Copyright © 2002 IFAC
15th Triennial World Congress, Barcelona, Spain

CSP GENERATION FROM PETRI-NETS MODELS

Daniel Riera [*] **Miquel A. Piera** [*] **Antoni Guasch** [**]

[*] *Enginyeria de Sistemes i Automàtica,
Universitat Autònoma de Barcelona, Barcelona, Spain
{Daniel.Riera,MiguelAngel.Piera}@uab.es*
[**] *Instituto de Robótica e Informática Industrial,
UPC/CSIC, Barcelona, Spain , Guasch@esaii.upc.es*

Abstract: The use of traditional production planning techniques is constrained by large numbers of decision variables, uncertainty in demand and time production, and non-deterministic system behaviour, characteristics intrinsic in manufacturing. The aim of this paper is to present a methodology that combines the modelling power of petri-nets (PN) to represent both manufacturing architecture and production logistics, together with the optimisation performance given by constraint programming (CP). While PN can represent the entirety of any system, CP is effective in solving large problems, especially in area of planning. The foundations to generate a Constraint Satisfaction Problem (CSP) from a PN are given. *Copyright* © 2002 IFAC

Keywords: Methodology, Petri-nets, Constraint Satisfaction Problems, Planning, Production systems

1. INTRODUCTION

In the last few years, many methods and tools have been developed to improve production performance in the manufacturing industry. These approaches try to tackle changes in production objectives such as *high production diversity* (instead of *high production volume*), *make to order* (instead of *make to stock*), and *zero stock* (instead of *just in time*) policies. Although OR (Operations Research) methodologies have been proved to perform well in front of some types of problems, they fall short to tackle present flexible manufacturing scheduling production demands.

Petri-nets (PN) have shown to be successful tools for modelling Flexible Manufacturing Systems (FMS) due to several advantages such as the conciseness of embodying both the static structure and the dynamics, the availability of the mathematical analysis techniques, and its graphical nature (Jensen, 1997; Silva and Valette, 1989; Zimmermann *et al.*, 1996). Furthermore, PN are very suitable to model and visualize patterns of behaviour comprising concurrency, synchronization and resources sharing, which are the main characteristics of a FMS.

Thus, PN formalism is used to specify all the events that lead to a system state change (e.g. beginning or ending of a production activity) together with all the preconditions that should be satisfied in order an event could occur (e.g. raw material and resources requirements, etc.). Note that a PN model of a production system is enough to simulate its behaviour, which means that the model formalizes both:

- Architecture Production Constraints: Automated transport units (i.e. conveyors, robots, manipulators, etc.) bind the material flow in the production system.
- Logistic Production Constraints: Despite flexible production mechanisms allow different scheduling policies, product recipes reduce subsystem interaction due to several factors such as the operation sequence in which the raw material must be processed.

Constraint Programming (CP) has been mainly chosen because of its good optimisation performance. Since CP is usually embedded in declarative programming, the user does not need to write an algorithm to solve the problem but only to model the problem to be solved. Therefore, once the model is generated, CP can optimise it without requiring an expert to rule it.

In this paper, the main aspects of a PN analysis methodology to deal automatically with all the constraints of a production system are presented. The aim of this methodology is to automatize the constraints modelling phase of a production system — described previously in PN formalism — in order to generate a quasi-optimal scheduling for a particular system state and production goal, making the optimisation phase transparent to the user (See Fig. 1).

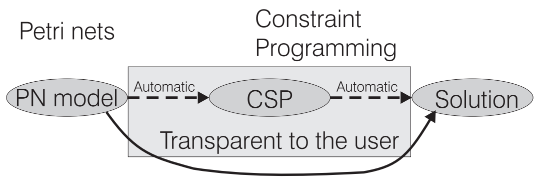


Fig. 1. Presented methodology

It should be noted that despite there are several PN analysis tools to improve system performance, those can be used only when the PN model shows certain characteristics, which is not always the case of PN models of production systems. On the other hand, despite CP offers a powerful technology to deal with the best planning policy for a production system, it requires a deep knowledge in both the production system and the CP technology, and time to develop the CP model. Thus, the present work tries to combine the main advantages of both methodologies in order to offer good scheduling policies, minimizing the time spent in the modelling and optimization tasks.

The foundations for the analysis on a PN model and the generation of the CP model are presented. This analysis is made by detecting structures in the PN, which are translated directly into the CSP constraints.

In sections 2 and 3 PN and CP backgrounds are presented. Section 4 introduces the purposed methodology. Section 5 illustrates this methodology by means of an example. Finally, Sections 6 and 7 discuss the benefits, and present the conclusions and future work.

2. PETRI-NETS BACKGROUND

A *Petri-net* (PN) is a particular kind of directed graph, together with an initial state called the *initial marking*. An ordinary PN is a 5-tupla $N = (P, T, I, O, M_0)$:

- $P = \{p_1, \ldots, p_n\}$ is the set of places, represented graphically by circles.
- $T = \{t_1, \ldots, t_m\}$ is the set of transitions, represented graphically by bold lines or rectangles.
- $I : (P \times T) \rightarrow \mathbb{N}$ is a function that defines the weight of directed arcs from places to transitions.
- $O : (T \times P) \rightarrow \mathbb{N}$ is a function that defines the weight of directed arcs from transitions to places.
- M_0 is the initial marking.

A *marking* is an array that assigns to each place a non-negative integer. If a marking assigns to place p a value k ($k \in \mathbb{Z}^+$), p is marked with k tokens, represented graphically by black dots.

A transition t_i is said to be enabled by a marking M, if $\forall p \in P : M(p) \geq I(p, t_i)$. The firing transition generates a new marking M' which can be computed by withdrawing $I(p_k, t_i)$ tokens from each p_k input place of t_i, and by adding $O(p_j, t_i)$ tokens to each p_j output place of t_i.

In manufacturing terms, *transitions* are used to model operations (firing a transition can represent a task or process initiation or an ending of a task), *places* are used to model buffers and resources status, connecting *arcs* specify logical relationships and resource constraints among operations, and *tokens* represent material and resources conditions.

3. CONSTRAINT PROGRAMMING BACKGROUND

Constraints arise in most areas of human endeavour. A constraint is simply a logical relation among several unknowns (or variables), each taking a value in a given domain. The constraint thus restricts the possible values that variables can take. CP is the study of computational systems based on constraints. The main idea is to solve problems by stating constraints (requirements) about the problem area and, consequently, finding a solution satisfying all the constraints.

The earliest ideas leading to CP may be found in the **Artificial Intelligence** (AI) with the *scene labelling* problem (Waltz, 1975) and the *interactive graphics* (Sutherland, 1963).

Gallaire (1985) and Jaffar and Lassez (1987) noted that logic programming was just a particular kind of CP. The basic idea behind **Logic Programming** (LP), and declarative programming in general, is that the user states *what* has to be solved instead of *how* to solve it, which is very close to the idea of constraints.

Recent advances promise that CP and **Operations Research** (OP), can exploit each other, in particular, the CP can serve as a roof platform for integrating various constraint solving algorithms including those developed and checked to be successful in OR.

Then, CP combines ideas from a number of fields including Artificial Intelligence, Combinatorial Algorithms, Computational Logic, Discrete Mathematics, Neural Networks, Operations Research, Programming Languages and Symbolic Computation.

The problems solved using CP are called Constraint Satisfaction Problems (CSP). A CSP is defined as:

- a set of variables,
$$X = \{x_1, \ldots, x_s\}$$
- for each variable x_i, a finite set D_i of possible values (its *domain*), and
- a set of *constraints* restricting the values that the variables can simultaneously take.

Such as Barták (1999) says: "Constraint programming is an emergent software technology for declarative description and effective solving of hard real life problems, especially in areas of planning and scheduling".

4. PETRI-NET MODELS TO CP MODELS

4.1 *CSP Components Identification*

In order to analyse the PN model and generate the CSP problem, the first step is to identify the elements composing the CSP. These elements are:

- Variables

  There is a variable for each firing of a PN transition. Hence, every transition is associated to a list of times. The length of this list is given by the number of times the transition is fired (or a higher bound if unknown). These are called *transition variables*.

  The second set of variables are those representing the number of firings of the transitions. These variables are called *firing variables*.

  Apart of these, there are also *Boolean variables* which indicate the paths followed by tokens in bifurcations.
- Domains

  The domains on the *transition variables* are defined using the knowledge about the problem. Usually the fact of reducing the domains on the variables makes the search faster. *Firing variables* depend on the bounds found by the *Transitions Firings Bounding* algorithm (See Alg.1).
- Constraints

  They are generated in the PN structures detection phase (See Section 4.3) and restrict the paths which can be followed by every token and the times when transitions can be fired.

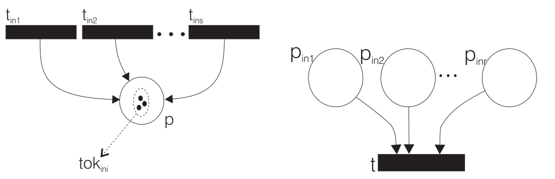
4.2 *Calculation of the Bounds of the Transitions Firings*

Since the variables are the firings of the transitions in the PN, it is necessary to bound the number of times they happen (Λ). Otherwise the number of variables would become unmanageable.

An algorithm which depends on the PN and the initial state (M_0)of the problem is proposed:

Algorithm 1. Transitions Firings Bounding

Step 1: Initialisation of Λ_0 by applying rules T→P and P→T (See below) for each place and transition respectively. First T→P is applied to every place in the PN and later P→T calculates the initial λ–values for every transition.



(a) T→P (b) P→T

Fig. 2. General cases for place and transition feeding

T→P finds the maximum number of tokens each place $p \in P$ can contain during all the system run. This is calculated applying to the general case shown in Fig.2(a) the formula presented in Eq.1.

$$tok(p) = tok_{\text{ini}}(p) + \sum_{i=1}^{m} \lambda_i \cdot O_{t_i,p} \qquad (1)$$

where $tok_{\text{ini}}(p)$ is the initial number of tokens of place p in the initial marking (M_0).

P→T initialises the number of times each transition $t \in T$ might be fired during the system run. The number or expression (if there are unknown values) is given by the application of the formula in Eq.2 to the general case shown in Fig.2(b).

$$\lambda(t) = \min_{i=1,\ldots,n} \left\lfloor \frac{\mathtt{tok}(p_i)}{I_{p_i,t}} \right\rfloor \mid I_{p_i,t} \neq 0 \qquad (2)$$

Step 2: Propagation of the λ*–values* in Λ_i from its values in Λ_{i-1}. In this step both numbers and expressions are propagated.

Step 3: Resolution of the expressions in Λ_i. If a expression representing λ_j contains itself (λ_j), it is not considered, and is substituted by infinity (∞). After this simplification, `min` functions are applied, if possible.

Step 4: `if` $\exists \lambda_j \in \Lambda_i \mid \lambda_j \notin \mathbb{Z}^+$ `then go to` **Step 2** `else` **End**.

4.3 *Petri-net Structures Detection*

In this step, the PN structures are extracted in order to generate the constraints and reduce the search space. Each structure corresponds to a set of constraints which are automatically generated:

4.3.1. *Structure R1* R1 constraints (See Eq.3) are generated for every single transition in the PN. This is not a structure but a way of removing symmetries from the problem. Since there is a variable for each firing of a transition, by using R1, the values of these variables (and hence the firings) are given a unique order.

$$\begin{aligned} & t_{i_j} \geq t_{i_{j-1}} \\ & \forall j = 2, \ldots, l, \;\; \texttt{length}(T_i) = l \end{aligned} \tag{3}$$

4.3.2. *Structure R2* This structure includes two consecutive transitions (with a direct path between them). Each firing time of the exit transition is related with one of the source transition (See Eq.4).

$$\begin{aligned} & t_{i_j} \geq t_{k_j} + \texttt{time}_k \\ & \forall j = 1, \ldots, l, \;\; \texttt{length}(t_i) = \texttt{length}(t_k) = l \end{aligned} \tag{4}$$

where $\texttt{time}_k$ is the processing time associated to transition k.

These constraints are not generated if both transitions (forming R2) belong to a more complex structure (i.e. F, J or FJ).

4.3.3. *Structure IS* This represents an initial stock (i.e. an isolated place feeding two or more transitions). IS structure does not add temporal constraints but constraints related with the number of firings of the exit transitions (See Eq.5).

$$\begin{aligned} & \sum \lambda_i \leq tok_{\text{ini}}(p_{\text{in}}) \\ & \forall i \mid \texttt{feeds}(p_{\text{in}}, t_i) \end{aligned} \tag{5}$$

where p_{in} is the place representing the initial stock.

4.3.4. *Structure F* This structure is formed by a single transition, which feeds a set of transitions through a single place (See Fig.3). A list of *Boolean variables* are used to select the path followed by each token in the structure. These variables enable or disable constraints depending on their correspondence to the selected path (See Eq.6).

$$\begin{aligned} & t_{\text{out}_i} \geq (t_{\text{in}_j} + \texttt{time}_{\text{in}}) \cdot B_{\text{out}_j} - B_{\text{diff}} \cdot M \\ & \forall j = 1, \ldots, n, \;\; \texttt{length}(T_{\text{in}}) = n, \\ & \forall i = 1, \ldots, m, \;\; \texttt{length}(T_{\text{out}}) = m, \\ & B_{\text{diff}} = \texttt{diff}(i, \sum_{k=1}^{j} B_{\text{out}_k}) \end{aligned} \tag{6}$$

where function $\texttt{diff}$ returns zero (0) if the parameters are equal and one (1) if they are different, and M is an integer that dominates the expression (a ‘big-M’ term).

4.3.5. *Structure J* This is complementary to structure F. A set of transitions feed a single transition through a unique place (See Fig.4). *Boolean variables* are also necessary in this case (See Eq.7).

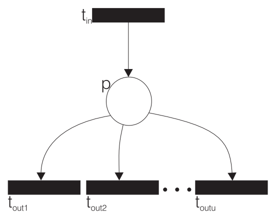


Fig. 3. Structure F

$$\begin{aligned} & t_{\text{out}_i} \geq (t_{\text{in}_j} + \texttt{time}_{\text{in}}) \cdot B_{\text{in}_i} - B_{\text{diff}} \cdot M \\ & \forall j = 1, \ldots, n, \;\; \texttt{length}(T_{\text{in}}) = n, \\ & \forall i = 1, \ldots, m, \;\; \texttt{length}(T_{\text{out}}) = m, \\ & B_{\text{diff}} = \texttt{diff}(j, \sum_{k=1}^{i} B_{\text{in}_k}) \end{aligned} \tag{7}$$

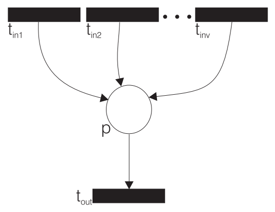


Fig. 4. Structure J

4.3.6. *Structure JF* This is the most complex structure, where a list of input transitions feeds an output list of transitions (See Fig.5). This structure also requires the use of *Boolean variables* (See Eq.8).

$$\begin{aligned} & T_{\text{out}_i} \geq (T_{\text{in}_j} + \texttt{time}_{\text{in}}) \cdot B_{\text{in,out}_j} - B_{\text{diff}} \cdot M \\ & \forall j = 1, \ldots, n, \;\; \texttt{length}(T_{\text{in}}) = n, \\ & \forall i = 1, \ldots, m, \;\; \texttt{length}(T_{\text{out}}) = m, \\ & B_{\text{diff}} = \texttt{diff}(i, \sum_{in} \sum_{k=1}^{j} B_{\text{in}_k}) \end{aligned} \tag{8}$$

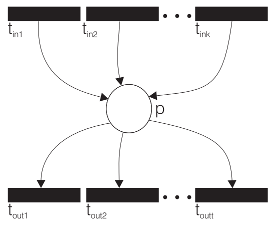


Fig. 5. Structure JF

4.4 *CSP Labelling Definition*

Since the aim of this methodology is to find the optimum, not only a solution has to be found but optimality must be proved. Although a possibility is to generate the complete solutions tree, this would mean a high computational cost.

On the other hand optimality can be proved by labelling the variable to optimise first, and selecting its values from better to worse solutions. This means that, in the moment a solution is found, all the possible better solutions have been already rejected. Optimality is proved and further search is not necessary.

Transition and *Boolean variables* are labelled only once for each instance of the variable to optimise (any of their feasible values gives an optimal solution). Paths are selected by labelling *Boolean variables* first. *Transition variables* are labelled later.

5. EXAMPLE

Example 1. Process and Assemble Factory

The system is composed by two machines which perform the next operations: M1 makes two different operations depending on the type of piece it is working with. This is a shared resource. Hence it is necessary to plan the order the pieces are put into it. On the other hand, M2 is used to join two processed pieces (A and B). The system can be seen in Fig. 6.

The aim is, given 10 pieces of type A and 5 pieces of type B, to perform 5 processed and assembled pieces.

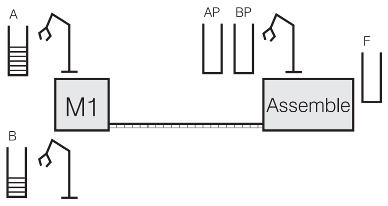


Fig. 6. The studied system

5.1 *Petri-net of the System*

The PN of the studied system (See Fig.7) has the next components:

Places:

P1: Stock of pieces A.
P2: Stock of pieces B.
P3: Piece A being processed in M1.
P4: Piece B being processed in M1.
P5: M1 free.
P6: Stock of pieces A processed.
P7: Stock of pieces B processed.
P8: M2 free.
P9: M2 assembling a final piece.
P10: Stock of assembled pieces.

Transitions:

T1: Move piece A to M1 (1 hour).
T2: Move piece B to M1 (1 hour).
T3: Retire piece A from M1 (2 hours).
T4: Retire piece B from M1 (4 hours).
T5: Move processed pieces A and B to M2 (1 hour).
T6: Retire final piece from M2 (3 hours).

Initial marking:

$$M_0 = [10, 5, 0, 0, 1, 0, 0, 1, 0, 0]$$

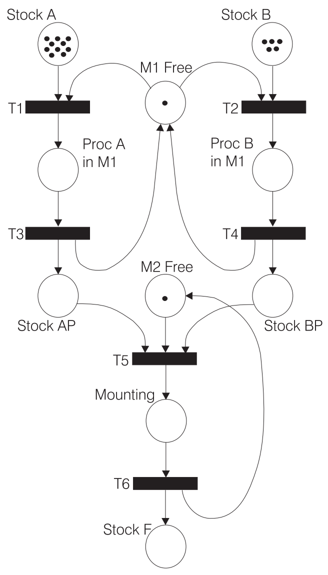


Fig. 7. The Petri-net of the system

5.2 *Transitions Firings Bounding*

In order to calculate the higher bounds of the transitions firings, Algorithm 1 is applied to the PN:

$$\begin{aligned}
\Lambda_0 &= [\mathtt{min}(10, 1+\lambda_3), \mathtt{min}(5, \lambda_4), \lambda_1, \lambda_2,\\
&\quad \mathtt{min}(\lambda_3, \lambda_4, 1+\lambda_6), \lambda_5]\\
\Lambda_1 &= [\mathtt{min}(10, 1+\lambda_1), \mathtt{min}(5, \lambda_2), \mathtt{min}(10, 1+\lambda_3),\\
&\quad \mathtt{min}(5, \lambda_4), \mathtt{min}(\lambda_1, \lambda_2, 1+\lambda_5),\\
&\quad \mathtt{min}(\lambda_3, \lambda_4, 1+\lambda_6)] =\\
&= [\mathtt{min}(10, \infty), \mathtt{min}(5, \infty), \mathtt{min}(10, \infty),\\
&\quad \mathtt{min}(5, \infty), \mathtt{min}(\lambda_1, \lambda_2, \infty),\\
&\quad \mathtt{min}(\lambda_3, \lambda_4, \infty)] =\\
&= [10, 5, 10, 5, \mathtt{min}(\lambda_1, \lambda_2), \mathtt{min}(\lambda_3, \lambda_4)]\\
\Lambda_2 &= [10, 5, 10, 5, \mathtt{min}(10, 5), \mathtt{min}(10, 5)] =\\
&= [10, 5, 10, 5, 5, 5] = \Lambda
\end{aligned}$$

5.3 *Structures Found*

The structures found and automatically translated into constraints are as follows:

R1: [T1], [T2], [T3], [T4], [T5] and [T6]
R2: [T1 $\rightarrow$ T3], [T2 $\rightarrow$ T4] and [T5 $\rightarrow$ T6]
IS: $\emptyset$
F: $\emptyset$
J: [T3, T4, T6 $\rightarrow$ T5]
JF: [T3, T4 $\rightarrow$ T1, T2]

5.4 *Results*

The firing times found after the optimisation phase are the next:

T1=[20, 23, 26, 29, 37]
T2=[0, 5, 10, 15, 32]
T3=[21, 24, 27, 30, 38]
T4=[1, 6, 11, 16, 33]
T5=[23, 27, 31, 35, 40]
T6=[24, 28, 32, 36, 41]

Then, the optimal time for the performance of 5 pieces (processed and assembled) is 44 hours ($T6_5 + \texttt{time}_6$).

6. BENEFITS OF THE PURPOSED METHODOLOGY

The proposed approach improves several aspects of actual scheduling tools, some of which are:

- The specification of the logistics of complex production systems using the PN formalism together with the presented analysis tool is a useful procedure to improve the overall system performance.
- The use of the CP technology to avoid local optimisation in front of global optimisation gives better solutions.
- The possible validation of the scheduling policy by means of a PN simulator. Note that the CP model does not consider the stochastic aspects of manufacturing systems.

7. CONCLUSIONS AND FUTURE WORK

The presented methodology has proved to work for academic examples. Although only structural constraints extracted from the PN have been presented, new constraints are being studied. These may raise from relations like symmetries, t-invariants, etc. and also constraints which can be added by experts on the studied system. The addition of these constraints makes the search faster. Likewise, more complex PN models are going to be considered (Wang and Wu, 1998).

Another work line is the hierarchial structuring of the search in order to perform the optimisation in different steps: splitting the system in more general structures first and, specializing later.

8. REFERENCES

Barták, R. (1999). Constraint programming: In pursuit of the holy grail. *Proceedings of WDS99 (invited talk)*.

Gallaire, H. (1985). Logic programming: Further developments. *IEEE Symposium on Logic Programming*.

Jaffar, J. and J.L. Lassez (1987). Constraint logic programming. *The ACM Symposium on Principles of Programming Languages*.

Jensen, K. (1997). *Coloured Petri Nets: Basics Concepts, Analysis Methods and Practical Use*. Vol. 1. Springer-Verlag. Berlin.

Silva, M. and R. Valette (1989). Petri nets and flexible manufacturing. *Lecture Notes in Computer Science, vol. 424, Advances in Petri Nets* pp. 374–417.

Sutherland, I. (1963). Sketchpad: a man-machine graphical communication system. *Proc. IFIP Spring Joint Computer Conference*.

Waltz, D.L. (1975). *Understanding line drawings of scenes with shadows, in: Psycology, of Computer Vision*. McGraw-Hill. New York.

Wang, L. and S. Wu (1998). Modeling with colored timed object-oriented petri nets for automated manufacturing systems. *Computers and Industrial Engineering* **34**(2), 463–480.

Zimmermann, A., K. Dalkowski and G. Hommel (1996). A case study in modeling and performance evaluation of manufacturing systems using colored petri nets. *Proceedings of the 8th European Simulation Symposium (ESS '96)*.