A PETRI NET-BASED DEADLOCK CONTROL POLICY FOR FLEXIBLE ASSEMBLY SYSTEMS

Naiqi Wu, MengChu Zhou*, and Elzbieta Roszkowska#

Dept. of ME, Guangdong Univ. of Tech., Guangzhou 510090, P. R. China
*Dept. of ECE, New Jersey Institute of Technology, Newark, NJ 07102-1982, USA
#Institute of Engineering Cybernetics, Wroclaw Univ. of Tech., ul. Janiszew skiego
11/17, 50-372 Wroclaw, Poland

Abstract: This paper presents a Petri net-based method for deadlock control in flexible assembly systems (FAS). Instead of a process-oriented modeling method, a resource-oriented Petri net modeling method is used. A new control policy is thus formulated and proved to outperform the existing one in the literature. The deadlock problem in FAS is often more difficult since deadlock can arise from improper base component flow and part flow, as well as ill-synchronized assembly operations.

Keywords: Petri net, assembly system, deadlock avoidance, and modeling and analysis

1. INTRODUCTION

Deadlock resolution in automated manufacturing systems (AMS) has been widely studied, for example (Zhou and DiCesare, 1991; Ezpeleta, et al., 1995; Banaszak and Krogh, 1990; Lawley, 1999; Fantì and Zhou, 2004). Recently, the deadlock problem in another AMS class characterized by non-sequential resource requirement became a hot topic (Fantì, et al., 1997; Hsieh, 2004; and Roszkowska, 2004). Flexible assembly system (FAS) is such a system. In FAS, parts need to be mounted on base components of the products. Often, bases and pallets are presented and delivered separately with parts. Both bases and parts take spaces in the assembly process. They cannot be finished without each other. Fantì, et al. (1997) and Hsieh (2004) studied the deadlock avoidance problem in FAS by using digraphs and Petri net, respectively. An assembly process is realizable if and only if there exists a feasible execution sequence (Roszkowska and Wojcik, 1993). By using Petri nets, they studied FAS with fork/join material flow, and proved that the realizability problem is NP-complete and it is NP-hard to find the maximum permissive deadlock avoidance policy. A control policy is proposed based on buffer space reservation (Roszkowska, 2004). This paper studies the same problem. Resource-oriented Petri net (ROPN) (Wu and Zhou, 2001; 2004a) is used to model it. An algorithm to calculate the realizable resource requirements and deadlock avoidance control policy are presented.

2. A FLEXIBLE ASSEMBLY SYSTEM

An FAS in Fig. 2.1 is adopted from (Roszkowska, 2004). It has two robots r1,2 and three workstations w1,3,5. There are an input buffer b0 and output buffers b1 for w1, b2 for w2, and b3 for w3. Between them, b4 can be accessed by w1 and w2, and b5 by w2 and w3. The robots handle the transport of materials from/to system input/output and between buffers. r1 delivers trays with parts and unpalletized subassemblies, while r2 handles pallets with base. In the assembly process, trays with parts and subassemblies are held in b4 or b5, and pallets with base are put onto b3,5.

When a workstation performs an operation, it may take the parts in a tray at b4 or b5 and mounts them onto the base in b3,5. Notice that workstations themselves cannot hold any component.

A and B-products are assembled concurrently with their assembly processes in Fig. 2.2, where “tray” denotes trays with parts, and “base” denotes base components. Take A-part as an example. After r1 takes a tray with parts to be mounted onto the base into b4, r2 transports a base into b0, then w1 mounts the parts in b4 onto the base in b0, and the finished one is moved into b1, while the tray with other parts remains in b4. Then r2 can deliver the base into b2 and
w₂ performs an operation on it, and after that it remains in b₃. It can then be delivered into b₁ by r₂. If, at the same time, the tray in b₄ is delivered into b₅ by r₁, then w₃ is ready to perform its assembly. After that, the finished product remains in b₅ and the tray in b₅, is ready to perform its assembly. After that, the buffer spaces are then released. When w₃ does its operation, it takes two parts from b₅, one is delivered to b₂, and the other from the central storage. It can then be delivered into b₃ by r₂. If, w₂ performs an operation on it, and after that it remains in b₂. It can then be delivered into b₃ by r₂, and other parts from the central storage are put into b₃ and b₄, from b₄, and the other from the central storage.

\[ N = \{0, 1, 2, \ldots\}; \]
\[ O: P \times T \rightarrow N \text{ is an output function}; \]
\[ M: P \rightarrow N \text{ is a marking representing the numbers of tokens in places with } M₀ \text{ denoting the initial marking}; \]
\[ K: P \rightarrow \{1, 2, 3, \ldots\} \text{ is a capacity function where } K(p) \text{ represents the maximal number of tokens that place } p \text{ can hold at a time}. \]

The \textit{preset} of transition \( t \) is the set of all input places to \( t \), i.e., \( \tau(t) = \{p \in P: I(p, t) > 0\} \). The \textit{postset} of \( t \) is the set of all output places from \( t \), i.e., \( \tau'(t) = \{p \in P: O(p, t) > 0\} \). Similarly, \( p \)'s \textit{preset} \( \tau(p) = \{t \in T: O(t, p) > 0\} \) and \textit{postset} \( \tau'(p) = \{t \in T: I(t, p) > 0\} \).

\textbf{Definition 3.1}: A transition \( t \in T \) in a finite capacity PN is enabled if for all \( p \in P \),
\[ M(p) \geq I(p, t) \] and
\[ K(p) \geq M(p) - I(p, t) + O(p, t) \]
If a transition is enabled, it can fire. Firing an enabled transition \( t \) in marking \( M \) yields
\[ M'(p) = M(p) - I(p, t) + O(p, t) \]
Definition 3.1 means that \( t \) is enabled and can fire if all the places in \( t \) have enough tokens and all the places in \( t \) have enough free spaces. Thereafter, when condition (1) is met, \( t \) is said to be process-enabled. When condition (2) is met, \( t \) is resource-enabled. Thus, \( t \) is enabled only if it is both process and resource-enabled. A sequence of firings results in a sequence of markings. \( M_i \) is said to be reachable from \( M_0 \) if there exists a sequence of firings that transforms \( M_0 \) to \( M_i \). The set of all markings reachable from \( M_0 \) is denoted by \( R(M_0) \). A place \( p \) is said to be full if \( M(p) = K(p) \). A transition in a PN is live if it can fire at least once in some firing sequence for every marking \( M \) reachable from \( M_0 \). A PN is live if every transition is live. The liveness of a PN assures that all events or activities in the model can happen. It implies the deadlock-free operation of an FAS if its PN model is live.

\[ \text{PN Models for Resources: Buffers are different from workstations and robots. The former is called H-resource and the latter G-resource. An H-resource is modeled by an H-place as shown in Fig. 3.1(a). A token in it represents a part occupying a space in the buffer, no matter whether it is just sitting there or being processed. Its self-loop transition, i.e., } t_5, \text{ represents assembly. A G-resource is modeled by a G-place in Fig. 3.1(b). Unlike an H-place, a token in it represents that the G-resource is available. Firing } t \text{ in Fig. 3.1(b) implies that the resource performs an operation. After its firing, the token comes back and the resource becomes available again.} \]

\[ \text{PN Models for Individual Products: The first basic operation is material delivery by robots. It can be} \]
modeled by a G-resource model in Fig. 3.1 (c). Firing \( t \) implies a part moved from \( p_1 \) to \( p_2 \) by \( r \).

There are four situations for the operations performed by a workstation as modeled and explained in Fig. 3.2. Place \( w \) models a G-resource and others model H-resources. These PN models are called primitives. Consider Fig. 3.2(d). It indicates that when the assembly is completed the base and part still occupy the buffer spaces. However, if this operation is the last one for a product, then the completed product can be moved away and will no longer occupy buffer spaces. Thus, this model becomes Fig. 3.3 (b).

![Fig. 3.2. PN primitives: (a) \( w \) processes a base in \( p \); (b) \( w \) assembles all the parts in \( p_1 \) to base in \( p_2 \); (c) \( w \) assembles some parts in \( p_2 \) to base in \( p_1 \) and moves it to \( p_3 \); and (d) \( w \) assembles some parts in \( p_2 \) to base in \( p_1 \), after that they remain in \( p_1 \) and \( p_2 \).](image)

By using Fig. 3.1-3, we present ROPN for individual products. For easy understanding, we use \( b_i, i = 0, 1, \ldots, 5 \), to name the places corresponding to buffers, \( w_{1,3} \) for three workstations, and \( r_{1,2} \) for two robots. Place \( p_0 \) represents the central storage for FAS, which hosts all the raw and final pieces. It is shown in (Wu, 1997) that by modeling the G-resources in this way deadlock resulting from processes performed by G-resources can be eliminated. Thus, in the sense of deadlock avoidance, these G-places and their associated arcs can be removed from the model. In this way, the PNs for assembling products A and B are obtained as shown in Figs. 3.4 (a) and (b) named A and B-subnets for short.

**ROPN for the Whole System**: We can obtain the ROPN for the whole system by merging the PNs for the individual products (Wu and Zhou, 2001). The union of the two subnets in Fig. 3.4 leads to Fig. 3.5. We use solid, dashed, and bold-solid lines to denote the individual part flows for products A and B, and shared one, respectively.

![Fig. 3.5. The ROPN for the whole system](image)

The ROPN obtained is compact, but cannot describe the material flow exactly. For example, when a token is in \( b_2 \), one may not know which of \( t_8, t_{16}, \) or \( t_{17} \) it enables. This problem can be solved by introducing colors into ROPN resulting in colored ROPN (CROPN). When we obtain the ROPN by merging the subnets we allow only one transition between two places and in the same direction, so some transitions come from only one subnet, others are from multiple subnets. If \( t \in T \) is from a single subnet, we call \( t \) a nonshared transition. If it is from \( k \geq 2 \) subnets, we call it \( k \)-shared. For example, \( t_1 \) and \( t_{20} \) are nonshared, but \( t_2, t_3, \) and \( t_9 \) are 2-shared.

**Definition 3.1**: \( C(t_i) = \{c_i\} \) is the only color for a nonshared transition \( t_i \in T \), and \( C(t_i) = \{c_{iA}, c_{iB}, \ldots, c_{iK}\} \) for a multiple-shared transition.

A multiple-shared transition represents the assembly of multi-products. The assembly of different products is distinguished by the transition’s colors.

**Definition 3.2**: If \( t_i \in p_i \), the tokens in place \( p_i \) enabling \( t_i \) have the color \( f_i \) associated with color \( c_i \).

In Fig. 3.5, \( t_1, t_5, t_6, t_8, t_{14}, t_{15}, t_{17}, \) and \( t_{20} \) are single-colored, but \( t_2, t_3, \) and \( t_9 \) have two colors named as \( c_{2A} \) and \( c_{2B} \), \( c_{3A} \) and \( c_{3B} \), and \( c_{9A} \) and \( c_{9B} \), respectively. For example, if there is a token in \( b_1 \) representing an A-base with color \( f_{AB1} \), then \( t_5 \) with color \( c_5 \) can fire, but if it represents a B-base with color \( f_{BB4} \), then \( t_{20} \) with color \( c_{20} \) can fire. Transition \( t_3 \) has two colors, when there is a token in \( b_0 \) representing an A-base with color \( f_{AB1} \) and an A-part...
in b4 with color fAP1, t3 can fire with color c3A. Note that all the tokens in p0 should have colors. For simplicity, we just put α in p0 to indicate α tokens.

The color of a token may change. However, such change occurs only when an assembly transition fires. In Fig. 3.5, in p0, color fAB1 is used for a token representing an A-base enabling t1, fAP1 for an A-part enabling t9, fBP1 for a B-base enabling t2, fAPi for a B-part enabling t9, and fBP2 for a B-part enabling t9. When a token with color fAP1 enters b0 by firing t1, and a token with color fAP1 comes to b4 by firing t12 with color c2A, these two tokens then enable t3 with color c3A. After firing t3, their colors change into fAB2 and fAP2, respectively.

4. REALIZABLE RESOURCE REQUIREMENT

Given a set of product types to be processed in FAS concurrently, the set of buffers with their capacities, the FAS layout, and product routes, can each operation be executed at least once? This problem is called realizability problem that is NP-complete (Roszkowska, 2004). This implies that to find a minimal realizable resource requirement (RRR) is NP-complete. However, RRR is needed as a necessary condition for deadlock control. This paper intends to find a simple algorithm for RRR. To complete the assembly of a product, its base must first be released into the system. Nevertheless, RRR can be reduced if a part to be mounted onto a base is released into the system just before FAS needs it.

Definition 4.1: Assume that in CROPN of FAS, ∃t, ∃*t = {p0}, pα ∈t, and tα ∈pα, where tα is an assembly transition with color c. To fire tα, it requires tokens in pα, i = 1, 2, ..., k, k ≥ 1, with color αi, and tokens with color cαi that move into pα by firing t. In marking M, ∃i, ∃M(pα)(ai) > I(pα, ta)(ai, c) and t fires, then firing t is a premature where M(pα)(ai) is the number of tokens of color αi in pα and I(pα, ta)(ai, c) represents the number of arcs from pα to tα with respect to token color αi and transition color c.

For example, to perform the final assembly of A, there should be an A-base in b3, an A-part from b4, and another A-part from p0. If the former two items are not ready at b3 and b4, firing t9 to release A-parts into b4 is of no use for the assembly process but occupies buffer spaces. Thus, such premature firing of t9 should be avoided. Assume that there are β types of products to be assembled, product i needs di parts to be mounted onto its base. Thus, if FAS is required to assembly one product for each type, there are totally α = β + \sum di parts including β bases.

Algorithm 4.1: Find RRR for assembling one product of each type concurrently.

Step 1: Initialization: let K(p) = \infty for all p ∈ P = {p0, p1, p2, ..., pn} in the CROPN, put α tokens with their colors in p0 and set the marking of the CROPN M = M0 with all tokens in p0, and let R[] = 0 be an m-vector with integer values;

Step 2: Find T E ⊆ T such that ∀t ∈ T E is enabled in the current marking M. If T E = ∅ then stop, otherwise go to the next step;

Step 3: Find the set of transitions T EF, such that ∀t ∈ T EF whose firing is premature, and set T E = T E - T EF do the following:

1) Form a firing sequence f in T F, and the order can be set arbitrarily;
2) Calculate M' = M'f, and let M = M';
3) For i = 1 to m do
   If M(pi) > R[i], R[i] = M(pi);
4) Go to Step 2.

When algorithm 4.1 ends, vector R[] returns RRR.

Example 1: Find RRR in the CROPN shown in Fig. 3.6 via Algorithm 4.1. We obtain RRR for b0, b1, b2, b3, b4, and b5 are 1, 2, 1, 1, 2, and 1, respectively.

Base on Algorithm 4.1, we know that for the set of products to be processed in FAS all the operations can be executed at least once. Thus, according to the definition of realizability in (Roszkowska, 2004), the assembly process is realizable.

Property 4.1: For the given set of product types, the assembly process is realizable with the buffer capacity obtained by Algorithm 4.1.

Assume that there are m places (buffers) and n transitions in CROPN of FAS. Then Algorithm 4.1’s computational complexity is as follows.

Property 4.2: The computational complexity to find the realizable buffer capacity requirements by Algorithm 4.1 is O(αm(n + m)).

The solution found by Algorithm 4.1 is shown to be feasible for the execution of a single product for each type of product. However, we still do not know if it is feasible when there are multiple products for each type. By Algorithm 4.1, we can find R[i] by releasing one product of a type i into the CROPN for product type i. Based on R[i], i ∈ {1, 2, ..., β}, RRR for deadlock avoidance with multiple products in FAS can be calculated as follows.

Definition 4.2: Place p is said to be an assembly place if ∃t ∈ P such that t is an assembly transition. Further, if t is for the assembly of product type i, it is said that i is a product type assembled in p.

For an assembly place p, let AS(p) denote the set of product types that are assembled in p, NAS(p) denote the set of product types that use p in the assembly process but are not assembled in p. Clearly, we have AS(p) ∩ NAS(p) = ∅. If AS(p) = ∅, p is said to be an non-assembly place. Let Z[k] = \sum R[i]k, i ∈ AS(p) and W[k] = max(R[k]), j ∈ NAS(p), then R[k] = Z[k] + W[k] is the minimal resource requirement for deadlock control with multiple products in the system, where k is the kth place.

Example 2: Find RRR for deadlock control in the CROPN shown in Fig. 3.5. From the CROPNs shown in Fig. 3.4 (a) and (b), we obtain R[1] = (1, 1, 1, 1, 1,
2) and $R_B = (1, 1, 1, 1, 1, 1)$. Based on Definition 4.2 we obtain $R[] = (2, 1, 2, 2, 3, 3)$ that is smaller than $(2, 1, 2, 2, 3, 3)$ obtained in (Roszkowska, 2004).

5. DEADLOCK AVOIDANCE POLICY

We assume that the buffer capacity in the system is greater than or equal to $R[]$. Deadlock due to base component and part flows takes place in circuits. In Fig. 3.5, if circuits $C_1 = \{b_0, t_3, b_1, t_5, b_2, t_{17}, b_0\}$ and $C_2 = \{b_2, t_6, b_3, t_{13}, b_2\}$ are full, deadlock occurs. Consider a subnet shown in Fig. 5.1 where $K(p_i) = 1, i=1-4$. Assume that $t_3$ is an assembly transition for A-products. It requires an A-base and an A-part in $p_1$ and $p_4$, respectively. If at some time, $p_3$ has A-base and $p_4$ B-parts, although there may be B-base in $p_1$ and A-parts in $p_4$, assembly cannot be done and a deadlock occurs. Hence, to avoid deadlock in FAS we should avoid deadlock resulting from base component flow, part flow and assembly.

![Fig. 5.1. PN subnet for assembly operations](image-url)

The buffers belonging to a workstation can be seen as a special set of buffers denoted by $B_i$ for workstation $i$. It may contain only one buffer. $B_i$ is called a buffer group. We also notice that there is some $t \in p_0$ and $t^*$ is an input place of an assembly transition such that firing $t$ releases a part into the system. Such a set of transitions is denoted by $T_{ray}$. For example, in the CROPN shown in Fig. 3.8, $t_2$ and $t_6$ are such transitions.

**WZ-Policy**: A transition $t \in T$ in CROPN for FAS in marking $M$ is enabled and its firing changes $M$ to $M'$. Then $t$ can fire only if all the following conditions are met:

1) Assume that there are $k$ groups of buffers $B_{1-k}$ for the base. Then in $M'$ there are at least $k-1$ free buffer spaces such that at most one group $B_i$ is full;
2) Assume that there are $k$ buffers $b_{1-k}$ for the parts to be mounted onto the base. Then in $M'$ there are at least $k-1$ free buffer spaces such that at most one buffer is full;
3) Assume $t \notin T_{ray}$, and its firing moves $V_h$ tokens of type-h product into $p_i$. Let $M(p_i)(k)$ denote the number of tokens representing type k product in $p_i$ in $M$. Further assume $t_3 \in T_{ray}$, and its firing moves $U_h$ tokens of type-h product into $p_i$. Then
   a) $K(p_i) - W[i] - \max(M(p_i)(k), R_i[i]) - M(p_i)(b) - \sum_{h \in AS(i)} U_h \geq V_h$, if $h \in AS(i)$;
   b) $K(p_i) - Z[i] - M(p_i)(b) \geq V_h$, if $h \in NAS(i)$;
4) Assume $t \in T_{ray}$, its firing moves $U_h$ tokens of type-h product with color $C_1$ into $p_i$, and these tokens together with $V_h$ tokens of type-h product in $p_j$ and $Y_h$ tokens with color $C_2$ in $p_i$ enable assembly transition $t_a$. Then $M(p_i)(h) \geq V_h$, $M(p_i)(C_2) \geq Y_h$, and $M(p_i)(C_1) = 0$.

Conditions 1-2 ensure that the base and part flows will not be deadlocked, respectively. Condition 3 ensures that the state shown in Fig. 5.1 can never occur. Condition 4 avoids any premature firing so that the buffer spaces are effectively used. Conditions 3 and 4 make no restriction on an non-assembly place.

**Theorem 5.1**: The CROPN for an FAS is deadlock-free if WZ-policy is applied.

**Proof**: See (Wu and Zhou, 2004b).

Although WZ-policy is a group of sufficient conditions, it is not too conservative. For we require only $n-1$ free space for the base component and part transportation, respectively, with $n$ groups of buffers. In general, in FAS each buffer has multiple spaces. Hence the restriction imposed is not too much. We impose less restriction on the number of parts in FAS, rather restrict the ratio of parts among the types.

**Theorem 5.2**: WZ-policy is less conservative than R-policy (Roszkowska, 2004).

**Proof**: See (Wu and Zhou, 2004b).

**Theorem 5.3**: The complexity for implementing WZ-policy is $O(|P|\beta)$, where $P$ is the place set in CROPN and $\beta$ is the number of product types.

6. ILLUSTRATIVE EXAMPLE

**Example 3**: Consider the FAS shown in Fig. 2.1. The capacity of $b_{0,5}$ is 2, 4, 4, 3, 6, and 5, respectively. The CROPN is shown in Fig. 3.5. By using R-policy (Roszkowska, 2004), a marking is reached such that 1, 1, 3 A-bases are in $b_{0,1}$, $b_{2,1}$, and $b_{3,2}$, respectively. 3, 3 A-parts are in $b_{0,2}$, $b_{5,2}$, respectively. 1, 1, 1 B-bases are in $b_{0,3}$, $b_{1,3}$, and $b_{5,3}$, respectively. 1, 1 B-parts are in $b_{0,4}$, $b_{5,4}$, respectively. In this marking $t_{12}$ is forbidden by R-policy. This implies that no more jobs can be released into the system.

Consider the system under WZ-policy. From Section 4, $R_A = (1, 1, 1, 1, 1, 2)$ and $R_B = (1, 1, 1, 1, 1, 2)$. Places $b_0$ and $b_{2,5}$ are assembly places. In this marking, it is easy to verify that Condition 3) is satisfied for both products A and B. There are three groups of places for the base flow $B_1 = \{b_{0,1}, b_{1}, b_{2,1}\}$, $B_2 = \{b_{2}\}$, and $B_3 = \{b_{3}\}$, Condition 1 is satisfied at this marking. The free spaces in $b_{0,1}$ and $b_{5,2}$ are 1 and 1, or Condition 2 is also satisfied. This implies that this marking is a legal marking for WZ-policy and can be reached. However, there are more free spaces in the buffers than what is required by WZ-policy, so more parts can be released into the system if other conditions are satisfied. In fact, in this marking, $t_{12}$ is
enabled and firing it does not violate WZ-policy, so it can fire.

In fact, we have \( \sum_{i=0}^{3} K(b_i) = 13 \). By Condition 1, only two free spaces are needed. This implies that at most 11 products can be released into the system for assembly. Assume that initially \( M_0(p_0) > 0 \) and for any \( i \), \( M_0(b_i) = 0 \), the evolution from \( M_0 \) is presented in Table 6.1 where the number in the brackets after a pace represents the capacity of the place. In the table, \( \varnothing \) is used for a token of A-base with color \( f_{ABi} \) and \( \ominus \) for a token of A-base with color \( f_{AB2} \); \( \ominus \) for A-part with color \( f_{APi} \); \( \varnothing \) for A-part with color \( f_{APi} \); \( \ominus \) for B-base with color \( f_{BB1} \); \( \ominus \) for B-base with color \( f_{BB2} \); \( \ominus \) for B-base with color \( f_{BB3} \); \( \ominus \) for B-base with color \( f_{BB4} \); \( \ominus \) for B-part with color \( f_{BP1} \); \( \ominus \) for B-part with color \( f_{BP2} \), and \( \ominus \) for B-part with color \( f_{BP3} \). It is shown that totally, 11 products are released into the system for assembly. R-policy allows only 9 products to be released into it. This example shows Theorem 5.2’s correctness.

Table 6.1. The token evolution from \( M_0 \)

<table>
<thead>
<tr>
<th>Marking</th>
<th>Type</th>
<th>( b_2(2) )</th>
<th>( b_3(4) )</th>
<th>( b_4(3) )</th>
<th>( b_5(6) )</th>
<th>( b_6(5) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( M_1 )</td>
<td>A</td>
<td>( \varnothing )</td>
<td>( \varnothing )</td>
<td>( \ominus )</td>
<td>( \ominus )</td>
<td>( \ominus )</td>
</tr>
<tr>
<td>B</td>
<td>( \ominus )</td>
<td>( \ominus )</td>
<td>( \varnothing )</td>
<td>( \varnothing )</td>
<td>( \varnothing )</td>
<td>( \varnothing )</td>
</tr>
<tr>
<td>( M_2 )</td>
<td>A</td>
<td>( \varnothing )</td>
<td>( \varnothing )</td>
<td>( \ominus )</td>
<td>( \ominus )</td>
<td>( \ominus )</td>
</tr>
<tr>
<td>B</td>
<td>( \ominus )</td>
<td>( \ominus )</td>
<td>( \varnothing )</td>
<td>( \varnothing )</td>
<td>( \varnothing )</td>
<td>( \varnothing )</td>
</tr>
<tr>
<td>( M_3 )</td>
<td>A</td>
<td>( \varnothing )</td>
<td>( \varnothing )</td>
<td>( \ominus )</td>
<td>( \ominus )</td>
<td>( \ominus )</td>
</tr>
<tr>
<td>B</td>
<td>( \ominus )</td>
<td>( \ominus )</td>
<td>( \varnothing )</td>
<td>( \varnothing )</td>
<td>( \varnothing )</td>
<td>( \varnothing )</td>
</tr>
<tr>
<td>( M_4 )</td>
<td>A</td>
<td>( \varnothing )</td>
<td>( \varnothing )</td>
<td>( \ominus )</td>
<td>( \ominus )</td>
<td>( \ominus )</td>
</tr>
<tr>
<td>B</td>
<td>( \ominus )</td>
<td>( \ominus )</td>
<td>( \varnothing )</td>
<td>( \varnothing )</td>
<td>( \varnothing )</td>
<td>( \varnothing )</td>
</tr>
<tr>
<td>( M_5 )</td>
<td>A</td>
<td>( \varnothing )</td>
<td>( \varnothing )</td>
<td>( \ominus )</td>
<td>( \ominus )</td>
<td>( \ominus )</td>
</tr>
<tr>
<td>B</td>
<td>( \ominus )</td>
<td>( \ominus )</td>
<td>( \varnothing )</td>
<td>( \varnothing )</td>
<td>( \varnothing )</td>
<td>( \varnothing )</td>
</tr>
<tr>
<td>( M_6 )</td>
<td>A</td>
<td>( \varnothing )</td>
<td>( \varnothing )</td>
<td>( \ominus )</td>
<td>( \ominus )</td>
<td>( \ominus )</td>
</tr>
<tr>
<td>B</td>
<td>( \ominus )</td>
<td>( \ominus )</td>
<td>( \varnothing )</td>
<td>( \varnothing )</td>
<td>( \varnothing )</td>
<td>( \varnothing )</td>
</tr>
</tbody>
</table>

7. CONCLUSIONS

In FAS, base components are transported with pallets, and parts to be mounted onto them are transported with no pallets but via trays. When an assembly operation is performed by using some parts in a tray, the tray still occupies a buffer space. In this way, an assembly/disassembly material flow is formed. In such systems, deadlock can occur in the base component flow, part flow and assembly. Thus, it is a great challenge to avoid deadlock in FAS. This paper proposes to use resource-oriented Petri nets to capture the discrete event dynamics of FAS concisely and presents a deadlock control policy. The policy is computationally efficient and better than the existing one. It may be used to FAS in (Hsieh, 2004).

REFERENCES


