IMPLEMENTATION OF A DEADLOCK AVOIDANCE POLICY FOR MULTIPART REENTRANT FLOW LINES USING A MATRIX-BASED DISCRETE EVENT CONTROLLER

Jose Mireles Jr, Member IEEE, Frank L. Lewis, Fellow IEEE, Ayla Gürel

Instituto de Ingeniería y Tecnología de la Universidad Autónoma de Ciudad Juárez, Ave. del Charro 610 Nte. Juárez MÉXICO. jmireles@arri.uta.edu.

Automation & Robotics Research Institute, of The University of Texas at Arlington (UTA) 7300 Jack Newell Blvd. S., Fort Worth, TX 76118-7115, USA. flewis@controls.uta.edu

Electric & Electronics Eng. Dept. Eastern Mediterranean Univ. P.O. Box 95, Gazi Magosa Mersin 10, TURKEY. gurel@eenet.ee.emu.edu.tr

ABSTRACT

A deadlock avoidance supervisory controller for Discrete Event (DE) Systems is implemented. The DE controller uses a novel rule-based matrix dispatching formulation (US patent received). This matrix formulation makes it direct to write down the DE controller from standard manufacturing tools such as the bill of materials or the assembly tree. It is shown that the DE controller's matrix form equations plus the Petri Net marking transition equation together provide a complete dynamical description of DE systems. Deadlock-free dispatching rules are derived by performing circular wait analysis (CW) for possible deadlock situations. We analyze the so-called critical siphons, certain critical subsystems and resources to develop a DE controller that guarantees deadlock-free dispatching by limiting the work-in-progress in the critical subsystems associated with each CW. This is the least-restrictive dispatching policy that avoids deadlock. The deadlock-free dispatching rules are implemented by the DE controller on a three-robot, two machine reentrant flow line, the Intelligent Material Handling cell at the Automation and Robotics Research Institute of UTA. Technical information given includes the development of the deadlock-free controller in LabVIEW®.

Keywords: Deadlock Avoidance, Petri nets, Discrete Event Systems, Reentrant flow lines, Intelligent control.

1. INTRODUCTION

A problem in Flexible Manufacturing Systems or Discrete Event Systems (DES) is job sequencing when some resources are shared. While some resources manipulate or machine single parts in a DES, others manipulate or machine multiple parts for several products in the manufacturing process. If jobs are not correctly sequenced in the latter case, serious problems in the performance of the DES can be obtained, including blocking and system deadlock [1-8]. Deadlock occurs when blocking develops in a circular wait [9,10] situation, which is a fatal condition that eventually stops all activity in the flow lines involved. Therefore, it is very important that the DE controller properly sequences jobs and assigns resources.

Many approaches exist to control the resource dispatching sequences in manufacturing systems, including First-In-First-Out (FIFO), First-Buffer-First-Serve (FBFS), Last-Buffer-First-Serve (LBFS), Earliest-Due-Date (EDD), Least-Slack (LS), and others [11]. Rigorous analysis for some of these algorithms has been done for the case of unbounded buffer lengths. For instance, [12,13] has shown that LBFS yields bounded buffer stability in the case of a single part reentrant flow-line (RF). However, in actual manufacturing systems, the buffer lengths are usually finite, which introduces the possibility of system deadlock. Moreover, little rigorous work has been done for dispatching in multipart RF (MRF) with finite buffers. In this paper we provide a technique for avoiding deadlock in MRF with finite buffers by restricting the work-in-progress (WIP) in certain “critical subsystems”. This is a rigorous notion related to the idea of ‘CONWIP’ [14]. All computations are performed using straightforward matrix
algorithms, including computation of the critical subsystems, for any RFL in a certain general class.

This paper presents the development and implementation of an augmented discrete event controller for multipart reentrant flow lines that is based on the decision-making matrix formulation introduced in [15-17]. We describe the DE controller (DEC) formulation, and show how to analyze and compute in matrix notation the circular waits, their critical siphons and certain critical subsystems, and a matrix test for the presence/absence of critical resources. The CRs are critical structured-placed resources that might lead to possible Second Level Deadlock (SLD) situations in MRF/FMRF (the mathematical definition of these CR is provided later). Based on these constructions, we integrate deadlock-free dispatching rules into our augmented DEC matrix formulation by limiting the work-in-progress (WIP) in these critical subsystems. This is the least-restrictive dispatching policy that avoids deadlock. We provide an implementation example of the augmented computationally efficient DEC on a 3 robot Intelligent Material Handling (IMH) robotic workcell at UTA's Automation & Robotics Research Institute (ARRI). A detailed exposition of the development of the DEC for the workcell is given, including all steps needed to implement the controller. Technical information includes this implementation in a graphical environment, LabVIEW.

2 MATRIX-BASED DISCRETE EVENT CONTROLLER

A novel Discrete Event Controller (DEC) for manufacturing workcells was described in [16, 18-21]. This DEC is based on matrices, and it was shown to have important advantages in design, flexibility and computer simulation. The definition of the variables of the Discrete Event Controller is as follows. Let \( v \) be the set of tasks or jobs used in the system, \( r \) the set of resources that implement/perform the tasks, \( u \) the set of inputs or parts entering the DES. The DEC Model State Equation is described as

\[
\bar{x} = F_v \otimes v \oplus F_r \otimes r \oplus F_u \otimes u \oplus F_{\text{uc}} \otimes u_C
\]

(1)

where: \( \bar{x} \) is the task or state logical vector,
\( F_v \) is the job sequencing matrix,
\( F_r \) is the resource requirements matrix,
\( F_u \) is the input matrix,
\( F_{\text{uc}} \) is the conflict resolution matrix, and
\( u_C \) is a conflict resolution vector.

This DEC equation is performed in the AND/OR algebra. That is, multiplication \( \otimes \) represents logical “AND,” addition \( \oplus \) represents logical “OR,” and the over-bar means logical negation (used as in [18]). From the model state equation, the following four interpretations are obtained. The job sequencing matrix \( F_v \) reflects the states to be launched based on the current finished jobs. It is the matrix used by [22-24] and others, and it can be written down from the manufacturing Bill of Materials. The resource requirement matrix \( F_r \) represents the set of resources needed to fire possible job states this is the matrix used by [25,26]. The input matrix \( F_u \) determines initial states fired from the input parts. The conflict resolution matrix \( F_{\text{uc}} \) prioritizes states launched from the external dispatching input \( u_C \), which has to be derived via some decision making algorithm [11,27]. The importance of this equation is that it incorporates matrices \( F_v \) and \( F_r \), previously used in heuristic manufacturing systems analysis, into a rigorous mathematical framework for DE system computation.

For a complete DEC formulation, one must introduce additional matrices, \( S_v \) and \( S_r \), as described next. The state logic obtained from the state equation is used to calculate the jobs to be fired (or task commands), to release resources, and to inform about the final products produced by the system. These three important features are obtained by using the three equations:

\[
\text{Start Equation (task commands)}
\]

\[
\nu_S = S_v \otimes x
\]

(2)

\[
\text{Resource Release Equation}
\]

\[
r_S = S_r \otimes x
\]

(3)

\[
\text{Product Output Equation}
\]

\[
y = S_y \otimes x
\]

(4)

Figure 1 shows the DEC based on the matrix formulation as used to control job sequences and resource assignment of a workcell. Subscript “s” on the vectors \( \nu \) and \( r \) denotes “start.” Thus, \( \nu \) and \( r \) are outputs from the workcell measured by sensors, while \( \nu_s \) and \( r_s \) are commands to the workcell to begin jobs or set resources as “released.”

2.1 MATRIX FORMULATION AND PETRI-NETS

There is a very close relationship between the DEC just described and Petri Net (PN) tools. The Incidence Matrix [28,30] of the PN equivalent to the DE controller is obtained by defining the activity completion matrix and the activity start matrix as:

Activity Completion Matrix:

\[
F = [F_u \ F_v \ F_r \ F_{\text{uc}}]
\]

(5)

Activity Start Matrix:

\[
S = [S_u^T \ S_v^T \ S_r^T \ S_{\text{uc}}^T]'
\]

(6)

Then, the PN’s Incidence Matrix is defined as

\[
M = S^T - F = [S_u^T - F_u \ S_v^T - F_v \ S_r^T - F_r \ S_{\text{uc}}^T - F_{\text{uc}}],
\]

(7)

where \( S_u \) and \( S_{\text{uc}} \) are zero matrices.
If we define the set $X$ containing the elements of $x$ (the state controller vector), and $A$ as the set of activities containing the vectors $v$ and $r$, (i.e. $A = \{v, r\}$), then it has been shown that $(A, X, F^T, S)$ is a PN \cite{18,30}. This allows one to directly draw the PN of a system given the matrices $F$ and $S$ or vice versa.

The elements of matrices $F$ and $S$, are either ‘zero’ or ‘one’. $F^T$ is the PN input incidence matrix and $S$ is the PN output incidence matrix. The $f_{ij}$ elements of $F$, ($F_i$) which are set to ‘one’, state that to fire transition $x_i$, the job $v_j$ (resource $r_j$) needs to be finished (available); the $s_{ij}$ elements of $S$, ($S_i$) set to ‘one’, indicate that to start job $v_j$, the transition $x_i$ needs to be finished (the resource $r_i$ is released after the transition $x_j$ is finished).

If the marking vector $m(t)$ from a PN is defined as $m(t) = [u(t)^T, v(t)^T, r(t)^T, u_j(t)^T]^T$. (8)

For a specific time iteration $t$, then the PN marking transition equation \cite{28} is

$$m(t + 1) = m(t) + F^T x = m(t) + [S^T - F] x(t). \quad (9)$$

2.1 COMPLETE DYNAMICAL DESCRIPTION FOR DES

A major gap in PN theory has been its inability to provide a complete dynamical description of a DES. The marking transition equation (9) provides a partial description \cite{28}, but it is not known in the literature how to generate the allowable firing vector $x(t)$. This deficiency is repaired by using the matrix-based DEC controller equation (1) together with the PN marking transition equation. The key is to note that the vector $x(t)$ in (9) is identical to the vector $x$ in the DEC equation (1) at time $t$.

To put the DEC eqs. into a format convenient for simulation, one may write (1) as $\bar{x} = F \otimes \overline{m}$ or

$$x(t) = F \otimes \overline{m}(t) = [F_v \ F_r \ F_w] \oplus [u \ v \ r \ u_j](t). \quad (10)$$

The complete dynamical description of the DES, as described in \cite{21}, is provided by the PN marking eq. (9), plus the DEC equation (10). Note that we are using the Timed Places Petri Net (TPPN) representation of PNs \cite{31}.

To simulate DES on a digital computer, one must split (9) into two parts. Thus, $m(t)$ is split into two vectors, the pending marking vector $m_p$ and available resources and finished jobs marking vector $m_a$,

$$m(t) = m_p(t) + m_a(t). \quad (11)$$

The pending marking vector contains all new jobs and any unfinished jobs or currently in process,

$$m_p(t + 1) = m_p(t) + S^T x(t). \quad (12)$$

The available marking vector equation takes away tokens from $m_a$ corresponding to which resources are no longer available and the jobs that have just finished.

$$m_a(t + 1) = m_a(t) - F x(t). \quad (13)$$

Note that adding (12) and (13) gives (9).

For the case of manufacturing MRF, where one or more resources are shared for different activities, $m_p$ and $m_a$ are crucial information vectors while dispatching subsequent jobs. For instance, $m_p$ and $m_a$ are needed for the deadlock avoidance and conflict resolution dispatching rules. A complete simulation of discrete event systems using this technique is provided in \cite{20,21}.

3 MATRIX ANALYSIS OF MRF INTERNAL STRUCTURE AND DEADLOCK

In this section we present a unified technique for deadlock-free dispatching, and show how to implement some notions from other papers using matrices. This yields computationally efficient algorithms for analyzing the structure of MRF and deadlock-free dispatching. The least restrictive deadlock-free policy is given here. In the next section, we show an example illustrating these constructions.

Consider the definition of Multiple Reentrant Flowlines (MRF) and the assumptions considered in \cite{32}, which
basically define the sort of discrete-part manufacturing systems that can be described by a Petri net. The assumptions are:
- No preemption. A resource cannot be removed from a job until it is complete.
- Mutual exclusion. A single resource can be used for only one job at a time.
- Hold while waiting. A process holds the resources already allocated to it until it has all resources required to perform a job.
- In this paper we also assume there are no machine failures.

For the DE systems we consider in our analysis, the following are their particularities:
- Each job uses only one resource.
- After each resource executes one job, it is released immediately.

For the class of MRF satisfying these assumptions and definitions, deadlock can occur only if there is a circular wait relation among the resources [5,32]. Circular wait relations are ubiquitous in reentrant flowlines and in themselves do not present a problem. However, if a circular wait relation develops into circular blocking, then one has deadlock. But, as long as dispatching is carefully performed, the existence of circular wait relations presents no problem for regular systems [32]. The definition of a ‘regular system’ is given later.

3.1 CIRCULAR WAITS: SIMPLE CIRCULAR WAITS AND THEIR UNIONS.

In this section we present a matrix procedure to identify all circular waits (CW) in MRF systems. CWs are special wait relationships among resources described as follows. Given a set of resources $R$, for any two resources $r_i, r_j \in R$, $r_i$ is said to wait for $r_j$, denoted $r_i \rightarrow r_j$, if the availability of $r_i$ is an immediate requirement to release $r_j$, or equivalently, if there exists at least one transition $x \in \cdot r_i \cap \cdot r_j$. Circular waits among resources are a set of resources $r_a, r_b, r_c, \ldots, r_w$, which wait relationships among them are $r_a \rightarrow r_b \rightarrow \ldots \rightarrow r_w$, and $r_w \rightarrow r_a$. The simple Circular Waits (sCW), are primitive CWs which do not contain other CWs.

To identify such sCW, a wait relation digraph of resources has to be constructed first [33]. A digraph $D=(R,E)$, where $R$ is the set of nodes and $E=\{a_{ij}\}$ is the set of edges, with $a_{ij}$ drawn if $r_i \rightarrow r_j$ (in other words, each $a_{ij}$ represents all transitions in $\cdot r_i \cap \cdot r_j$). The digraph of resources is easily obtained from the matrix formulation of the system, by getting $W = (S, F)$. (14)

Unfortunately, to be able to analyze the MRF system and its possible deadlock structures we need to identify all CWs, not only simple circular waits. The entire set of CWs are the sCW plus the circular waits composed of unions of non-disjoint sCW (unions through shared resources among sCW.) In figure 8, we show a LabVIEW diagram that calculates all CWs from the sets of all sCW; it uses Gurel’s algorithm (from [8]) in MATLAB script code but uses matrices for efficiency of computations.
Using this diagram/code, we obtain two resulting matrices, $C_{\text{out}}$ and $G$. $C_{\text{out}}$ provides the set of resources which compose every CW (in rows), that is, an entry of ‘one’ on every $(ij)$ position means that resource $j$ is included in the $i^{th}$ CW. $G$ provides the set of composed CWs (rows) from unions of $sCW$ (columns), that is, an entry of ‘one’ on every $(ij)$ position means that $j^{th}$ sCW is included in the $i^{th}$ composed CW. In section 5.1 one can find the resulting matrices for a given example.

3.2 DEADLOCK ANALYSIS: IDENTIFYING CRITICAL SIPHONS AND CRITICAL SUBSYSTEMS.

In this section, we apply PN and matrix-based notions to calculate specific PN-place sets associated with each CW. The determination of these sets is required so that we can identify possible circular blocking (CB) [4,6,8,32] phenomena or deadlock situations. After computing the sets, we will provide computationally efficient matrix-based algorithms for a least restrictive deadlock-free dispatching policy. These sets are highly tied to siphons associated with each CW. A siphon set has a behavioral property that if it is token-free under some marking, then it will remain token-free under each successor marking. Such property may lead to CB, i.e. deadlock. A set of places $S$ is a siphon if and only if for all places $p_i \in S$ one has $\bullet p_i \subseteq U_j \{ p_j \}$ for some $\{ p_j \} \subseteq S$.

Three important sets associated with the CWs C are the siphon-job sets $J_s(C)$, the critical siphons, $S_c(C)$, and critical subsystems, $J_d(C)$. The critical siphon of a CW is the smallest siphon containing the CW. Note that if the critical siphon ever becomes empty, the CW can never again receive any tokens. This is, the CW has become circular blocking. The siphon-job set, $J_s(C)$, is the set of jobs which, when added to the set of resources contained in CW C, yields the critical siphon. The critical siphons of that CW C are the conjunction of sets $J_s(C)$ and C. The critical siphons of the CWs CC are the job sets $J_s(C)$ from that C not contained in the siphon-job set $J_s(C)$ of C. That is $J_s(C) = J(C) \setminus J_s(C)$. The job sets of CW C are defined by $J(C) = \cup_{r \in C} J(r)$, for $J(r) = r^* \cap J$, where J is the set of all jobs.

We now provide computational tools to determine the siphon-job sets $J_s(C)$, the critical siphons, $S_c(C)$, and critical subsystems, $J_d(C)$, for every CW C. To determine such sets, we need to calculate the set of adding transitions $T^+_C = \{ C \setminus C^* \}$ and clearing transitions $T^-_C = C^* \setminus C^\star$. $T^+_C$ are the set of transitions that, when fired, add tokens to the CW C. On the contrary, $T^-_C$ are the set of transitions which, when fired, subtract tokens from C. $C^\star$ and $C^*$ are the set of input and output transitions from C. These sets of transitions are important in keeping track of the tokens inside every CW C, and hence in determining the status of tokens inside the critical siphon.

In order to implement efficient real-time control of the DES, we need to compute these sets in matrix form. Thus, the intermediate quantities $C^\star$ and $C^*$ in matrix form for each CW are denoted $dC$ and $Cd$ respectively, computed as,

$$dC = C_{\text{out}} S_c$$

and

$$Cd = C_{\text{out}} F_v$$

Now, we are able to calculate the adding transitions

$$T^+_C = \{ C \setminus C^* \}$$

and the clearing transitions $T^-_C = C^* \setminus C^\star$. In matrix form

$$T_p = dC - (dC \land Cd),$$

and

$$T_m = C_d - (C_d \land dC),$$

where operation $A \land B$ represents an element-by-element logical AND operation between matrices A and B.

For each circular wait, these matrix forms contain the set of transition vectors $T^+_C$ and $T^-_C$ arranged in the rows of matrices $T_p$ and $T_m$, respectively. That is, an entry of ‘one’ on every $(ij)$ position in matrix $T_p$ ($T_m$), means that $j^{th}$ transition is a $T^+_C$ ($T^-_C$) transition belonging to that $i^{th}$ composed CW.

In terms of these constructions, matrix form sets are described next, indicating ‘one’ on every entry $(i,j)$ for places that belong to that set existing in every $i^{th}$ CW. The job sets described earlier for each CW C, $J(C)$, in matrix form (for all CWs arranged in rows) are described by

$$J_C = dC F_v = C_d S_v^\top.$$  

The siphon-job sets are defined for each $i^{th}$ CW C, as $J_s(C) = J(C) \cap T^+_C$. In matrix notation, we can obtain them for all CWs by

$$J_s = T_p F_v.$$  

There is a shortcut way to identify these siphon-job sets without calculating $T_p$. However, later we will need construction $T_p$ to identify presence/absence of critical resources, which we will define later. This shortcut way in matrix form is

$$J_s = J_C \land (C_d F_v).$$

This mathematical shortcut facilitates the calculation of these sets only if the system fulfills conditions presented in the beginning of section 3.4.

The critical subsystems, $J_d(C) = J(C) \cap J_s(C)$, which is defined later, in matrix form for all CWs C, are obtained by

$$J_s = J_C \land (C_d F_v).$$

Such representation is similar of that presented by [8]. In their work, they present another way to calculate the critical subsystems from the p-invariant covering job sets, not belonging to the critical siphons job set.
4 DEADLOCK AVOIDANCE

In terms of the constructions just given, we now present a minimally restrictive resource dispatching policy that guarantees absence of deadlock for multi-part reentrant flow lines. To efficiently implement in real time a DE controller with this dispatching policy we use matrices for all computations. We consider the case where the system is regular, that is, it cannot contain the Critical Resources (CR) (so-called structured bottleneck resources or 'key resources' [6,32] existing in Second Level Deadlock (SLD) structures [7,34]). For this case, we describe in section 4.2 a mathematical test to verify that the MRF/FMRF system is regular. If that is not the case, we can still use this matrix formulation, but with a different dispatching policy designed for systems containing second level deadlock structures. We will present such dispatching policy for FMRF systems having CR in a forthcoming work.

4.1 DISPATCHING POLICY

In this section we consider dispatching for regular systems. A matrix test for nonregularity is given in the next section. In [8] was given a minimally restrictive dispatching policy for regular systems which avoids deadlock for the class of MRF considered in this paper. To understand this policy, note that, for this class of systems, a deadlock is equivalent to a circular blocking (CB). There is a CB if and only if there is an empty circular wait (CW). However, this is possible (for regular systems) iff the corresponding critical siphon is empty. By construction, this is equivalent to all jobs of the circular wait being in the Critical Subsystem. In terms of PN, there is a deadlock iff all tokens of the CW are in the Critical Subsystem.

Therefore, the key to deadlock avoidance is to ensure that the WIP in the Critical Subsystems is limited to one less job than the total number of initial tokens in the CW (i.e. the total number of resources available in the CW). Due to the necessity and sufficiency of all the conditions just outlined, this MAXWIP policy is the least restrictive policy that guarantees absence of deadlock. It is very easy to implement. Preliminary off-line computations using matrices are used to compute the Critical Systems. A supervisor is assigned to each Critical Subsystem (CS) who is responsible for off-line computations using matrices are used to compute the WIP in the Critical Subsystems. It is very easy to implement.

Preliminary

In our implementation example, in every DE iteration, we use FBFS dispatching policy. Generally, FBFS maximizes WIP and machine percent utilization. However, it is known that FBFS often results in deadlock. Therefore, we combine FBFS with our deadlock avoidance test (23). Thus, before we dispatch the FBFS resolution, we examine the marking outcome with our deadlock policy. If this resulting outcome does not satisfy (23), then the algorithm denies or pre-filters in real time the firing and we apply again the FBFS conflict resolution strategy for the next possible allowable firing sequence. Therefore, using FBFS while permitted, we will try to satisfy in most of the current status of the cell the case m(J_o(C_i)) = m_d(C_i)-1. The later condition is the so called MAXWIP policy, defined in [35].

The dispatching policy for the case one has a regular system follows three main steps:

First, based on the structure of the system defined by matrices F and S, we use (22) to obtain for all CWs its corresponding critical subsystems J_o(CW). In matrix form (having all CWs), J_o^max has m rows as CWs, and n columns as total # resource-jobs in the system.

Second, for every DE-iteration, one must calculate from the current marking vector, m_current, the corresponding possible successor-marking vector, m_possible - Equation (9), which can be split into eqs. (12) and (13) provide this possible successor (m_d(t+1) = m_possible, m_d(t) = m_current). However, for a given m_current vector, since it is possible to have enabled more than one transition r_i, for r_i be a shared resource, marking m_possible can have negative elements due to (13). That is, it is possible that the marking vector m_possible has negative number(s) in the r(t) section from the general marking vector m(t), if more than one resource-jobs, v(t), are attempt to fire from one single shared resource, r(t)=r_i (verify from (8) that r(t) is part of m(t), as well as v(t) and u_i(t)). That is why the marking m_possible has to be 'filtered' by a conflict resolution dispatching policy to deny negative elements.

We 'filter' negative numbers from m_possible by setting to zero the non-desired resource-jobs elements in vector u_i(t) from the current marking vector m_current. u_i(t) is strategically preset full of ones in m_current before one starts every new DE iteration (set of these three steps), and adjusted during the calculation of m_possible to solve any possible conflict on shared resources. This is how we are able to calculate m_possible having no negative numbers.

Third, once selected the candidate m_possible, to avoid deadlock one must verify the condition m(J_o(C_i))<m_d(C_i) for every CW C_i. This can be accomplished by extracting from marking m_possible the v(t) vector, defined as v_possible and performing m_possible having no negative numbers.

\[ ||J_o^\wedge v_possible|| < ||C_out|| \]
for J_o be i^{th} row vector from J_o. C_{out} be the i^{th} row (or i^{th} CW) vector from C_{out} and ||V|| be the Σv_{i} for v_{i} be the i^{th} element of vector V.

If for any DE iteration, (24) does not hold, one must eliminate the resource-job from v_{possible} that is attempting to cause circular blocking in that i^{th} CW. This elimination can be accomplished by a high order conflict resolution among different machines (notice that for this case, this resolution is not among resource-jobs from a single resource-machine). The resource-jobs that might attempt to cause deadlock problems are found by

\begin{equation}
\text{v}_{\text{problem}} = (v_{\text{possible}} \cap v_{\text{current}})^{1 \text{st}} \text{ row} \text{ of } J_{o}
\end{equation}

The high order conflict resolution strategy has to be accomplished among elements from each v_{problem}, and the chosen resource-job not be fired must be cleared (set to zero) from u_{o} from m_{current} and restart second step.

To be prepared for this high-order conflict resolution strategy for the DEC implementation, one must decide among strictly trap-job sets contained in vectors J_o (for each i^{th} CW). That is, one must decide among resource-jobs contained in i^{th} row from J_o when the following condition holds

\begin{equation}
||J_{o,i}|| \Rightarrow ||C_{out}||, \text{ (26)}
\end{equation}

for ||J_{o,i}|| be the number of jobs contained in J_o present in the current v_{possible} vector from the i^{th} CW. In words, condition (26) verifies condition m_{i}(J_o(CW))<m_{o}(CW) for the attempted firing vector v_{possible}, and if (26) holds, resolution among jobs in vector v_{problem} has to be solved. Such resource-jobs are contained in the same i^{th} row from J_o.

4.2 IDENTIFYING CRITICAL AND KEY RESOURCES, SECOND LEVEL DEADLOCK.

There is a certain pathological case which requires extreme care in deadlock-avoidance dispatching. This case occurs where there exist the so called Second Level Deadlock (SLD) [7] structures in the system. SLD exist on the presence of Critical Resources (CR) and Key Resources (KR) [6]. These structures are identified by analyzing interdependencies in circular wait relationships and its siphons [6,32] and/or by accomplishing digraph connectivity-analysis among circular waits or cycles on resources and jobs [7,34]. We refer to CR as the structured-bottleneck resources [32], not the well known timed-bottleneck resources. We now define these structures using matrices.

The following refinements are needed later to define CR and KR in matrix form. These sets indicate ‘one’ on every entry (i,j) for places that belong to that set existing in every i^{th} CW, C_i.

The trap-job set for every i^{th} CW C_o, defined as J_o(C_i) = J(C_i) \cap T_{C_i}^{\dagger}, is computed in the i^{th} row of the matrix

\begin{equation}
J_o = T_m S_v^{T} \text{. (27)}
\end{equation}

The siphon-trap-job sets, S_{Q}(C_i) = J(C_i) \cap J_{o}(C_i), are the intersection of the siphon-job and trap-job sets, which in matrix notation, they are

\begin{equation}
J_{o} = J(C_i) \cap J_{o}(C_i) \text{. (28)}
\end{equation}

The strictly siphon-job set is defined as J (C_i) = J(C_i) \cap J_{o}(C_i), and in matrix form it is

\begin{equation}
J_{S} = J(C_i) \cap J_{o}(C_i) \text{. (29)}
\end{equation}

The strictly trap-job set is defined as J (C_i) = J(C_i) \cap J_{o}(C_i), and in matrix form it is

\begin{equation}
J_{T} = J(C_i) \cap J_{o}(C_i) \text{. (30)}
\end{equation}

The neutral-job set is defined as J (C_i) = J(C_i) \cap J_{o}(C_i), and in matrix form it is

\begin{equation}
J_{N} = J(C_i) \cap J_{o}(C_i) \text{. (31)}
\end{equation}

For the calculation of CR, we will need to identify the precedent transitions T_{pos}(C_i) and the posterior transitions T_{pos}(C_i) for the associated critical subsystems. They are defined as

\begin{align}
T_{pre}(C_i) = \{ J_{o}(C_i) \cap J_{o}(C_i) \}, \text{ and (32)}
\end{align}

These earlier described critical subsystems, C_{R_i}(C_i), which are needed in our deadlock-free algorithm, can also be defined from these quantities for all CW by

\begin{align}
J_{o} = J_{o} \cap J_{o} \text{. (33)}
\end{align}

To define critical resources (CR), we must determine the presence of cyclic circular wait (CCW) loops in the DE system. These specify a particular combination of circular waits that needs special attention for deadlock-free dispatching [6,32] and are a requisite for existence of CR. To identify whether if the system has cyclic circular wait (CCW) loops, let C_i and C_j be two circular waits with

\begin{align}
T_{pos}(C_i) \cap T_{pre}(C_j) \neq \emptyset \text{ and } T_{pre}(C_i) \cap T_{pre}(C_j) \neq \emptyset. \text{ (36)}
\end{align}

If that is the case, then we got a CCW. The matrix test to find CCW loops among all CWs is

\begin{equation}
C_{CCW} = (T_{pre} T_{pos}) \cap (T_{pre} T_{pos}^{T}) \text{. (37)}
\end{equation}

If we define ||C_i|| = Σc_{ij}, for c_{ij} be the (i,j) element of matrix C_i, then, if ||C_{CCW}|| > 0 we have an irregular system, otherwise, the system is regular. If we have CCW loops, C_{CC} is a symmetric matrix having non-zero elements in each c_{ccw_{ij}}, for CW i (indicated by row i from C_{CCW}) and CW j (column) be respectively C_i and C_j from (19).
The intersections of transitions that interconnect such CCWs are needed to define CR. We can use matrix $C_{CW}$ and the precedent and posterior matrix transitions $T_{pre}$ and $T_{pos}$ to identify such transitions.

$$\hat{T}_{pre} = (C_{CW} \cap T_{pre})$$

$$\hat{T}_{pos} = (C_{CW} \cap T_{pos})$$

We call them the cyclic precedent and cyclic posterior transitions, (38) and (39) respectively.

The definition of a Critical CW is as follows: Given PN $N$, and its initial marking $m_0$, let $\{C_i, C_j\}$ be a CCW such that $|C_i \cap C_j| = 1$, and let $C_i \cap C_j = \{r_c\}$, obviously, $r_c$ is a shared resource. Then, if $T_{pos}(C_i) \cap T_{pre}(C_j) \subseteq r_c^*$ and $T_{pre}(C_i) \cap T_{pos}(C_j) \subseteq r_c^*$, then $\{C_i, C_j\}$ is said to be a critical CW ($CW_{rc}$) and $r_c$ is called its critical resource. If in addition $m_0(r_c) = 1$, then $r_c$ is called a key resource.

Then, since we already identify the cyclic precedent and cyclic posterior transitions for all CCW in the system, we can proceed to identify the critical resources using the following straightforward matrix formula

$$Res_{CW} = (\hat{T}_{pos} F_r) \cap (\hat{T}_{pre} F_r)$$

$Res_{CW}$ provides, for each CW, the set of critical resources shared with other CWs in one or more CCW. If this matrix is zero, there are no critical resources and hence no key resources. Further analysis of shared critical resources will be described in a forthcoming paper soon.

5 IMPLEMENTATION

Consider the Multipart Reentrant Flow-line problem shown in figure 3. The Intelligent Material Handling cell from ARRI is capable to ‘manufacture’ the products from this MRF problem. We use term ‘manufacture’ since our machines are simulated machines. However, our resources are real robotic systems which we intend to perform the MRF sequence problem. The PN structure corresponding to this MRF system is shown in figure 4. The $F_r$, $S^T_r$, $F_r$, and $S^T_r$ matrices for this PN structure are shown in LabVIEW discrete format in figure 5 (each black circle in matrices is a ‘one’ and the others are ‘zero’, the $i^{th}$ rows are the corresponding $t_i$ transitions for each system.)

5.1 DETERMINING THE FMRF STRUCTURE

To calculate the regularity of the system, one can develop steps from sections 3 and 4.2. The conclusion of these constructions is that this system is regular. An example showing identification of an irregular system is provided in [36]. In the remaining of this paper, we implement a deadlock-free dispatching policy for the regular system from figures 3-4.

We created a small program in LabVIEW that uses matrices $S_r^T$ and $F_r$, to internally calculate the digraph matrix $W$, (14), and use MATLAB script code to get the simple circular waits, $sCW$, for this system. The diagram/code for this small program is shown in figure 6. Not all MATLAB code is shown in this diagram because the MATLAB script window was resized, the complete MATLAB code is available at the web page http://arri.uta.edu/acs/jmireles/PhD_study.htm (it is called StringM.m) Figure 7 shows the representation of the LabVIEW function which diagram is figure 6. The digraph of resources, $W$ matrix is shown in figure 8. The output matrix, shown as $L$ (for loops) in figures 6 and 7, contains all simple circular waits (sCW) from the system and is shown in figure 9.

From the number of rows in matrices from figure 10, we can see that we found two sCW in system A, and three sCW in system B. System A has one sCW, sCW1a composed by R3 and M1 and other composed by resources R2 and M2, sCW2a. System B has one sCW, sCW1b, composed by M1 and R1, other composed by M2 and R2, sCW2b, and the third one composed by resources C1, C2, R1 and R2, sCW3b.

5.3 IMPLEMENTATION OF DEADLOCK AVOIDANCE POLICY ON REGULAR SYSTEMS

To implement the deadlock-free dispatching rules, we proceed to determine the constructions (15) to (22). Equation (15) needs the resulting matrix $C_{out}$, Gurel’s algorithm which calculates all composed circular waits. For our system A, $C_{out}$ is equal to matrix $sCW$, meaning that there is no shared resource interconnecting any sCW, and that the system A does not have composed circular waits, i.e. $G$ matrix is a $2x2$ identity matrix (see end of section 3.1.) Since we do not have shared resources among $sCW$, we can say that there is not CR in system A because critical resources are resources that interconnect $sCW$ via $tpos$ and $tpre$ transitions.

For the implementation of the deadlock-free policy, we now calculate the critical subsystems matrix $J_o$. Such matrix can be found by using the LabVIEW diagram shown in figure 10, which determines constructions (15) to (22). Figure 11 shows the resulting $J_o$ matrix.
**Figure 4.** Petri Net system structure.

**Figure 5.** $F_v$, $S_v^T$, $F_r$, $S_r^T$ matrices for system from fig. 3 (in LabVIEW discrete format).

**Figure 6.** Code for the Number-string algebra that calculates sCW. It does not show all MATLAB code (see appendix 1).
Figure 12 shows the implemented controller in LabVIEW diagram/code. This controller verifies condition (23), based on the calculated critical subsystems (22), for every attempting firing of state $x$ (10) which updates marking vector (9). In case the attempting state $x$ fails condition (23), the controller denies the specific firing, by denying the availability of the resource attempting to fire and fill the critical subsystem, i.e. empty the critical siphon which will lead to circular blocking, deadlock. A detailed explanation of this controller is provided in [36].

Figure 13 shows the Discrete Events of the implementation run for system from our example. This figure was obtained in real-time directly from our DE controller implementation in LabVIEW. On the figure the discrete duration of the robotic jobs are shown. Every line represents a discrete event for every robotic job, and it has only two states, high and low (or ON and OFF), meaning executing robotic job and not executing such robotic job respectively. Notice that for every robot/resource (R1x, R2x and R3x), only one robotic job goes high at any time.

6. Conclusions.
A deadlock avoidance supervisory DE controller was developed for multipart reentrant flow-line systems. This DE controller uses a rule-based matrix dispatching formulation. On-line deadlock-free dispatching rules were implemented by the DE matrix controller. This was accomplished by analyzing circular waits for possible deadlock situations while analyzing the so-called critical siphons, certain critical subsystems and the presence of critical resources. We also presented a matrix formulation that identifies critical and key resources shared among circular resource loops that lead to second level deadlock structure. In addition to deadlock-free dispatching rules, conflict resolution strategies are implemented by the DE controller in a multipart reentrant flow line. Technical information given included the development of the deadlock-free controller implemented in LabVIEW. Future research on second level deadlock structures having critical resources will be performed, as well as routing problems on free choice multipart reentrant flow lines will be analyzed.
Figure 12. Main loop of the DE controller implementation in LabVIEW.

Figure 13. LabVIEW real time implementation output of the DE controller.

References