ON JUST IN TIME CONTROL OF FLEXIBLE MANUFACTURING SYSTEMS VIA DIOID ALGEBRA

Michel ALSABA, Jean-Louis BOIMOND and Sébastien LAHAYE *

* *LISA, 62 avenue Notre Dame du Lac - Angers, France*
** *Email: [alsaba, boimond, lahaye]@istia.univ-angers.fr*

Abstract: This paper deals with the just in time control of flexible manufacturing systems when reference input is *a priori* known. The control is based on basic results of residuation in dioid algebra and a local scheduling rule of *Early Due Date* (to solve possible conflicts).

Keywords: Flexible manufacturing system, timed Petri nets, timed event graphs, timed state graphs, heap of pieces, scheduling, dioid, residuation, just in time control.

1. INTRODUCTION

This work deals with the Just in Time (JIT) control of Flexible Manufacturing Systems (FMS), *i.e.*, how to delay system inputs at the latest such that system outputs occur before the desired reference input? The JIT control we propose requires an *a priori* known reference input. FMS are characterized by the production of several types of parts, which is specified by given operating sequences of tasks and some resources (like machines or robots) shared between several tasks.

The system we consider is such that each type of parts crosses successive stages arranged in a linear manner as a production line. A systematic construction of FMS using Petri nets is proposed in (Amar *et al.*, 1992). As an illustrative example let us introduce the FMS described by figure 1 in which three types of parts (A, B, C) are treated. A part A visits the following sequence of tasks: machine $M1$ (the presence of two tokens in place $P3$ means that the machine can process two parts simultaneously and independently), then machine $M2$ (also used to process parts B). A part B visits the following sequence of tasks: machine $M3$ (also used to process parts C), then machine

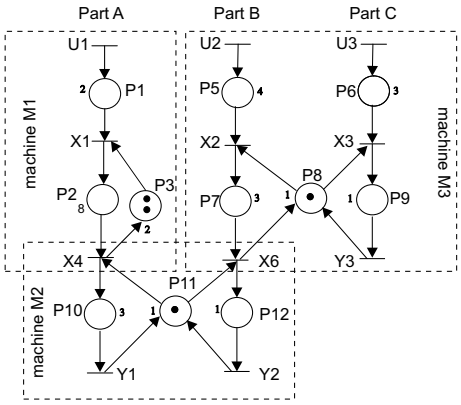

Fig. 1. Example of FMS.

$M2$. Finally a part C is simply processed by the machine $M3$.

It is known that there is no optimal solution, in general, for the JIT control problem of considered systems, due to the scheduling of the shared resources. Such a problem, analogous to a jobshop scheduling problem, is NP-hard when a ressource is shared by 3 jobs or more (Sotskov, 1991). The alternative solution to manage a shared ressource is based on a local scheduling rule. The proposed control method leads to subdivide Petri nets into different graphs.

We use Timed State Graphs (TSG) to represent choices phenomena, see the shared resources of machine $M2$, or $M3$, in figure 1. In this paper, we use a task scheduling rule called *Early Due Date*, or *Jackson* rule (Jackson, 1955). It is simple, efficient and gives in general good results when the criterion is based on lateness, which is the case in this paper (the lateness $L_i = C_i - d_i$ where C_i is the completion time of task i and d_i is the due date of task i). Applied on the basic TSG described by the figure 2, where are only represented two types of tasks for clarity reason, this scheduling rule is obtained simply by arranging tasks $I1O1$ and $I2O2$ by order of increasing desired achievement dates indicated by reference input (tasks $I1O1, I2O2$ correspond to the treatment of pieces $1, 2$ respectively). The scheduling being fixed, we propose a JIT optimal control method of safe TSG (*i.e.,* one token at most in each place), in the sense that the resource is attributed at the latest to tasks. The control is based on heap model of TSG.

We use Timed Event Graphs (TEG) to represent synchronization and delay phenomena. Well known results on residuation in $(\max, +)$ algebra allow the computation of a JIT optimal control of these graphs, see (Cohen *et al.*, 1989). Note that TEG are not necessarily safe, see machine $M1$ described in figure 1, contrary to TSG supposed safe.

This modeling approach appears to us better than a global transformation of a timed Petri net into a TEG, as proposed in (Trouillet and Benasser, 2002). In fact, the transformation of a timed Petri net into an equivalent TEG supposes an *a priori* conflits arbitration for each given reference input, which could be costly in calculation time due to possibly important dimension of resulting TEG. In contrast, the representation of a timed Petri net is independent of the considered scheduling, and in consequence, of the reference input behavior, which is particularly interesting in our case, in the sense that any reference input trajectory can be considered.

This paper is organized as follows. In the second section are recalled some basic facts about dioid, residuation theory, timed Petri nets and heap models. We model and control FMS in the third section, by subdividing Petri nets in TSG and TEG. As an illustration, control algorithm is applied to the system represented in figure 1.

2. PRELIMINARIES

2.1 Dioid

See (Cohen *et al.*, 1989), (Baccelli *et al.*, 1992, §4) for an exhaustive presentation of dioid theory.

A *dioid* $\mathcal{D}$ is a set endowed with two inner operations denoted by $\oplus$ (addition) and $\otimes$ (multiplication), both associative and both having a neutral element denoted by ε and e respectively, such that $\oplus$ is also commutative and idempotent (*i.e.*, $a \oplus a = a$). The $\otimes$ operation is distributive with respect to $\oplus$, and ε is absorbing (*i.e.*, $\varepsilon \otimes a = a \otimes \varepsilon = \varepsilon$). The symbol $\otimes$ is often omitted.

A dioid $\mathcal{D}$ is *complete* if it is closed for infinite sums and if the product distributes over infinite sums too, *i.e.*, if $\forall c \in \mathcal{D}$ and $\forall \mathcal{A} \subseteq \mathcal{D}$,
$c \otimes \left(\bigoplus_{x \in \mathcal{A}} x \right) = \bigoplus_{x \in \mathcal{A}} c \otimes x.$
The upper bound, noted $\top$, of a complete dioid is the sum of all its elements and it is absorbing for the addition (*i.e.*, $\forall a, \top \oplus a = \top$).

An *order relation*, noted $\succeq$, can be associated with a dioid $\mathcal{D}$ by the following equivalence: $\forall\, a, b \in \mathcal{D}, a \succeq b \Leftrightarrow a = a \oplus b$. This order confers upon a complete dioid a structure of complete lattice. So we can introduce an operator Inf, noted $\wedge$, verifying: $\forall\ a, b \in \mathcal{D},\ a \succeq b \Leftrightarrow b = a \wedge b$.

The set $\mathbb{R} \cup \{\pm\infty\}$, endowed with the max operator as sum and the classical sum as product, is a complete dioid, usually denoted by $\overline{\mathbb{R}}_{\max}$ with $\varepsilon = -\infty, e = 0$ and $\top = +\infty$.

If $\mathcal{D}$ is a dioid, the set $\mathcal{D}^{n \times n}$ of $n \times n$ matrices and entries in $\mathcal{D}$ is also a dioid where sum and product are defined by:
$$(A \oplus B)_{ij} = A_{ij} \oplus B_{ij}, (A \otimes B)_{ij} = \bigoplus_{k=1}^{n} A_{ik} \otimes B_{kj}.$$

2.2 Residuation theory

A presentation of this theory in the context of dioid is given in (Baccelli *et al.*, 1992, §4.4). The residuation theory provides, under some assumptions, the *greatest* solution to the inequality $f(x) \preceq b$, where f is an isotone mapping (*i.e.,* $a \preceq b \Rightarrow f(a) \preceq f(b)$) defined over ordered sets.

An isotone mapping $f : \mathcal{D} \to \mathcal{F}$, where $\mathcal{D}$ and $\mathcal{F}$ are ordered sets, is a *residuated mapping* if for all $b \in \mathcal{F}$ the upper bound of the subset $\{x \in \mathcal{D} \mid f(x) \preceq b\}$ exists and belongs to this subset.

Theorem 1. (Baccelli *et al.*, 1992, §4.4.2) Let $f : \mathcal{D} \to \mathcal{F}$ be an isotone mapping from the complete dioid $\mathcal{D}$ into the complete dioid $\mathcal{F}$. The following statements are equivalent:

(i) Mapping f is *residuated*.
(ii) There exists a unique isotone mapping $f^\sharp : \mathcal{F} \to \mathcal{D}$, called *residual*, such that $f \circ f^\sharp \preceq id_{\mathcal{F}}$ and $f^\sharp \circ f \succeq id_{\mathcal{D}}$ where $id_{\mathcal{F}}$ and $id_{\mathcal{D}}$ are identity mappings in $\mathcal{F}$ and $\mathcal{D}$ respectively.

Mappings $L_a : x \mapsto ax$ and $R_a : x \mapsto xa$ defined over a complete dioid $\mathcal{D}$ are both residuated. Their

residuals are usually denoted by $L_a^\sharp(x) = a\backslash x$ and $R_a^\sharp(x) = x/a$ respectively.

Let $A \in \mathcal{D}^{m\times n}, B \in \mathcal{D}^{m\times p}, C \in \mathcal{D}^{n\times p}$, then matrix residuation in function of scalar residuation is defined by:
$A\backslash B \in \mathcal{D}^{n\times p}, (A\backslash B)_{ij} = \bigwedge_{l=1}^{m} A_{li}\backslash B_{lj}$,
$B/C \in \mathcal{D}^{m\times n}, (B/C)_{ij} = \bigwedge_{k=1}^{p} B_{ik}/C_{jk}$.
Moreover, matrix residuation satisfies:
$B/(AC) = (B/C)/A \in \mathcal{D}^{m\times m}$,
$(AC)\backslash B = C\backslash(A\backslash B) \in \mathcal{D}^{p\times p}$.

2.3 Petri nets

See (Murata, 1989) for an exhaustive presentation of Petri nets. Considered timed Petri nets are nets with only holding times associated to places.

A *Timed Petri Net* (TPN) is a 5-tuple $\mathcal{G} = (\mathcal{T}, \mathcal{P}, \mathcal{F}, M, \tau)$, where $\mathcal{T}$ is the finite set of transitions, $\mathcal{P}$ is the finite set of places, $\mathcal{F} \subseteq (\mathcal{P}\times\mathcal{T}) \cup (\mathcal{T}\times\mathcal{P})$ is the set of arcs, $M : \mathcal{P} \to \mathbb{N}$ is the initial marking and $\tau : \mathcal{P} \to \mathbb{N}$ is the holding time.

We denote by $x^\bullet$ (resp., ${}^\bullet x$) the set of direct successors (resp., predecessors) of a node (place or transition) x. We call *language* of the Petri net $\mathcal{G}$ the set $L \subset \mathcal{T}^*$ of firing sequences starting from M and resulting in a reachable marking, where $\mathcal{T}^*$ are the set of words on the alphabet $\mathcal{T}$.
Transitions are fired according to the following rule. We assume that a transition T_i becomes enabled at instant t, then the firing of T_i occurs in two steps:
At instant t, one token is removed from each place $p \in {}^\bullet T_i$;
At instant t, one token is added in each place $p \in T_i^\bullet$ and can contribute to the enabling of the transitions in $p^\bullet$ after instant $t + \tau_p$.

2.4 Heap models and heap automata

See (Gaubert and Mairesse, 1999) for an exhaustive presentation of heap automata where it is shown that heap models are particularly interesting for the successive evaluation of a large number of schedules in safe TPN. Such heap automata can be seen as special $(\max, +)$ automata, which compute, with a computational complexity independent of the schedule, the height of heaps of pieces to have a makespan evaluation. Heap models are used here to model TSG in order to develop a control method which comes down to addressing a model inversion problem.
The heaps vertical axis indicates its height and its horizontal axis is for a finite number of slots. A *piece* is a solid (possibly not connected) "block" occupying some of the slots, with staircase-shaped upper and lower contours, see figure 3. With an ordered sequence of pieces, we associate a heap by pilling up the pieces, starting from an horizontal ground. A piece is only subject to vertical translations and occupies the lowest possible position, provided it is above ground and the pieces previously piled up.

A *heap model* is a 5-tuple $\mathcal{H} = (\mathcal{T}, \mathcal{R}, R, l, u)$, where:
$\mathcal{T}$ is a finite set whose elements are called pieces.
$\mathcal{R}$ is a finite set whose elements are called slots.
$R : \mathcal{T} \to \mathcal{P}(\mathcal{R})$ gives the subset of slots occupied by a piece.
$l : \mathcal{T}\times\mathcal{R} \to \mathbb{R}\cup\{-\infty\}$ gives the height of the lower contour of the piece at the different slots.
$u : \mathcal{T}\times\mathcal{R} \to \mathbb{R}\cup\{-\infty\}$ gives the height of the upper contour of the piece at the different slots (by construction, we have: $u \geq l$).
By convention, $l(a,r) = u(a,r) = -\infty$ if $r \notin R(a)$ and $min_{r\in R(a)} l(a,r) = 0$ (a piece can not be lower than the ground).

We will interpret a k length word $\omega = a_1...a_k \in \mathcal{T}^*$ as the heap obtained by pilling up the k pieces $a_1, ..., a_k$ (in this logical order). We define the *upper contour* of the heap ω as the $card(\mathcal{R})$-dimensional row vector $x_\mathcal{H}(\omega)$, where $x_\mathcal{H}(\omega)_r$ is the height of the heap on slot r. For example, we have relatively to the heap of pieces that corresponds to the word $I1O1$ represented in figure 3: $x_\mathcal{H}(I1O1)_1 = \tau_1, x_\mathcal{H}(I1O1)_2 = \tau + \tau_1, x_\mathcal{H}(I1O1)_3 = -\infty$. The horizontal ground assumption yields $x_\mathcal{H}(e) = (0, ..., 0)$ where e denotes the empty word. The *height* of the heap ω is $y_\mathcal{H}(\omega) = max_{r\in\mathcal{R}} x_\mathcal{H}(\omega)_r$.

A heap automaton is 4-tuple $\mathcal{A} = (Q, I, F, \mathcal{M})$, where:
Q is a finite set (of *states*);
$I \in \mathbb{R}_{max}^{1\times Q}$ and $F \in \mathbb{R}_{max}^{Q\times 1}$ are the *initial* and *final* vectors respectively;
$\mathcal{M}$ is a morphism ($\mathcal{M}(a\otimes b) = \mathcal{M}(a)\otimes\mathcal{M}(b)$) defined by: $\mathcal{M} : \mathcal{T}^* \to \overline{\mathbb{R}}_{\max}^{Q\times Q}, a \mapsto \mathcal{M}(a) = I \oplus [\tilde{l}(a,.)]^t u(a,.)$, where I is the identity matrix defined by $I_{ii} = e = 0, I_{ij} = \varepsilon = -\infty, i \neq j$, where $\tilde{l}(a,i) = -l(a,i)$ if $l(a,i) \neq \varepsilon$ and $\tilde{l}(a,i) = \varepsilon$ otherwise, and where $\tilde{l}(a,.), u(a,.)$ are viewed as row vectors.

We verify that:

$$\begin{cases} x_\mathcal{H}(e) &= I_\mathcal{R}^t, \\ x_\mathcal{H}(\omega a) &= x_\mathcal{H}(\omega)\mathcal{M}(a), \\ y_\mathcal{H}(\omega) &= x_\mathcal{H}(\omega) I_\mathcal{R}, \end{cases} \quad (1)$$

where $I_\mathcal{R}$ is the $card(\mathcal{R})$-dimensional column vector whose entries are equal to e.

Theorem 2. (Gaubert and Mairesse, 1999) Let $\mathcal{H} = (\mathcal{T}, \mathcal{R}, R, l, u)$ be a heap model. The heap automaton $(\mathcal{R}, I_\mathcal{R}^t, I_\mathcal{R}, \mathcal{M})$, associated with the heap model $\mathcal{H}$, recognizes the upper contour $x_\mathcal{H}$ and the height $y_\mathcal{H}$, which means that $\forall\omega \in \mathcal{T}^*$ we have:

$x_{\mathcal{H}}(\omega) = I_{\mathcal{R}}^t \mathcal{M}(\omega)$,
$y_{\mathcal{H}}(\omega) = I_{\mathcal{R}}^t \mathcal{M}(\omega) I_{\mathcal{R}}$.

A useful interpretation of a heap model consists in viewing pieces as tasks and slots as resources. Each task "a" requires a subset of the resources (given by $R(a)$) during a certain amount of time $(u(a,r) - l(a,r)$ for a resource $r \in R(a))$. In fact, in TPN context, the maps $y_{\mathcal{H}}$ and $x_{\mathcal{H}}$ correspond to *dater* functions in the sense that $x_{\mathcal{H}}(\omega)_r$ represents the achievement time of task r when the word ω is applied.

Theorem 3. (Gaubert and Mairesse, 1999) Let $\mathcal{G} = (\mathcal{T}, \mathcal{P}, \mathcal{F}, M, \tau)$ be a safe TPN with language L. Then the heap model $\mathcal{H} = (\mathcal{T}, \mathcal{P}, R, l, u)$, with:
$\forall a \in \mathcal{T}, R(a) = a^\bullet \cup {}^\bullet a$,
$\forall a \in \mathcal{T}, \forall p \in a^\bullet, u(a,p) = \tau_p$,
$\forall a \in \mathcal{T}, \forall p \in {}^\bullet a \backslash a^\bullet, u(a,p) = 0$, where "\" is the set substraction,
$\forall a \in \mathcal{T}, \forall p \in R(a), l(a,p) = 0$,
is such that:
$\forall \omega \in L, x_{\mathcal{G}}(\omega) = x_{\mathcal{H}}(\omega), y_{\mathcal{G}}(\omega) = y_{\mathcal{H}}(\omega)$,
which means that *dater* functions of vector $x_{\mathcal{G}}$ and of the scalar $y_{\mathcal{G}}$ of the net $\mathcal{G}$ coincide respectively with the upper contour $x_{\mathcal{H}}$ and the height $y_{\mathcal{H}}$ of the associated heap model $\mathcal{H}$.

Firing times of a safe TPN are recognized by a heap automaton as a result of the previous two theorems. For example, let us consider the basic TSG described by figure 2 which could correspond to the machine M2 or M3 of figure 1.

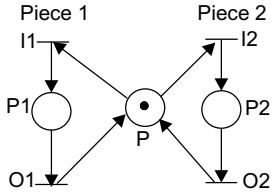

Fig. 2. A basic TSG.

The associated heap model is defined by:
$\mathcal{T} = \{I1, I2, O1, O2\}, \mathcal{R} = \mathcal{P} = \{P1, P, P2\}$;
$R(I1) = R(O1) = \{P1, P\}$,
$R(I2) = R(O2) = \{P, P2\}$;
$u(I1, .) = [\tau_1, 0, -\infty], l(I1, .) = [0, 0, -\infty]$,
$u(I2, .) = [-\infty, 0, \tau_2], l(I2, .) = [-\infty, 0, 0]$,
$u(O1, .) = [0, \tau, -\infty], l(O1, .) = [0, 0, -\infty]$,
$u(O2, .) = [-\infty, \tau, 0], l(O2, .) = [-\infty, 0, 0]$.

We have represented in figure 3 the heaps of pieces associated with words $I1, O1$ and $I1O1$. We directly read the values $x_{\mathcal{H}}(I1O1) = [\tau_1, \tau_1\tau, -\infty]$ and $y_{\mathcal{H}}(I1O1) = \tau_1\tau$ where $\tau_1\tau$ correspond to $\tau_1 + \tau$ in usual algebra.

The corresponding heap automaton is defined by the 4-tuple $(\mathcal{P}, (e\ e\ e), (e\ e\ e)^t, \mathcal{M})$ where $\mathcal{M}$ is defined as follows:

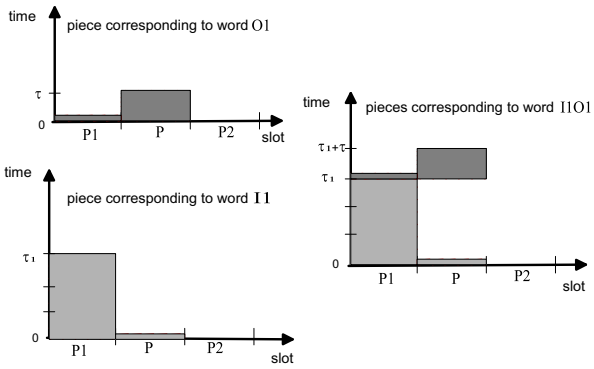

Fig. 3. Heaps of pieces associated with words $I1, O1$, and $I1O1$.

$$\mathcal{M}(I1) = I \oplus \begin{pmatrix} -e \\ -e \\ \varepsilon \end{pmatrix} \begin{pmatrix} \tau_1 & e & \varepsilon \end{pmatrix} = \begin{pmatrix} \tau_1 & e & \varepsilon \\ \tau_1 & e & \varepsilon \\ \varepsilon & \varepsilon & e \end{pmatrix},$$

$$\mathcal{M}(O1) = I \oplus \begin{pmatrix} -e \\ -e \\ \varepsilon \end{pmatrix} \begin{pmatrix} e & \tau & \varepsilon \end{pmatrix} = \begin{pmatrix} e & \tau & \varepsilon \\ e & \tau & \varepsilon \\ \varepsilon & \varepsilon & e \end{pmatrix},$$

in the same way,

$$\mathcal{M}(I2) = \begin{pmatrix} e & \varepsilon & \varepsilon \\ \varepsilon & e & \tau_2 \\ \varepsilon & e & \tau_2 \end{pmatrix}, \mathcal{M}(O2) = \begin{pmatrix} e & \varepsilon & \varepsilon \\ \varepsilon & \tau & e \\ \varepsilon & \tau & e \end{pmatrix}.$$

As a result we have, $\mathcal{M}(I1O1) = \mathcal{M}(I1)\,\mathcal{M}(O1) = \begin{pmatrix} \tau_1 & \tau_1\tau & \varepsilon \\ \tau_1 & \tau_1\tau & \varepsilon \\ \varepsilon & \varepsilon & e \end{pmatrix}$ and $\mathcal{M}(I2O2) = \begin{pmatrix} e & \varepsilon & \varepsilon \\ \varepsilon & \tau_2\tau & \tau_2 \\ \varepsilon & \tau_2\tau & \tau_2 \end{pmatrix}$.

3. MODELING AND CONTROL OF FMS

We deal with the JIT control of FMS modeled by TPNs: given desired output transitions firing dates defined by the *dater* function $Z = \{z(k)\}_{k=0,\ldots,k_f}$, find the *latest* input transitions firing dates $U = \{u(k)\}_{k=0,\ldots,k_f}$ such that output transitions firing dates $Y = \{y(k)\}_{k=0,\ldots,k_f}$ occur *before* the given ones. In a production context, it amounts to satisfying the customer demand while minimizing internal stocks. This output tracking problem is optimally solved *via* residuation theory for TEG (Cohen *et al.*, 1989).
In the method we propose, TPN are subdivided into TEG and TSG in order to separate the synchronization phenomena from the choice phenomena. Modeling and control of these graphs are described in the two following subsections. The JIT control applied to TSG is such that the scheduling of a shared resource is obtained by using the *Early Due Date*, or *Jackson* rule (Jackson, 1955), *i.e.,* by ordering tasks according to increased achievement desired dates indicated by the reference input.

3.1 Modeling and control of TEG

TEG are well adapted to model synchronization phenomena and correspond to linear dynamic systems in dioid algebra (see (Cohen *et al.*, 1989),

(Baccelli *et al.*, 1992)). Let us consider the dioid $\overline{\mathbb{R}}_{\max}$ to linearly represent the dynamic behavior of a TEG that corresponds to the following state representation:

$$\begin{cases} x(k) = Ax(k-1) \oplus Bu(k), \\ y(k) = Cx(k). \end{cases} \quad (2)$$

Given a reference input Z, the JIT control solution, noted u_{JIT}, for the model described by the equations (2) is defined by the backtracking costate equations:

$$\begin{cases} \xi(k) = A \backslash \xi(k+1) \wedge C \backslash Z(k), \\ u_{JIT}(k) = B \backslash \xi(k), \end{cases} \quad (3)$$

where ξ represents the costate vector, see (Baccelli *et al.*, 1992, §5.6).

3.2 Modeling and control of TSG

A heap automaton can be seen as a $(\max, +)$ linear system whose dynamics is driven by letters. For a given sequence of letters $a_1...a_k$, we set $x(k) = x(a_1...a_k)$, then equations (1) could correspond to the following recurrence relation at step k:

$$\begin{cases} x(k) = x(k-1)\mathcal{M}(a_k), \\ y(k) = x(k)F, \end{cases} \quad (4)$$

for $k \geq 1$ and $x(0) = (0, ..., 0)$.
Considering the basic TSG of figure 2 where $I1, I2$ are input transitions and $O1, O2$ are internal (and output) transitions of the graph. The components of the row state vector $x(k)$ are dates of availability of token numbered k in places $P1, P, P2$ respectively.
Such state equations represent a non stationary system, let $A(k) = \mathcal{M}(a_k)$ be the associated matrix at step k, we have $a_k = I1O1$, or $I2O2$ which depends of the sequence choice at step k. Let $F_1 = (e, \varepsilon, \varepsilon), F_2 = (\varepsilon, \varepsilon, e), F_3 = (e, e, \varepsilon), F_4 = (\varepsilon, e, e)$ be the *interface vectors* joining *daters of transitions* to *daters of places* as described above. Finally, adding inputs to the model, the non autonomous state equations become:

$$\begin{cases} x(k) = x(k-1) \otimes A(k) \oplus u(k) \otimes B(k), \\ y(k) = x(k) \otimes C(k), \end{cases} \quad (5)$$

where $u = (I1, I2), x = (x_{P1}, x_P, x_{P2}), y = (O1, O2)$;
$B(k) = F_1\mathcal{M}(I1O1)$, resp. $F_2\mathcal{M}(I2O2), C(k) = F_1^t$, resp. F_2^t, depending on the sequence $I1O1$, resp. $I2O2$, applied at step k.

Similarly to (Lahaye *et al.*, 1999) we compute:
$x(k) = \bigoplus_{j \leq k} u(j)B(j)\Phi(j,k)$ and
$y(k) = \bigoplus_{j \leq k} u(j)B(j)\Phi(j,k)C(k)$,
where $\Phi(j,k)$ is the transition matrix given by:

$$\Phi(j,k) = \begin{cases} \text{not defined} & \text{if } j > k, \\ I & \text{if } j = k, \\ A(j+1)A(j+2)...A(k) & \text{otherwise.} \end{cases}$$

Then, we deduce that the JIT control, noted $u_{JIT_{Ii}}$, is given by:
$u_{JIT_{Ii}}(k) = \bigwedge_{k \leq j} Z_{Oi}(j) \not\backslash (B(k)\Phi(k,j)C(j))$,
$= \bigwedge_{k \leq j} Z_{Oi}(j) \not\backslash (F_i(k)A(k)A(k+1)..A(j)F_i^t(j))$,
where the reference input $Z_{Oi}(j) = Z_{O1}(j)$, resp. $Z_{O2}(j)$ (*i.e.*, the desired output of Y_1, resp. Y_2) according to sequence $I1O1$, resp. $I2O2$, occurring at step j.
Let $\Delta(k) = \bigwedge_{k \leq j} Z_{Oi}(j) \not\backslash (A(k)A(k+1)..A(j)F_i^t(j))$ the costate vector, we obtain (using section 2.2) the following costate equations:

$$\begin{cases} \Delta(k) = (\Delta(k+1) \wedge Z_{Oi}(k) \not\backslash F_i^t(k)) \not\backslash A(k), \\ u_{JAT_{Ii}}(k) = \Delta(k) \not\backslash F_i(k), \end{cases}$$

The use of Jackson scheduling rule, which schedules a resource while respecting the order of increased desired achievement dates indicated by reference input, corresponds to apply the following control algorithm at step k (with $k = l + h$):
If $Z_1(l) \preceq Z_2(h)$ then

$$\begin{cases} \Delta(k) = (\Delta(k+1) \wedge Z_{O2}(h) \not\backslash F_2^t) \not\backslash \mathcal{M}(I2O2), \\ u_{JAT_{I2}}(h) = \Delta(k) \not\backslash F_2, \\ h = h - 1, \end{cases} \quad (6)$$

else

$$\begin{cases} \Delta(k) = (\Delta(k+1) \wedge Z_{O1}(l) \not\backslash F_1^t) \not\backslash \mathcal{M}(I1O1), \\ u_{JAT_{I1}}(l) = \Delta(k) \not\backslash F_1, \\ l = l - 1, \end{cases} \quad (7)$$

where $u_{JAT_{I1}}(l)$(resp. $u_{JAT_{I2}}(h)$) corresponds to input transition firing date $I1(l)$(resp. $I2(h)$). It clearly appears that scheduling rule of the shared resource carries out a selection among the sequences $I1O1$ and $I2O2$ at each step k of the algorithm. Let us note that the test $Z_{O1}(l) \preceq Z_{O2}(h)$ gives a priority to the first production line $(I1, O1)$ when the reference input date is the same for the two lines. The obtained control is equivalent to the optimal JIT control of TEG, corresponding to TPN obtained by considering the used scheduling.

3.3 Modeling and control of FMS

Production lines of FMS is an assembly of TSG and TEG joined in series which leads to a simple construction of FMS models. Let us consider by example the FMS of figure 1 which corresponds to the combination of subgraphs described in figure 4. Note that the subgraphs TEG, TSG1 and TSG2 are coupled, in the sense of merging some transitions (see transition X_4, X_6 in figure 1 or figure 4 which both belong to two subgraphs).

Given Z a reference input, we start the residuation from the end, *i.e.* at step k_f, as usual in backward algorithm. A special care has to be done to interfaces between different subgraphs

because some transitions, called interface transitions, are common to these subgraphs. Due to the coupling described above the costate vectors are mutually effected. Let Δ_1, Δ_2 be the costate vectors associated to the subgraphs TSG1, TSG2 respectively. If the single resource of machine 2 is affected to part A then the first equation in (7) becomes $\Delta_2(k_2) = \Delta_2(k_2+1) \not| \mathcal{M}(I1O1) \wedge Z_1(l) \not| (\mathcal{M}(I1O1){F_1}^t) \wedge (\xi_1(l+2) - 2) \not| {F_3}^t$ (the last term reflects the influence of subgraph TEG) and the second equation in (7) becomes $\xi_4(l) = \Delta_2(k_2) \not| F_1$ where $k_2 = l + h$. If the single resource of machine 2 is affected to part B then the first equation in (6) becomes $\Delta_2(k_2) = \Delta_2(k_2 + 1) \not| \mathcal{M}(I2O2) \wedge Z_2(h) \not| (\mathcal{M}(I2O2){F_2}^t) \wedge \Delta_1(k_1 + 1) \not| (1{F_4}^t F_3)$, the last term reflects the influence of the last sample calculated of the resource assignment time in subgraph TSG1 projected into TSG2. Dually in the last case, subgraph TSG2 will affect the costate variable of subgraph TSG1 according to the equation: $\Delta_1(k_1) = \Delta_1(k_1 + 1) \not| \mathcal{M}(I1O1) \wedge \Delta_2(k_2) \not| (\mathcal{M}(I1O1){F_1}^t F_2)$. Finally if the last resource is affected to part C then the costate vector is calculated using: $\Delta_1(k_1) = \Delta_1(k_1+1) \not| \mathcal{M}(I2O2) \wedge Z_3(g) \not| (\mathcal{M}(I2O2){F_2}^t)$ and $\xi_3(g) = \Delta_1(k_1) \not| F_2$ where $k_1 = h + g$. The other state vectors are calculated using equation (3), for example $\xi_1(l) = \xi_4(l) - 8$. In this paragraph, the control method is described on a particular example for a better understanding. A generalization of this approach is currently being formalized.

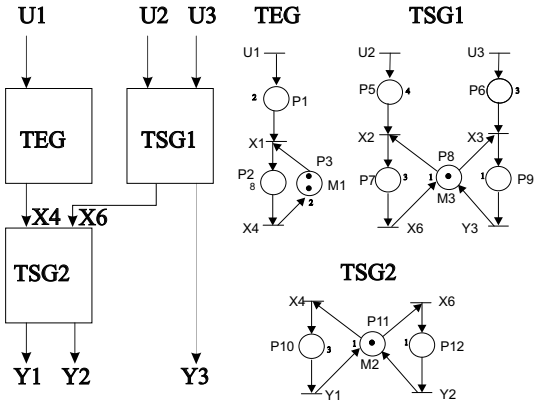

Fig. 4. Block diagram corresponding to FMS of figure 1.

3.4 Numerical application

Let us apply the previous control algorithm on example of figure 1. The reference input we consider is:

$$Z = \begin{pmatrix} 18 & 20 & 42 & 43 & 45 \\ \varepsilon & 19 & 21 & 31 & 41 \\ \varepsilon & 19 & 24 & 37 & 40 \end{pmatrix},$$

We apply the previous algorithm, step by step, beginning from the end $k_f = 5$, we obtain from the first interface:
$\xi_4 = \begin{pmatrix} 10 & 16 & 32 & 38 & 42 \end{pmatrix}$ and $\xi_1 = \begin{pmatrix} 2 & 8 & 24 & 30 & 34 \end{pmatrix}$, we obtain from the second interface:
$\xi_6 = \begin{pmatrix} \varepsilon & 14 & 20 & 26 & 30 \end{pmatrix}, \xi_2 = \begin{pmatrix} \varepsilon & 11 & 17 & 23 & 27 \end{pmatrix}$ and $\xi_3 = \begin{pmatrix} \varepsilon & 15 & 21 & 36 & 39 \end{pmatrix}$. Finally we find the input
$U_{JIT_1} = \begin{pmatrix} 0 & 6 & 22 & 28 & 32 \end{pmatrix}, U_{JIT_2} = \begin{pmatrix} \varepsilon & 7 & 13 & 19 & 23 \end{pmatrix}$, $U_{JIT_3} = \begin{pmatrix} \varepsilon & 12 & 18 & 33 & 36 \end{pmatrix}$.
The system output corresponding to U_{JIT} is given by:

$$Y = \begin{pmatrix} 13 & 19 & 35 & 41 & 45 \\ \varepsilon & 15 & 21 & 27 & 31 \\ \varepsilon & 16 & 22 & 37 & 40 \end{pmatrix}.$$ We can verify that $Y \preceq Z$.

4. CONCLUSION AND PERSPECTIVES

In this paper we have considered the just in time control problem of flexible manufacturing system. It is presented for a particular example. At present, we are working on a formalization which allows the generalization of systems classes, we can consider, for this control method.

REFERENCES

Amar, S., E. Craye and J.C Gentina (1992). A Method Of Hierarchical Specification And Prototyping Of FMS. In: *Proceedings of the IEEE-ETFA*. Vol. 1. pp. 44–49.

Baccelli, F., G. Cohen, G.J. Olsder and J.-P. Quadrat (1992). *Synchronization and Linearity: An Algebra for Discrete Event Systems.* Wiley and Sons.

Cohen, G., P. Moller, J-P. Quadrat and M. Viot (1989). Algebraic Tools for the Performance Evaluation of Discrete Event Systems. *IEEE Proceedings: Special issue on Discrete Event Systems* **77**(1), 39–58.

Gaubert, S. and J. Mairesse (1999). Modeling and Analysis of Timed Petri Nets Using Heaps of Pieces. *IEEE Trans. on Automatic Control* **44**(4), 683–697.

Jackson, J.R. (1955). Scheduling a Production Line to Minimize Maximum Tardiness. Research report 43. University of California. Los Angeles. Management Science Research Project.

Lahaye, S., J.-L. Boimond and L. Hardouin (1999). Optimal Control of (Min,+) Linear Time-Varying Systems, Petri Nets and Performance Models. Proceedings of PNPM'99. Zaragoza Spain. pp. 170–178.

Murata, T. (1989). Petri Nets: Properties, Analysis and Applications. In: *Proceedings of the IEEE.*. Vol. 77(4). pp. 541–580.

Sotskov, Y.N. (1991). The Complexity of Scheduling Problems with 2 and 3 Jobs. In: *Eur. J. Oper. Res.* Vol. 53. pp. 326–336.

Trouillet, B. and A. Benasser (2002). Cyclic Scheduling Problems with Assemblies: An Approach Based to the Search of Initial Marking in a Marked Graph. *IEEE Trans. on Systems, Man and Cybernetics.*