A New Matrix Model for Discrete Event Systems: Application to Simulation

Diego A. Tacconi and Frank L. Lewis

Simulation schemes for discrete event (DE) systems based on a new DE matrix formulation are presented. This new formulation is a hybrid system with logical and algebraic components that allows fast, direct design and reconfiguration of rule-based controllers for manufacturing systems. It applies for general DE systems that include shared resources, dispatching, circular waits, and variable part routing. A certain DE matrix state equation together with the familiar Petri net marking transition equation yield a complete dynamical description of a DE system. Our goal in this article is to show that this provides a simplified computer tool that allows efficient simulation and modeling for DE systems.

Introduction

As in many engineering fields, the design, simulation, and analysis of discrete event (DE) systems can be carried out using mathematical models. There are many approaches to modeling, simulation, and controls design for DE systems [8], including Petri nets [12, 17, 22, 31, 32, 34], alphabet-based approaches [36], perturbation methods [6, 19, 20], control theoretic techniques [25, 28, 29], expert systems design [9, 24], and so on. Though these techniques allow the development of models, it is often difficult to apply them to complex practical systems, large-scale interconnected systems, or systems characterized by standard industrial techniques including the bill of materials (BOM). In this article we are presenting a simplified computer software tool that allows efficient simulation and modeling for DE systems. This tool is based on a new matrix-based formulation for discrete event systems that affords radically new techniques for modeling, analysis, simulation, and control of flexible manufacturing systems (FMS) [21, 27]. The matrix formulation is a hybrid system [1, 2, 48] with logical and algebraic components that applies for general DE systems with shared resources, dispatching, and variable part routings. The key is to separate the functions of the FMS plant from those of the FMS controller. The matrices come from standard industrial engineering data structure techniques, including the BOM, assembly tree, and resource requirements matrix, and are straightforward to write down for large-scale interconnected manufacturing systems using notions of block matrices. Although the simulations presented all involve some sort of manufacturing system, the formulation is also applicable for other DE systems, including autonomous guided vehicles (AGV), communication networks, and computer operating systems.

The authors are with the Automation and Robotics Research Institute, The University of Texas at Arlington, 7300 Jack Newell Blvd., S. Fort Worth, TX 76118-7115, telephone 817-272-5972, fax 817-272-5989, dtacconi@arri.uta.edu.

Overview of Discrete Event System Simulation

Since analytical results are often difficult to obtain, particularly for transient analysis, the performance of FMS, including scheduling and dispatching rules and other algorithms, has often been studied using simulation [3, 10, 23, 30, 38]. There are many approaches to simulation of discrete event systems. However, modeling actual manufacturing systems for simulation is complicated, with program-specific languages being required. There are available many packages for simulation of manufacturing systems (SIMFACTORY II.5, Gert [7], etc.), Petri nets (Design/CPN, Grafcet [11], TORA, etc.), queueing systems ([5]), and general DE systems (SIMAN, Simscript II.5, Simula, Smalltalk-80, GPSS, Extend, Slam [35]). Object-oriented techniques are used in [4], knowledge-based approaches in [39], and Prolog in [33]. A system theory approach, TCT [47], for simulation of supervisory controllers with subsystem analysis is now available; this package models a DE system using a rule-based logical component plus an algebraic component. The large number of techniques available show the complications arising from simulation of DE systems. Many of these tools use brute force approaches that do not take advantage of the protocol structures of manufacturing flow lines, assembly lines, and job shop systems.

Manufacturing Systems

There are several standard structures of manufacturing systems, including the reentrant flow line, the assembly line, and the job shop. In a flexible manufacturing system (FMS), the controller should be distinguished from the physical portion of the FMS, which is captured in the notion of a manufacturing facility—a set of resources (including the machines, tools, fixtures, robots, transport devices, buffers, etc.), each of which has a distinct function. Each one can denote a pool of more than one machine that performs the same function.

The resources operate on parts; the job sequence for part type is a sequence of jobs required to produce a finished product. The sequence of jobs may be obtained from a task decomposition, assembly tree, bill of material (BOM), etc. [26, 42]. Once the sequence of jobs for a part type has been assigned, resources must be assigned to perform the jobs based on the facilities available. The ordering of the jobs for a given part type can be either fixed or variable; for instance, in an application it may be allowable either to drill then machine a part, or to machine and then drill the

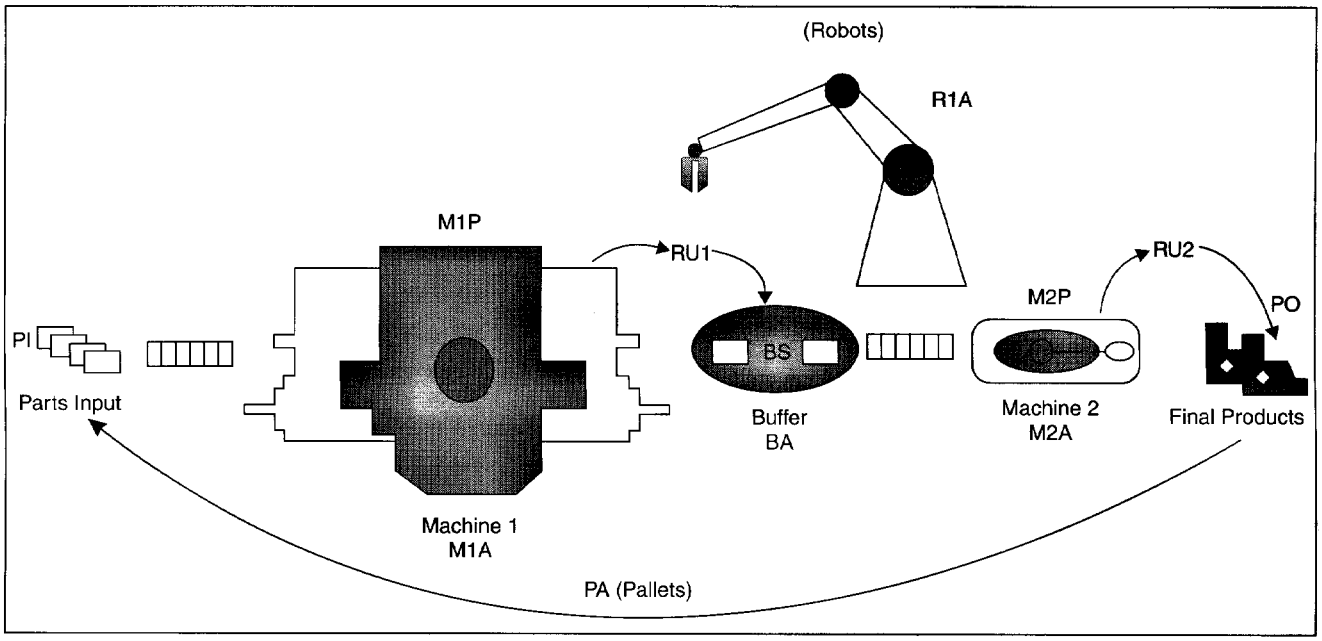


Fig. 1. DEMS workcell.

part. Likewise, the resources assigned to each job can be either fixed or variable. For instance, either of two machines of different types (e.g., from different resource pools) might be capable of performing a given drilling job. In the general job shop, the sequencing of jobs is not fixed, or the assignment of resources to the jobs is not fixed. The effect is that part routing decisions must be made during processing.

In the flow line, the sequence of jobs for each part type is fixed and the assignment of resource pools to the jobs is fixed. The result is that parts of each type visit the resources in the same sequence, though different part types may have different sequences. A flow line is said to be reentrant if any part type revisits the same resource pool more than once in its job sequence [25, 26]. This occurs if the same resource is assigned to different jobs in the part's sequence. The reentrant flow line has also been called the job shop with fixed part routing.

In the reentrant flow line or the job shop, the parts can revisit the same resource pool more than once, and/or the same resource pools may be used to service several parts of different types. Therefore, certain resources are shared, either by parts of the same type at different stages of their processing, or across parts of different types. Thus, one is faced with a decision at each shared resource involving which part to process next. If the buffer sizes are finite, this dispatching decision is a crucial one which can cause severe problems in a manufacturing system if not properly made.

Petri Nets

Event-driven systems are growing in popularity and complexity. This is motivating the use of well-organized design methodologies to avoid failures and to optimize performance. These systems usually have characteristics such as concurrence, conflicts, priorities, mutual exclusions, shared resources, and many others. These properties are difficult to handle; however, the design of these systems, and thus their simulations, can be carried out using Petri nets (PN) [12, 17, 22, 31, 33, 34, 40, 43, 49, 50]. There are many varieties of Petri nets, from binaries, which are simple to analyze, to colored nets, which allow the modeling of more complex systems but have fewer analytic results. A Petri net (PN) is nothing but a bipartite (e.g., having two sorts of nodes) digraph described by $\mathcal{P}, \mathcal{T}, I, O$, where $\mathcal{P}$ is a set of places, $\mathcal{T}$ is a set of transitions, I is a set of (input) arcs from places to transitions, and O is a set of (output) arcs from transitions to places. Considering a PN as a digraph with nodes $\mathcal{V} = \mathcal{P} \cup \mathcal{T}$ and arcs $\mathcal{A} = I \cup O$, one may speak of PN paths, circuits, and elementary circuits. In a well-defined PN, the occurrences of places and transitions alternate along any of these.

In our application, the PN places represent manufacturing resources and jobs, and the transitions represent decisions or rules for resource assignment/release and starting jobs. A standard representation for a reentrant flow line is given in Fig. 1. The PN representation for the same system is shown in Fig. 2, where the places are drawn as circles and the transitions as bars. Note that along the part path, some resources (e.g., Robot) is used more than once. The part path in the figure has a set of pallets denoted by PA; one pallet is needed to hold each part entering the cell. In Fig. 2, places and transitions alternate along the part paths. Places ending in $\mathcal{P}$, all on the job paths, correspond to jobs in progress. Places ending in $\mathcal{A}$ correspond to the availability of resources. Denote the set of jobs for part type j as $\mathcal{J}_j$ and the set of all the jobs as $\mathcal{J} = \cup_j \mathcal{J}_j$. It is noted that the *part input places PI* and *part output places PO* are not included as jobs. Places that occur

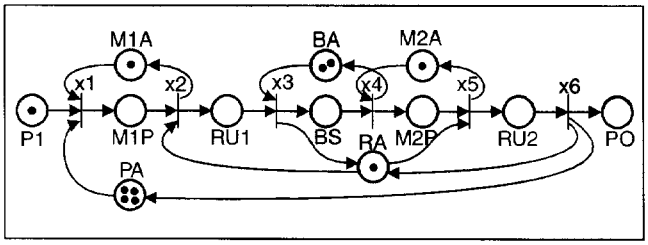


Fig. 2. Petri-net representation.

off the part paths represent the availability of resources; the set of resources $\mathcal{R}$ is the set of all such places. The set of resources may be partitioned as $\mathcal{R} = \mathcal{R}_{ns} \cup \mathcal{R}_s$, with $\mathcal{R}_{ns}$ the nonshared resources and $\mathcal{R}_s$ the shared resources. The set of PN places is given by $\mathcal{P} = \mathcal{R} \cup \mathcal{J}$, the set of resources plus the set of jobs. It is important to note that all transitions occur along the part paths. The tokens (drawn as dots) within the places shown in the figure denote the initial numbers of resources assigned to the resource pools. For instance, the tokens within the place BA in Fig. 2 denote that initially there are two buffer spaces available; the tokens in the place PA indicate initial availability of four pallets for part type 1.

PN are very difficult to design for specific FMS of reasonable complexity, and to modify if objectives, products, or resources change. There is no repeatable design algorithm for PN. Several researchers [12, 17, 22, 37, 49] are confronting such PN deficiencies. "Top-down" and "bottom-up" design algorithms are emerging, and the shared-resource allocation problem is being confronted using concepts such as series mutual exclusion (SME), parallel mutual exclusion (PME), forbidden states, etc. A major problem is that to date there is *no complete mathematically rigorous evolution equation for a PN*, so that properties such as reachability must be verified for each given system using simulation. Moreover, to accommodate manufacturing design algorithms in the PN framework, it is necessary to introduce colored PN, hierarchical PN, generalized PN, multiple types of places, or other esoteric notions that quickly go beyond the experience of the manufacturing engineer and invalidate most PN analysis techniques.

Matrix Model For Discrete Event System

This section describes a new DE matrix description that makes it straightforward to model and simulate DE systems. From the DE matrix representation of a FMS, the PN representation may easily be derived if desired. The DE matrix representation is written down using manufacturing engineering concepts. Some standard manufacturing engineering data structure techniques are very useful for conceptualizing manufacturing processes and providing a limited capability for analysis. The *Bill of Materials (BOM)* can be described as a matrix in which the *(i,j)* entry is equal to the number of subassemblies/parts of type *j* needed to produce one subassembly/part of type *i* [13]. BOM information is an integral part of all MRP systems [44]. The BOM contains similar information as the *assembly tree* [46], which shows the task decomposition of jobs needed to manufacture a product.

Steward's *job sequencing matrix (JSM)* [14, 41, 45], comes directly from the assembly tree [46] or the BOM. In this matrix, the columns and rows correspond to jobs, and an *(i,j)* entry of 1 indicates that job *j* is a prerequisite for job *i*. The job sequencing matrix is very useful for representing the partial orderings needed for sequencing manufacturing jobs; it has been shown that a lower triangular JSM corresponds to a causal ordering of jobs, and that information on the hierarchical subsystem structure of a process can be extracted by raising the JSM to various powers.

The resource requirements matrix (RRM) comes directly from the resources available to perform jobs, as reflected, for instance, in the subassembly tree, which is an assembly tree annotated to indicate the resources assigned to the jobs [46]. In this matrix, the columns correspond to resources (tools, fixtures, machines, robots) and rows correspond to jobs; an *(i,j)* entry of 1 indicates that resource *j* is needed for job *i*. As resources change, it is very easy to modify the RRM, and Kusiak has shown that RRM provides the basis for dispatching of shared resources (MDR rule).

The Matrix Discrete Event Model

A new rule-based DE matrix model is now described that allows assembly/job sequencing, then addition of resources, then deadlock analysis and avoidance, and facilitates dispatching/routing design. The model matrices provide a framework for *rigorous analysis* of an FMS, including its structure, circular waits, siphons, and deadlock. The model also allows a very convenient computer simulation of FMS. The DEDS model is based on a matrix formulation (not the max-plus algebra) where each matrix has a well-defined function for job sequencing, resource assignment, and resource release. The matrix discrete event (DE) model is described by the following equations:

Matrix DE Model State Equation:

$$\bar{x} = F_v \bar{v}_c + F_r \bar{r}_c + F_u \bar{u} + F_D \bar{u}_D \ , \tag{1}$$

Start Equation:

$$v_s = S_v x, \tag{2}$$

Resource Release Equation:

$$r_s = S_r x, \tag{3}$$

Product Output Equation:

$$y = S_y x. \tag{4}$$

These are *logical* equations, where addition denotes logical 'or', multiplication denotes logical 'and', and the overbar indicates logical negation. They are very easy to write down for a specific FMS. F_v is the *job sequencing matrix* of Steward it is determined from the BOM (Elsayed and Boucher 1994) or assembly tree (Wolter et al. 1992). Element $F_v(i,j)$ is equal to 1 if job *j* is required as an immediate precursor to job *i* (equiv. in the BOM, if subassembly *j* is required to produce subassembly *i*). F_r is the *resource requirements matrix* of Kusiak (1992), which is assigned by the shop-floor engineer depending on the available manufacturing facilities. It has element $F_r(i,j)$ equal to 1 if resource *j* is required for job *i*. Steward's sequencing matrix F_v and the resource requirements matrix F_r have long been used as heuristic design aids by industrial engineers, with some possibility for limited analysis (as described, e.g., by Warfield (1973) in the case of F_v and Kusiak (1992) in the case of F_r). The logical matrix model elevates these design tools to formal computation elements. The job start matrix S_v and the resource release matrix S_r are *new matrices that must be introduced to obtain a complete matrix description of DE systems*; they are equally direct to write down. Matrix S_v has element $S_v(i,j)$ equal to 1 if job *i* should be started when all the requirements of logical component x_j are satisfied, so that x_j has been set high. In the flow line this matrix has diagonal 1's. In the job shop, it has multiple ones in the same column

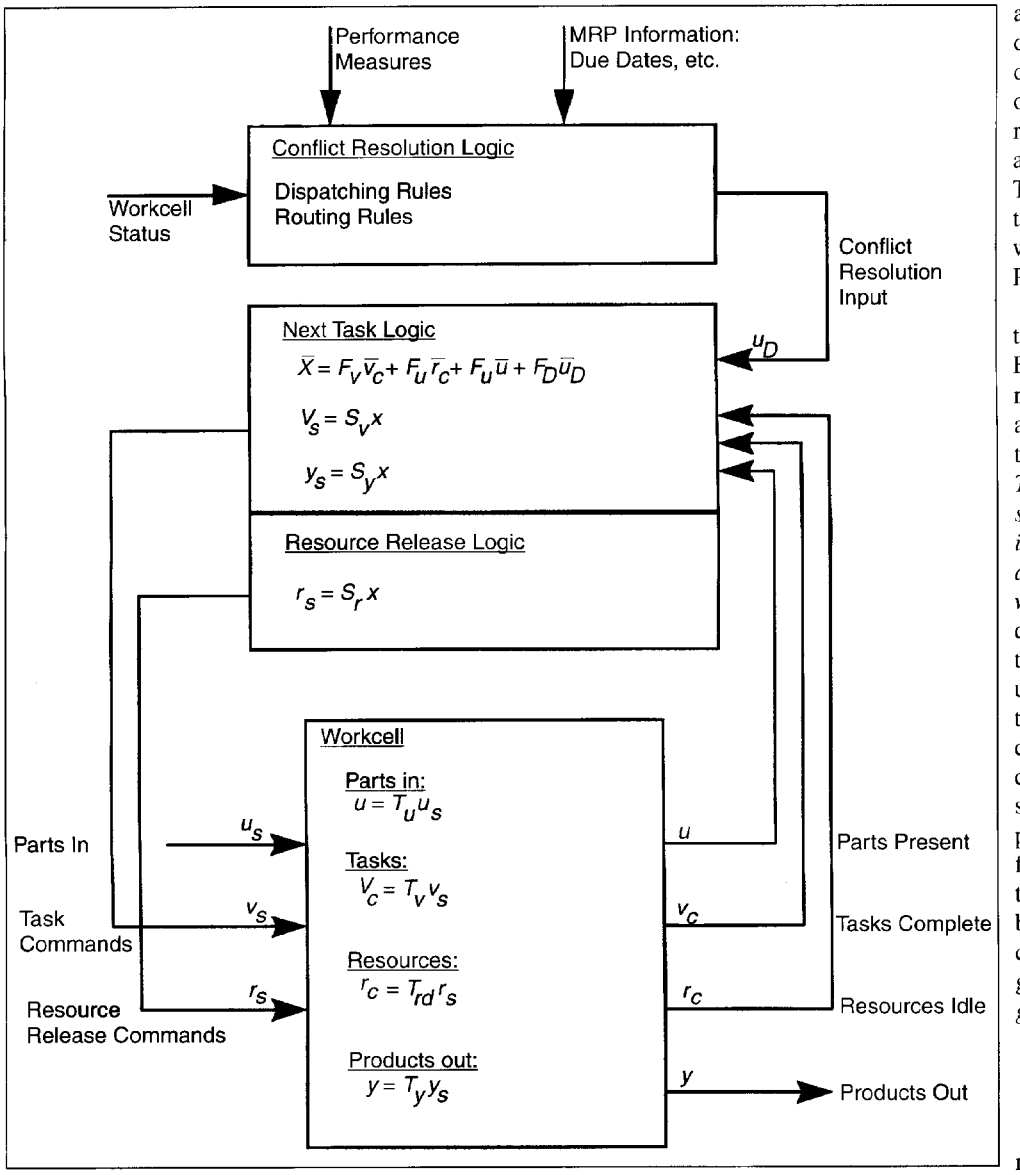


Fig. 3. Matrix-based DE model.

corresponding to job routing decisions. Matrix S_r has element $S_r(i,j)$ equal to 1 if resource i should be released when logical state component x_j is set high. Input u represents raw parts entering the cell and y products leaving the cell.

The DE model, shown in Fig. 3, observes the *status outputs* of the workcell, namely, vector v_c, whose entries of '1' represent completed jobs and vector r_c, whose entries of '1' represent resources currently available. The *DE model state equation* (1), analogous to the matrix differential equation $\dot{x} = Ax + Bu$ in control system theory, checks the conditions required for performing the next jobs in the manufacturing system. Based on these conditions, stored in the logical state vector x, the job start equation (2) computes which jobs are activated and may be started, and the resource release equation (3) computes which resources should be released (due to completed jobs). Then the DE model sends *commands* to the workcell dynamics, namely vector v_s, whose '1' entries denote which jobs are to be started, and vector r_s, whose '1' entries denote which resources are to be released. Products out are given by (4). (Subscript c denotes complete or current, s denotes start). The upper third of Fig. 3 represents shared-resource conflict resolution and dispatching activities. This article focuses on the matrix model (1)-(4) and the workcell, as captured in the PN transition equation (7).

It is easy to show that with the or/and matrix algebra, Equations (1)-(4) become mathematical equations that allow formal computation of the rules in the FMS model. *This allows: (a) computer simulation and (b) computer implementation of the model as a controller on an actual workcell.* The operations required in the DE model equations can easily be carried out using standard real-time control software on a personal computer. It is noted that the coefficient matrices in (1) are sparse, so that real-time computations are very easy even for large manufacturing systems. Moreover, (1) is nothing but a rulebase, so that the rules can be fired using efficient algorithms such as the *Rete algorithm* [16].

PN from Discrete Event Matrices

It is straightforward to derive the PN description of a manufacturing system from the matrix DE model equations (1)-(4) [21, 27]. This allows all the PN analysis tools to be used for DEDS analysis within the matrix DE model framework. It results in a *repeatable design algorithm for Petri nets for DE controllers* that formalizes some work in the literature (e.g. "top-down" and "bottom-up" design [49]). In fact, given the FMS model equations, define the *activity completion matrix F and the activity start matrix S as*

$$F = \begin{bmatrix} F_u & F_v & F_r & F_y \end{bmatrix}, \quad S = \begin{bmatrix} S_u^T & S_v^T & S_r^T & S_y^T \end{bmatrix}. \qquad (5)$$

Define $\mathcal{X}$ as the set of elements of controller state vector x, and $\mathcal{A}$ (activities) as the set of elements of the job and resource vectors v and r. Then $(\mathcal{A}, \mathcal{X}, F, S^T)$ is a Petri net. Here F_y and S_u are introduced to complete the matrices and they are nothing but a vector of zeros. This result identifies F as the input incidence matrix and S^T as the output incidence matrix of a PN, so that the PN incidence matrix is given by

```
% Simple DE simulation
   clear all;
   Fv = [0 0 0 0 0; 1 0 0 0 0; 0 1 0 0 0; 0 0 1 0 0; 0 0 0 1 0; 0 0 0 0 1];
   Fr = [1 1 0 0 0; 0 0 0 0 1; 0 0 0 1 0; 0 0 1 0 0; 0 0 0 0 1; 0 0 0 0 0];
   Sv = [1 0 0 0 0 0; 0 1 0 0 0 0; 0 0 1 0 0 0; 0 0 0 1 0 0; 0 0 0 0 1 0];
   Sr = [0 0 0 0 0 1; 0 1 0 0 0 0; 0 0 0 0 1 0; 0 0 0 1 0 0; 0 0 1 0 0 1];
   Fu = [1 0 0 0 0 0]';
   Fy = [0 0 0 0 0 0]';
   Su = [0 0 0 0 0 0];
   Sy = [0 0 0 0 0 1];
   F = [Fu Fv Fr Fy];
   S = [Su' Sv' Sr' Sy']';
   M = S' - F;

% Initial Conditions
% m(to) = [PI v r Ud PO];
% m(to) = [PI M1P RU1 BS M2P RU2 PA M1A M2A BA RA PO];
   output(1,:) = ['  PI M1P RU1  BS M2P RU2  PA M1A M2A  BA  RA  PO'];
   m = [1 0 0 0 0 0 4 1 1 2 1 0]';
   output(2,:) = sprintf(' %2d ', m);

% Running the simulation
   for i = 3:10;
      x = multoa(not(F), not(not(m)));
      m = m + (M' * x);
      output(i,:) = sprintf(' %2d ', m);
   end

% Displaying the results
   output
```

Fig. 4. MATLAB file "DE1.M".

```
function R = multoa(X,Y)  % Matrix "and/or" multiplication

[nx, mx] = size(X);
[ny, my] = size(Y);
for i = 1:nx
   for j = 1:my
      R(i,j) = ((X(i,1)) | (Y(1,j)));
      for k = 2:mx
         R(i,j) = R(i,j) & ((X(i,k)) | (Y(k,j)));
      end
   end
end
```

Fig. 5. MATLAB file "MULTOA.M".

$$M = S^T - F = \left[S_u^T - F_u \quad S_v^T - F_v \quad S_r^T - F_r \quad S_y^T - F_y\right]. \quad (6)$$

Based on this result, a PN such as the one in Fig. 2 is easily drawn for a system described in matrix form. In this figure, initial markings (m_0) have been added; this is accomplished by determining the number of resources available in the workcell. The result makes it patently clear that *the PN is the closed-loop description of the workcell plus controller.* As such, all attempts to directly draw a PN for a workcell are fraught with danger, as they are equivalent to an attempt to draw the workcell PN and design the dispatching/routing controller all in one step.

Petri Net Marking Transition Equation

According to PN theory, a column vector p indexed by the set of places $\mathcal{P}$ is called the PN p-vector (place vector). The *PN marking vector* is the marking vector $m(p)$, where the marking of p, is the number of tokens in p. Given a vector of places $p = [p1\ p2 \ldots pq]^T$, the marking $m(p)$ is the vector $m(p) = [m(p1)\ m(p2) \ldots m(pq)]^T$ of markings of the individual places. It is common to simplify the notation so that $m(t)$ denotes the marking vector $m(p)$ at time t. In terms of the PN incidence matrix, one can write the PN marking transition equation

$$m(t_2) = m(t_1) + M^T \cdot x = m(t_1) + \left[S^T - F\right]^T \cdot x, \quad (7)$$

```
1.    function Y = not(X)

2.    [nx, mx] = size(X);
3.    for i = 1:nx
4.       for j = 1:mx
5.          if (X(i,j) < 1)
6.             Y(i,j) = 1;
7.          else
8.             Y(i,j) = 0;
9.          end
10.      end
11.   end
```

Fig. 6. MATLAB file "NOT.M".

where $m(t)$ is the PN marking vector at time t, $t_1 < t_2$, and x is a vector denoting which transitions have fired between times t_1 and t_2; element $x_i = n_i$ if the *i-th* transition has fired n_i times in the interval. Unfortunately, this equation is not a complete description of a PN since it does not take into account the order of firing of the transitions, nor whether a given transition can actually fire at any point in time.

Thus, the matrix approach to PN has yielded some valuable insight [34], but has never been extended to provide a complete description of the firing dynamics of a PN. In subsequent sections of this article, this deficiency is corrected. A matrix formulation of discrete event systems is given that, together with the PN marking transition equation (7), provides a complete dynamical description that can be used for analysis and computer simulation.

Complete Dynamical Description of Discrete Event Systems

The matrix formulation provides the rigorous framework needed for the analysis and simulation of discrete event system (DES). Thus the DE model presented here, designed to simulate this sort of system, is based on this framework. Denoting the discrete event iteration number by k, the following equations give the basic description of a DES.

$$m_{k+1} = m_k + M^T \cdot x_k \,. \tag{8}$$

Equation (8) is the Petri-net marking transition equation as introduced in the previous section, where m_k is the marking vector and M is the incidence matrix [1]. This equation attempts to provide a complete dynamical description of a PN. However, this equation alone cannot be used without an allowable firing vector x_k. Assuming that this firing vector is given, then using Equation (8) we can determine the next marking vector based on the previous one. Remember that M is fixed and defined by the structure of the system itself. Therefore Equation (1) has to be introduced in order to determine the allowable firing vector x_k. This equation may be written as

$$x_k = \overline{\overline{F}} \oplus \overline{\overline{m}}_k = \overline{\overline{\begin{bmatrix} F_u & F_v & F_r & F_y \end{bmatrix}}} \oplus \overline{\overline{\begin{bmatrix} PI & v & r & PO \end{bmatrix}_k^T}} \,. \tag{9}$$

Equation (9), as opposed to (8), cannot be computed in standard matrix algebra. The overbar denotes logical negation; given a natural number, its negation is equal to zero if the number is greater than zero, and 1 otherwise. The double overbar only indicates that the logical negation is performed twice, making projecting the natural number vector m_k into a vector of zeros and ones. The $\oplus$ symbol indicates that the operation to be performed is somehow like a matrix multiplication, but with the operations of multiplication and addition replaced by "or" and "and", respectively. On the example demonstrated in the next section, this operation is described on the MATLAB function in Fig. 5. These two equations constitute a hybrid system [1, 2, 48] with logical and algebraic components (see the example below).

Using these equations it is very easy to write a computer program in MATLAB, MATRIX$_X$, C, or any computer language, to simulate a discrete event system. Once the equations have been programmed, the program can easily simulate different discrete event systems by only changing the matrix description of the system (i.e., M and F). Moreover, it is straightforward to compute and plot versus time various performance measures, such as resource percent utilization, buffer lengths, part throughputs, and so on.

Simulation of Discrete Event Systems: An Example

The use of the Equations (8) and (9) in the matrix framework will now be illustrated by a reentrant-flow-line example. Though the example is a simple one, the techniques extends directly to complex systems. In fact, the basic structure of Fig. 4 can be considered a sort of "Runge-Kutta" integrator for all discrete part DE systems, i.e., those that can be represented as a PN; only the matrices F and S need to be changed. In this case, the example consists of two machines *M1* and *M2*, which could be either available (*M1A*, *M2A*) or processing a part (*M1P*, *M2P*); a buffer (*BA*, *BS*); and a shared robot (*RA*, *RU1*, *RU2*).

As shown in Fig. 1, a part enters the workcell in *PI*, it is drilled by the first machine (*M1*), moved by the robot (*RU1*) to the buffer (*BS*), polished by the second machine (*M2*), and moved again by the robot (*RU2*) to *PO*, which is the cell output. The figure also shows that pallets (*PA*) are needed for the first machine and for the derivative subassemblies. Fig. 2 shows the same example represented as a Petri-net.

With the description proposed in the previous paragraph, we can define the marking vector as $m_k = [PI\ M1P\ RU1\ BS\ M2P\ RU2\ PA\ M1A\ M2A\ BA\ RA\ PO]^T$. Evaluating successively the firing vector x_k from Equation (9) and the next state of the system from the Petri-net marking transition Equation (8), one can see the progressive evolution of the net. These few calculations show the power and simplicity of the basic equations and thus the usefulness of the matrix framework in DES simulations.

A simple MATLAB example is included in Fig. 4. The first few lines define the system matrices as explained by the matrix DE model. These matrices correspond to the same discrete event system shown in Figs. 1 and 2. In this case, with the initial conditions established in *line 18* (Fig. 4), the evolution of the net is simply computed by the successive execution of *lines 22* and *23* (Fig. 4). These two lines are the MATLAB equivalent of Equations (8) and (9). The output of this program shows the marking vector for successive iterations, and thus the simulation of the DE system under consideration.

```
% Timed DE simulation

% v = [M1P RU1 BS M2P RU2]
% r = [PA M1A M2A BA RA]
% ud = [ud1 ud2]
   clear all;
   Fv = [0 0 0 0 0; 1 0 0 0 0; 0 1 0 0 0; 0 0 1 0 0; 0 0 0 1 0; 0 0 0 0 1];
   Fr = [1 1 0 0 0; 0 0 0 0 1; 0 0 0 1 0; 0 0 1 0 0; 0 0 0 0 1; 0 0 0 0 0];
   Fud = [0 0;  1 0;  0 0;  0 0;  0 1;  0 0];
   Sv = [1 0 0 0 0 0; 0 1 0 0 0 0; 0 0 1 0 0 0; 0 0 0 1 0 0; 0 0 0 0 1 0];
   Sr = [0 0 0 0 0 1; 0 1 0 0 0 0; 0 0 0 0 1 0; 0 0 0 1 0 0; 0 0 1 0 0 1];
   Sud = [0 0 0 0 0 0;  0 0 0 0 0 0];
   Fu = [1 0 0 0 0 0]';
   Fy = [0 0 0 0 0 0]';
   Su = [0 0 0 0 0 0];
   Sy = [0 0 0 0 0 1];
   F = [Fu Fv Fr Fud Fy];
   S = [Su' Sv' Sr' Sud' Sy']';
   M = S' - F;

% Initial Conditions
% m(to) = [PI v r Ud PO];
% m(to) = [PI M1P RU1 BS M2P RU2 PA M1A M2A BA RA ud1 ud2 PO];
   mf = [9 0 0 0 0 0 4 1 1 2 1 0 0 0]';
   mi = [0 0 0 0 0 0 0 0 0 0 0 0 0 0]';
   PNtimes = [0 2.8 .4 1.2 3 .5 .2 0 0 0 0 0 0 0]';
   ud1 = 12; ud2 = 13;

   TotalTime = 25;
   TimePeriod = .1;
   Num_iter = TotalTime / TimePeriod;

   T = [PNtimes PNtimes PNtimes PNtimes PNtimes];
   U(:,1) = [0 0 0 0 0 0 0 0 0 0 0 0 0 0]';
   W(:,1) = [0 0 0 0 0 0 0 0 0 0 0 0 0 0]';
   N(:,1) = mf + mi;
   time(1) = 0; % // Initial time

% Running the simulation
   for i = 2:Num_iter;

   % Conflict resolution Subroutine
      mf(ud1) = 1;
      mf(ud2) = 1;
      x = multoa(not(F), not(not(mf)));
      conflict = (mf - (F' * x));
      if any(conflict < 0)
         % Random dispatching
         if (fix(2*rand(1)) == 0)
            mf(ud1) = 1; mf(ud2) = 0;
         else
            mf(ud1) = 0; mf(ud2) = 1;
         end
      end

   % Deadlock Avoidance Subroutine (MAXWIP)
      x = multoa(not(F), not(not(mf)));
      mtest = mf + (M' * x);
      if ((mtest(3)+mtest(4)+mtest(5)) >= 3)
         % /// RU1-BS-M2P --- ud1 ///
         mf(ud1) = 0;
         mf(ud2) = 1;
      end

      x = multoa(not(F), not(not(mf)));
      mf = mf + (-F' * x);
      mi = mi + (S * x);

   % Elapsed Times Subroutine
      for j = 1:length(mf);
         % Computing Utilization (U), wasted time (W), number of tokens (N)
```

Fig. 7. MATLAB file "DE2.M".

```
63.            if (mi(j) > 0)
64.               U(j,i) = U(j,i-1) + TimePeriod;
65.            else
66.               U(j,i) = U(j,i-1);
67.            end
68.            if (mf(j) > 0)
69.               W(j,i) = W(j,i-1) + TimePeriod;
70.            else
71.               W(j,i) = W(j,i-1);
72.            end
73.            N(:,i) = mf + mi;
74.            if (mi(j) > 0)
75.               for k = 1:mi(j)
76.                  if (T(j,k) > 0)
77.                     T(j,k) = T(j,k) - TimePeriod;
78.                  else
79.                     for kt = 2:mi(j)
80.                        T(j,kt-1) = T(j,kt);
81.                     end
82.                     T(j,mi(j)) = PNtimes(j);
83.                     mf(j) = mf(j) + 1;
84.                     mi(j) = mi(j) - 1;
85.                  end
86.               end
87.            end
88.         end

89.         time(i) = time(i-1) + TimePeriod;
90.      end

91.      % Displaying the results
92.      plot(time,N(2,:)+12,'y-',time,N(3,:)+10,'y-',time,N(5,:)+8,'y-',time,N(6,:)+6,'y-
93.      ',time,N(8,:)+4,'y-',time,N(9,:)+2,'y-',time,N(11,:),'y-');
94.      AXIS([0 TotalTime -1 14])
95.      text((1.02)*TotalTime,12.5,'M1P');
96.      text((1.02)*TotalTime,10.5,'RU1');
97.      text((1.02)*TotalTime,8.5,'M2P');
98.      text((1.02)*TotalTime,6.5,'RU2');
99.      text((1.02)*TotalTime,4.5,'M1A');
100.     text((1.02)*TotalTime,2.5,'M2A');
101.     text((1.02)*TotalTime,0.5,'RA');
102.     xlabel('Time');
```

Fig. 7. MATLAB file "DE2.M" (continued).

Timed Simulations

It is easy to see from the MATLAB example in Fig. 4 how simple it is to simulate a basic DE system. However, any real workcell has nonzero times for performing operations, namely for performing jobs and setting up resources once they are released. These times are interpreted herein as *timed PN places*. In this situation, one must split the place transition Equation (8) into two parts, adding also a subroutine that determines by computing elapsed times which currently ongoing operations are completed. To accomplish this, each *i-th* place is envisioned as having two parts. The "input" part is m_i, where tokens are positioned as the place input transitions fire. After the time associated with that place has lapsed, the token in m_i is moved to the final part m_f; then it becomes available to fire the place output transitions. A subroutine must be written to compute, at each DE iteration k, which currently ongoing workcell jobs and/or resource releases are completed, appropriately moving entries of m_i into m_f. Then, the following equations replace (8):

$$m_f(k+1) = m_f(k) - F^T \cdot x(k),$$
$$m_i(k+1) = m_i(k) + S \cdot x(k).$$

Fig. 7 shows the MATLAB code for a timed simulation. Once again, the same DE system is implemented, but now the code has been modified to take into account the restrictions imposed by timed simulations. Mainly, Equation (8) has been replaced by *lines 58* and *59* (Fig. 7), as explained in this section. Also, a *elapsed times subroutine* has been included (*lines 60-88*, Fig. 7). Now with this new code, we are able to compute different performance measures, such as resource percent utilization (Fig. 10), wasted time, overall activity of the DE system and so on.

Dispatching: Conflict Resolution Subroutine

In Fig. 4, the initial conditions (e.g., number of jobs in the system, $PI = 1$) were selected so that there is no shared resource conflict. In reality, shared resources present a problem in that failure to properly dispatch the shared resource can lead to deadlock. In the example shown in Fig. 4, this could happen if $PI > 3$. However, it is straightforward to simulate complex systems that contain shared resources using (8) and (9). To accomplish this, an extra step must be inserted between Equations (9) and (8). The vector x_k as initially computed should be considered as a *requested or proposed* transition firing vector (*line 39*, Fig. 7). If it results in any negative values in m_{k+1}, then there was a shared-

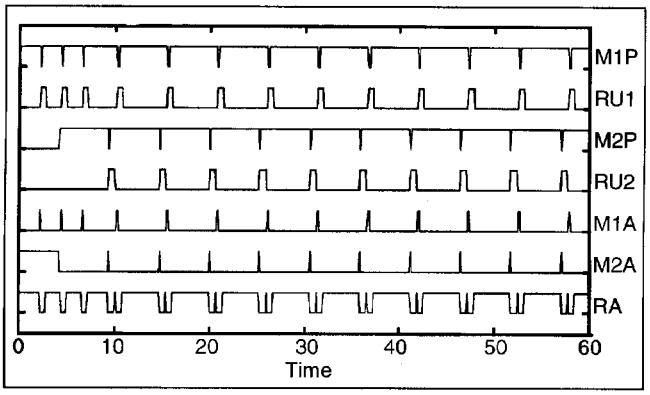


Fig. 8. Execution of a DE system.

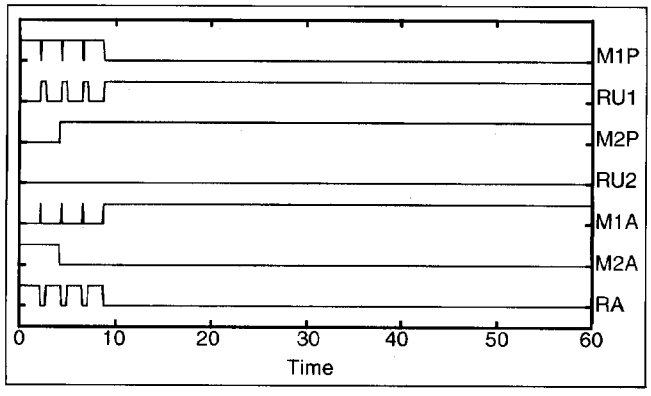


Fig. 9. Reaching deadlock.

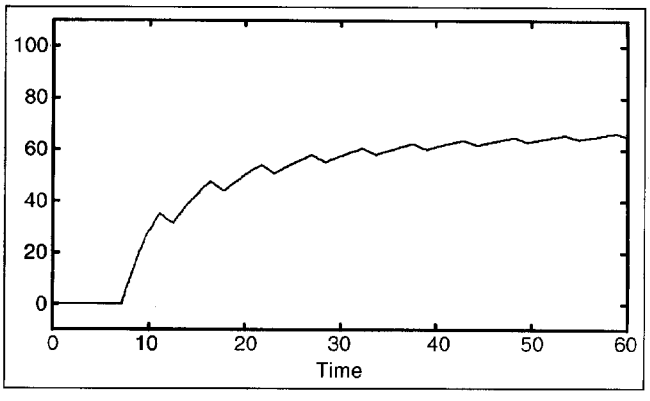


Fig. 10. Resource percent utilization.

resource assignment problem (*lines 40-41*, Fig. 7). In this case, one must call a dispatching subroutine that decides which job involving that shared resource to perform and propose a new x_K; this amounts to selecting the dispatching input. The job selection is determined by extending the marking vector to include dispatching places u_D. Then, m_{k+1} is computed and one proceeds with the iteration. The dispatching subroutine can contain any dispatching rule desired (*lines 42-47*, Fig. 7). As you can see a different choice of dispatching inputs is selected depending whether or not a shared-resource assignment problem has occurred. In this case the subroutine is implementing random dispatching.

Simulation Results

Using this technique we are able to plot the evolution of a discrete event system versus time. In Fig. 8, we see the operation of several machines and robots in our reentrant-flow-line example. For instance, in the last line the state of the shared robot is shown (when RA is high, this means that the robot is available).

The following plot (Fig. 9) shows the same system with a different controller. In this case, deadlock is reached such that after some time there is no more activity in the DE system. Such a case was produced by implementing an erroneous deadlock avoidance policy.

As another example, Fig. 10 shows the percent utilization of the shared-robot. Assuming that the system is deadlock free, all the plots reach some steady-state value. In the same way that this plot was computed, one can plot versus time various performance measures, such as resource percent utilization, the elapsed times, part throughputs, and so on.

Conclusions

A new matrix-based formulation of discrete event systems was adopted that, together with the PN marking transition equation, provides a full dynamical description of a DE system. Complete simulation equations were provided for a DE system. The matrices in the DE model description come directly from industrial engineering tools such as the bill-of-material, assembly tree, and resource requirements matrix. Using these matrices, algorithms were given to simulate and model discrete event systems. The results show the power and simplicity of our approach.

References

[1] H. Alayan and R.W. Newcomb, "Binary Petri net Relationships," *IEEE Trans. Circuits and Systems*, vol. CAS-34, no. 5, pp. 565-568, May 1987.

[2] P. Antsaklis, W. Kohn, A. Nerode, and S. Sastry, *Hybrid Systems II*, Springer-Verlag, Berlin, 1995.

[3] J. Banks, and J.S. Carson, *Discrete Event System Simulation*, Prentice Hall, Englewood Cliffs, NJ, 1984.

[4] D.A. Bodner, S. Dilley-Schneider, S. Narayanan, U. Sreekanth, T. Govindaraj, L.F. McGinnis, C.N. Mitchell, "Object-Oriented Modeling and Simulation of Automated Control in Manufacturing," *Proc. Int. Conf. Robotics and Automation*, vol. 3, pp. 83-88, May 1993.

[5] D.Y. Burman, F.J. Gurrola-Gal, A. Nozari, S. Sathaye, and J.P. Sitarik, "Performance Analysis Techniques for IC Manufacturing Lines," *AT&T Technical Journal*, vol. 65, no. 4, pp. 46-56, July/Aug. 1986.

[6] X. Cao and Y.C. Ho, "Models of Discrete Event Dynamic Systems," *Control Systems Magazine*, vol. 10, no. 4, pp 69-76, 1990.

[7] C.R. Cash and W.E. Wilhelm, "A Simulation Model for Use in Designing Robotic Assembly Cells," *J. Manufacturing Sys.*, vol. 7, no. 4, pp. 279-291.

[8] C.G. Cassandras, *Discrete Event Systems*, Aksen Assoc. Inc., Boston, 1993.

[9] C.L.P. Chen and C. Wichman, "A CLIPS Rule-Based Planning System for Mechanical Assembly," *Proc. NSF DMS Conf.*, pp. 837-841, Atlanta, 1992.

[10] R.W. Conway, W.L. Maxwell, and L.W. Miller, *Theory of Scheduling*. Reading, MA, Addison-Wesley, 1967.

[11] R. David and H. Alla, "Petri Nets for Modeling of Dynamic Systems—A Survey," *Automatica*, vol. 30, no. 2, pp. 175-202, 1994.

[12] A.A. Desrochers, *Modeling and Control of Automated Manufacturing Systems*, IEEE Computer Society Press, 1990.

[13] E.A. Elsayed and T.O. Boucher, *Analysis and Control of Production Systems*. 2nd ed., Prentice Hall, Englewood Cliffs, 1994.

[14] S.D. Eppinger, D.E. Whitney, and R.P. Smith, "Organizing the Tasks in Complex Design Projects," *Proc. ASME Int. Conf. Design Theory and Methodology*, pp. 39-46, September 1990.

[15] M.R. Garey, and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness*. Freeman, San Francisco, CA, 1979.

[16] J. Giarratano and G. Riley, *Expert Systems*, PWS-Kent Publishing Co., Boston, 1989.

[17] D. Gracanin, P. Srinivasan, K. Valavanis, "Parametrized Petri nets: Properties and Applications to Automated Manufacturing Systems," *Proc. IEEE Mediterranean Symp. New Directions in Control and Automation*, pp. 48-55, June 1994.

[18] S.C. Graves, "A Review of Production Scheduling," *Operations Research*, vol. 29, pp. 646-675, August 1981.

[19] Y.C. Ho, "Dynamics of Discrete Event Systems," *Proc. IEEE*, pp. 3-6, Jan. 1989.

[20] Y.C. Ho and X. Cao, *Perturbation Analysis of Discrete Event Systems*, Kluwer, Boston, 1991.

[21] H. Huang, F.L. Lewis, O.C. Pastravanu, and A. Gurel, "Flowshop Scheduling Design in an FMS Matrix Framework," *Control Engineering-Practice*, vol. 3, no. 4, pp. 561-568, 1995.

[22] M.D. Jeng and F. DiCesare, "A Synthesis Method for Petri Net Modeling of Automated Manufacturing Systems with Shared Resources," *Proc. IEEE Conf. Decision and Control*, pp. 1184-1189, December 1992.

[23] J.P.C. Kleijnen and C. Standridge, "Experimental Design and Regression Analysis in Simulation: An FMS Case Study," *European Journal of Operational Research*, vol. 33, pp. 257-261, 1988.

[24] M. Klein, "Supporting Conflict Resolution in Cooperative Design System," *IEEE Trans. Sys., Man, and Cybernetics*, vol. 21, no. 6, pp. 1379-1390, November/December 1991.

[25] P.R. Kumar and S.P. Meyn, "Stability of Queueing Networks and Scheduling Policies," *Proc. IEEE Conf. Decision and Control*, pp. 2730-2735, December 1993.

[26] A.H. Levis, N. Moray, and B. Hu, "Task Decomposition and Allocation Problems and Discrete Event Systems," *Automatica*, vol. 30, no. 2, pp. 203-216, 1994.

[27] F.L. Lewis, O.C. Pastravanu, and H.-H. Huang, "Controller Design and Conflict Resolution for Discrete Event Manufacturing Systems," *Proc. IEEE Conf. Decision and Control*, pp. 3288-3293, San Antonio, December 1993.

[28] S.H. Lu and P.R. Kumar, "Distributed Scheduling Based on Due Dates and Buffer Priorities," *IEEE Trans. Automat. Control*, vol. 36, no. 12, pp. 1406-1416, December 1991.

[29] P.B. Luh and D.J. Hoitomt, "Scheduling of Manufacturing Systems Using the Lagrangian Relaxation Technique," *IEEE Trans. Automat. Control*, vol. 38, no. 7, July 1993.

[30] T.E. Morton, and D.W. Pentico, *Heuristic Scheduling Systems*, John Wiley, New York, 1993.

[31] T. Murata, "Petri Nets: Properties, Analysis and Applications," *Proc. IEEE*, vol. 77, no. 4, pp. 541-580, April 1989.

[32] T. Murata, N. Komoda, K. Matsumoto, and K. Haruna, "A Petri Net-Based Controller for Flexible and Maintanable Sequence Control and Its Applications in Factory Automation," *IEEE Trans. Ind. Electronics*, vol. IE-33, no. 1, pp. 1-8, February 1986.

[33] U. Negretto, "Control of Manufacturing Systems," *Intelligent Design and Manufacturing*, A. Kusiak, ed., New York: Wiley, 1991.

[34] J.L. Peterson, *Petri Net Theory and the Modeling of Systems*, New Jersey: Prentice-Hall, 1981.

[35] A.A.B. Pritsker, *Introduction to Simulation and Slam II*, Halsted Press, New York, 1986.

[36] P.J. Ramadge and W.M. Wonham, "The Control of Discrete Event Systems," *Proc. IEEE*, vol. 77, pp. 81-98, 1989.

[37] S. Ramaswamy, K.P. Valavanis, P. Srinivasan, and A. Steward, "A Coordination Level H-EPN Based Error Recovery Model for Hierarchical Systems," *Proc. IEEE Mediterranean Symp. New Directions in Control and Automation*, pp. 56-61, June 1994.

[38] L. Schruben, "Simulation Modeling with Event Graphs," *Communications of the ACM*, vol. 26, no. 11, 1983.

[39] R.E. Shannon, "Knowledge-Based Simulation Techniques for Manufacturing," *Int. J. Production Res.*, vol. 26, no. 5, pp. 953-973, 1988.

[40] K. Srihari, C.R. Emerson and J.A. Cecil, "Modeling Manufacturing with Petri Nets," *CIM Review*, pp. 15-21, Spring 1990.

[41] D.V. Steward, "The Design Structure System: A Method for Managing the Design of Complex Systems," *IEEE Trans. Engineering Management*, pp. 71-74, August 1981.

[42] E. Szczerbicki, "Modelling and Identification of Manufacturing Systems: Decomposition Stage," *Int. J. Systems Sci.*, vol. 24, no. 8, pp. 1509-1518.

[43] S.-H. Teng and J.T. Black, "Cellular Manufacturing Systems Modeling: The Petri Net Approach," *J. Manufacturing Systems*, vol. 9, no. 1, pp. 45-54, 1990.

[44] T.E. Vollman, W.L. Berry, and D.C. Whybark, *Manufacturing Planning and Control Systems*. 3rd. ed., Homewood, Ill., Irwin, 1993.

[45] J.N. Warfield, "Binary Matrices in System Modeling," *IEEE Trans. Systems, Man, Cybern.*, vol. SMC-3, no. 5, pp. 441-449, September 1973.

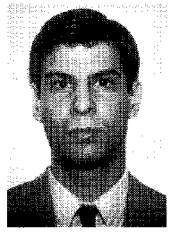
[46] J. Wolter, S. Chakrabarty, and J. Tsao, "Methods of Knowledge Representation for Assembly Planning," *Proc. NSF Design and Manuf. Sys. Conf.*, pp. 463-468, January 1992.

[47] W.M. Wonham, *Notes on Control of Discrete-Event Systems*, Pub. ECE 1636F, Dept. of Elect. Eng., Univ. of Toronto, Canada, 1996.

[48] Y.Y. Yang, D.A. Linkens, S.P. Banks, "Modelling of Hybrid Systems Based on Extended Coloured Petri Nets," *Hybrid Systems II*, Springer-Verlag, Berlin, 1995.

[49] M.-C. Zhou, F. DiCesare, A.D. Desrochers, "A Hybrid Methodology for Synthesis of Petri Net Models for Manufacturing Systems," *IEEE Trans. Robotics and Automation*, vol. 8, no. 3, pp. 350-361, June 1992.

[50] M.-C. Zhou and F. DiCesare, *Petri Net Synthesis for Discrete Event Control of Manufacturing Systems*, Boston: Kluwer, 1993.



Diego A. Tacconi is a graduate research assistant at the Automation & Robotics Research Institute. He received his B.S. in electrical engineering from the University of Texas at Arlington in 1994. He is currently completing his M.S. degree, also at the University of Texas at Arlington. He is a member of IEEE, Eta Kappa Nu, and Tau Beta Pi. His research interests include advanced controls, robotics, real-time implementation of control systems, and discrete event system simulations.

Frank L. Lewis was born in Wurzburg, Germany, and studied in Chile and Scotland. He obtained the bachelor's degree in physics/electrical engineering and the master's of electrical engineering degree at Rice University in 1971. He spent six years in the U.S. Navy, serving as navigator aboard the frigate U.S.S. Trippe (FF-1075), and executive officer and acting commanding officer aboard U.S.S. Salinan (ATF-161). In 1977 he received the master's of science in aeronautical engineering from the University of West Florida. In 1981 he obtained the Ph.D. degree at The Georgia Institute of Technology in Atlanta, where he was employed from 1981 to 1990 and is currently an adjunct professor. He was awarded the Moncrief-O'Donnell Endowed Chair in 1990 at the Automation and Robotics Research Institute of the University of Texas at Arlington.