A UNIFIED FRAMEWORK FOR DESCRIBING THE DYNAMICS OF PULL CONTROL POLICIES WITH BATCH PRODUCTION

M. DI MASCOLO J-M. BOLLON

Laboratoire d'Automatique de Grenoble
BP46 - 38402 Saint Martin d'Hères Cedex FRANCE
Email: Maria.Di-Mascolo@inpg.fr

Abstract: This paper presents a unified framework that enables to express the dynamics of pull control policies with the same set of canonical functions. This canonical formulation allows to identify under what parameter values two different policies have the same dynamics behavior. It applies to many pull control policies, and also to manufacturing systems producing batches. A computational algorithm that enables to calculate efficiently the parameters of the canonical formulation for each policy is derived. This algorithm relies on the use of $(\min, +)$ algebra tools. *Copyright*$^{©}$ *2005 IFAC.*

Keywords: Production/inventory systems, pull control policies, (min,+) algebra, queueing systems, batch production

1. INTRODUCTION

This paper deals with make-to-stock pull control policies for multi-stage production/inventory systems. There are different ways of expressing pull control policies and it is often very difficult to see if the behaviors of two policies are identical. In a recent paper (Bollon *et al.*, 2004), we proposed a unified framework to describe the dynamics of some pull policies and we applied it to basestock, kanban, generalized kanban, and extended kanban. We also showed that this formulation allows to derive some properties for each policy and also to find all identical dynamics between two systems, with a systematic approach. This was illustrated by the comparison between the extended kanban and the generalized kanban mechanisms.

In this paper our first aim is to show that the same canonical formulation applies to a large class of other pull control policies, and also to systems producing batches. Secondly we develop an algorithm for an efficient computation of the formulation's parameters. This algorithm relies on the use of a shortest path search.

The class of pull policies we consider here can be modelled with a queueing network, divided into two parts, one representing the flow of material going downstream through the stocks and manufacturing processes, and the other representing the flow of information going upstream to control the circulation of materials. The pull control policies we consider here differ according to the way the information is transmitted upstream. For example, the kanban (Monden, 1983), uses a local transfer of information, whereas the basestock (Lee and Zipkin, 1992) uses a global flow of information. There are also some hybrid policies that combine these traditional policies, and thus local and global information flows. We can quote, among others, the generalized kanban (Buzacott, 1989) and the extended kanban (Dallery and Liberopoulos, 2000) that are both combinations of kanban and basestock policies. In (Liberopoulos and Dallery, 2000), many other hybrid policies are presented. The class of pull policies we consider here include all the policies described above and their extensions to systems producing batches, such as those presented in (Liberopoulos and Dallery, 2003). Many other policies can be included too, as can be seen in section 4.

This paper is organized as follows: in section 2, we present the modelling assumptions that are used for the systems studied in this paper. In section 3, we briefly present our formulation and give some elements that are necessary to the derivation of the new algorithms presented in the paper. In section 4, we show how this canonical formulation can be extended to more general pull control policies with batch production. We derive an efficient computational algorithm that enables to find the parameters of the canonical representation.

2. MODELLING ASSUMPTIONS

The production/inventory systems that we consider here are divided into N stages. Each stage i consists of an output buffer, denoted by P_i, which contains the finished parts of the stage, and a manufacturing process, denoted by F_i, which is used to supply the output buffer of the stage. We assume that raw parts are always available upstream the system. Demands which cannot be immediately satisfied are backlogged in a queue denoted by D_N.

We consider the case of stages in series, and we assume that there is no blocking at the entry of each manufacturing system and at the exit. We also assume that we have a mono-product system and that processes can produce batches of size Q_i at stage i. The size of batches for demands is Q_{N+1}. We assume that Q_i is a multiple of Q_{i+1}, and $Q_{N+1} = 1$, as usually assumed when dealing with batches (see (Axsäter and Rosling, 1993) and (Liberopoulos and Dallery, 2003) for example). This choice prevents the system from having useless remaining parts in a stage, and it also enables to simplify the calculations.

In each manufacturing process, we use a push control mechanism, but the different stages are coordinated using a pull mechanism. With a pull control policy the production is pulled downstream by the demand. Whenever a demand is received, it is transmitted directly or indirectly to upstream stages in order to maintain a certain level in stock. The way this transmission of information is done depends on the control policy that is used.

When a part leaves a manufacturing process, a stock, or when a demand arrives, an information can be sent upstream the system to allow one or

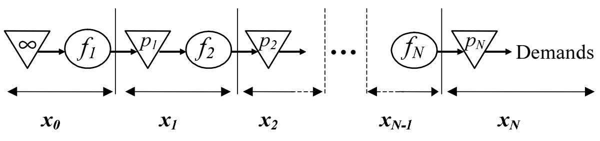

Fig. 1. State variable for an N-stage line

several parts to be processed. For example in a kanban policy when a part leaves a stock, a kanban label is sent to the upstream manufacturing process. This kanban allows a new part to enter the process. For a basestock policy, each demand is sent to all the manufacturing process inputs to allow the process of a new part.

The processing times or the times between two demand arrivals can be either deterministic or stochastic with general distribution. We only assume that the control mechanism is instantaneous.

3. PRINCIPLES OF OUR FORMULATION OF PULL CONTROL POLICIES

3.1 Elements and notations for our unified formulation of policies

We use a capital letter with an upright font to denote objects like manufacturing processes F_i or stores P_i. The scalar value for the number of parts present in an object is denoted by a small letter written with an italic font. For example, the number of parts in F_i, P_i or D_N is respectively f_i, p_i and d_N. If we need the value of these quantities at a given time t, we write $f_i(t)$, $p_i(t)$, etc.

We prove in (Bollon, 2001), that the state of the system can be expressed with a state vector $\boldsymbol{X}$, whose components are denoted by x_i, for $i = 1, \cdots, N$, and are defined as follows:

$$x_N = p_N - d_N \quad \text{and} \quad x_i = p_i + f_{i+1}. \qquad (1)$$

These components are illustrated in figure 1. A component x_i is equal to the sum of parts present between two consecutive manufacturing outputs. When the quantity x_N is positive, it represents the number of available finished goods and when it is negative, it represents the number of unsatisfied demands. The vector $\boldsymbol{X}$ gives the size of the different inventory positions which are always positive except for the last one, which represents finished goods minus demands. Each occurrence of an event (an outgoing part from a manufacturing process or a demand arrival) changes the vector $\boldsymbol{X}$. Conversely, if a vector change is known, then the event that has occurred is known too. In other words, all the events are detected if the vector $\boldsymbol{X}$ is known at every moment.

Let us now consider the values f_i. When they can be expressed as a function of $\boldsymbol{X}$ they are denoted by $f_i(\boldsymbol{X})$ [1]. The components of the N-dimensional function $\boldsymbol{F}(\boldsymbol{X})$ are defined by $f_i(\boldsymbol{X})$ for i varying from 1 to N. The knowledge of this vector $\boldsymbol{F}(\boldsymbol{X})$ enables to find the dynamics of the system: If $\boldsymbol{X}$ is known at every moment, events are known and the state of the system given by the values of p_i, f_i and d_N is known with equation (1). Then, to describe the policy, we only have to search for $\boldsymbol{F}(\boldsymbol{X})$, which is what we are going to do in the following section.

3.2 Principle of calculation of the $f_i(\boldsymbol{X})$ functions

Basic notions of $(\min,+)$ *algebra.*

The $(\min,+)$ algebra (Baccelli *et al.*, 1992) is often used to model the behavior of timed event graphs. These graphs are special cases of timed Petri nets. They are composed of places and transitions. The places must have no more than one input and one output transition. The transitions are timed with a deterministic delay.

Some variables called counters are used to describe the system. They are associated with transitions and count the number of tokens that have gone through a transition by a certain time. At the beginning, all the counters have the value zero. With counters and $(\min,+)$ algebra, inequalities can be written to describe the dynamics of the system.

The control mechanisms in a make-to-stock pull control policy can be expressed by an event graph. In this case $(\min,+)$ algebra will be useful to describe the policies. Note that an equivalent representation of an event graph can be obtained using a queueing network. It is this queueing network representation that we are using in the remainder of the paper.

The set $\mathbb{R}\cup\{+\infty\}$ associated with the min operator and the usual addition has a dioid algebraic structure called $\mathbb{R}_{\min}$. Then, the operators $\oplus$ and $\otimes$ will stand respectively for the min and the usual addition. The neutral element for $\oplus$ is infinity and its notation is ε. For $\otimes$ the neutral element is zero and will be denoted by e. The use of (min,+) algebra and its notations enables us to obtain concise expressions for the formulations we are looking for. It enables us also to facilitate the calculation as explained in the following.

Link between $(\min,+)$ *algebra and the graphs*

For the special case of a timed event graph where all delays on transitions are zero, if transitions are fired at the earliest, then it is possible to express every counter function $y_i(t)$ with input counters $z_i(t)$ and the initial number of tokens of each place. In other words, if we denote by $\boldsymbol{Z(t)}\in\mathbb{R}^m_{\min}$ the vector having for components the counters $z_i(t)$ of the m input transitions Z_i and by $\boldsymbol{Y(t)}\in\mathbb{R}^n_{\min}$ the vector having for components the counters of the n others transitions Y_i, then $\boldsymbol{Y(t)}$ is the solution of the following system of inequalities ($\leqslant$ is the natural order):

$$\boldsymbol{Y(t)} \leqslant \boldsymbol{Y(t)} \oplus B\,\boldsymbol{Z(t)}, \qquad (2)$$

where A and B are matrices which belong respectively to $\mathbb{R}^{n\times n}_{\min}$ and $\mathbb{R}^{n\times m}_{\min}$. The maximal solution of system (2) can be given in an explicit way, using the "Kleene star" operator A^*:

$$\boldsymbol{Y(t)} = A^*B\,\boldsymbol{Z(t)} \quad \text{with} \quad A^* = \bigoplus_{k\geqslant 0} A^k, \quad (3)$$

where A^0 is the identity matrix in $\mathbb{R}_{\min}$, where each component is equal to $e=0$ on the diagonal and $\varepsilon=+\infty$ for the other elements.

This result can be interpreted as a path search in a graph where vertices correspond to transitions and arcs correspond to places. Near a place there are two transitions, one before and one after, the vertices related to them are respectively the beginning and the ending of the arc associated with this place. The initial number of tokens present in the place will be the weight of the arc. The star operator, named "Kleene star", gives the value of $y_i(t)$; this value can also be similarly given by the minimum value for $j=1,\cdots,m$ of a sum adding $z_j(t)$ and the shortest directed path starting from Z_j and ending at Y_i.

This link between $(\min,+)$ algebra and the graphs through the Kleene star, enables to derive efficient algorithms to compute the canonical formulations, as we are going to see in the next sections.

Structure of pull control systems and required counters

A production line using a pull control policy can be divided into two parts (see figure 2):

- The first part is composed of stocks P_i and manufacturing processes F_i where the production flow takes place. Each stock is connected to a synchronization station on which the second part of the system acts.
- The second part is the mechanism of the pull policy. It controls the production requirement flow of information moving upstream the line.

In the production part of the system we use a counter at the end of each manufacturing process F_i (represented by an oval in figure 2) and each buffer P_i (represented by a queue, linked to a synchronization station in figure 2). The first counter, denoted by u_i, detects the outgoing of products

[1] Note that f_i can be a function of the time (then we have a scalar value inside the brackets) or a function of the state (then we have a vector inside the brackets).

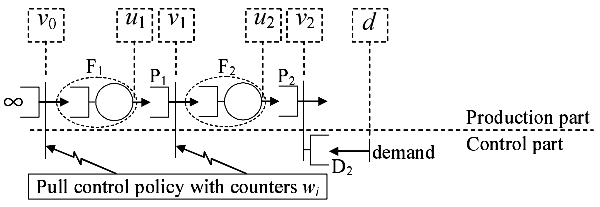

Fig. 2. Main counters of a two-stage production line

from the manufacturing process and the other, denoted by v_i, counts the release of available parts from stocks. If a synchronization station coincides with a counter u_i or v_i, then we denote it by U_i or V_i respectively. In the control part of the system, an input counter d is used to detect the arrival of demands. The pull control policy may require additional synchronization stations W_i, which we associate with counters denoted by w_i.

For an N-stage line, an arc connects the arrival of a demand, represented by a transition called D, to a queue D_N at the synchronization station V_N. Depending on the policy, some other arcs move from a transition D, V_i, U_i or W_i to an upstream transition V_j or W_j.

To calculate the $\boldsymbol{F}(\boldsymbol{X})$ function, we first use the previously defined counters to obtain the function of the time $\boldsymbol{F}(t)$ whose components are, for $i = 1, \cdots, N$ (with usual notations):

$$f_i(t) = f_i(0) + v_{i-1}(t) - u_i(t) \ . \tag{4}$$

Moreover counters $u_i(t)$ and $d(t)$ can be linked to variables x_i through equations (5), for $i = 1, \cdots, N$ (with usual notations):

$$\begin{aligned} x_i(t) - x_i(0) &= u_i(t) - u_{i+1}(t) \text{ and} \\ x_N(t) - x_N(0) &= u_N(t) - d(t) \, . \end{aligned} \tag{5}$$

Considering that the control part of the system is an event graph and that the $u_i(t)$ and $d(t)$ are inputs of this event graph, then the $v_{i-1}(t)$ can be expressed in terms of $u_i(t)$ and $d(t)$ using a $(\min, +)$ algebra, which can be expressed as functions of x_i using (5). They can then be removed and replaced by an expression using $\boldsymbol{X}(t)$ in equation (4). If all the counters are removed and replaced by the $\boldsymbol{X}(t)$ components, we can obtain the function $\boldsymbol{F}(\boldsymbol{X})$ from $\boldsymbol{F}(t)$ by substituting $\boldsymbol{X}$ for $\boldsymbol{X}(t)$.

3.3 A canonical formulation

In (Bollon *et al.*, 2004), we calculated the functions $\boldsymbol{F}(\boldsymbol{X})$ for four pull control policies: base-stock, kanban, extended kanban and generalized kanban.

For these policies, the function $f_i(\boldsymbol{X})$ always has the same structure, that we call the canonical formulation. This formulation is given in equation (6), where the fractions stand for the usual minus:

$$\begin{aligned} f_1(\boldsymbol{X}) &= \bigoplus_{j=1}^{N+1} \left(C_{(1,j)} \Big/ \bigotimes_{k=1}^{j-1} x_k \right) \quad \text{and} \\ f_i(\boldsymbol{X}) &= \bigoplus_{j=i}^{N+1} \left(C_{(i,j)} \Big/ \bigotimes_{k=i}^{j-1} x_k \right) \oplus x_{i-1} \\ \text{for } i &= 2, \cdots, N. \end{aligned} \tag{6}$$

The parameters $C_{(i,j)}$ of this formulation define the policy and are calculated from the control parameters of each policy, namely, S_k, the maximum level of finished products of each stage k, and K_k, the number of kanbans in each stage k.

In (Bollon and Di Mascolo, 2004), we prove that this formulation enables to describe a more general class of pull policies than those studied in (Bollon *et al.*, 2004). An idea of the proof is the following: the values for $f_i(t)$ are given by equation (4). The difficulty relies in the way to express counters v_i as functions of counters u_i. The control mechanisms of the make-to-stock pull control policy can be modelled by an event graph with no delay on transitions. Then a system of inequalities as in (2) can be set, with system input transitions being U_i and D. Other transitions are V_i and $\mathrm{W}_{(i,j)}$. The solution of this system is obtained thanks to a shortest path search in a graph where vertices correspond to synchronization stations. This solution enables to express counters $v_i(t)$ and $w_{(i,j)}(t)$ as functions of counters $u_k(t)$ and $d(t)$.

4. CANONICAL FORMULATION FOR MORE GENERAL PULL CONTROL POLICIES WITH BATCH PRODUCTION

4.1 Existence of a canonical formulation for a class of pull control policies producing batches

In this part we are going to show that it is possible to describe a class of pull policies producing batches with a formulation similar to the one given by equations (6), using a new notation[2]. This class is defined in proposition 1 and is illustrated in figure 3. In this figure, a generic stage i is represented in grey. Most of the queues and transitions are not necessarily present. Entities present in the upper part of the figure are required. Those present in the lower part of the figure can appear one or several times, or can be dismissed. So manufacturing processes F_i, stocks P_i, queue D_N, synchronization stations V_{i-1}, U_i and D, arcs going downstream and the arc going

[2] we denote by $\lfloor \alpha \rfloor$ the highest integer smaller than or equal to α and by $\lfloor \alpha \rfloor_\beta = \beta \left\lfloor \frac{\alpha}{\beta} \right\rfloor$ the highest multiple of β smaller or equal to α

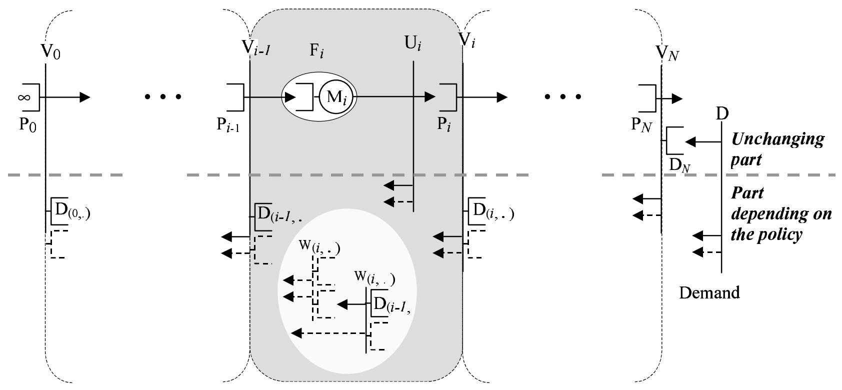

Fig. 3. Description of a class of policies having a canonical formulation

from D to D$_N$ are all required. The arcs that are going upstream can reach any of the upstream queues that are oriented in the same direction. The synchronization station U$_i$ which follows F$_i$ enables to locate the counter u_i and it often contains only one arc and has no utility for the policy itself .

Note that this class of policies with batches, described in proposition 1 below, includes the extensions of basestock, kanban, generalized kanban, extended kanban and many other hybrid policies to systems producing batches, like, for example, the installation and the echelon stock (Q,r)-policies (Axsäter and Rosling, 1993) and the hybrid policies described in (Liberopoulos and Dallery, 2003).

Proposition 1

The policy of a queueing system producing batches can be defined by the functions

$$f_1(\boldsymbol{X}) = \left\lfloor \bigoplus_{j=0}^{N} \left(C_{(1,j)} \Big/ \bigotimes_{k=1}^{j} \lfloor x_k \rfloor_{Q_{k+1}} \right) \right\rfloor_{Q_1} \quad \text{and}$$

$$f_i(\boldsymbol{X}) = \left\lfloor \bigoplus_{j=i-1}^{N} \left(C_{(i,j)} \Big/ \bigotimes_{k=i}^{j} \lfloor x_k \rfloor_{Q_{k+1}} \right) \oplus \lfloor x_{i-1} \rfloor_{Q_i} \right\rfloor_{Q_i} \quad \text{for } i = 2, \cdots, N \tag{7}$$

if it is defined as follows:

The queueing system in its production part is composed of :

- Manufacturing processes F$_i$ including an infinite queue followed by a station M$_i$ producing batches of size Q_i. Inputs and outputs of the process are batches of size Q_i.
- Synchronization stations U$_i$ (associated with the batch counter $u_i(t)$) having a unique input for batches of size Q_i coming from M$_i$ and an output going to the queue P$_i$. Each output of U$_i$ delivers batches of size Q_i.
- Stocks with queues P$_i$. The raw parts queue P$_0$ is never empty.
- Synchronization stations V$_i$ (associated with the batch counter $v_i(t)$) having an input coming from P$_i$ and an output going to F$_{i+1}$ or out of the system for $i = N$. Every output from V$_i$ are batches of size Q_{i+1}. When batches are outgoing from V$_i$, Q_{i+1} parts are taken in each queue linked with it.

In the control part, the system is composed of :

- Queues D$_{(i,j)}$ linked to V$_i$ or W$_{(i+1,k)}$, for $i = 0, \cdots, N-1$.
- A queue D$_N$ linked to V$_N$ and fed by D.
- Synchronization stations V$_i$, U$_i$ or W$_{(i,j)}$ (respectively associated with counters $v_i(t)$, $u_i(t)$ and $w_{(i,j)}(t)$) with outputs going to some queues D$_{(k-1,l)}$ linked on V$_{k-1}$ or W$_{(k,m)}$, where $1 \leqslant k \leqslant i \leqslant N$. In a same stage we assume that synchronization stations W$_{(i,j)}$ are ordered so that they cannot be linked to W$_{(i,m)}$ for $j \leqslant m$. When batches are outgoing from W$_{(i,j)}$, Q_i parts are taken in each queue linked with it. 3.

Note that if we consider the special case when processes are composed of an infinite input queue followed by a server processing batches of size Q_i, then the canonical formulation (6) can still be used, and the number of parts in process for a stage i is given by $f_i(\boldsymbol{X})$ (Bollon and Di Mascolo, 2004).

4.2 An algorithm to compute the canonical formulation

The algorithm 1 below is able to calculate the canonical formulation (7) for any policy consistent with proposition 1 – notations are identical. It uses the links that exist between $(min, +)$ algebra and the graphs. The core of this algorithm is a shortest path search.

Algorithm 1

We assume that $f_i(0) = 0$ for i varying from 1 to N.

//Construction of a temporary graph G used for //calculation

Add vertices to the empty graph G, corresponding to each V_i, U_i, D and $\mathrm{W}_{(i,j)}$; name them as the associated synchronization stations.

Add an arc from a vertex U_k, V_{k-1}, $\mathrm{W}_{(k,l)}$ or D to a vertex V_{i-1} or $\mathrm{W}_{(i,m)}$ every time an arc exists between the corresponding synchronization stations. The weight on the arc is given by $\lfloor d_{(i-1,j)} \rfloor_{Q_k}$ (or $\lfloor d_{(i-1,j)} \rfloor_{Q_N+1}$ if the arc is coming from D) where $d_{(i-1,j)}(0)$ is the initial number of tokens present in the queue $\mathrm{D}_{(i-1,j)}$ at the end of the arc.

Add an arc from the vertex U_k to the vertex V_k, for k varying from 1 to N. The weight on the arc is given by $\lfloor p_k(0) \rfloor_{Q_{k+1}}$.

```
// $f_i$ calculations :
For $i$ from 1 to $N$ do
  For $j$ from $i$ to $N$ do
    $C_{(i,j)} = \left( \bigotimes_{k=i}^{j-1} \lfloor p_k(0) \rfloor_{Q_{k+1}} \right) \otimes g(j, i-1)$
  End for $j$
  $C_{(i,N+1)} = \frac{\left( \bigotimes_{k=i}^{N} \lfloor p_k(0) \rfloor_{Q_{k+1}} \right)}{d_N(0)} \otimes g(i-1)$
  If $i = 1$ then
    $f_1(\boldsymbol{X}) = \left\lfloor \bigoplus_{j=1}^{N+1} \left( C_{(1,j)} \Big/ \bigotimes_{k=1}^{j-1} \lfloor x_k \rfloor_{Q_{k+1}} \right) \right\rfloor_{Q_1}$
  Else
    $f_i(\boldsymbol{X}) = \left\lfloor \bigoplus_{j=i}^{N+1} \left( C_{(i,j)} \Big/ \bigotimes_{k=i}^{j-1} \lfloor x_k \rfloor_{Q_{k+1}} \right) \oplus \lfloor x_{i-1} \rfloor_{Q_i} \right\rfloor_{Q_i}$
  End if
End for $i$
```

were $g(j, i-1)$ is the shortest path in G from U_j to V_{i-1} and $g(i-1)$ is the shortest path in G from D to V_{i-1}.

Note that in Algorithm 1, the number of customers in queues is rounded off to a certain multiple, so some customers may not always be useful. We give a proof for this algorithm in (Bollon and Di Mascolo, 2004).

5. CONCLUSION

In this paper, we have extended the unified framework for describing and comparing the dynamics of pull control policies presented in (Bollon *et al.*, 2004), to describe a larger class of pull control policies, including those with batch production. This formulation allows to derive some properties for each policy and also to find all identical dynamics between two systems, with a systematic approach (Bollon and Di Mascolo, 2004).

A computing algorithm for calculating the parameters of this formulation has been derived when manufacturing lines are modelled with a queueing network. This algorithm relies on the shortest path search and on the use of $(\min, +)$ algebra tools, that facilitate calculation and enable to obtain concise expressions.

We expect to find some more general results and applications with this formulation. For example it is possible to extend it to deal with systems containing assembly (or disassembly) between stages.

REFERENCES

Axsäter, S. and K. Rosling (1993). Installation vs. echelon stock policies for multilevel inventory control. *Management Science* **39**(10), 1274–1280.

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

Bollon, J-M. (2001). Etude de différentes politiques de pilotage de systèmes de production. PhD thesis. Institut National Polytechnique de Grenoble, LAG.

Bollon, J-M. and M. Di Mascolo (2004). Use of path algebra for a unified description of general pull control policies including batch production. Technical report. Institut National Polytechnique de Grenoble, LAG.

Bollon, J-M., M. Di Mascolo and Y. Frein (2004). Unified framework for describing and comparing the dynamics of pull control policies. *Annals of Operation Research, special issue on Stochastic Models of Production-Inventory Systems* **125**, 21–45.

Buzacott, J. A. (1989). Queueing models of kanban and MRP controlled manufacturing systems. *Engineering Cost and Production Economics* **17**, 3–20.

Dallery, Y. and G. Liberopoulos (2000). Extended kanban control system: Combining kanban and base stock. *IIE Transactions on Design and Manufacturing* **32**(4), 369–386.

Lee, Y-J. and P. Zipkin (1992). Tandem queues with a planned inventories. *Operations Research* **40**(5), 936–947.

Liberopoulos, G. and Y. Dallery (2000). A unified framework for pull control mechanisms in multi-stage manufacturing systems. *Annals of Operations Research* **93**, 325–355.

Liberopoulos, G. and Y. Dallery (2003). Comparative modelling of multi-stage production-inventory control policies with lot sizing. *International Journal of Production Research* **41**(6), 1273–1298.

Monden, Y. (1983). *Toyota Production system.* Norcross, Georgia, Industrial Engineering and Management Press (Institute of Industrial Engineers).