unbounded net has an infinite reachability graph. But it has been shown that a reachability graph is infinite if and only if, it has infinite paths. These paths can be cut by identifying repetitive firing sequences. The obtained graph is called the Karp's graph [7] and is finite for every net. If the number of tokens increases while firing a repetitive sequence, then the places in the corresponding vertex is marked by a the symbol ω. So verifying boundedness consists in searching this symbol in the Karp's graph. For instance the net of figure 2.9 is bounded and has a finite reachability graph. The last behavioural property presented is deadlock-freeness. A marking is deadlocked if it enables no transition. So in a deadlock-free system, there always exists a transition to fire (i.e. an operation to do). That property can be checked using the finite reachability graph: it is sufficient to check that there is no leaf in the graph (i.e. a vertex without any successor).

Structural properties are studied by using an algebraic representation of the net. From the structure of the net an incidence matrix can be defined (as in a classical graph theory). Markings are vectors which are indexed on the set of places. The number of transitions in a firing sequence can be stored in a characteristic vector. Let us denote C the incidence matrix, X a characteristic vector, and M_0 the initial marking of the net. Every reachable marking M verifies the marking equation: $M = M_0 + CX$. Every reachable marking satisfies a linear algebraic system of equations [5]. But take care, the converse is not true: com-

puting M and/or X using the marking equation can lead to spurious solutions (i.e. markings and sequences that are not feasible on the net while playing the token game). As a consequence many algorithms working on structural properties are semi-decision algorithms (verifying a necessary or a sufficient condition but not both).

Invariant (also called semiflows) refers to stable situations in the net behaviour whatever the initial marking is. The weighted sum of tokens in a set of places is called place invariant. It is always the case for a set of places modelling renewable resources or mutual exclusion sub-systems. Computing place-invariants can be easily done by solving the system ${}^tCX = 0$ (i.e. the kernel of the transposed matrix C in classical linear algebra) with the Fourier-Motzkin's algorithm [2]. For instance in figure 2.6 the places pending and invoiced constitute a place-invariant since no orders are introduced or deleted in the system. Another kind of invariant deals with repetitive firing sequences (that lead from one marking to the same marking). These invariants based on transitions can be easily computed by solving the system $CX = 0$. Since it is the dual system of the place invariant one, it can be solved by the same algorithm. Structural properties can then be efficiently semi-decided using classical linear programme solvers.

2.5 The Natural Language Description of the Specifications

2.5.1 Case 1

An order is defined by a set of references of products. For each reference is known the ordered quantity, inventory levels and its status which are natural numbers. The status of the order is defined by two different variables upon natural numbers. The first variable contains one if the order is not invoiced, and zero otherwise, and the second variable stores one if the order is invoiced and zero otherwise. The system provides an operation (invoice order) that can be executed if, and only if, every reference is available according to the ordered quantity. When the operation is completed, the operation decrements the pending order variable, increments the invoiced order variable and updates the quantities in stocks.

2.5.2 Case 2

The ordering system is an extension of case study 1, that allows dynamic arrivals and cancellations of orders, and dynamic arrivals of raw of materials (i.e. products in the stock). The definitions of orders and stocks are the same as in case 1. The system provides four new operations to input new products in the stock, to input new orders, to cancel orders not yet invoiced, and to delete invoiced orders. The operation introducing products in stock increases the value of

the variables associated to these references. The operation which introduces new orders defines a set of references and their ordered quantities. Both cancellation operations decrease the status variables, as defined in case 1. An operation that invoices an order is defined per order in the same manner than in case 1.

2.6 Conclusion

Through this chapter, we have outlined the strong points as well as the weaknesses of the Petri nets. When the order flow is completely defined at the modelling step, Petri nets provide a very nice and concise modelling of the system. The graphical support is very helpful, since it gives a good synthetic view of the system in its whole, points out the different objects which constitute it, and describes their interactions.

The structural approach is here completely suitable to the problem, since it uses the firing rule in order to model the invoicing of orders. Moreover, the model we get with this approach can be analysed, using all the analysing facilities of Petri nets. The limits of this approach come from the fact that the orders have to be specified in the model. If we want to consider any possible order, that is to consider the order as a parameter, our previous approach does not work anymore, and we stumble to the main weakness of the model: the test to zero fails, so does any comparison between markings. Thus we have to change the power of the model, and to integrate inhibitor arcs. But we get then the power of a Turing Machine, which forbids any analysis of the model.

Thus, the data/action approach can be used only in order to play the token game (in our case, the process will end, so does the token game). Now, if we take an order flow into account, we use a high level coloured net, here again, any analysing facility fails. But we can get an instantaneous description of the system, at any time, and perform some analysis or play the token game, from the state of the system at that time. Finally, we did not present any solution including the scheduling of the orders, since it would have required the use of fifos or of priorities and inhibitor arcs, and would have supposed a rather complicated net, dedicated to the deletion of the unsatisfied order, and the restoration of the stock, after a partial handing over of products for a finally non satisfied order.

As a general conclusion, Petri nets seem to be rather suitable for the modelling of the first case (since the invoicing of order is directly modelled by the firing rule), but for the second case, their weaknesses (no comparison between markings, the structure cannot dynamically be modified) are too important, and we cannot provide a good solution, unless we use an extended model with the power of the Turing machine. Thus, for the second case, the modelling with Petri nets is not really suitable.

References

1. Choquet-Geniet A., Vidal-Naquet G. (1993) Petri nets and parallel systems. Armand Colin (in french)

2. Colom J.M., Silva M. (1991) Convex Geometry and Semiflows in P/T Nets. A Comparative Study of Algorithms for Computation of Minimal P-Semiflows. in:Advances in Petri nets'90, 79–112, Springer Verlag
3. Desel J., Esparza J. (1995) Free-Choice Petri nets, Cambridge Tracts in Computer Science n° 40, Cambridge University Press
4. Design/CPN, tool Homepage: http://www.daimi.au.dk/designCPN/
5. David R., Alla H. (1992) Petri nets and Grafcet, Prentice-Hall
6. Jensen K. (1997) Coloured Petri nets, Basic concepts, Analysis Methods and Practical Use. Monographs in Theoretical Computer Science, Springer Verlag
7. Karp R.M., Miller R.E. (1969) Parallel program schemata, Journal of Computer System Sciences 3:147–195
8. Murata T. (1989) Petri nets : Properties, Analysis and Applications, Proceedings of the IEEE 77(4):541–580
9. Peterson J.L. (1981) Petri net theory and the modelling of systems, Prentice-Hall
10. Petri C.A. (1962) Kommunikation mit Automaten. (German) Schriften des Rheinisch-Westfälischen Institutes für instrumentelle Mathematik an der Universität Bonn, Nr. 2, Bonn