International Journal of Management Science
Vol 3, No 2, November 1997

SPECIFICATION AND CONTROLLER SYNTHESIS FOR THE HIERARCHICAL CONTROL OF FMS*

JIN-TAE CHANG
Consulting Operations, Samsung Data Systems, Korea

HUN-TAI KIM
Department of Industrial Engineering, Daejin University, Korea

SUK-HO KANG**
Department of Industrial Engineering, Seoul National University, Korea

ABSTRACT

Developing FMS controllers has been a difficult problem largely because of the variety of the system configuration. The purpose of this paper is to develop a method of building an FMS controller. The controller consists of control module and execution module. A hierarchically layered structure of these modules is proposed. The control module generates abstract-level execution requested by identifying a set of activities that can be executed without creating any irregular state. The execution module transmits the requests to physical device controllers and reports back the completion of the requests to the control module. Both of these two modules use Petri Net-based models. In this paper, a controllable Petri Net model is automatically synthesized from declarative specifications provided by a user. An execution Petri Net model for the execution module is designed to ensure the consistency between the control module and the real target system. The controller operates in MMS on TCP/IP and UNIX environment.

1. INTRODUCTION

An important perspective is that the successful implementation and operation of an FMS often lies on the development of safe and effective control software[5, 6]. An FMS would be composed of several components provided by different

* This research was supported by Korea Science and Engineering Foundation grant 941-1000-021-2.
** Dept. of I.E. S.N.U. San 56-1 Shillim-Dong Kwanak-Gu Seoul 151-742 Korea

vendors. Consequently, it has many variations in its configurations. This complexity and diversity can be partly managed by a formal and generic control model.

Petri Nets has largely been studied as a control model for FMSs. But to build an Petri Control Model is very hard work and to test completeness of the model is also difficult [1,2,4,7,8]. The purpose of this research is to develop a method to synthesize a Petri Net control model for an FMS directly from declarative user specifications that guarantee the safe operations of the system. The proposed controller consists of control module, execution module and deadlock avoidance module.

This paper introduce a method to generate the control module using contents of the informations that users support. The structure of the execution module is presented. The role of the execution module is to maintain integrity between the control module and physical FMS composed of MMS devices. MMS is sets of ISO/OSI protocol supporting manufacturing processes[3].

The format of this paper is as follows. First, overall control structure is described in chapter 2. Then required informations to build the control model are suggested in Chapter 3. A method to synthesize a controllable Petri Net is described in chapter 4. Execution structure is defined in chapter 5. Finally, conclusions in the chapter 6.

2. OVERALL STRUCTURE

Figure 1 describes the overall structure of proposed FMS contoller. The controller keeps state informations and send scheduler a set of available activities that can be executed without creating any irregular states. Transition chosen by scheduler is transformed to physical operations by the controller and the executions of them are managed.

The controller consists of deadlock avoidance module, *CtrlPPN* (Controllable Production Petri Net), *eXePPN* (Execution Production Petri Net), and *COM* (Composite Operation Monitor).

The roles of *CtrlPPN* are as follows.

1) To maintain a system status in the control model and find the set of physically executable operations.
2) To receive a safe transition from the deadlock avoidance module.
3) To command the operation of selected transition and uncontrollable transitions to execution module.

eXePPN maintains integrity between logical information of *CtrlPPN* and

physical FMS system via *COM*. *COM* converts the logical information of the *eXePPN* to operation information that can be recoginized by MMS.

Addressed in this paper are those elements in the shaded region which includes *CtrlPPN* and *eXePPN*. Required informations and method to synthesize *CtrlPPN* and the structure of *eXePPN* are detailed in following three chapters.

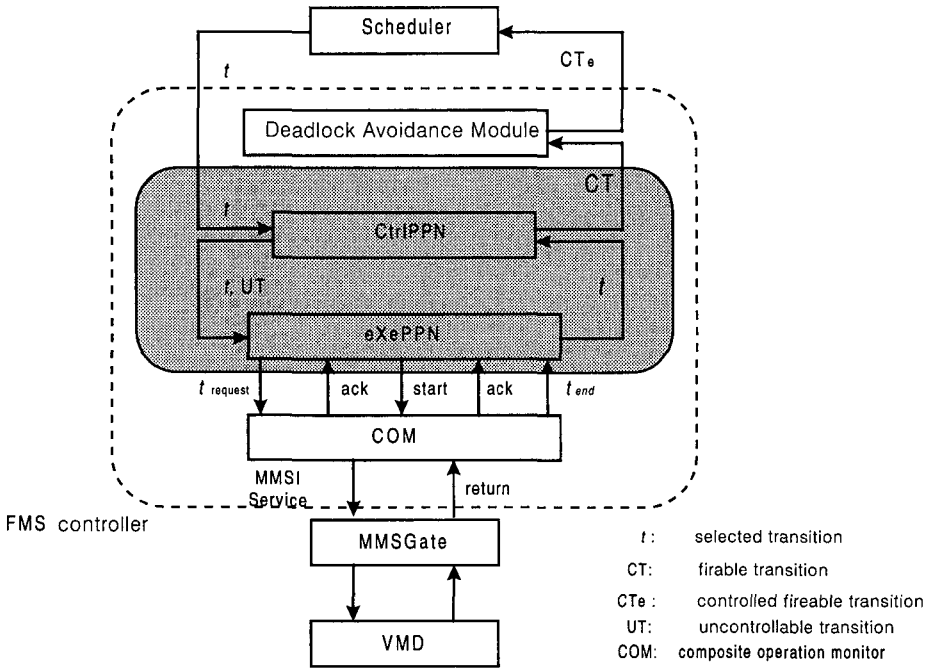


Figure 1. Overall control structure

3. REQUIRED INFORMATIONS

Developing a proposed controller requires some informations that should be indetified by user such as the components of the system, parts, operations and part route, moving paths between locations, and part programs. To recognize the information given by user, six domains are defined. Each domain can have subdomains.

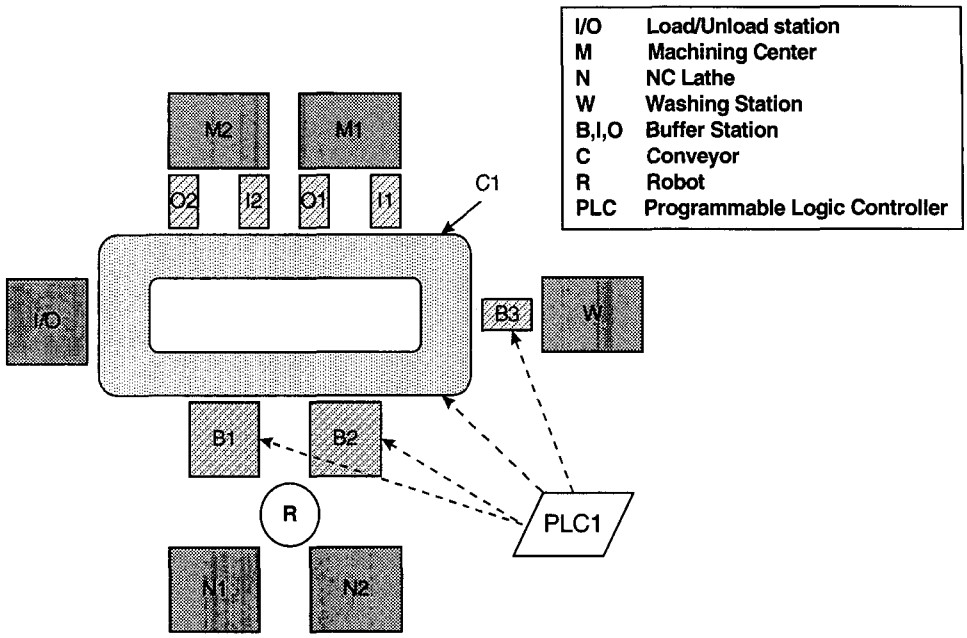


Figure 2. An FMS example

For convenience shake, we will use an FMS example described in figure 2. N1 and N2 are NC lathes, M1 and M2 are machining centers. I1 and I2 represent input buffer station for each machining center, O1 and O2 are output buffer station. Each maching center controls each buffer station. B1 is an input and output buffer station for N1 and B2 is for N2. Robot R sets up a part on B1 or B2 to N1 or N2. W is a washing station and B3 is a buffer station for W. I/O is an load/unload station. C1 is a conveyor. PLC1 is a programmable logic controller that governs C1, B1, B2, and B3.

3.1. Resources

Resources mean production equipments such as machines, material handlers, buffers or storages, and controllers. Each resource has a different name and can retain only one part or pallet at a time. *RSC* is a domain for resources. *RSC* has subdomains as follows.

MH: to represent material handlers such as a robot for moving a part, conveyor, and AGVs. In our example, R, C1, I1, I2, O1, O2, B3 are in *MH* . But B1 and B2 are not in *MH* .

MF: to represent resources that process the operatations of part routes. Again, *MF* has subdomains of *MF_W_P* and *MF_W_O_P* . If the part should be manufactured fixed on a pallet at a machine, this machine is in *MF_W_P* domain. If not, this machine is in *MF_W_O_P* domain. In our example I/O, M1, M2, W are *MF_W_P* domain resources and N1, N2 are *MF_W_O_P* domain resources. In the case of I1, I2, O1, O2, B3, they are in both *MF* and *MH* domain. This kind of resources are classified into *MFH* domain.

ST: to represent resources that can be used as a part storage. In our example, B1, B2, B3, I1, I2, O1, O2 are included in *ST* domain.

CTL: to represent programmable logic controller or machine controller itself. *C(RSC)* is a function of resource to map corresponding controller. In our example, N1, N2, M1, M2, I/O, W, PLC1 are in *CTL* domain. $C(R)=R$, $C(C1)=PLC1$, $C(B1)=PLC1$, $C(B2)=PLC1$, $C(B3)=PLC1$, $C(N1)=N1$, $C(N2)=N2$, $C(I/O)=I/O$, $C(M1)=M1$, $C(M2)=M2$, $C(I1)=M1$, $C(I2)=M2$, $C(O1)=M1$, $C(O2)=M2$.

3.2 Unit Paths

Let's define effective location first. Effective location is a place on which part or pallet can reside excluding the resource included in *MH* or *CTL* domains. *EL* is a set of all effective locations. $R(EL_i)$ means a corresponding resource for effective location EL_i. A unit path is a path between two effective locations that can be directly connected by a material handling resources. *U_PATH* represents domain for unit paths. $up(EL_i, EL_j)$ is a unit path between effective location EL_i and EL_j. A path is a set of sequences that consist of stream of unit paths. Let a path between EL_i and EL_j be $path(EL_i, EL_j)$. In our exmple, a path from I/O to N1 is as follows.

$path(I/O, N1) = ((up(I/O, B1), up(B1, N1)), (up(I/O, B2), up(B2, N1)))$.

U_PATH has two subdomains. If we can get a information from the resource A that a part is transfromed from EL_i to a material handling resource

Table 1. Unit Paths of the example FMS
* represents the unit path is included in U_PATH_ACK domain

from \ to	I/O	B1	B2	B3	N1	N2	W	M1	I1	O1	M2	I2	O2
I/O		C1 *	C1 *	C1 *					C1 *			C1 *	
B1	C1 *			C1 *	R	R							
B2	C1 *			C1 *	R	R							
B3	C1 *	C1 *	C1 *				B3		C1 *			C1 *	
N1		R	R										
N2		R	R										
W				B3									
M1										O1			
I1								I1					
O1	C1 *			C1 *								C1 *	
M2													O2
I2											I2		
02	C1 *			C1 *					C1 *				

and another part can occupy the resource A, the unit path is included in U_PATH_ACK domain. In the other case the unit path is in included in $U\_PATH\_NO\_ACK$. $R_{MH}(up_i)$ represents a MH domain resource in the case of $up_i \in U\_PATH$. Table 1 illustrates the unit paths of the example FMS.

3.3 Parts

$PART$ is a domain for part.

3.4 Operations

Operation is a logical term that represents a set of manufacturing processes in a certain resource. Each operation is unique in the same system. However, this does not mean that every operation consists of different part programs. OP is a domain for operations. Two type of subdomains are defined. One is T_OP and the other is O_OP. T_OP domain is concerned with part moving operations. An operation included in T_OP domain is defined as a tuple

$(op, rsc, up, pp, next\_op, prg)$. op is a operation identifier. rsc is a resource that operation is undertaken. up is an unit path. pp is a part route. $next\_op$ is a succeeding O_OP domain operation. prg is a part program. O_OP is a subdomain for manufacturing processes. An operation included in O_OP domain is defined as (op, rsc, prg).

3.5 Part Programs

We assume that a part program for an operation is not processed in two machines at the same time. P is a domain for part programs. An operation consists of a set of part programs. These part programs have partial orders. Let I_P is an index set of part programs in P. We define $prg_k = (V_{prg}, E_{prg})$, $k \in I_P$ as a ADG(Acyclic Directed Graph) that represents partial orders between part programs. V_{prg} is a set of nodes for part programs and E_{prg} is a set of directed arcs representing direct precedence between part programs. $v \in V_{prg}$ contains a information for the controller which processes the part program. Figure 3 decribes a part program graph for an T_OP domain operation.

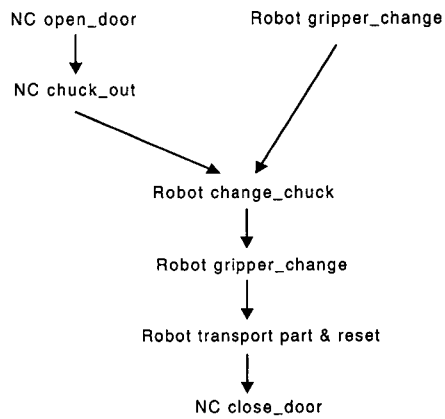


Figure 3. An example part program graph for T_OP domain operation

3.6 Part Routes

A part routes is a sequence of O_OP domain operations for a part. PR is a domain for part route. There are partial orders between operations in a part route. In this paper, a part route is represented as a OR-graph. Let $PR_j = (V_{O\_OP}, E_{O\_OP})$ be a graph for the part route of $part_j$, where $j \in I_{PR}$ and I_{PR} is a set of index of part routes in PR. V_{O_OP} is a set of nodes for

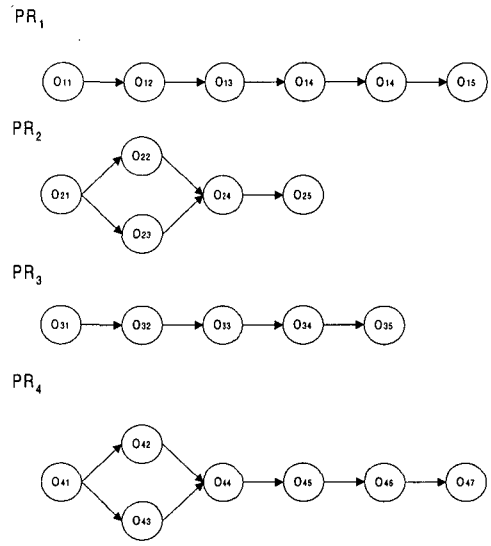


Figure 4. Part routes of the example FMS

O_OP domain operations and E_{O_OP} is a set of directed arcs representing the precedence between operations in a part route. Figure 4 represents part routes in the example FMS. O_{11}, O_{21}, O_{31}, O_{41} are loading operations at I/O station. O_{15}, O_{35}, O_{47} are unloading operations at I/O station. O_{12}, O_{22} are manufacturing operations at N1. O_{14}, O_{23} are operations at N2. O_{32}, O_{42} are opeations at M1. O_{33}, O_{43}, O_{45} are operations at M2. O_{13}, O_{14}, O_{24}, O_{34}, O_{44}, O_{46} are washing operations at W.

4. Control Model

In this chapter, a method to develop an *CtrlPPN* is described.

4.1 Unit path Graph

$AG=(V_{AG}, E_{AG})$ is an Accessibility Graph that represents an unit path between effective locations. This can be built from unit paths and effective location informations. Physical moving path of a part can be found using this graph.

$v \in V_{AG}$ is a node that represents an effective location, $label(v)$ mean the name of the effective location. $v \in V_{AG}$ has an operation information of

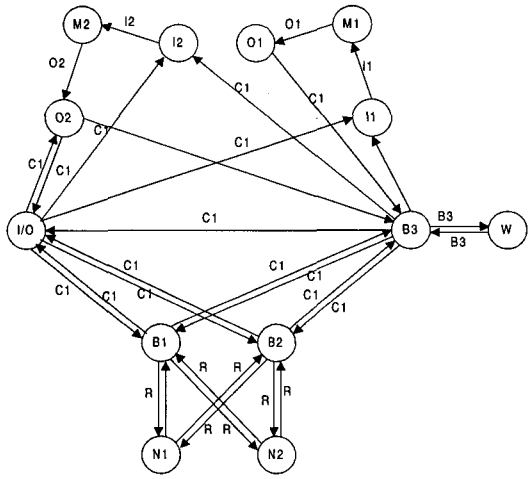


Figure 5. AG of the example FMS

O_OP domain and $op(v)$ is the name of $v \in V_{AG}$. $e \in E_{AG}$ is an arc defined when a unit path exists between the effective locations. $label(e)$ is name of an MH domain resource. Figure 5 is an AG of the example.

4.2 Part move path graph

$LG_j = (V_{LG_j}, E_{LG_j})$, $j \in I_{PR}$ is a graph that represents the move path for each part P_j that has a part route PR_j. LG_j is created based on the PR_j and AG. $LG = \bigcup_{j \in I_{PR}} LG_j$. $v \in V_{LG}$ is a node that represents an effective location and contains the resource information for that location. For each $u \in V_{AG}$, $label(v) = label(u)$ and $op(v) = op(u)$. $e \in E_{LG}$ represents a unit path to an effective location that is required to process the next operation in the part route. For each $a \in E_{AG}$, $label(e) = label(a)$. $pl_{LG}(e)$ retains the index information of the part route, that is, $pl_{LG}(e) = j$ for $e \in E_{LG_j}$. $LG_j = (V_{LG_j}, E_{LG_j})$ can be generated by procedure 1.

Procedure 1. to create LG

step 1. Define $v \in V_{LG_j}$ for each $u \in V_{PR_j}$. $label(v) = label(u)$ and $op(v) = op(u)$

step 2. For each (v_a, e_i, v_b), $v_a, v_b \in V_{PR_j}, e_i \in E_{PR_j}$ of AG ,

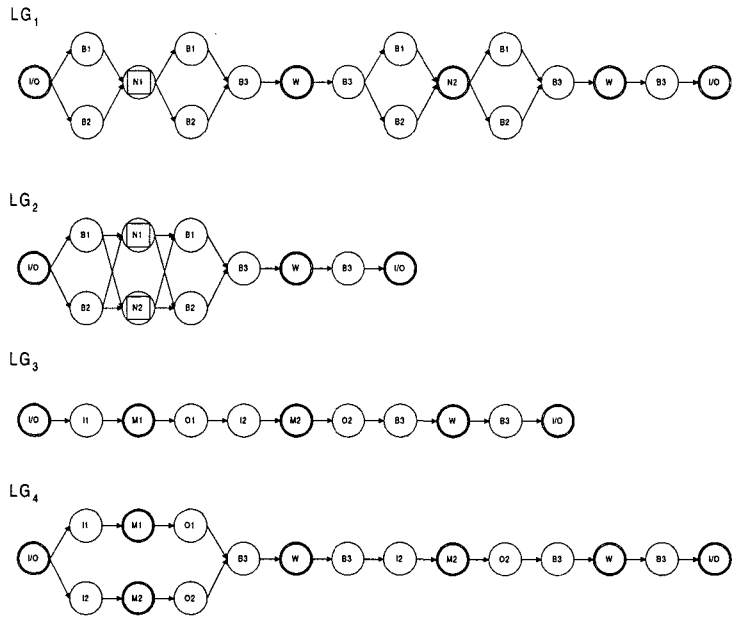


Figure 6. LG of the FMS example

find $path(label(v_a), label(v_b)) = \{path_1, \ldots, path_l\}$ such that,
$path_k = up(label(v_a), E_{k_1}), up(E_{k_1}, E_{k_2}), \ldots up(E_{k_n}, label(v_b))$, $k = \{1, \ldots, l\}$.

step 3. For each $k = \{1, \ldots, l\}$, create node v_{k_r} between v_a and v_b such that $label(v_{k_r}) = E_{k_r}$, $r = \{1, \ldots, n\}$, and connect arc $e_{k_a}, e_{k_b}, e_{k_r}$ for each between v_a and v_{k_1}, $v_{k_n}$and v_b, v_r and v_{r+1}, $(r = \{1, \ldots, n-1\})$.
$label(e_{k_a}) = up(label(v_a, E_{k_1}))$, $label(e_{k_b}) = up(E_{k_n}, label(v_b))$,
$label(e_{k_r}) = up(E_{k_r}, E_{k_{r+1}})$. $pl_{LG}(e) = j$ for each e.

Figure 6 represents the LG of the example. Double circles, squares surrounded by circles, and circles represents effective locations of which resources are included in MF_W_P, $MF\_W\_O\_P$, ST for each.

4.3 Production Petri Nets : *PPN*

PPN called as production petri net is a Petri Net whose places mean effective locations and transitions represent operations. This can be formalized as follows.

$$PPN_j = (P_{PPN_j}, T_{PPN_j}, I_{PPN_j}, O_{PPN_j})$$

$$P_{PPN} = P_{EL} = P_{EL_{MF}} \bigcup P_{EL_{ST}}$$
$$P_{EL_{MF}} = P_{EL_{MF\_W\_P}} \bigcup P_{EL_{MF\_W\_O\_P}}$$
$$PPN = \bigcup_{j \in I_{PR}} PPN_j$$

PPN_j is transformed from LG_j. P_{PPN} is a set of places and T_{PPN} is a set of transitions. I_{PPN} is a set of input arcs from transitions to places and O_{PPN} is from places to transitions. $P_{EL_{MF\_W\_P}}$, $P_{EL_{MF\_W\_O\_P}}$, $P_{EL_{ST}}$ are sets of places for each MF_W_P, $MF\_W\_O\_P$, ST domain resources. Each place has a information about the efftective location for each node of LG_j and each transition has informations about the unit path and the index for part route. $I(t_i, p_k)$ is an input arc from place p_k to transition t_i. $O(t_i, p_k)$ is an input arc from transition t_i to place p_k. $\cdot t_i$ is a set of places that have $I(t_i, p.)$ and $t_i \cdot$ is a set of places that have $O(t_i, p.)$. $\cdot p_k$ is a set of transitions that have $O(t., p_k)$ and $p_k \cdot$ is a set of transitions that have $I(t., p_k)$. $el : P_{PPN} \rightarrow EL$ is a function from places to effective locations. PPN_j can be generated by Procedure 2.

Procedure 2. to generate PPN_j

step 1. Let $A = \{v_1, \ldots, v_n\}$ $v_i \in V_{LG_j}$ is a set of nodes such as $R(label(v_i)) \in MF\_W\_O\_P$. $\cdot A_k = \{v_{(k,1)}, \ldots, v_{(k,m)}\}$ is a set of precedence nodes for node $v_k \in A$ that is directely connected by some arcs with v_k and $A_k \cdot = \{v_{(k,m+1)}, \ldots, v_{(k,2m)}\}$ is a set of succeeding node for $v_k \in A$ that is directely connected by some arcs with v_k. Let $label(\cdot A_k)$ ia set of label for each node in $\cdot A_k$. $|\cdot A_k| = |A_k \cdot|$ and $label(\cdot A_k) = label(A_k \cdot)$. Remove all acrs that are connected with node v_k and create nodes $v_k^1, \ldots, v_k^m$.
$label(v_{(k,r)}) = label(v_k^r) = label(v_{(k,r+m)})$. Create arcs from $v_{(k,r)}$ to v_k^r and from v_k^r to $v_{(k,r+m)}$. Let's call this new graph as $LG_j' = (V_{LG_j}', E_{LG_j}')$.

step 2. Define $p_i \in P_{PPN_j}$ for each node $v_i \in V_{LG_j}'$. $el(p_i) = label(v_i)$, $op(p_i) = op(v_i)$

step 3. Let $V^S_{LG_j}$ is a set of starting nodes and $V^F_{LG_j}$ is a set of ending nodes of LG_j Create transition t_{source_x} for each starting node $v_x \in V^S_{LG_j}$ and transition t_{sink_y} for each ending node $v_y \in V^F_{LG_j}$ Connect arcs such as $t_{source_x} \cdot = v_x$, $\cdot t_{sink_y} = v_y$.
Create t_k for each tuple (v_i, e_k, v_l), $v_i, v_l \in V_{LG_j}'$, $e_k \in E_{LG_j}'$ such that $\cdot t_k = p_i$, $t_k \cdot = p_l$. Let up_{t_k} be a unit path information and $pl_{PPN}(t_k)$ be an index information of part route retained by t_k. $up_{t_k} = label(e_k)$ and $pl_{PPN}(t_k) = pl_{LG}(e_k)$.

Figure 7 represents PPNs obtained by applying procedure 2 to LG of firgure 6. Squares surrounded by circles represent the places included in $P_{EL_{MF\_W\_O\_P}}$ domain, double circles are places included in $P_{EL_{MF\_W\_P}}$ domain and single circles means the palces included in $P_{EL_{ST}}$ domain.

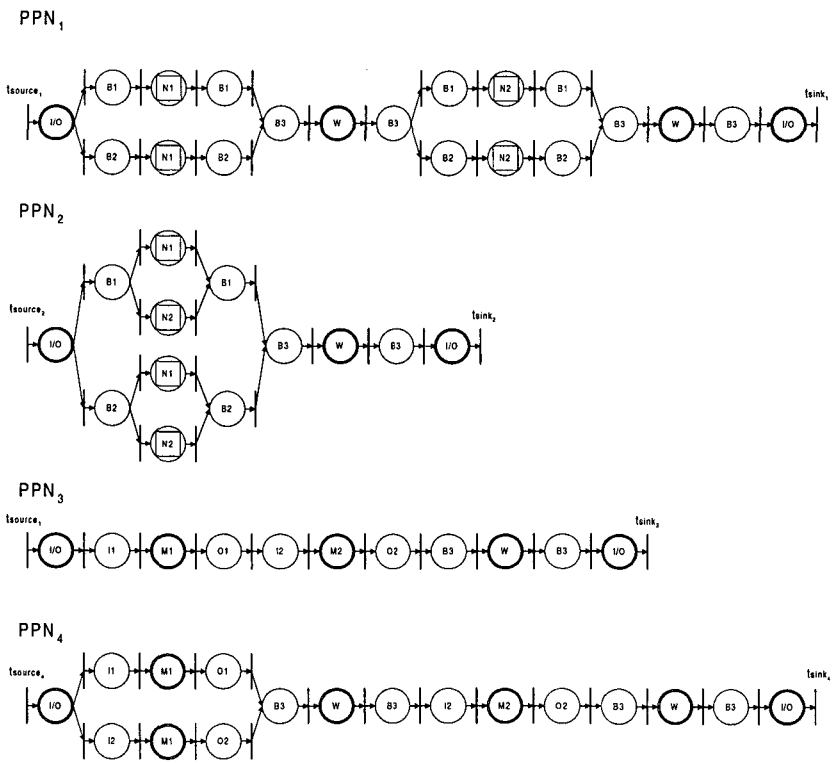


Figure 7. PPN for FMS example

4.4 Uncontrolled Production Petri Net : *UPPN*

UPPN is a Petri Net which is generated from each PPN_j based on the shared resources. *UPPN* is defined formally as follows.

$$UPPN=(P_{UPPN}, T_{UPPN}, I_{UPPN}, O_{UPPN})$$
$$P_{UPPN}=P_{PPN}\cup P_{R(EL)}\cup P_{MH}\cup P_{R(MH)}$$
$$P_{MH}=P_{MH\_ACK}\cup P_{MH\_NO\_ACK}$$

UPPN guarantees the mutual exclustion of shrared resources using the information of the binary status of resources. $P_{R(EL)}$ is a set of places to represent ST and MF. $P_{R(MH)}$ is a set of places to represent MH domain resources. P_{MH} is a set of places to represent part moving. P_{MH_ACK} is a set of places from which we can get a information about the availability of the resource for EL_i when a part has been occupied that resource. For a place in $P_{MH\_NO\_ACK}$, we cannot get that kind of information. *UPPN* can be generated by prodecure 3.

Procedure 3. to generate *UPPN*

step 1. (to create resource places $P_{R(EL)}$ for each ST, MF domain resource)

For each PPN_j, $j\in I_{PR}$, repeats followings. Create $p_k^R\in P_{R(EL)}$, $p_k\in P_{PPN_j}$, for each (t_i, p_k, t_{i+1}), $t_i, t_{i+1}\in T_{PPN_j}$, $p_k\in P_{PPN_j}$. Make a token marking for each $p_k^R\in P_{R(EL)}$ and connect arcs.

i) If $p_k\in P_{EL_{ST}}$, (t_{i-1}, p_{k-1}, t_i) and $p_{k-1}\in P_{EL_{MF\_W\_O\_P}}$, then go to step 2.

ii) If p_{k+1} is included in $P_{EL_{MF\_W\_P}}$ for each $p_k\in P_{EL_{ST}}$, $(t_{i+1}, p_{k+1}, t_{i+2})$, create arc such that $\cdot p_k^R=\{t_{i+1}\}$, $p_k^R\cdot=\{t_i\}$, then go to step 2.

iii) If $p_k\in P_{EL_{MF}}$ then create an arc such that $\cdot p_k^R=\{t_{i+1}\}$, $p_k^R\cdot=\{t_i\}$, then go to step 2.

iv) If p_{k+1} is included in $P_{EL_{MF\_W\_O\_P}}$ for each $(t_{i+1}, p_{k+1}, t_{i+2})$, find p_w that is most precedent place in the succeeding places satisfying $R(el(p_k))=R(el(p_w))$ and create arc such that $\cdot p_k^R=\{t_{l+1}\}$, $R(p_k)\cdot=\{t_l\}$ for each (t_l, p_w, t_{l+1}).

Figure 8. (a) describes some part of the results of applying step 1 to

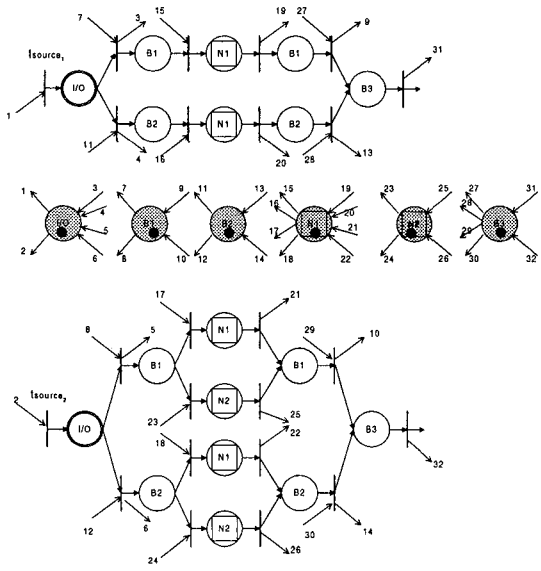
PPN_1 and PPN_2 in figure 7. The places filled with gray color represent resource places.

step 2. (to create a set of MH domain resource places and connect arcs)

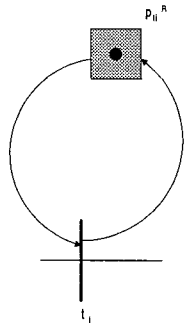
For each $j \in I_{PR}$ repeats followings. Create a domain resource place $p_{t_i}^R \in P_{R(MH)}$ if a MH resource place of up_{t_i} was not defined for t_i where $t_i \in T_{PPN_j}$ excluding t_{source_j} and t_{sink_j}. $\cdot\, p_{t_i}^R = \{t_i\}$, $p_{t_i}^R \cdot = \{t_i\}$. If $R_{MH}(up_{t_i}) \in MFH$, $R_{MH}(up_{t_i}) = R(p_k^R)$ and $P_k^R \in P_{R(EL)}$ was created in step 1 skip the process of this transition and go to next transition. Figure 8. (b) shows the step 2.

step 3. (to create a set of places to represent part moving and connect arcs)

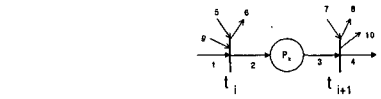
For each $j \in I_{PR}$, repeats followings. For each $t_i \in T_{PPN_j}$ excluding t_{source_j} and t_{sink_j}, Split t_i into $t_i^1 \in T_{UPPN}$ and $t_i^2 \in T_{UPPN}$ and create $p_i' \in P_{MH}$ that represents the state that part is moving. t_i^1 means the start of part moving and t_i^2 means the completion of part moving. Let upt_t be a unit path information retained $t \in T_{UPPN}$ and upp_p be a unit path information retained $p \in P_{MH}$. Then, $upt_{t_i^1} = up_{t_i}$, $upp_{p_i'} = up_{t_i}$. $pl_{UPPN}(t_i^1) = pl_{PPN}(t_i)$. If $upp_{p_i'}$ is included in U_PATH_ACK domain, p_j' is contained in P_{MH_ACK}, else



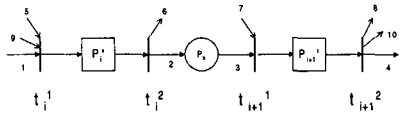
(a)



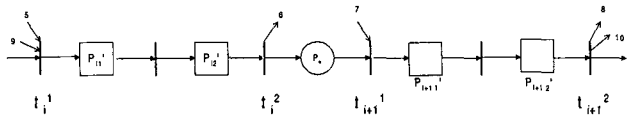
(b)



(1)

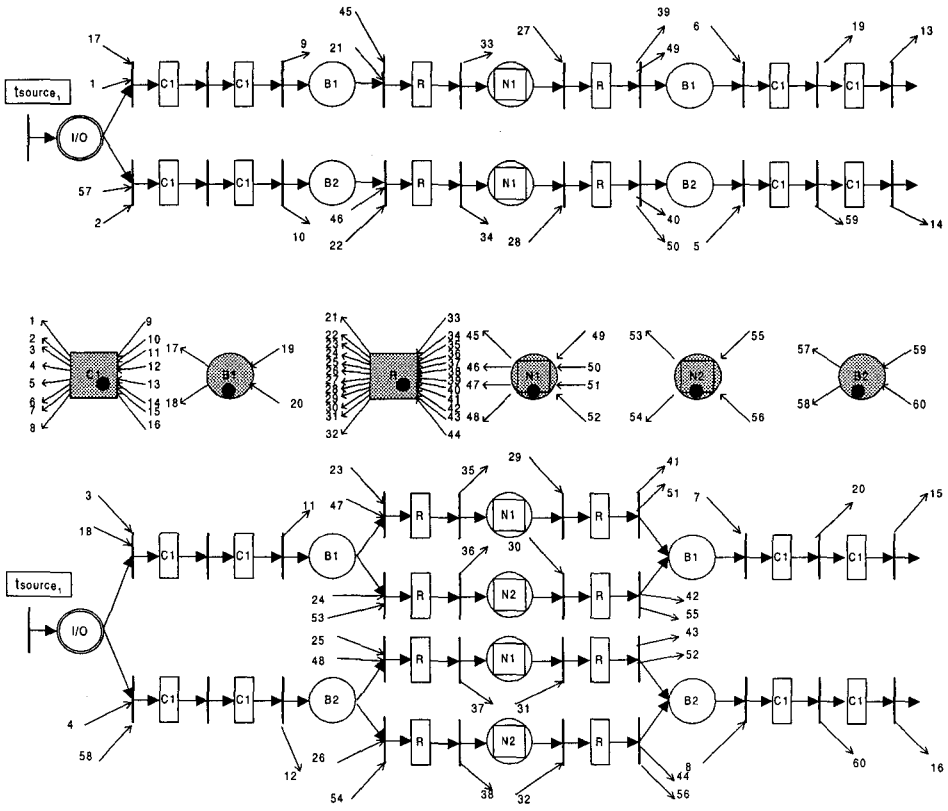


(2)



(3)

(c)



(d)

Figure 8. An example to create an UPPN

contained in $P_{MH\_NO\_ACK}$. If $p_i' \in P_{MH\_NO\_ACK}$, connect arc as i), if $p_i' \in P_{MH\_ACK}$, connect arc as ii).

i) $\cdot t_i^1 = \cdot t_i$, $t_i^1 \cdot = \{p_i'\}$, $\cdot t_i^2 = \{p_i'\}$, $t_i^2 \cdot = t_k \cdot$

For example, (t_i, p_k, t_{i+1}) in figure 8(c)(2) is transformed to $(t_i^1, p_i', t_i^2, p_k, t_{i+1}^1, p_{i+1}', t_{i+1}^2)$.

ii) Mark t_i^1 as UP_ACK_MARK. As like figure 8(c)(3), p_i' is splited into $p_{i_1}' \in P_{MH}$ and $p_{i_2}' \in P_{MH}$, arcs are connected as follows.

$\cdot p_{i_1}' = \{t_i^1\}$, $p_{i_1}' \cdot = \{t_i'\}$, $\cdot p_{i_2}' = \{t_i'\}$, $p_{i_2}' \cdot = \{t_i^2\}$

4.5 Controllable Production Petri Nets : *CtrlPPN*

CtrlPPN is obtained by adding some functional places to *UPPN*. *CtrlPPN* is defined formally as follows.

$$CtrlPPN = (P_{CtrlPPN}, T_{CtrlPPN}, I_{CtrlPPN}, O_{CtrlPPN})$$
$$P_{CtrlPPN} = P_{UPPN} \cup P_{O\_OP} \cup P_{CTRL} \cup P_{ACK}$$

P_{CTRL} is a set of control places, P_{ACK} is a set of response places that alert the state transition of *eXePPN*. P_{O_OP} is a set of places representing the state in which operation for a part in a *MF* domain resource is ready or operation is finished. A token in the contol place means that the corresponding transition is selected by Deadlock avoidance module as secure transition. Response place guarantees the run time integrity between the status of logical model and the status of real system. *CtrlPPN* can be generated by following procedure 4.

Procedure 4. to generate *CtrlPPN*

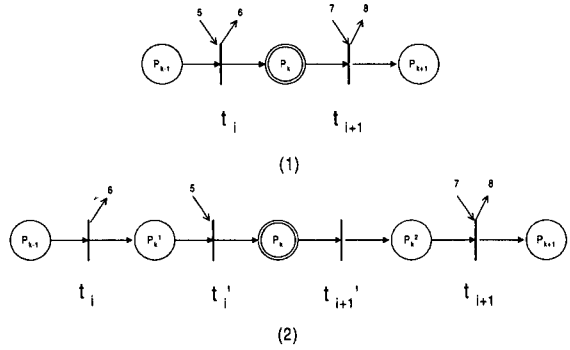
step 1. (to create a set of places P_{O_OP})

Create p_k^1, $p_k^2 \in P_{O\_OP}$ and t_i' ' $t_{i+1} \in T_{CtrlPPN}$ such that $p_{k-1}, p_{k+1} \in P_{MH}$ and $p_k \in P_{EL_{MF}}$ for each $(p_{k-1}, t_i, p_k, t_{i+1}, p_{k+1})$ and connect arcs as follows.

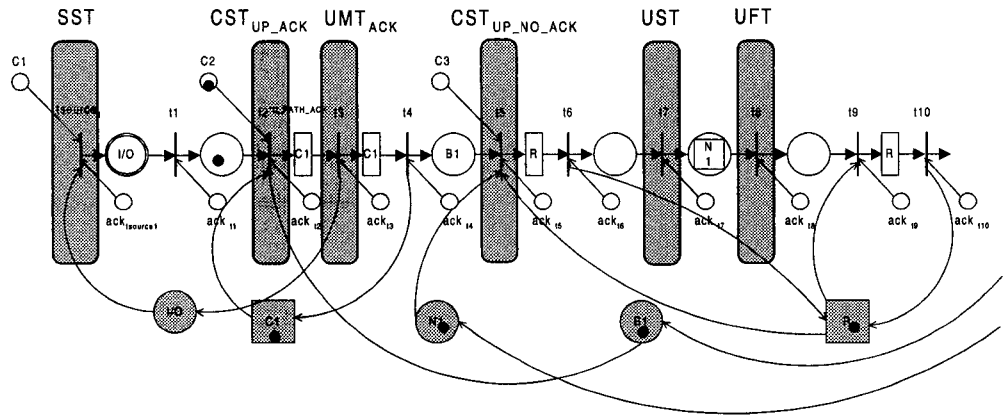
$\cdot t_i = \{p_{k-1}\}$, $t_i \cdot = t_i \cdot - \{p_k\} \cup \{p_k^1\}$, $\cdot t_{i'} = p_k^1 \cup (\cdot t_i - \{p_{k-1}\})$, $t_{i'} \cdot = \{p_k\}$,

$\cdot {}'t_{i+1} = \{p_k\}$, ${}'t_{i+1} = \{p_k^2\}$, $\cdot t_{i+1} = \cdot t_{i+1} - \{p_k\} \cup \{p_k^2\}$, $t_{i+1} \cdot = t_{i+1} \cdot$

In case of $t_i = t_{source}$, $\cdot t_i = \{\}$. If $t_{i+1} = t_{sink}$, $t_{i+1} \cdot = \{\}$.



(a)



(b)

Figure 9. An example of *CtrlPPN*

Figure 9(a) describes the step 1.

step 2. (to create a set of P_{CTRL} and a set of P_{ACK})

i) Create $p_i^c \in P_{CTRL}$ for each $t_i \in T_{CtrlPPN}$ and t_{source_j} such that $I(t_i, p_k^R)$, $p_k^R \in P_{R(EL)}$ are defined, and connect $I(t_i, p_i^c)$. p_i^c is marked when the t_i is decided as a safe firable transition from a deadlock avoidance module.

ii) Create $p_i^a \in P_{ACK}$ for all transtion $t_i \in T_{CtrlPPN}$, and connect $I(t_i, p_i^a)$. $p_i^a \in P_{ACK}$ is marked when *eXePPN* reports start or finish of certain operation. Figure 9(b) is an example of *CtrlPPN* that is generated from a part of figure 8(d) based on Procedure 4.

4.6 Transition Classes

Transitions of *CtrlPPN* can be devided into seven classes. Dividing factors are existence of a control place, type of succeeding place, *U_PATH_ACK* marking and t_{source}, t_{sink}. Figure 9(b) shows each one.

$CST_{UP\_NO\_ACK}$(Controllable Start Transition)

(Definition) A set of transitions $t \in T_{CtrlPPN}$ that satisfies following conditions.

1) no *U_PATH_ACK* making.
2) $\cdot t$ contains places such as $p^c \in P_{CTRL}$ and $p_k \in P_{EL} \bigcup P_{O\_OP}$

This type of transition represents the start of an *T_OP* domain operation. *CtrlPPN* sends a tuple $(t\_class, p_t^a, p_{t'}^a, up_t, pl_t, next\_o\_op)$ to *eXePPN*. *t_class* is the transition class, p_t^a is a response transition corresponding to t, $p_{t'}^a$ is a response transition of succeeding transition $t' \in UFT$, up_t is an unit path, pl_t is an index of part route and *next_o_op* is the first $op(p_k)$ of $p_k \in P_{EL_{MF}}$ that is found along the succeeding places of t.

UMT_{ACK} (Uncontrollable Moving Transition with ACK)

(Definition) A set of transitions $t \in T_{CtrlPPN}$ that has *U_PATH_ACK* mark.

Transition t is firable when the system reports that the corresponding resource is available.

CST_{UP_ACK}(Controllable Start Transition with U_PATH_ACK)

(Definition) A set of transitions $t \in T_{CtrlPPN}$ that satisfies following conditions.

1) with *U_PATH_ACK* marking.
2) $\cdot t$ contains places such as $p^c \in P_{CTRL}$ and $p_k \in P_{EL} \bigcup P_{O\_OP}$

CtrlPPN sends an information of tuple $(t\_class, p_t^a, p_{t'}^a, up_t, pl_t, next\_o\_op)$ to *eXePPN* to request the start of *T_OP* domain operation.

$p^a_{t'}$ is a response places of $t' \in UFT$ that is succeeding transition of $t' \in UMT_{ACK}$.

UST(Uncontrollable Start Transition)

(Definition) A set of transitions $t \in T_{CtrlPPN}$ that satisfies following conditions.

1) no U_PATH_ACK marking.
2) $\cdot t$ does not contains places $p^c \in P_{CTRL}$.
3) $t \cdot$ contains $p_k \in P_{EL_{MF}}$.

These transitions represent the start of O_OP domain operations. These transistions are executed automatically whenever firable. *CtrlPPN* sends an information of tuple $(t\_type, p^a_t, p^a_{t'}, op(p_k))$ to *eXePPN* to request the start of O_OP domain operation.

UMT_{ACK}(Uncontrollable Moving Transition with ACK)

(Definition) A set of transitions $t \in T_{CtrlPPN}$ that have U_PATH_ACK mark.

Transition t is firable when the system reports that corresponding resource is available. The response places of the transition t is preceding transition $t' \in CST_{UP\_ACK}$. *CtrlPPN* sends this informatin to *eXePPN*.

UFT(Uncontrollable Finish Transition)

(Definition) A set of transitions $t \in T_{CtrlPPN}$ that satisfies following conditions.

1) no U_PATH_ACK marking.
2) preceding transition t' is not contained in UMT_{ACK}
3) $\cdot t$ does not contains places $p^c \in P_{CTRL}$.
4) $t \cdot$ contains $p_k \in P_{EL_{MF}} \bigcup P_{MH}$.

SST(Source Start Transition)

(Definition) A set of t_{source} transitions

These transitions have control place, while not included in UST . If they are selected as execution transitions, they act like UST transitions.

5. Execution of Transition

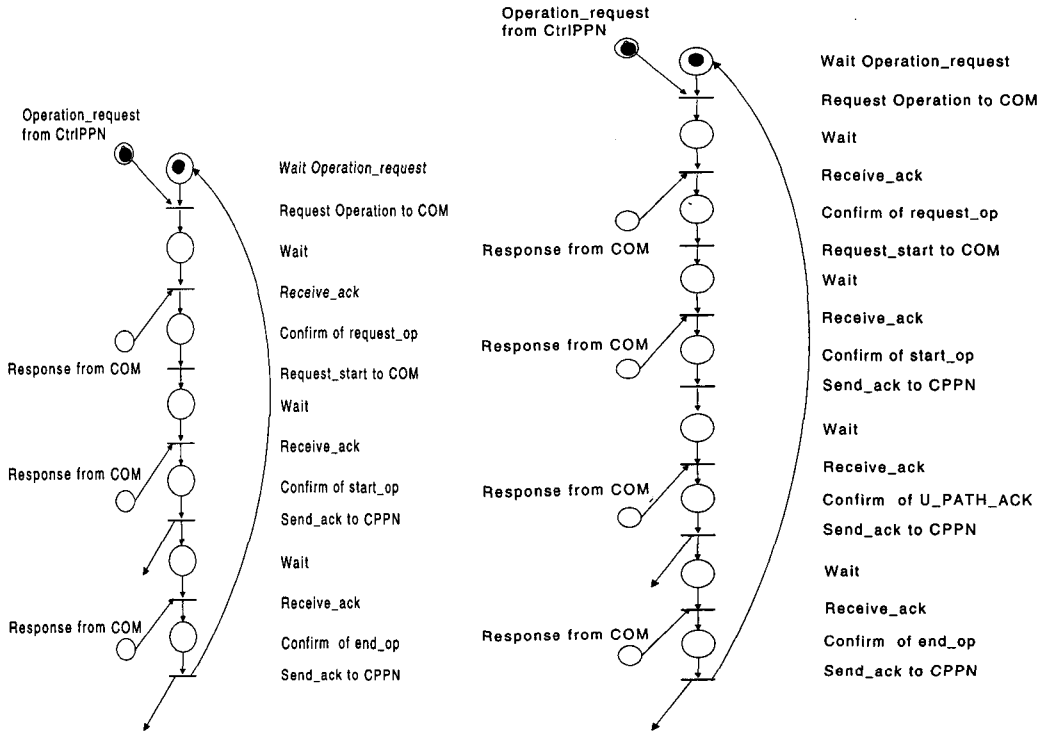
$eXePPN$ is a Petri Net representing a protocol to communicate with $CtrlPPN$ and COM. It performs control function for each corresponding transition class of $CtrlPPN$. There are two types of $eXePPNs$.

5.1 $eXePPN_{UP\_NO\_ACK}$

$eXePPN_{UP\_NO\_ACK}$ is operated when the calling transition class of $CtrlPPN$ is SST, $CST_{UP\_NO\_ACK}$ or UST . $eXePPN_{UP\_NO\_ACK}$ requests to COM to start a operation and wait response of accepting the start request. Recieving response, $eXePPN_{UP\_NO\_ACK}$ commands COM to start a operation and sends a response to $CtrlPPN$. Finishing the operation, COM sends a corresponding message, and $eXePPN_{UP\_NO\_ACK}$ alarms this result to p^a_t of $CtrlPPN$. The informations of start operation command for T_OP domain operation are $(up_t, pl_t, next\_o\_op)$. For O_OP domain operations $eXePPN_{UP\_NO\_ACK}$ sends $op(p_k)$. Figure 10 (a) describes $eXePPN_{UP\_NO\_ACK}$.

5.2 $eXePPN_{UP\_ACK}$

$eXePPN_{UP\_NO\_ACK}$ is operated when the calling transition class of $CtrlPPN$ is CST_{UP_ACK}. $eXePPN_{UP\_NO\_ACK}$ requests to COM to start a operation and wait response of accepting the start request. Recieving response, $eXePPN_{UP\_ACK}$ commands COM to start a operation and sends a response to $CtrlPPN$. Finishing the operation, COM sends a corresponding message, and $eXePPN_{UP\_ACK}$ alarm this result to p^a_t of $CtrlPPN$. COM sends another message that resource is available then $eXePPN_{UP\_NO\_ACK}$ alarm this result to p^a_t. of $CtrlPPN$. Figure 10 (b) describes $eXePPN_{UP\_ACK}$.



(a) $eXePPN_{UP\_NO\_ACK}$ (b) $eXePPN_{UP\_ACK}$

Figure 10. *eXePPN*

6. Conclusion

A systematic approach to generate FMS controller is developed. Required informations are identified and procedures are developed to synthesize of controllable Petri Net which is a logical model of the operation of FMS. Successive applications of four procedures transfrom informations on paths among resources and part routes to Controllable Petri Net.

The proposed controller has a hierarchically layered structure represented in figure 11. There is a consistency between layers. Controllable Production Petri Net and deadlock avoidance module guarantee deadlock-free and safe transitions in logical level. Execution Production Petri Net and Composite Operation Monitor maintain integrity between controllable Petri Net and physical FMS by managing the execution of physical operations associated with the logical transition.

Proposed controller was built and tested in MMS environment on UNIX host with TCP/IP as communication protocols.

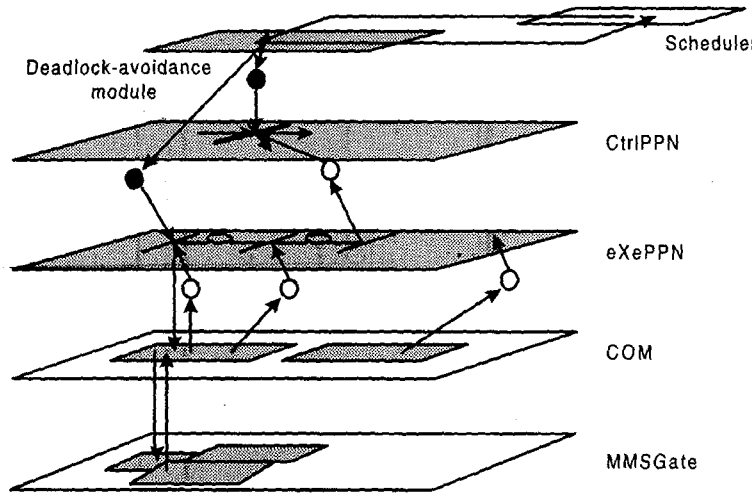


Figure 11. Hierarchical Control Structure

REFERENCES

[1] Giua, A. and F. DiCesare, "Petri Net Structural Analysis for Supervisory Control," *IEEE Trans. on Robotics and Automation*, Vol.10, No.2(1994), 185-195.

[2] Hsieh, F.S. and S.C. Chang, "Dispatching-Driven Deadlock Avoidance Controller Synthesis for Flexible Manufacturing Systems," *IEEE Trans. on Robotics and Automation*, Vol.10, No.2(1994), 196-209.

[3] ISO/IEC 9506-I *Industrial automation systems - Manufacturing Message Specification - Part 1: Service Definition*, ISO, 1990.

[4] Jeng, M.D., "Generation and Analysis of Synthesis Rules for Petri Nets with Applications to Manufacturing," *Proc. of IEEE Conf. on Systems, Man, and Cybernetics* (1994), San Antonio, Texas, 664-669.

[5] Mettala, E.G., R.A.Wysk & S.Joshi, "CIMGEN-A CASE Tool for CIM Development," *Proc. 3rd ORSA/TIMS Conference on FMS*, 1989.

[6] Naylor, A.W. and R.A. Volz, "Design of Integrated Manufacturing System Control Software," *IEEE Trans. on Systems, Man, and Cybernetics*, Vol.17, No.6(1987), 881-897.

[7] Viswanadham, N., Y. Narahari, and T.L. Johnson, "Deadlock Prevention and Deadlock Avoidance in Flexible Manufacturing Systems Using Petri Net Models," *IEEE Trans. on Robotics and Automation*, Vol.6, No.6(1990), 713-723.

[8] Zhou, M.C. and F. Dicesare, *Petri Net Synthesis for Discrete Event Control of Manufacturing Systems*, Kluwer Academic Publishers, Boston, 1993.