MODELING AND SIMULATION OF MANUFACTURING SYSTEMS

S. Bogdan, Z. Kovačić, N. Smolić-Ročak

LARICS – Laboratory for Advanced Robotics and Intelligent Control Systems
Dept. Of Control and Computer Engineering, Faculty of EE&C
University of Zagreb, CROATIA
http://flrcg.rasip.fer.hr
stjepan.bogdan@fer.hr

Abstract

In this paper we present a methodology for modeling and simulation of dynamical discrete event systems (DDES), predominantly flexible manufacturing systems (FMS). Proposed technique is based on the matrix representation of a manufacturing system. Although prerequisites that are required for an event to start are given by static model of DES, we are not able to tell in which particular moment these prerequisites are met, i.e. we do not know when the event actually starts. In real applications on actual manufacturing processes, we will be sensing the completion of prerequisite jobs by either using sensors (e.g., proximity, tactile, etc.) or via notification from the machines or resources. On the other hand, for the purpose of computer simulation, we must find a way to keep track of time lapsed in the processing of jobs. To keep track of jobs time durations, we incorporated the system dynamics into the matrix model in a form of an *extended lifetime*. That is, a real number d_i, called a lifetime, is associated with each task in an MS. Problem that we analyze in the paper is that when described as a bill-of-material (BOM), or in some other engineering form, the job sequence does not disclose potential difficulties that might develop when the structure of an MS, which executes this particular sequence, is determined. In the paper we show how two of these potential difficulties, conflict and deadlock, can be exposed by using static and dynamic simulations of an MS. Based on the matrix model, these simulations provide a complete insight in the system performance.

Keywords: Manufacturing Systems, Matrix-algebra, Modeling, Simulation

Presenting Author's biography

Stjepan Bogdan received his Ph.D.E.E. in 1999, M.S.E.E. in 1993, and B.S.E.E. in 1990 at the University of Zagreb, Croatia. He is currently an assistant professor at the Department of Control and Computer Engineering, Faculty of EE&C, University of Zagreb. His areas of interest are robotics, discrete event systems, and autonomous and intelligent systems. He received a Fulbright scholarship in 1996/97 and worked as a visitor researcher in the Automation and Robotics Research Institute, University of Texas at Arlington, with research group of Prof. Frank L. Lewis. He is a principle investigator and a project leader of several R&D projects founded by industry and government. He is a coauthor of three books and numerous papers published in journals and presented at conferences. He is a member of KoREMA, the IEEE, Sigma Xi, and a vice-president of Croatian Robotics Society.

ISBN 978-3-901608-32-2

1 Introduction

Today, virtual models provide a very inexpensive and convenient way for complete factory design. Instead of building real systems, a designer first builds new factory layouts and defines resource configurations in the virtual environment and refines them without actual production of physical prototypes. Allowing clear visualization of all potential problems caused by the layout, virtual modeling and dynamic simulation of manufacturing processes has traced a completely new route to analysis and design of MSs [1–3].

A factory layout design, physical modeling, control synthesis, performance analysis, dynamic simulation and visualization of robotized manufacturing systems have become much easier and more effective with specialized programs for virtual-factory modeling and simulation. Some virtual-factory simulators originated from the academia [4–6], but most of them are sophisticated products of leading robot manufacturers and independent companies.

In general, each of those MS simulators can be partitioned in the following modules: virtual objects designer, factory layout designer, trajectory generator, and functional testing module. The aim of this paper is to propose a method for MS analysis in functional testing module. Described algorithm represents a mathematical core of such a module.

Functional testing has a goal to connect a physical setup with the plan of the simulated MS. Functional testing is concerned with a job-sequence definition, setting of MS parameters, conflict and deadlock analysis at the local and global level (at the robot workcell or robot station, and at the whole MS level), synthesis of control logic, study of different job-scheduling strategies, simulation and visualization of dynamic phenomena during MS operation. Having a plan of a manufacturing process and all necessary MS data, functional testing should help the MS designer to reach a reliable and objective MS performance evaluation.

The paper is organized in the following way. In the next chapter we present matrix representation of discrete event systems. The system matrices and logical equations are given and described. In paragraph 3 a concept of extended lifetime is explained and timed matrices are defined. Implementation of proposed method is given in paragraph 4 by using a simulation example.

2 Matrix representation of DES

Generally, the complete task plan could be given by the system matrices $\mathbf{F}_v$, $\mathbf{S}_v$, $\mathbf{F}_r$, $\mathbf{S}_r$, which are specified by higher-level planners, or may be written down in manufacturing systems given the BOM or the assembly tree plus resource-availability information [7-11]. Additionally, these matrices can easily be extracted from plans generated by typical planning software, including hierarchical planners. Since each matrix has a well-defined function for job sequencing, resource assignment, and resource release, they are straightforward to construct as well as easy to modify in the event of goal changes, resource changes, or failures; that is, they accommodate task planning as well as *task replanning*.

Denoting the discrete event iteration number with k, we can calculate the logical state vector x each time an event takes place, *i.e.* a job is completed, resource becomes idle or part enters the system:

$$\bar{\mathbf{x}}(k) = \mathbf{F}_{\mathrm{v}} \triangle \bar{\mathbf{v}}_{\mathrm{c}}(k-1) \triangledown \mathbf{F}_{\mathrm{r}} \triangle \bar{\mathbf{r}}_{\mathrm{c}}(k-1) \triangledown \mathbf{F}_{\mathrm{u}} \triangle \bar{\mathbf{u}}(k-1) \quad (2.1)$$

where $\mathbf{F}_{\mathrm{r}}$ is *resource-requirements matrix*, $\mathbf{F}_{\mathrm{r}}(i,j) = 1$ if resource j contributes to construction of the ith component of the logical state vector, otherwise $\mathbf{F}_{\mathrm{r}}(i,j) = 0$, $\mathbf{F}_{\mathrm{v}}$ is *job-sequencing matrix*, $\mathbf{F}_{\mathrm{v}}(i,j) = 1$ if job j contributes to construction of the ith component of the logical state vector, otherwise $\mathbf{F}_{\mathrm{v}}(i,j) = 0$, $\mathbf{F}_{\mathrm{u}}$ is *input matrix*, $\mathbf{F}_{\mathrm{u}}(i,j) = 1$ if an input j contributes to construction of the ith component of the logical state vector, otherwise $\mathbf{F}_{\mathrm{u}}(i,j) = 0$, $\mathbf{v}_{\mathrm{c}}$ is *a job-completed vector*, $\mathbf{r}_{\mathrm{c}}$ is called *an idle-resource vector*, and $\mathbf{u}$ is a system input.

The equations describing the consequent parts of rules can be rewritten in the same way:

$$\begin{aligned} \mathbf{v}_{\mathrm{s}}(k) &= \mathbf{S}_{\mathrm{v}} \triangle \mathbf{x}(k) \\ \mathbf{r}_{\mathrm{s}}(k) &= \mathbf{S}_{\mathrm{r}} \triangle \mathbf{x}(k) \\ \mathbf{y}(k) &= \mathbf{S}_{\mathrm{y}} \triangle \mathbf{x}(k) \end{aligned} \quad (2.2)$$

where $\mathbf{S}_{\mathrm{v}}$ is *job-start matrix*, $\mathbf{S}_{\mathrm{v}}(i,j) = 1$ if the jth component of the logical state vector is a prerequisite to start job i, otherwise $\mathbf{S}_{\mathrm{v}}(i,j) = 0$, $\mathbf{S}_{\mathrm{r}}$ is *resource-release matrix*, $\mathbf{S}_{\mathrm{r}}(i,j) = 1$ if the jth component of the logical state vector is a prerequisite to start the release of resource i, otherwise $\mathbf{S}_{\mathrm{r}}(i,j) = 0$, $\mathbf{S}_{\mathrm{y}}$ is *output matrix*, $\mathbf{S}_{\mathrm{y}}(i,j) = 1$ if the jth component of the logical state vector is a prerequisite for output i, otherwise $\mathbf{S}_{\mathrm{y}}(i,j) = 0$.

In (2.1) and (2.2) operations are carried in so called *and/or algebra* denoted $\triangle$ and $\triangledown$, where multiplication is replaced by AND, and addition is replaced by OR.

In order to be able to link recursive equations (2.1) and (2.2) we have to relate a job-completed vector $\mathbf{v}_{\mathrm{c}}$ with a job-start vector $\mathbf{v}_{\mathrm{s}}$, and an idle-resource vector $\mathbf{r}_{\mathrm{c}}$ with a resource-release vector $\mathbf{r}_{\mathrm{s}}$. According to its

definition, the components of vector $\mathbf{v}_c$ correspond to completed operations, hence, each time a job is completed, the number of parts held by this particular job is increased. At the same time, if a job contributes to a rule(s) that is fulfilled, an already processed part(s) leaves the job and proceeds through the system. In other words

$$\begin{aligned}\mathbf{v}_c(k) &= \mathbf{v}_c(k-1) + \mathbf{S}_v\mathbf{x}(k) - \mathbf{F}_v^T\mathbf{x}(k) \\ &= \mathbf{v}_c(k-1) + \left[\mathbf{S}_v - \mathbf{F}_v^T\right]\mathbf{x}(k)\end{aligned} \tag{2.3}$$

where multiplications and additions are carried out in the standard way.

By following the same reasoning one can find the number of idle resources and the number of finished products in step *k* as

$$\begin{aligned}\mathbf{r}_c(k) &= \mathbf{r}_c(k-1) + \mathbf{S}_r\mathbf{x}(k) - \mathbf{F}_r^T\mathbf{x}(k) \\ &= \mathbf{r}_c(k-1) + \left[\mathbf{S}_r - \mathbf{F}_r^T\right]\mathbf{x}(k) \\ \mathbf{y}(k) &= \mathbf{y}(k-1) + \mathbf{S}_y\mathbf{x}(k)\end{aligned} \tag{2. 4}$$

Equations (2.3) and (2.4) can be written in the following form

$$\begin{aligned}\bar{\mathbf{x}}(k) &= \mathbf{F}_{\Delta}\bar{\mathbf{m}}(k-1) \quad , \quad \mathbf{m}(0) = \mathbf{m}_0 \\ \mathbf{m}(k) &= \mathbf{m}(k-1) + \left[\mathbf{S} - \mathbf{F}^T\right]\mathbf{x}(k)\end{aligned} \tag{2.5}$$

with

$$\mathbf{S} = \begin{bmatrix}\mathbf{S}_u \\ \mathbf{S}_v \\ \mathbf{S}_r \\ \mathbf{S}_y\end{bmatrix} \quad , \quad \mathbf{F}^T = \begin{bmatrix}\mathbf{F}_u^T \\ \mathbf{F}_v^T \\ \mathbf{F}_r^T \\ \mathbf{F}_y^T\end{bmatrix} \quad , \quad \mathbf{m}(k) = \begin{bmatrix}\mathbf{u}(k) \\ \mathbf{v}_c(k) \\ \mathbf{r}_c(k) \\ \mathbf{y}(k)\end{bmatrix}$$

where $\mathbf{S}_u = [\mathbf{0}], \mathbf{F}_y = [\mathbf{0}]$ are null-matrices required for keeping matrix dimensions consistent. Usually, matrix **S** is called the *activity-start matrix*, and matrix **F** is called the *activity-completion matrix*.

3 A concept of extended lifetime

The matrix model derived in the previous paragraph describes only logical (static) properties of an MS. Under the assumption that there are no machine failures, every task that starts will actually finish in a finite time, hence:

$$\begin{aligned}v_{ci}(t) &= v_{si}(t - d_{vi}) \quad , \\ r_{ci}(t) &= r_{si}(t - d_{ri}) \quad .\end{aligned} \tag{3. 1}$$

where d_{vi} and d_{ri} are lifetimes of operation v_i and resource release r_i respectively.

The final goal of an MS modeling and analysis is to prepare the ground for design of an appropriate dispatching supervisor. The nature of this supervisor is determined by its computer-based implementation, usually in a form of a PLC. Since the execution of an algorithm on a PLC is cyclic, the moment in which the supervisor detects completion of an operation does not necessarily coincide with the actual moment in which an operation is finished. Therefore, from the supervisor point of view, the operation lifetime is not d_i but $d_i + \varepsilon_i$ (Figure 2.1) [12, 13]. We can rewrite (3.1) as

$$\begin{aligned}v_{ci}^s(kT_s) &= v_{si}(kT_s - d_{vi} - \varepsilon_{vi}) = v_{si}\left((k - n_{vi})T_s\right), \\ r_{ci}^s(kT_s) &= r_{si}(kT_s - d_{ri} - \varepsilon_{ri}) = r_{si}\left((k - n_{ri})T_s\right) \quad .\end{aligned} \tag{3. 2}$$

where $n_iT_s \geq d_i > (n_i - 1)T_s$, T_s is the supervisor sampling (cycle) interval, and n_i is an integer representation of the lifetime expressed in number of sampling intervals . It is apparent that the sampling interval should be small enough to provide an accurate dynamic model.

Introduction of a *shift (delay) operator q* in (3.2) gives the vector form as

$$\begin{aligned}\mathbf{v}_c(q) &= \mathbf{T}_v(q)\mathbf{x}(q), \\ \mathbf{r}_c(q) &= \mathbf{T}_r(q)\mathbf{x}(q),\end{aligned} \tag{3. 3}$$

where $\mathbf{T}_v$ and $\mathbf{T}_r$ are operations and resources release *delay matrices* with elements representing operations lifetimes. Delay matrices are obtained by replacing each entry "1" in $\mathbf{S}_v$ and $\mathbf{S}_r$ with a shift operand representation of corresponding lifetime.

Due to the existence of shared resources, transformation of the second equation in (3.3) requires additional explanation. Namely, each non-shared resource in **r** has its corresponding operation in **v** which is responsible for its release. In the same time, a shared resource that is represented by one component in vector **r**, has several operations in **v** it could be released from. As release lifetimes associated with these operations generally differ, the row in $\mathbf{T}_r$ that corresponds to a shared resource could have two or more different entries.

Conversion of equations (3.3) into recursive form, suitable for simulation, can be done in the same way as in the case of static recursive model (2.5).

$$\mathbf{v}_c(q) = q^{-1}\mathbf{v}_c(q) + \mathbf{T}_v(q)\mathbf{x}(q) - \mathbf{F}_v^T\mathbf{x}(q) \quad , \qquad (3.4)$$

$$\mathbf{r}_c^s(q) = q^{-1}\mathbf{r}_c^s(q) + \mathbf{T}_r(q)\mathbf{x}(q) - \mathbf{F}_r^T\mathbf{x}(q) \quad . \qquad (3.5)$$

Finally, the dynamic matrix model of an MS is obtained by including the shift operator q in logical state vector equation:

$$\begin{aligned} &\overline{\mathbf{x}}(q) = \mathbf{F}_\Delta q^{-1}\overline{\mathbf{m}}(q) \quad , \quad \mathbf{m}(0) = \mathbf{m}_0 \quad , \\ &\mathbf{m}(q) = q^{-1}\mathbf{m}(q) + \left[\mathbf{T}(q) - \mathbf{F}^T\right]\mathbf{x}(q) \quad , \end{aligned} \qquad (3.6)$$

where

$$\mathbf{T}(q) = \begin{bmatrix} \mathbf{S}_u \\ \mathbf{T}_v(q) \\ \mathbf{T}_r(q) \\ \mathbf{S}_y \end{bmatrix} .$$

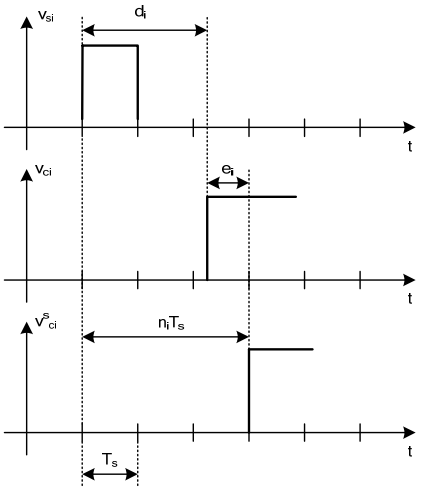

Figure 2.1. Extension of the operation lifetime for the system dynamics modeling.

By comparing (2.5) with (3.6) one can notice that the main difference between two models is in matrix **S** which is replaced with delay matrix $\mathbf{T}(q)$. Before we give an example of system dynamics modeling based on (3.6), there are two issues that have to be further discussed. The simulation of a dynamic model is done in the way that each element $\mathbf{T}(q)(i,j)$ of delay matrix, that is not equal to 0, is associated with a *clock*, denoted $C(i,j)$, containing the time passed after the job has been started. All clocks are initially set to zero. When the rule for starting a particular task is satisfied, the corresponding clock is activated. Then, in each sampling interval all active clocks are checked. If some clock is found to be equal to or greater than the corresponding task lifetime, defined as an entry of the delay matrix, the particular task is considered completed. In that case the entry of vector **m** matching this task is incremented. Such realization of model (3.6) is valid as long as there are no resources that can process more than one part at a time. If there exists such a resource, then simulation algorithm must be modified in a straightforward manner, by expanding the number of clocks for each additional part processed simultaneously by the resource. For example, if $\mathbf{T}(q)(i,j) = q^{-5}$ stands for some task that lasts 5 sampling intervals and can process 3 parts in the same time, then it is associated with a so called multipart clock, that is, $C(i,j,1)$, $C(i,j,2)$ and $C(i,j,3)$. The first part entering the task activates $C(i,j,1)$, the second one $C(i,j,2)$ and the third part $C(i,j,3)$. Having its own clock, each part can be tracked separately.

The second issue that needs additional clarification when one considers realization of the dynamic matrix model is related to so called "hidden" parts. Let us assume that rule x_i, which has job v_i in its prerequisite part and job v_j in its consequent part, is satisfied in the sampling interval k. Further on, let processing of the part in v_j follows immediately after processing in v_i. Then, according to (3.6), term $\mathbf{F}^T\mathbf{x}(k)$ removes the part from v_i, i.e. corresponding component of vector **m** is decreased. Processing of the part in v_j starts in the same sampling interval k, but due to the operation lifetime, the part will be completed n_{vj} sampling intervals later, i.e. the component of vector **m** that corresponds with operation v_j will be increased with delay. Therefore, one is not able to tell where the part is if only vector $\mathbf{m}(k)$ is tracked. However, the results of system performance analysis in the sense of system throughput, resources utilization, etc., are not influenced by the existence of hidden parts.

4 A simulation example

Let us consider the system shown in Figure 4.1. The lifetimes of work cell operations are given in Table 1. Release of buffer BA, which lasts 2.75 seconds, is the shortest task in the work cell, thus, we choose the simulation sampling interval to be $T_s = 1$ [s]. Extended lifetimes for this sampling interval are specified in the third column of Table 4.1.

There are ten different tasks in the system, and two of them can hold two parts simultaneously, buffer operation BP and buffer release B. Accordingly, simulation requires eight standard and two multipart clocks. As in the case of the static simulation, we assume that only one part enters the system at the initial step and all resources are idle at the beginning, consequently, $\mathbf{m}(0) = \mathbf{m}_0 = [1\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 2\ 1\ 0]^T$.

Results obtained by simulation are shown in Figure 4.2. Upon entering the system, the part has been processed in machine A. After 76 sampling intervals (graph MAP) the part is removed from the machine

into the buffer, which can be clearly seen on graph R – robot is idle while the part is processed in machine A, then it moves the part (10 sampling intervals) and finally it is released (6 sampling intervals). The part advances through the system and after 211 samples (see graph RP2 that represents the last operation of the system) it leaves the work cell.

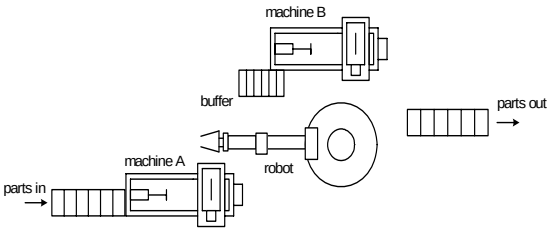

Figure 4.1. The work cell layout for the assembly tree in Figure 1.

Table 4.1. Lifetimes of the work cell tasks

Operation	Lifetime d_i [s]	Extended lifetime n_i
MAP (drill)	76	76
RP1 (move 1)	10	10
BP (buffer)	3.5	4
MBP (grind)	113	113
RP2 (move 2)	7.5	8
release of MA	15	15
release of B	2.75	3
release of MB	10	10
release of R (after RP1)	5.75	6
release of R (after RP2)	4.25	5

In order to get a complete insight in the system dynamic properties we have to simulate a situation with several parts being processed simultaneously. This situation is closer to the real conditions in which the system is fed by parts with predetermined frequency (or stochastically).Given that manufacturing systems are generally designed to work periodically, this kind of simulation provides results that can be used for calculations of production cycles, resources utilizations, bottleneck machines, etc.

Graphical representation of results obtained in case a new part is available each time robot R is idle, is given in Figure 4.3.

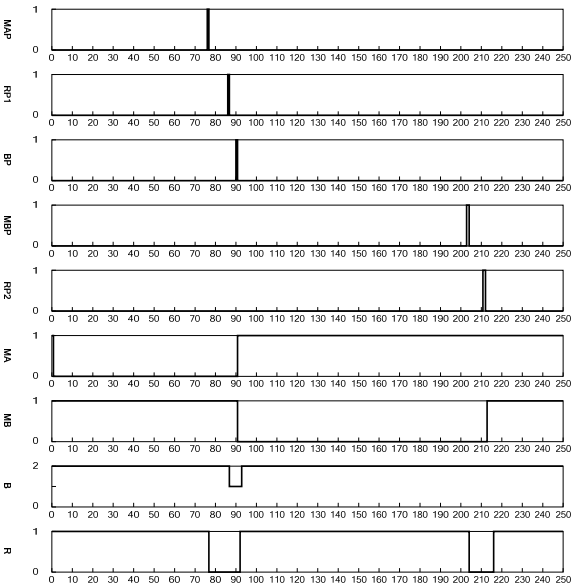

Figure 4.2. Graphical representation of results obtained by the dynamic simulation (one part processed).

Several observations regarding system performance can be made from attained results. We see that the first part leaves the system after 211 samples, same as in the previous simulation when only one part has been passed through the work cell. After that, the time period between departures of two consecutive parts from the system is equal to 123 sampling intervals, which corresponds to the sum of processing and release lifetimes of machine B (see Table 4.1). Hence, simulation confirmed, as we expected, that this machine is the system bottleneck since it is the slowest one according to Table 4.1.

We conclude this discussion with a note on another interesting phenomenon that is revealed by results of the dynamic model simulation. From the graphical representation of the first operation in the system, MAP, it is evident that 10 parts have entered the work cell. On the other hand, only 5 parts have arrived at the output. The other 5 parts got trapped in the system; all resources are occupied and none of them can be released since they are all waiting for each other. This condition is known as a *circular blocking* and it is equivalent to already mentioned deadlock. Analysis of graphs in Figure 4.3. can clearly show how the system came into deadlock. In sampling interval k = 806 machine A just finished processing of 9th part. In the same time sample buffer B is full (BP = 2 for k = 806), machine B is processing 6th part and robot R is idle.

Prerequisites of rule x_2, MAP is completed and robot R is idle, are met, thus, task in the consequent part, RP1, is started. Since buffer is full, robot cannot complete RP1. A part that is supposed to leave the buffer and make room for a new one is blocked by the part in machine B which waits to be cleared by

robot R that is already holding a part. Resources wait for each other, the system is deadlocked and parts cannot proceed through the line.

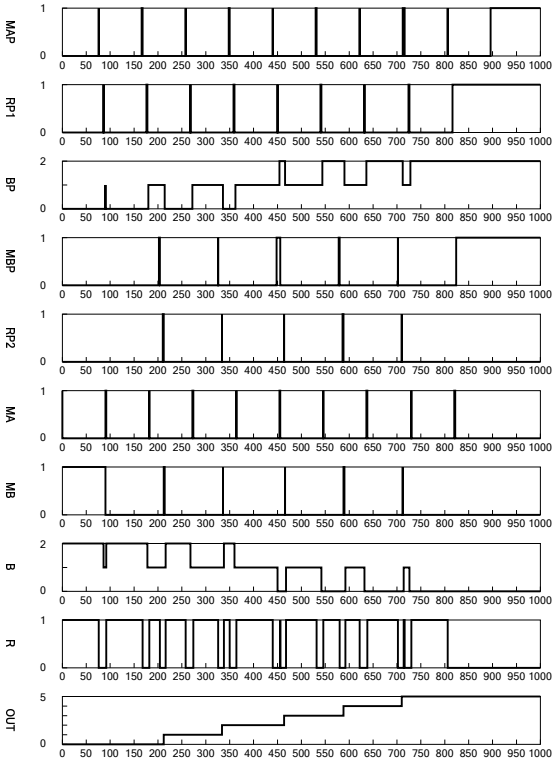

Figure 4.3. Results obtained by simulation based on dynamic matrix model (several parts processed).

At the end of the example, let us reorder the job sequence in the work cell by exchanging positions of machines A and B, i.e. instead of drill the first operation in the sequence is grind. The dynamic matrix model is changed correspondingly and simulation results are shown in Figure 4.4. It can be noticed that deadlock is avoided and the system has cyclic activities. Parts are leaving the work cell with a period of 123 sampling intervals. Operational time of particular resource can be easily determined from graphs corresponding to its idleness and activity. For example, graph B clearly shows that buffer is underutilized as it never accommodates more than 1 part, i.e. the system could work correctly with a 1-slot buffer. As expected, the slowest machine is operational 100 % of time (graphs MBP and MB), while activity periods of other two resources are approximately 24 % for robot (graphs RP1, RP2 and R) and 74 % for machine A (graphs MAP and MA).

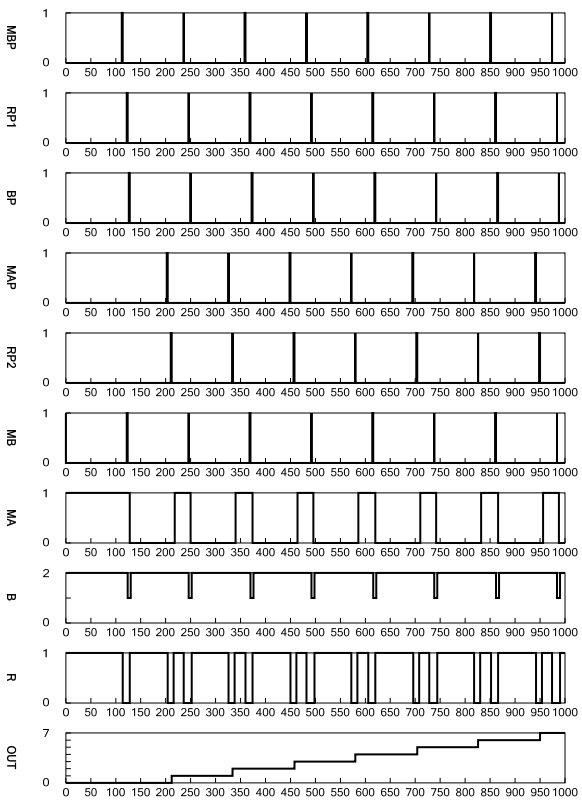

Figure 4.4. Results obtained by simulation based on dynamic matrix model (reordered job sequence).

References

[1] Viswanadham N, Narahari Y. Performance Modeling of Automated Manufacturing Systems. New Jersey: Prentice Hall, 1992.

[2] Vince J. Virtual Reality Systems. Reading, MA: Addison-Wesley, 1995.

[3] Mayr H. Virtual Automation Environments – Design, Modeling, Visualisation, Simulation. New York Basel: Marcel Dekker, 2002.

[4] Ge S S, Lee T H, Gu D L, Woon L C. A One Stop Solution in Robotic Control System Design, IEEE Rob. Aut. Mag. 2000;7:3:42–54.

[5] Corke P. Robotic Toolbox for Matlab, CSIRO Manufacturing Science and Technology, http://www.cat.csiro.au/cmst/, visited 2005.

[6] Choi B, Park B, Ryu H Y. Virtual Factory Simulator Framework For Line Prototyping, J. of Advanced Man. Sys., World Scientific Publishing Company 2004;3:1:5–20.

[7] Tacconi DA, Lewis FL. A New Matrix Model for Discrete Event Systems: Application to Simulation, IEEE Control Systems Magazine 1997;17;5:62-71.

[8] Lewis FL, Gurel A, Bogdan S, Docanalp A, Pastravanu OC. Analysis of Deadlock and Circular Waits using a Matrix Model for Flexible Manufacturing Systems, Automatica 1998;34:9:1083-1100

[9] Gurel A, Bogdan S, Lewis FL. Matrix Approach to Deadlock-Free Dispatching in Multi-Class Finite Buffer Flowlines, IEEE Transactions on Automatic Control 2000;45;11:2086-2090

[10] Mireles J, Lewis FL. Intelligent Material Handling: Development and Implementation of a Matrix-Based Discrete Event Controller, IEEE Transactions on Industrial Electronics 2001;48;6.

[11] Mireles J, Lewis FL, Gurel A. Deadlock Avoidance for Manufacturing Multipart Reentrant Flow Lines Using a Matrix-Based Discrete Event Controller, Int. J. Production Research 2002;40;13:3139-3166.

[12] Smolic-Rocak N, Bogdan S, Kovacic Z, Reichenbach T, Birgmajer B. Dynamic modeling and Simulation of FMS by using VRML, CD Proceedings of 15th IFAC World Congress 2002.

[13] Bogdan S, Lewis FL, Gurel A, Kovacic Z. Timed matrix-based model of flexible manufacturing systems, Proceedings of the IEEE International Symposium on Industrial Electronics 1999;3:1373-1378

ISBN 978-3-901608-32-2