Modeling and Simulation of Manufacturing Systems based on a Machine-Job Incidence Matrix

Ivica Sindicic, Exor d.o.o, Zagreb, Croatia
Tamara Petrovic, Stjepan Bogdan, University of Zagreb, Croatia

SNE Simulation Notes Europe SNE 20(3-4), 2010, 5-12, doi: 10.11128/sne.20.tn.09981

This paper presents a method for modeling and simulation of particular class of manufacturing systems. The method is based on so called Machine-Job Incidence Matrix (MJI) that is formed from Steward sequencing matrix and Kusiak machine-part incidence matrix. A model of the system in a form of MJI matrix is easy to comprehend. It can be put in direct relation with other manufacturing systems analysis tools, such as Petri nets and MS matrix model. Moreover, structural properties of MJI matrix offer direct insight in the system configuration, hence, providing a ground for the system analysis and supervisory controller design. Properties, such as circular waits and conflicts, can be determined straightforwardly by using simple matrix manipulations, thus, allowing design of sequencing control algorithms. An extension of MJI matrix in time domain offers a foundation for determination of recursive equation that can be used for simulation of system's dynamics. Although manufacturing systems have been used for validation of the proposed modeling technique, the method can be applied on other discrete event systems as well.

Introduction

Using today's classification of systems, manufacturing systems (MSs) can be treated as hybrid systems that contain a mixture of various dynamic behaviors—continuous and discrete control loops, Boolean variables related to process states, and discrete events, all embraced by a usually hierarchical decision-making overhead.

This means that an MS structure contains both hard and soft technology, first focused on the product fabrication, assembly and distribution, while later the focus is on the support and coordination of manufacturing operations. The MS's hard technology is split into several levels – from the factory level via the operating center, workcell and robotic station levels to a particular manufacturing process level. The accompanying soft technology is also split into several levels – from the highest strategy level, via lower planning, supervisory, and manipulating levels to the basic manufacturing task level.

Today, simulation 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 understanding of all potential problems caused by the factory layout and/or dispatching strategy, modeling and dynamic simulation of manufacturing processes has traced a completely new route to analysis and design of MSs [1–3]. Simulation of robotized manufacturing systems has become much easier and more effective with specialized programs for virtual-factory modeling and simulation. Many virtual-factory simulators have origin in the academia [4–8]. Each of these tools has a mathematical core in a form of an algorithm used to describe dynamic behavior of MS elements. In this paper we exploit so called machine-job incidence matrix (MJI) for purpose of deriving such an algorithm.

The paper is organized in the following way. In the next section, we describe construction of MJI matrix, which is based on two well-known matrices: resource requirements matrix [9], also known as *machine-part incidence matrix* (MPI), and Steward sequencing matrix [10], also referred as *design structure matrix* (DSM). In Section 1 an extension of MJI to design of MS recursive simulation model is given, followed by illustrative example. In Section 2 the recursive mathematical model of *free choice multiple reentrant flowlines* (FMRF) is presented in detail. We consider FMRF systems with multiple flowlines, where resources can hold an arbitrary number of parts simultaneously (*k-limited* systems). In Sections 2.5 and 2.6 a mathematical framework that allows testing of various control policies during the system simulation is presented. The recursive model forms the basis of the developed system simulator, which is presented in Section 3. We conclude the paper with final remarks and an outline for future work.

SNE 20/3-4, December 2010

1 Construction of the machine-job incidence matrix

We start with introduction of basic terms that will be used throughout the paper. Let Π be the set of distinct types of parts produced (or customers served) by a flexible manufacturing system (FMS). Then each part type $P_k \in \Pi$ is characterized by a predetermined sequence of job operations $J^k = \{J_1^k, J_2^k, \ldots, J_{L_k}^k\}$ with each operation employing at least one resource (note that some of these job operations may be similar, e.g. J_i^k and J_j^k with $i \neq j$ may both be drilling operations). We uniquely associate with each job sequence J^k, the operations of raw part-in, J_{in}^k, and finished product-out, J_{out}^k. Denote the system resources with $R = \{r_i\}_{i=1}^n$, where $r_i \in R$ can represent a pool of multiple resources each capable of performing the same type of job operation. In this notation, $R_k \subset R$ represents the set of resources utilized by job sequence J^k. Note that $R = \bigcup_{k\in\Pi} R^k$ and $J = \bigcup_{k\in\Pi} J^k$ represent all resources and jobs in a particular FMS. Since the system could be re-entrant, a given resource $r \in R^k$ may be utilized for more than one operation $J_i^k \in J^k$ (*sequential sharing*). Also, certain resources may be used in the processing of more than one part-type so that for some $\{l, k\} \in \Pi$, $l \neq k$, $R^l \cap R^k \neq \emptyset$ (*parallel sharing*). Resources that are utilized by more than one operation in either of these two ways are called *shared resources*, while the remaining are called *non-shared resources*. Thus, one can partition the set of system resources as $R = R_s \cup R_{ns}$, with R_s and R_{ns} indicating the sets of shared and non-shared resources, respectively, where $|R_s| = n_s$ and $|R_{ns}| = n_{ns}$, $n_s + n_{ns} = n$. For any $r \in R$ we define the resource job set $J(r)$. Obviously, $|J(r)| = 1$ (> 1) if $r \in R_{ns}$ $(r \in R_s)$. The previously defined job set J and resource set R are associated with MJI matrix in the form of vectors. A *job vector* $\boldsymbol{V}: J \to \aleph$ and a *resource vector* $\boldsymbol{R}: R \to \aleph$ represent the set of jobs and the set of resources corresponding to their nonzero elements. The set of jobs (resources) represented by $\boldsymbol{V}(\boldsymbol{R})$ is called the support of $\boldsymbol{V}(\boldsymbol{R})$, denoted $\sup \boldsymbol{V}(\sup \boldsymbol{R})$; i.e. given $\boldsymbol{V} = [v_1 v_2 \cdots v_q]^{\mathrm{T}}$, vector element $v_i > 0$ if and only if $v_i \in \sup \boldsymbol{V}$. In the same manner, given $\boldsymbol{R} = [r_1 r_2 \cdots r_q]^{\mathrm{T}}$, vector element $r_i > 0$ if and only if resource $r_i \in \sup \boldsymbol{R}$.

MJI matrix, used herein, describes *free-choice multiple re-entrant flowline* (FMRF) class of manufacturing systems [11]. This class has the following properties: each operation in the system requires one and only one resource with no two consecutive jobs using the same resource, i.e. $\forall J_i^k \in J$, $R(J_i^k) = 1$ and $R(J_i^k) \neq R(J_{i+1}^k)$, there are no assembly jobs, and there are shared resources in the system. In FMRF some jobs have the option of being machined in a resource from a set of resources (routing of jobs), and each resource might be used to machine different jobs. For each job that can be performed by more than one resource, there exists a material handling buffer (routing resources) that routes parts. A subclass of FMRF systems that does not allow routing of jobs, i.e. each job is executed by a single pre-assigned resource, is called *multiple re-entrant flowlines* (MRF). Having defined basic terms we proceed with MJI construction. DSM is a square matrix containing a list of tasks in the rows and columns with matrix elements indicating the execution sequence. In case of FMRF systems DSM is subdiagonal identity matrix of the following form

$$\Gamma = \begin{array}{c|cccccccccc} & J_1^1 & J_2^1 & \cdots & J_{L_1}^1 & J_1^2 & \cdots & J_{L_2}^2 & \cdots & J_1^m & \cdots & J_{L_m}^m \\ J_1^1 & 0 & 0 & \cdots & 0 & & & & & & & \\ J_2^1 & 1 & 0 & \cdots & 0 & & & & & & & \\ \vdots & \vdots & \vdots & \ddots & \vdots & & & & & & & \\ J_{L_1}^1 & 0 & 0 & \cdots & 0 & & & & & 0 & & \\ J_1^2 & & & & & 0 & & & & & & \\ \vdots & & & & & & \ddots & & & & & \\ J_{L_2}^2 & & & & & & & 0 & & & & \\ \vdots & & & & & & & & \ddots & & & \\ J_1^m & & & & & & & & & 0 & & \\ \vdots & & & & & & & & & & \ddots & \\ J_{L_m}^m & & & & & & & & & & & 0 \end{array}$$

The order of tasks in the rows or columns indicates the execution sequence. The relationships among the tasks are usually represented by '1' in the corresponding cells, i.e. if task j is immediate predecessor of task i then DSM element (i, j) is equal to 1, otherwise it is zero. The second matrix, used for system description, is MPI. It captures relations between resources and parts processed by a system. Generally, MPI has the following form,

$$\Theta = \begin{array}{c|cccc} & R_1 & R_2 & \cdots & R_q \\ P_1 & \mathbf{0/1} & \mathbf{0/1} & \ldots & \mathbf{0/1} \\ P_2 & \mathbf{0/1} & \mathbf{0/1} & \ldots & \mathbf{0/1} \\ \cdots & \ldots & \ldots & & \ldots \\ P_p & \mathbf{0/1} & \mathbf{0/1} & \ldots & \mathbf{0/1} \end{array}$$

where $0/1$ represents entry of 0 or 1 depending on relation between corresponding part and resource: if resource j is processing a part i then MPI element (i, j) is equal to 1, otherwise is zero.

Since the sequence $J^k = \{J_1^k, J_2^k, \ldots, J_{L_k}^k\}$ represents P_k processing order, by combining DSM and MPI matrices, we get general form of machine-job incidence matrix for an FMRF system as

$$\Lambda = \begin{array}{c|cccc} & R_1 & R_2 & \cdots & R_q \\ J_1^1 & 0/1 & 0/1 & \ldots & 0/1 \\ J_2^1 & 0/1 & 0/1 & \ldots & 0/1 \\ \ldots & \ldots & \ldots & & \ldots \\ J_{L^1}^1 & 0/1 & 0/1 & \ldots & 0/1 \\ J_1^2 & 0/1 & 0/1 & \ldots & 0/1 \\ \ldots & \ldots & \ldots & & \ldots \\ J_{L^2}^2 & 0/1 & 0/1 & \ldots & 0/1 \\ \ldots & \ldots & \ldots & & \ldots \\ J_1^m & 0/1 & 0/1 & \ldots & 0/1 \\ \ldots & \ldots & \ldots & & \ldots \\ J_{L^m}^m & 0/1 & 0/1 & \ldots & 0/1 \end{array}$$

In case job i is performed by resource j, matrix element (i, j) is equal to 1, otherwise is zero. It should be noted that, according to definition of FMRF class of systems, some operation in the system could be executed by several resources, hence, multiple ones would appear in corresponding row of MJI matrix. On the other hand, column comprising multiple entries of '1', represents shared resource. If one observes an MRF system, its machine-job matrix will have exactly one '1' in each row, and possibly multiple '1' in columns. Machine-job incidence matrix can be defined separately for each part type in an FMS.

2 Recursive system equations from MJI matrix

2.1 Introduction to modeling of FMRF systems

The system model should provide the insight into states of system jobs and resources. In other words, given the recursive system model and its initial state, the user should, in each discrete event iteration number k, be able to know which jobs are inactive, completed, which resource are available and which are not.

The basic property of FMRF systems is that these systems include jobs that can be performed by more than one resource from the resource pool. The main idea when developing its recursive model is the following: if there is a well-defined job routing strategy, in each step k, it is uniquely decided which resource will be assigned to each job, hence, in each step k system can be seen as an MRF system. Since the resource assigned to a certain job changes during the work of the system, for each step k the FMRF systems will be represented with structurally distinct MRF models.

This method will be explained by the example that follows. Let us consider a system with the following MJI matrix:

$$\Lambda = \begin{array}{c|cccc} & R_1 & R_2 & R_3 & R_4 \\ J_1^1 & 1 & 1 & 0 & 0 \\ J_2^1 & 0 & 0 & 0 & 1 \\ J_3^1 & 0 & 1 & 1 & 0 \end{array}$$

The system embraces three jobs J_1, J_2 and J_3 and four resources R_1, R_2, R_3 and R_4. Job J_1 can be executed by resources R_1 and R_2. Job J_2 is performed by resource R_4 and job J_3 by resources R_2 or R_3. Let us, for instance, define a control strategy such that jobs J_1 and J_3 are performed alternately by assigned resources, i.e. resource for job J_1 is assigned in the following order: (R_1, R_2, R_1, R_2…) and for job J_3: (R_3, R_2, R_3, R_2…). Since the first resources to perform considered jobs are R_1 and R_3, the MJI matrix of MRF substitute model $\mathbf{\Lambda}_s$ in the initial state is:

$$\Lambda_s = \begin{array}{c|cccc} & R_1 & R_2 & R_3 & R_4 \\ J_1^1 & 1 & 0 & 0 & 0 \\ J_2^1 & 0 & 0 & 0 & 1 \\ J_3^1 & 0 & 0 & 1 & 0 \end{array}$$

When one of the jobs J_1 and J_3 is started, the resource assigned to this job alternates, thus, the $\mathbf{\Lambda}_s$ matrix changes to:

a) job J_1 starts – next resource assigned to it is R_2

$$\Lambda_s = \begin{array}{c|cccc} & R_1 & R_2 & R_3 & R_4 \\ J_1^1 & 0 & 1 & 0 & 0 \\ J_2^1 & 0 & 0 & 0 & 1 \\ J_3^1 & 0 & 0 & 1 & 0 \end{array}$$

b) job J_3 starts – next resource assigned to it is R_2

$$\Lambda_s = \begin{array}{c|cccc} & R_1 & R_2 & R_3 & R_4 \\ J_1^1 & 1 & 0 & 0 & 0 \\ J_2^1 & 0 & 0 & 0 & 0 \\ J_3^1 & 0 & 1 & 0 & 0 \end{array}$$

c) both jobs J_1 and J_3 start – next resource assigned to them is R_2

$$\Lambda_s = \begin{array}{c|cccc} & R_1 & R_2 & R_3 & R_4 \\ J_1^1 & 0 & 1 & 0 & 0 \\ J_2^1 & 0 & 0 & 0 & 1 \\ J_3^1 & 0 & 1 & 0 & 0 \end{array}$$

SNE 20/3-4, December 2010

As we can see from the example, matrices that describe the system are MRF system matrices since they contain no multiple '1' in rows.

Given the basic idea behind the development of the FMRF system model, the general recursive procedure is given as:

Algorithm. Input: FMRF system's MJI matrix, Job routing strategy.

In each discrete event iteration step k do:

1. Determine $\mathbf{\Lambda}_s$ – MJI matrix of MRF system substitution, from the system state in step $(k-1)$ and the given routing strategy
2. Calculate the system state in step k based on MRF model ($\mathbf{\Lambda}_s$)

2.2 Basic recursive model of MRF system

The basic recursive model is developed for MRF system with the following properties: each resource can hold maximally one part at a time and there is one sample of each resource in the resource pool. Hence, each job or resource, seen as a place in Petri-net formalism, can contain maximally one token. Further, a system is autonomous, thus, a part can enter the system each time the resource assigned to the input operation is available. In other words, input buffers always contain parts waiting to be processed.

First, we define the following notation for the rest of the chapter: $\boldsymbol{x}(k)$ denotes the value of vector $\boldsymbol{x}$ in discrete step k, while $x_i(k)$ denotes the value of the i-th element of the vector x in discrete step k. The system MJI matrix is denoted as $\mathbf{\Lambda}_s$.

Definition 1 (completed jobs vector)
The *completed jobs vector* $\boldsymbol{v}$, is a column vector with dimension equal to the number of jobs in the system. $v_i = 1$ if i-th job is completed, otherwise $v_i = 0$.

Definition 2 (idle resource vector)
The *idle resource vector* $\boldsymbol{r}$, is a column vector with dimension equal to the number of resources in the system. $r_i = 1$ if the i-th resource is available, otherwise $r_i = 0$.

The state of the system is completely described by the values of completed jobs and idle resources vector. The relation between vector $\boldsymbol{v}$ and vector $\boldsymbol{r}$ is:

$$\boldsymbol{r} = \overline{(\mathbf{\Lambda}_s)^T \Delta \boldsymbol{v}} \tag{1}$$

Besides these two vectors, we define an auxiliary vector $\boldsymbol{r}_v$ with dimension equal to the number of jobs, whose i-th element, $\boldsymbol{r}_{vi}$, is equal to '1' if resource assigned to job i is available. Vector $\boldsymbol{r}_v$ is determined as follows:

$$\boldsymbol{r}_v = \mathbf{\Lambda}_s \Delta \boldsymbol{r} = \mathbf{\Lambda}_s \Delta \overline{(\mathbf{\Lambda}_s)^T \Delta \boldsymbol{v}} \tag{2}$$

The operations on matrices are defined in an *and*/*or* algebra, denoted Δ and ∇, where standard multiplication is replaced by logical *and* and standard addition by logical *or*. Given a natural number a, its negation $\bar{a}$ is such that $\bar{a} = 0$ if $a > 0$, otherwise $\bar{a} = 1$.

In general, the completed jobs vector $\boldsymbol{v}$ is determined as follows:

$$\boldsymbol{v}(k) = \boldsymbol{v}(k-1) + \boldsymbol{d}(k) \tag{3}$$

Given a certain job i, $d_i(k)$ equals the difference between the number of parts the job i starts processing $d_i^+(k)$ and the number of parts released by the job i in discrete event iteration step k, denoted by $d_i^-(k)$:

$$d_i(k) = d_i^+(k) - d_i^-(k) \tag{4}$$

The job i can start processing a part in step k if the previous job in line, $(i-1)$-th job, is completed and if the assigned resource is available:

$$d_i^+(k) = v_{i-1}(k-1) \cdot r_i(k-1) \tag{5}$$

The job i can release a part it holds in step k, if it is completed and if the resource assigned to the next job in line, $(i+1)$-th job, is available:

$$d_i^-(k) = v_i(k-1) \cdot r_{i+1}(k-1) \tag{6}$$

Definition 3 (vector shift)
Let $\boldsymbol{a}$ be a vector in $\Re^m$, $\boldsymbol{a} \in \Re^m$. Vector $\boldsymbol{b} \in \Re^m$, which is a result of upwards $(m = 1)$ or downwards $(m = -1)$ vector shift operation, denoted $\boldsymbol{b} = \uparrow_m \boldsymbol{a}$, is calculated as:

$$b_j = \begin{cases} 1 & (m = 1 \wedge j = n) \vee (m = -1 \wedge j = 1) \\ a_{j+m} & \text{otherwise} \end{cases}$$

Using vector shift operation, the overall change in value of completed jobs vector $\boldsymbol{v}$ can then be written as:

$$\begin{aligned} \boldsymbol{d}(k) &= \boldsymbol{d}^+(k) - \boldsymbol{d}^-(k) = \\ &= \left(\uparrow_{-1} \boldsymbol{v}(k-1)\right) \cdot \boldsymbol{r}_v(k-1) - \\ &\quad - \boldsymbol{v}(k-1) \cdot \left(\uparrow_1 \boldsymbol{r}_v(k-1)\right) \end{aligned} \tag{7}$$

A recursive mathematical model is obtained by combining equations (1 – 7) and it can be written in the following form

$$\begin{aligned}\boldsymbol{v}(k) &= \boldsymbol{v}(k-1) + \left(\uparrow_{-1} \boldsymbol{v}(k-1)\right)\boldsymbol{r}_{\mathrm{v}}(k-1) - \\ &\quad - \boldsymbol{v}(k-1)\cdot\left(\uparrow_{1} \boldsymbol{r}_{\mathrm{v}}(k-1)\right)\\ \boldsymbol{r}(k) &= \overline{(\boldsymbol{\Lambda}_{\mathrm{s}})^{\mathrm{T}} \,\Delta\, \boldsymbol{v}(k)}\\ \boldsymbol{r}_{\mathrm{v}}(k) &= \boldsymbol{\Lambda}_{\mathrm{s}} \,\Delta\, \boldsymbol{r}(k)\end{aligned} \tag{8}$$

where "·" denotes element-wise product of two vectors.

2.3 Recursive model of *k*-limited MRF systems

In general, MRF systems include resources that can hold more than one part simultaneously. These systems are called *k-limited* systems, where k denotes the maximum number of parts a single resource can hold at a time. The previously considered systems were 1-limited.

For k-limited systems, elements of vectors $\boldsymbol{v}$ and $\boldsymbol{r}$ can obtain values from zero to the maximum number of parts that the corresponding resource can hold – k. That is, if $k > 1$, completed jobs and idle resources vectors comprise integer values unlike binary values for $k = 1$.

If the resource i does not perform any job, the value of r_i corresponds to the maximum number of parts that resource i can hold. Each time a resource i starts processing a part, the value of r_i is decremented until it reaches zero, i.e. for $r_i = 0$ the resource is not available. We can see that the same reasoning was made for 1-limited systems. We introduce a new vector, $\boldsymbol{r}_{\mathrm{cap}}$, with dimension equal to the number of resources, with $(r_{\mathrm{cap}})_i$ equal to the maximal number of parts resource i can hold.

Further, although vectors $\boldsymbol{v}$ and $\boldsymbol{r}$ can obtain integer values for k-limited systems, vectors $\boldsymbol{d}^{+}(k)$ and $\boldsymbol{d}^{-}(k)$ should nevertheless be binary vectors. One can obtain a binary (0,1) value that corresponds to an integer value (0,1,2, …) by doing a double negation on integer number. Thus, the model that describes k-limited systems is structurally the same as model (8), with vectors $\boldsymbol{d}^{+}(k)$ and $\boldsymbol{d}^{-}(k)$ double negated. The model can be written as:

$$\begin{aligned}\boldsymbol{v}(k) &= \boldsymbol{v}(k-1) + \overline{\overline{\left(\uparrow_{-1} \boldsymbol{v}(k-1)\right)\cdot \boldsymbol{r}_{\mathrm{v}}(k-1)}} - \\ &\quad - \overline{\overline{\boldsymbol{v}(k-1)\cdot\left(\uparrow_{1} \boldsymbol{r}_{\mathrm{v}}(k-1)\right)}}\\ \boldsymbol{r}(k) &= \boldsymbol{r}_{\mathrm{cap}} - (\boldsymbol{\Lambda}_{\mathrm{s}})^{\mathrm{T}} \,\Delta\, \boldsymbol{v}(k)\\ \boldsymbol{r}_{\mathrm{v}}(k) &= \boldsymbol{\Lambda}_{\mathrm{s}} \,\Delta\, \boldsymbol{r}(k)\end{aligned} \tag{9}$$

2.4 Recursive model of MRF system with more than one flowline

The models introduced so far are valid for systems with a single flowline. If the system embraces more than one flowline, the previous models should be modified. First, to construct an MJI matrix of such system, we separate different flowlines with zero rows. This is convenient since in (F)MRF systems, at least one resource is assigned to each job, hence, zero rows cannot appear in any other place in matrix MJI. MJI matrix and vectors $\boldsymbol{v}$ and $\boldsymbol{r}$ are then constructed as follows:

$$\boldsymbol{\Lambda}_{\mathrm{s}} = \begin{bmatrix}\boldsymbol{\Lambda}_{\mathrm{s}1}\\ \mathbf{0}\\ \boldsymbol{\Lambda}_{\mathrm{s}2}\\ \mathbf{0}\\ \vdots\\ \mathbf{0}\\ \boldsymbol{\Lambda}_{\mathrm{s}n}\end{bmatrix},\quad \boldsymbol{v} = \begin{bmatrix}\boldsymbol{v}_1\\ \mathbf{0}\\ \boldsymbol{v}_2\\ \mathbf{0}\\ \vdots\\ \mathbf{0}\\ \boldsymbol{v}_n\end{bmatrix},\quad \boldsymbol{r} = \begin{bmatrix}\boldsymbol{r}_1\\ \mathbf{0}\\ \boldsymbol{r}_2\\ \mathbf{0}\\ \vdots\\ \mathbf{0}\\ \boldsymbol{r}_n\end{bmatrix}$$

If vectors $\boldsymbol{v}$ and $\boldsymbol{r}$ would be included in the previously given models, the results would be inaccurate due to inserted zeros. To neutralize the influence of these zeros in vectors $\boldsymbol{v}$ and $\boldsymbol{r}$ we introduce an auxiliary vector, $\boldsymbol{u}_{\mathrm{aux}}$:

$$\boldsymbol{u}_{\mathrm{aux}} = \boldsymbol{\Lambda}_{\mathrm{s}} \,\Delta\, \boldsymbol{I}_{m,1} \tag{10}$$

where $\boldsymbol{I}_{m,1}$ is a column vector with dimension m that is filled with '1's. According to equation (10), auxiliary vector $\boldsymbol{u}_{\mathrm{aux}}$ element is zero if it corresponds to zero row in MJI, otherwise it is one. If we denote with "∨" element-wise logical or operation, the recursive model for systems with more than one flowline can be written as:

$$\begin{aligned}\boldsymbol{u}_{\mathrm{aux}} &= \boldsymbol{\Lambda}_{\mathrm{s}} \,\Delta\, \boldsymbol{I}_{m,1}\\ \boldsymbol{v}(k) &= \boldsymbol{v}(k-1) + \\ &\quad + \left(\uparrow_{-1} \boldsymbol{v}(k-1) \vee \overline{\boldsymbol{u}}_{\mathrm{aux}}\right)\cdot \boldsymbol{r}_{\mathrm{v}}(k-1) - \\ &\quad - \boldsymbol{v}(k-1)\cdot\left(\uparrow_{1} \boldsymbol{r}_{\mathrm{v}}(k-1) \vee \overline{\boldsymbol{u}}_{\mathrm{aux}}\right)\\ \boldsymbol{r}(k) &= \overline{(\boldsymbol{\Lambda}_{\mathrm{s}})^{\mathrm{T}} \,\Delta\, \boldsymbol{v}(k)}\\ \boldsymbol{r}_{\mathrm{v}}(k) &= \boldsymbol{\Lambda}_{\mathrm{s}} \,\Delta\, \boldsymbol{r}(k)\end{aligned} \tag{11}$$

From equation (10) an attentive reader can conclude that auxiliary vector enables correct implementation of parts input and output jobs.

It should be noted that vector shift operation (Definition 3) need to be performed separately on each subvector, i.e.

$$\uparrow_m \boldsymbol{v} = \begin{bmatrix}\uparrow_m \boldsymbol{v}_1\\ \mathbf{0}\\ \uparrow_m \boldsymbol{v}_2\\ \mathbf{0}\\ \vdots\\ \mathbf{0}\\ \uparrow_m \boldsymbol{v}_n\end{bmatrix}.$$

2.5 Conflict resolution in recursive model of MRF system

In case two or more jobs, assigned to a single resource, can be started in the same step k, the system designer should define which of them should be performed. This kind of situation is called a conflict. From the mathematical point of view, conflict develops when more than one element of vector $\boldsymbol{v}^+$ that corresponds to a single resource is equal to '1'. Conflict is solved, depending on the resource dispatching strategy, in such a way that jobs of lower priority are forbidden i.e. the corresponding elements of $\boldsymbol{d}^+(k)$ are set to zero. The mathematical approach to conflict resolution is analogous to the one described in [11].

Let us define the following:

Definition 4 (conflict jobs vector)
The *conflict jobs vector* $\boldsymbol{v}_\mathrm{d}$ comprises information on the jobs performed by shared resources. $v_{\mathrm{d}i} = 1$ if job i is done by a shared resource, otherwise $v_{\mathrm{d}i} = 0$. The vector dimension equals the total number of jobs in the system. If a shared resource vector is denoted as $\boldsymbol{r}_\mathrm{s}$ ($r_{\mathrm{s}i} = 1$ if i-th resource is shared, otherwise $r_{\mathrm{s}i} = 0$), vector $\boldsymbol{v}_\mathrm{d}$ can be determined as follows:

$$\boldsymbol{v}_\mathrm{d} = \boldsymbol{\Lambda} \,\Delta\, \boldsymbol{r}_\mathrm{s} \tag{12}$$

Definition 5 (dispatching matrix)
The *dispatching matrix* $\boldsymbol{F}_\mathrm{d}$ is determined from the conflict jobs vector $\boldsymbol{v}_\mathrm{d}$ as follows:

$$f_{\mathrm{d}(i,j)} = \begin{cases} 1 & v_{\mathrm{d}i} = 1 \text{ and } j = \sum_{k=1}^{i} u_{\mathrm{d}k} \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

Definition 6 (dispatching vector)
The dispatching vector $\boldsymbol{u}_\mathrm{d}$ is a column vector with dimension equal to the number of conflict jobs in the system. If job i is of the highest priority among conflict jobs, $u_{\mathrm{d}i} = 1$, otherwise $u_{\mathrm{d}i} = 0$. The priorities of jobs depend on the applied conflict resolution strategy.

As we said earlier, in case of the conflict, the jobs of lower priority should be forbidden, hence, corresponding element of vector $\boldsymbol{d}^+(k)$ should be set to zero. Recursive matrix model for the system with conflict resolution is:

$$\begin{aligned} \boldsymbol{v}(k) = \boldsymbol{v}(k-1) + \\ + \left(\uparrow_{-1} \boldsymbol{v}(k-1)\right) \cdot \boldsymbol{r}_\mathrm{v}(k-1) \cdot \overline{\boldsymbol{F}_\mathrm{d} \,\Delta\, \overline{\boldsymbol{u}}_\mathrm{d}} - \\ - \boldsymbol{v}(k-1) \cdot \left(\uparrow_1 \boldsymbol{r}_\mathrm{v}(k-1)\right) \\ \boldsymbol{r}(k) = \overline{(\boldsymbol{\Lambda}_\mathrm{s})^\mathrm{T} \,\Delta\, \boldsymbol{v}(k)} \\ \boldsymbol{r}_\mathrm{v}(k) = \boldsymbol{\Lambda}_\mathrm{s} \,\Delta\, \boldsymbol{r}(k) \end{aligned} \tag{14}$$

The model in form of (14) is suited only for 1-limited systems.

To define a conflict resolution for k-limited systems, the element $\overline{\boldsymbol{F}_\mathrm{d} \,\Delta\, \overline{\boldsymbol{u}}_\mathrm{d}}$ needs to be included in the expression for $\boldsymbol{d}^+(k)$. Conflict resolution model for k-limited systems is then given as:

$$\begin{aligned} \boldsymbol{v}(k) = \boldsymbol{v}(k-1) + \\ + \overline{\left(\uparrow_{-1} \boldsymbol{v}(k-1)\right) \cdot \boldsymbol{r}_\mathrm{v}(k-1) \cdot \overline{\boldsymbol{F}_\mathrm{d} \,\Delta\, \overline{\boldsymbol{u}}_\mathrm{d}}} - \\ - \overline{\boldsymbol{v}(k-1) \cdot \left(\uparrow_1 \boldsymbol{r}_\mathrm{v}(k-1)\right)} \\ \boldsymbol{r}(k) = \boldsymbol{r}_\mathrm{cap} - (\boldsymbol{\Lambda}_\mathrm{s})^\mathrm{T} \,\Delta\, \boldsymbol{v}(k) \\ \boldsymbol{r}_\mathrm{v}(k) = \boldsymbol{\Lambda}_\mathrm{s} \,\Delta\, \boldsymbol{r}(k) \end{aligned} \tag{15}$$

Since vector $\boldsymbol{r}_\mathrm{cap}$ is filled with ones for 1-limited systems, model (15) is suitable both for 1-limited and k-limited systems with $k > 1$. If one wants to apply the same procedure for k-limited systems with more than one flowline, the auxiliary vector $\boldsymbol{u}_\mathrm{aux}$ should be included in model (15) as well:

$$\begin{aligned} \boldsymbol{u}_\mathrm{aux} = \boldsymbol{\Lambda}_\mathrm{s} \,\Delta\, \boldsymbol{I}_{m,1} \\ \boldsymbol{v}(k) = \boldsymbol{v}(k-1) + \\ + \overline{\left(\uparrow_{-1} \boldsymbol{v}(k-1) \vee \overline{\boldsymbol{u}}_\mathrm{aux}\right) \cdot \boldsymbol{r}_\mathrm{v}(k-1) \cdot \overline{\boldsymbol{F}_\mathrm{d} \,\Delta\, \overline{\boldsymbol{u}}_\mathrm{d}}} \\ - \overline{\boldsymbol{v}(k-1) \cdot \left(\uparrow_1 \boldsymbol{r}_\mathrm{v}(k-1) \vee \overline{\boldsymbol{u}}_\mathrm{aux}\right)} \\ \boldsymbol{r}(k) = \boldsymbol{r}_\mathrm{cap} - (\boldsymbol{\Lambda}_\mathrm{s})^\mathrm{T} \,\Delta\, \boldsymbol{v}(k) \\ \boldsymbol{r}_\mathrm{v}(k) = \boldsymbol{\Lambda}_\mathrm{s} \,\Delta\, \boldsymbol{r}(k) \end{aligned} \tag{16}$$

The model (16), with properly determined associated vectors $\boldsymbol{u}_\mathrm{d}$, $\boldsymbol{r}_\mathrm{cap}$, $\boldsymbol{u}_\mathrm{aux}$, comprises all previously considered system models: (8), (9), (11) and (15) and is therefore the most suitable for implementation.

2.6 Recursive model of FMRF system

As we stated earlier, the idea behind determination of the model of FMRF system is that, having a dispatching strategy, FMRF system can be represented as corresponding MRF system in each discrete event iteration step k.

The MJI matrix $\boldsymbol{\Lambda}_\mathrm{s}$, which is an MRF substitute of the FMRF system in step k, depends on the applied strategy.

Definition 7 (MRF substitute MJI matrix)
The MRF substitute MJI matrix $\mathbf{\Lambda}_s \in \Re^{m\times n}$ is a matrix with dimensions equal to dimensions of FRMF system MJI matrix $\mathbf{\Lambda}$. $\mathbf{\Lambda}_s$ is obtained as follows:

$$\Lambda_s(i,j) = \quad (17)$$
$$= \begin{cases} 0 & \Lambda(i,j) = 0 \\ 1 & \Lambda(i,j) = 1 \text{ and } \sum_{k=1}^{n} \Lambda(i,k) = 1 \\ u_l \in \{0,1\} & \Lambda(i,j) = 1 \text{ and } \sum_{k=1}^{n} \Lambda(i,k) \geq 2 \end{cases}$$

As we can see, the matrix $\mathbf{\Lambda}_s$ is a structural copy of matrix $\mathbf{\Lambda}$, where each '1' in rows that contain more than one '1' is substituted with corresponding control variable u_l. If we return to the example from the beginning, matrix $\mathbf{\Lambda}_s$ is determined as a function of control variables as:

$$\Lambda = \begin{array}{c|cccc} & R_1 & R_2 & R_3 & R_4 \\ J_1^1 & \mathbf{u_1} & \mathbf{u_2} & \mathbf{0} & \mathbf{0} \\ J_2^1 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} \\ J_3^1 & \mathbf{0} & \mathbf{u_3} & \mathbf{u_4} & \mathbf{0} \end{array}$$

Control variables have binary values depending on whether or not the corresponding resource performs the job. The values of control variables should be such that in every step k exactly one variable in the row is equal to '1'.

3 System simulator design

Based on the formulas presented in this paper the application called *MJIWorkshop*, which simulates the FMRF systems is developed. The simulator's main window is shown in Figure 1. The system is initially given by its MJI matrix and initial state and for each step k a new state value is determined. The state of the system is visually represented in form of a token matrix shown in Figure 2. The token matrix is structurally the same as the MJI matrix. The matrix element (i,j) contains a token in step k if job i is executed by resource j in step k. The token matrix is, as well as Petri nets, convenient for visualizing the discrete event systems.

Developed system simulator is standard Multiple-document Microsoft Foundation Class (MFC) application created with Microsoft Visual Studio 6.0 [12] and is written in C++. A reason to use MFC instead of .NET is an intention to get a program that is able to run without a problem even on older PCs and further, to make application portable, i.e. able to run from USB stick without need to install it. Internal structure of program is more or less similar to other multiple document programs created with Visual Studio wizard.

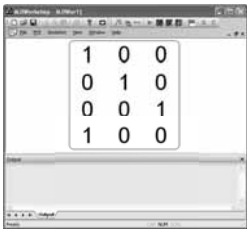

Figure 1. MJIWorkshop input window

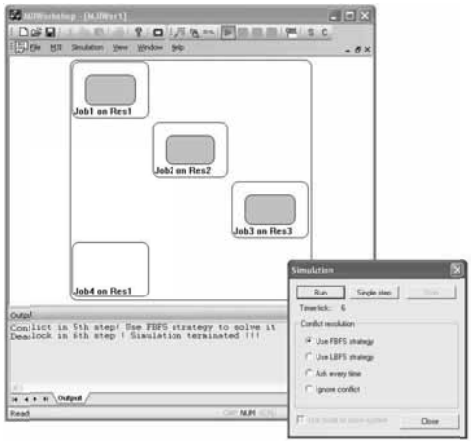

Figure 2. Visualizing system state in *MJIWorkshop* including conflict resolution window

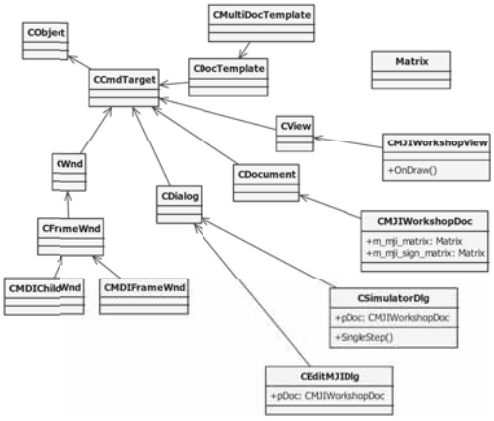

Figure 3. Simplified UML class diagram

It is standard document-view architecture software, which is mainly based on MFC classes `CDocument` and `CView`. Main class, which holds all necessary data, is `CMJIWorkshopDoc`.

This class, using its methods, performs editing and simulation by creating and running two other objects: `CEditMJIDlg` and `CSimulateDlg`. Class `CMJIWorkshopView` takes care of visualization.

Its object, using associated document, takes all necessary data to display the current system state on the screen, as shown in Figure 4. Unified Modeling Language (UML) class diagram, which shows main relationships between classes is given in Figure 3.

4 Conclusion and future work

Developed simulator is based on the model that consists of a single matrix: the machine-job incidence matrix. The MJI matrix comprises all information needed to fully model an FMRF system. Recursive models are given separately for basic types of systems, while the developed simulator encloses all of these models and implements the overall model of *k*-limited FMRF systems with more than one flowline, together with the job and conflict resolution. The current state of the system is obtained by performing various simple matrix operations, thus, the recursive model procedure is easy to implement and non-time-consuming. Matrix dimensions depend on the size and complexity of the analyzed system. The simulator is convenient for system analysis in design stage since it allows the user to apply various job and resource dispatching strategies on-line while simulating the system. In the future, operating times are to be included in the application together with the support for system performance analysis. Also, the connection with other simulating tools, such as Matlab and Petri.NET [16] is to be developed.

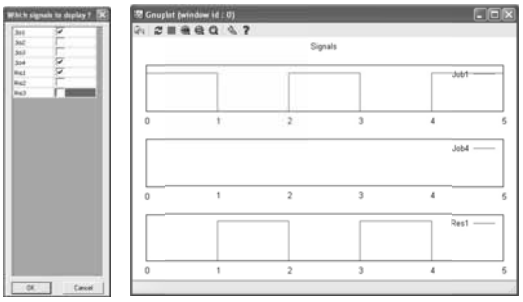

Figure 4. Signal visualization with GnuPlot

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] Gertz M W, Khosla P K. Onika: *A Multilevel Human-Machine Interface for Real-Time Sensor-Based Robotics Systems*, Proc. of SPACE 94: 4th Int. Conf. and Exposition on Engineering and Construction, 1994.

[5] Nethery J, Spong M W. Robotica: A Mathematica Package for Robot Analysis, IEEE Rob. Aut. Mag. 1994; 1: 1: 13–20.

[6] 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.

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

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

[9] Kusiak A., Ahn J. *Intelligent Scheduling of Automated Machining Systems*, Computer-Integrated Manufacturing Systems, 1992; 5; 1: 3-14.

[10] Steward, D.V. *The Design Structure System: A Method for Managing the Design of Complex Systems*, IEEE Transactions on Engineering Management, 1981; 28: 71-74.

[11] Bogdan S, Lewis FL, Mireles J, Kovacic Z. *Manufacturing Systems Control Design: a matrix based approach*, Springer, 2006.

[12] Visual Studio 6.0, [webpage] (2008, November 26) online: http://msdn.microsoft.com/enus/library/ms950417.aspx

[13] GnuPlot homepage, [webpage] (2008, November 3) Available at: *http://www.gnuplot.info/*

[14] PNML View - Cross platform viewer for PNML files, [webpage], (2008, October 15) Available online: *http://www.vanwal.nl/pnmlview/*

[15] Matrix C++ - Template class library with full source code, [webpage], (2008, October 2) Available online: *http://www.techsoftpl.com/matrix/*

[16] Genter G., Bogdan S., Kovacic Z., Grubisic I.: *Software tool for modeling, simulation and real-time implementation of Petri net-based supervisors*, Control Applications, 2007. CCA 2007. IEEE International Conference on , vol., no., pp.664-669, 1-3 Oct. 2007

Corresponding author: Stjepan Bogdan,
University of Zagreb, Faculty of EE&C,
Dept. of Control and Computer Engineering,
Laboratory f Robotics and Intelligent Control Systems
Unska 3, 10000, Zagreb, Croatia
stjepan.bogdan@fer.hr

Received & Accepted: MATHMOD 2009: -
Revised: September 10, 2009 –
Accepted: July 10, 2010 –