On the integral modelling of control and production management systems
van de Mortel - Fronczak, J.M.; Rooda, J.E.

Published in:
Advances in production management systems: proceedings of the 6th IFIP TC5/WG5.7 international conference on advances in production management systems, APMS'96, Kyoto, Japan, 4-6 November 1996

Published: 01/01/1996

Document Version
Accepted manuscript including changes made at the peer-review stage

Please check the document version of this publication:

- A submitted manuscript is the author's version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
- The final author version and the galley proof are versions of the publication after peer review.
- The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

Citation for published version (APA):

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 15. Dec. 2018
ON THE INTEGRAL MODELLING OF CONTROL AND PRODUCTION MANAGEMENT SYSTEMS

J.M. van de Mortel-Fronczak and J.E. Rooda

Department of Mechanical Engineering
Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
{vdmortel, rooda}@wlb.tue.nl

ABSTRACT

Because of the growing complexity of manufacturing systems, the design of a suitable control and production management is a difficult task. To lend the designers a helping hand, various reference models have been developed. They usually are described in an informal way. This paper presents a formal specification of a control architecture for job-shops based on their parallel character.

Keywords: Control architectures, modelling, parallelism.

1. INTRODUCTION

The design of industrial systems is a great challenge. First, the product must be designed that is to be manufactured. The second phase is the design of the primary (material flow) subsystem. Last, the control of the system must be designed. This secondary (information flow) subsystem takes care of a proper behaviour of the primary subsystem, so that products are delivered according to the requirements of the market.

Many of today’s industrial systems are very complex due to a large number of parallel actions that must be performed at the same time. To support the design process, a formalism is needed that can be used for the modelling of industrial systems. As industrial systems consist of in parallel performing components, attempts at describing them by means of sequential methods cannot be successful. Therefore, it seems justifiable to use a perception which exploits the parallel character of such systems.

As organisational aspects of production are of great importance, a lot of attention is payed to production planning methods, such as MRP and JIT (Chase and Aquilano, 1995). These methods of planning are based on an integral control of the stream of goods. In mass production, they provide good results. Other types of production systems are less easy to control in such a way. In particular, the difficulty of realising an “optimal” plan is painfully apparent in flexible production systems. While from the technological point of view, production in this kind of systems can be extremely flexible, inadequate planning methods substantially reduce this flexibility. Even if an appropriate plan can be determined, it remains an open question whether it is the best possible one. It is not surprising that this problem is so distinctly present within flexible manufacturing. In contrast to mass production, flexible manufacturing is directed at short run and single item production.

It is not easy to control machines and aggregates of machines and people because of the fact that many simultaneous operations take place, both in machines and in machine/person aggregates. Most of the concepts and techniques for analysis of manufacturing systems developed to date pay no attention to this simultaneous character. One viewpoint that is suitable for this purpose is the use of parallel descriptions. In this case, manufacturing systems are treated as collections of independent components that interact by exchanging information through communication channels.

In the past, we have developed simulation packages based on parallel descriptions and embedded in programming languages, such as Sole (Rooda, 1982) based on Simula 67 (Dahl, et al., 1970), S84 (Rooda, et al., 1984) based on Modular Pascal (Bron, 1982), and Process Tool (Wortmann and Rooda, 1990) based on Smalltalk-80 (Goldberg and Robson, 1989). An alternative viewpoint is the use of Petri Nets, as elaborated in (Silva and Valette, 1990) and (David and Alla, 1994).

The view of manufacturing systems as collections of co-operating independent components can be derived from general ideas of concurrent programming (Burns...
and Davies, 1993). The behaviour of components can be specified by concurrent programs. The resulting specifications are called models in the sequel. The language (Van de Mortel-Fronczak, et al., 1993; Van de Mortel-Fronczak and Roorda, 1995) we use to denote models is adopted from (Hoorn, 1991) and is a CSP-derivative (Hoare, 1985). Such models can be validated by sequential or distributed simulation (Misra, 1986). Moreover, an important feature is that CSP provides a basis for formal reasoning. The underlying mathematical framework can support the verification of models. With we can model the material flow as well as the control flow in industrial systems. Data and data processing functions facilitating quantitative analysis are specified separately.

The use of our specification language is illustrated by an application to the description of a hierarchical control architecture for job-shops.

The paper is organized as follows. In Section 2, a general description of the control structure for job-shops is presented. A detailed description of the control and production-management components can be found in Section 3. Concluding remarks of Section 4 complete the paper.

2. CONTROL ARCHITECTURE FOR JOB-SHOPS

A job-shop can be controlled by a single central controller. In this case, however, the parallel character of the job-shop is neglected and the controller usually is very complex. In practice, control systems consist of several components controlling different parts of the job-shop. They usually have a hierarchical structure. Different informal or partly formal specifications of control architectures can be found in the literature (e.g., Biemans, 1989; Haren and Williams, 1990). In order to properly support the design of control systems, formal specifications are needed. In this paper, a (part of a) formal specification of a control architecture for job-shops is presented, which is based on the work of Smit (1992) and Vrijen (1995).

In the control architecture derived from (Smit, 1992) and shown in Figure 1, controller C takes care of a proper functioning of a subsystem consisting of store S, transporter T, and a number of production departments D. Production departments have the same structure, which can be seen in Figure 3. As a consequence, the controllers from different layers exhibit a strong similarity. With the introduction of every layer, an additional store and a transporter are introduced. To keep the waiting times for material at a reasonable level, the number of layers should not be too large. In practice, two layers seem sufficient. For the purpose of this paper, the system consists of two production departments and each department consists of two identical machines. In every department, only operations of one type are performed.

The architecture presented explicitly models two scheduling functions of the control: releasing and sequencing. To this end, the following choices are made:

- fixed work-in-process rule for releasing
- first-in-first-out rule for sequencing

Allocating and dispatching are very simple in the model.

The components in the model co-operate with each other by exchanging material and information via communication channels (arrows in the figures) according to a communication protocol. We concentrate on information exchange, which implies that only control subsystems C and DC are discussed in the next section.

In the sequel, we assume that:

- raw material of only one type is used
- machines do not fail
assembling does not take place

As a consequence, only simple data structures and functions are needed in the model. The following data structures are used:

- **nat** stands for the natural numbers
- **void** is used to denote synchronization channels (no data is communicated)
- **order** = \( \langle \text{nat} \times \text{nat}^* \times \text{nat} \rangle \) — an (internal) order is represented by a triple with the order number, the list of operations (which are represented by numbers 1 and 2 corresponding to the departments; number 0 is reserved for the store), and the number of products; an instance of this type is \( o = \langle 107, [1, 1, 2, 1, 2], 15 \rangle \)
- **tjob** = \( \langle \text{nat} \times \text{nat} \times \text{order} \rangle \) — a transport job is represented by a triple with two numbers corresponding to the origin and the destination, and the associated order; an instance of this type is \( \langle 0, 2, o \rangle \)
- **pjob** = \( \langle \text{order} \times \text{nat}^* \rangle \) — a process job is represented by the pair with the order and the list of operations to be performed in the department; an instance of this type is \( \langle o, [1, 1] \rangle \)

Controller \( C \) has the following general functionality. It receives (internal) orders via channels \( fc \) and \( dtc \) (partly processed in a department), and prepares and issues via channel \( cl \) suitable transport jobs. Moreover, each time the material is delivered to a production department (which is modelled by a synchronization on \( tc \)), it passes the subsequent order via channel \( ed \) to the same department if this is allowed according to the fixed work-in-process rule. To this end, it receives via channels \( dsc \) the information from departments about their work-in-process. If the order is fully processed it leaves the controller via channel \( cf \).

Department controller \( DSC \) issues process commands derived from the orders received via channel \( cl \) from \( C \) to one of the machines via channels \( dcm \). It prepares and issues via channel \( dcl \) suitable transport jobs. Moreover, each time the (partly) finished products are delivered to the store (which is modelled by a synchronization on \( tc \)), it informs the controller about the released production capacity. If an order is completed, it is returned to the controller via channel \( df \).

### 3. Description of Components

To keep the specification of the controller comprehensible and to exploit the parallel character of the controlled system, the functionality of controller \( C \) is divided among four components: supervisor \( SC \), two order buffers \( B \), and transport controller \( TC \) (Figure 2).

These components are called processes in the specification language used. In the heading of each process the communication channels and, possibly, parameters are defined. For instance:

- **fc, tex** : `order in process SC defines input channels fc, tex over which orders are received`
- **set** : `tjob in process TC defines output channel set over which transport jobs are sent`
- **dsc** : `\( \langle \text{nat}^2 \rangle \) in process SC defines a pair of input channels dsc.0 and dsc.1 over which natural numbers are received`
- **tc** : `void in process TC defines a synchronization channel`
- **w** : `nat in process DSC defines a parameter w of type nat`

The body of each process consists of variable declarations followed by the executable statements. The executable part begins with some initial communication actions and/or variable initialization and, usually, contains an infinite repetition in which after a (conditional)
communication some other statements are performed. The enabled communication actions are executed in the first-in-first-out fashion.

Supervisor SC keeps the unfinished orders in the list os and the available production capacities of the departments in the (indexed) variable os. The initially available capacities (which can be assigned to parameter \( u \) of department controller DSC) are received from the departments via channels \( dsc.i \), in any order \((dsc.0?qt.0, dsc.1?qt.1)\). List os is initially empty \((os := []\)\). The following behaviour is repeated infinitely:

- if a new order is received \((fc?o)\), it is appended to the order list \((os := os + [o])\)

- if a part of the production capacity of a department is released \((i : dsc.i?q)\), it is registered by the supervisor \((qt.i := qt.i + q)\)

- if an order is received from the transport controller \((tcx?o)\) and it is finished \((o.1 = [])\), it is sent back via channel cf \((cf!o)\); if it is not yet finished \((o.1 \neq [])\), it is appended to the order list \((os := os + [o])\)

- if a department has available capacity for an order in the list os \((ex(os, qt.i, i) \neq 0)\), such an order can be sent to the order buffer associated with this department \(scb.i\), whereas \(tcx?o\), at which point the order is finished \((o.1 = [])\), the order list is not yet finished \((o.1 \neq [])\), it is appended to the order list \((os := os + [o])\)

The definitions of the functions used \((ex, \text{exor}, \text{rem}, \text{and work})\) can be found in (Vrijisen, 1995).

\begin{verbatim}
  [ o : order, os : order*, q : nat, qt : (nat^2)
    dsc.0?qt.0, dsc.1?qt.1
  ; os := []
  ; s[ [ fc?o o := os + [o] ]
    [ i : dsc.i?q qt.i := qt.i + q ]
    [ t cx?o [ o.1 = [] cf!o ]
      [ o.1 \neq []
        os := os + [o]
      ]
    [ i : ex(os, qt.i, i); scb.i\text{exor}(os, qt.i, i)
      o := exor(os, qt.i, i)
      sl[0, i + 1, o]
      os := \text{rem}(os, o)
      qt.i := qt.i-\text{work}(o)
    ]
  ].
\end{verbatim}

Order buffer B receives executable orders for a department from the supervisor \((scb?o)\) and stores them in the list os until the associated material arrives at the department \((tcx?o)\). Then the order for this material is passed to the department \((cd!o)\) and removed from the list \((os := \text{rem}(os, o))\).

\begin{verbatim}
proc B (scb, t cx : ?order, cd : !order) =
  [ o : order, os : order*
    os := []
    s[ [ scb?o os := os + [o]
      t cx?o cd!o; os := \text{rem}(os, o)
    ]
  ].
\end{verbatim}

Transport controller TC receives transport jobs from the supervisor via channel slct \((slct?)\) and stores them in the list ts \((ts := ts + [t])\). It can also receive orders from departments \((i : dts.i?q)\) what means that \((semi\text{-}finished)\) products must be fetched — to this end a transport job is formed and appended to \(ts\) \((ts := ts + [i + 1, 0, o])\). If the transportable is available \((a \text{ subsequent } tc^-\text{ has already taken place})\) and there are transport jobs, the first of them is released \((ts \neq [])\), the list of orders is removed from ts \((ts := tl(ts))\), and moved to ts \((ts := tl(ts))\) when this transport job is completed \((tc^-)\), the component associated with it \((\text{the supervisor or one of the order buffers})\) is informed \((tlx.(u.1)\))

\begin{verbatim}
proc TC (slct, dts : ?job, dt : !job, tc : !order^2) =
  [ o : order, t, u : vjob, ts : !job
    tc^- t := []
    ; s[ [ slct t := ts := ts + [t]
      [ i : dts.i?q t := ts + [i + 1, 0, o]
      [ ts \neq []
        t := tl(ts)
        ts := tl(ts)
        tc^- t := tlx.(u.1)\])
    ]
  ].
\end{verbatim}

The specification of the department controller is split into two components: department supervisor DSC and department transport controller DTC (Figure 4). Department supervisor transforms the orders received from the order buffer \((cd!o)\) into process jobs which are kept in the list os \((os := os + \text{plan}(o))\). Each process job applies to batches of at most 4 products \((4 \text{ is chosen arbitrarily})\) and determines the subsequent processing step at this department \((\text{possibly more than one operation of the same type})\). If os is not empty, the list of operations associated with the subsequent processing step of the first job in os can be sent to one of the two machines in the department \((i : os \neq [])\). Afterwards, the associated transport job is sent to DTC \((dts!(i, i + 1, h\text{d}(os)\))\), the processing job in question is stored in dt \((dt.i := dt.i + [h\text{d}(os)]\)) and then removed from os \((os := \text{tl}(os))\). When the processed
products are transported from a machine to the store, 
DSC receives the identification number of the machine from 
DTC (ds?q). Afterwards, the completed processing step is removed from the list of processing steps of 
the corresponding batch order (p := res(hd(q))), the released capacity (amount of work associated with it) 
is calculated (w := work(hd(at.q),0)) and the job is removed from at. Moreover, the released capacity is 
communicated to the supervisor SC, the completed batch order is added to the list fs (fs := 
add(fs,p)). Then, if the whole order is completed 
(re(os + ot.0 + ot.1,p0)), it is sent to TC (o := 
recor(fs,p0); dtcl0) and it is removed from fs (fs := 
rem(fs,o)).

The definitions of the additional functions used (plan, 
res, add, re, and recor) can be found in (Vrijen, 1995),
as well.

\[
\begin{array}{ll}
\text{proc DSC (} o: \text{order,} & dts: \text{:nat,} \\
\text{dst : \text{:tjob,} & dte : \text{:order,} \\
\text{,dem : \text{:nat}2,} & dse : \text{:nat,} \\
\text{w : \text{:nat}}) = & \text{seq} \\
\text{[} o, p : \text{order,} & os : \text{pjob}, \\
\text{fs : \text{:order}, & ot : \text{pjob2}}) \\
\text{,q : \text{:nat}} \\
\text{]} \text{; dse!w} \\
\text{; os := [; fs := [; ot := [\text{[;}] \\
\text{; \text{fs := [; os := os + plan(o) \\
\text{; i : os \neq [; dem!hd(os).i) \\
\text{1 --- dsl!(0, i+1,hd(os).0) \\
\text{; ot.i := ot.i + [hd(os)] \\
\text{; os := tl(os) } \\
\text{]} } \\
\text{; dts?q --- p := res(hd(at.q)) \\
\text{; w := work(hd(at.q),0) \\
\text{; ot.q := tl(at.q) \\
\text{; dse!w; fs := add(fs,p) \\
\text{; [re(os + ot.0 + ot.1,p0) \\
\text{--- o := recor(fs,p0) \\
\text{; dtcl0 \\
\text{; fs := rem(fs,o) \text{]} \text{]} \text{.} \\
\end{array}
\]

The department transport controller DTC can receive transport jobs from DSC (ds!t) and stores them in 
ts (ts := ts + [f]). It can also receive requests from both machines to fetch the (partly) finished products 
(i : mdc.i), whereupon a transport job is formed and appended to ts (ts := ts + [i+1, 0, 1]); \L stands for a 
do-not-care order and is used in the transport job, because at the machine there are only products of one 
order). If the transporter is available (a subsequent tdc~ has already taken place) and there are transport 
jobs, the first of them is released (ts \neq [; cl!hd(ts)), the origin of this transport job is stored in p, and the 
transport job itself is removed from ts (ts := tl(ts)). After the completion of this transport job (tdc~), 
if its origin was a machine, the identification number of the machine is sent to DSC ([p \neq 0 \rightarrow 
ds?q])

4. CONCLUDING REMARKS

The control architecture for job-shops discussed in this 
paper is successfully applied in (Van den Heuvel, 1996) 
for a wafer fab and in (Ensink op Kemna, 1996) for a 
machine manufacturer. The control architecture proved 
useful and not difficult to adopt in practical cases. 
Moreover, the scheduling functions chosen can easily 
be replaced by some other ones.

REFERENCES

Biemans, F.P.M. (1989). A Reference Model for Man-
ufacturing Planning and Control, PhD Thesis, 
Department of Electrical Engineering, Twente Uni-
versity of Technology, The Netherlands.

Bron, C. (1982). Modular Pascal Language Defini-
tion. Memo INF-82-10, Department of Computer 
Science, University of Twente, The Netherlands.

Burns, A. and G. Davies. (1993). Concurrent Pro-
gramming International Computer Science Series. 
Addison-Wesley, Reading.

and Operations Management. Irwin, Chicago.

Simula-67 Common Base Language. NCC Publica-
tion S-52, Norwegian Computing Centre.

ing of Dynamic Systems — A Survey. Automatica, 

Ensink op Kemna, S.E.A. (1996). Comparison of 
Two Modelling Techniques @ TUE and @ IFA. 
Masters Thesis, Department of Mechanical Engi-
neering, Eindhoven University of Technology, The 
Netherlands.

Addison-Wesley, Reading.


