Deadlock Avoidance In Mixed Capacity Flexible Manufacturing Systems

by

Sridhar Mohan

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science
Department of Industrial and Management Systems Engineering
College of Engineering
University of South Florida

Co-Major Professor: Suresh Khator, Ph.D.
Co-Major Professor: Ali Yalcin, Ph.D.
Tapas K. Das, Ph.D.

Date of Approval:
July 8, 2004

Keywords: real time control, colored petri nets, polynomial complexity

DEDICATION

To all my friends and family

ACKNOWLEDGEMENTS

I would like to thank Dr. Suresh Khator and Dr. Ali Yalcin for their patience and assistance in completing this thesis. I would like to express my grateful thanks for the help and advice given by Dr. Ali Yalcin who has been my inspiration. He is a great teacher in his style of teaching, a great mentor in influencing his students, and a great friend in his interaction. It is a great experience working with him and I look forward to learn many more things, with him as an example.

I owe my sincere thanks to Dr. Suresh Khator, for his guidance, encouragement, support and for helpful comments on the text. I would like to give special thanks to Dr. Tapas Das, for accepting to be on my committee, for his valuable suggestions on the problem and for being very cooperative.

I would like to express my thanks to my friend, guide and also, senior at the graduate school, Viswanath Sairaman for his discussions on the work, for motivating me in various ways, and for all the fun we shared during our graduate years.

I would like to make a special note of my friend Radhika Poolla, for all the care, great guidance, encouragement and support, all these years of my graduate studies at Tampa.

A hearty thanks goes to my best friend, Ashok Murugavel for all the great times shared, for all the fun and most of all, for being very dependable.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

DEADLOCK AVOIDANCE IN MIXED CAPACITY FLEXIBLE MANUFACTURING SYSTEMS

Sridhar Mohan

ABSTRACT

This research addressed the design and implementation of a polynomial-complexity deadlock avoidance controller for a flexible manufacturing cell modeled using Colored Petri Nets. The cell model is robust to changes in the part types to be manufactured in the system and is automatically generated using the interaction of the resources in the cell and the technological capabilities of the machines. The model also captures dynamic routing flexibility options. The framework introduced separates the cell model from the control logic allowing the system designer to implement and test various control algorithms using the same cell model. The controller adopts the neighborhood deadlock avoidance policy to resolve deadlocks and control the resource allocation decisions within the system. The evaluation of the performance of systems controlled by not maximally permissive algorithms is important in determining the applicability of the control algorithms. There are many polynomial time deadlock avoidance algorithms proposed for the control of general resource allocation systems. However, the permissiveness of these algorithms is not quantified and the applicability of these algorithms in terms of effective resource utilization remains unanswered. The performance of automated manufacturing cells controlled using the neighborhood deadlock avoidance policy is benchmarked by comparing its performance with other control policies .

CHAPTER 1

INTRODUCTION

A typical Automated Flexible Manufacturing Cell (FMS) consists of a set of Computer Numerically Controlled (CNC) machines and an automated transportation system, which connects the CNC machines. An FMS is capable of producing multiple part types efficiently in low to medium volume. The unfinished parts in an FMS are stored either in a centralized buffer of the FMS or local buffers of the machines. In an FMS, a job is first loaded into the cell from loading station and each operation of the job is performed by a CNC machine. When a machine required by the job is busy, the job waits either in the local input buffer of the machine or in the centralized buffer. When all the operations associated with the job are finished, the job is unloaded from the cell at the unloading station. An FMS is a type of a Resource Allocation Systems (RAS). An RAS is defined as a discrete event system with its entities referred to as resources. Ferreira, et. al. [32] defines 4 types of RAS as given below.

- A *Single Unit RAS* requires a single resource at each stage of its operation.
- A *Single Type RAS* requires one or more resources of the same type at each stage of its operation.
- A *Conjunctive RAS* requires one or more resources of any type at each stage of its operation.
- A *Conjunctive/Disjunctive RAS* requires one or more resources of any type at each stage of its operation with any sequence.

1.1 Representation of an FMS

Modeling FMS facilitates in understanding the interaction and the resource sharing between concurrent processes. An FMS can be modeled using Petri Nets [35, 8, 26, 7, 36, 40], Directed Graphs [37, 5, 10, 12, 42, 13], Finite State Automata [24, 41].

Petri Nets [19] are widely used for modeling [9, 3, 43] and analysis [6, 15, 14] of discrete event systems. Petri Nets capture the precedence relationships, synchronization concepts and concurrency associated with these system. A detailed literature review is presented in Chapter 2. Colored Petri Nets [7, 36, 16, 9] are extended Petri Nets in which the tokens have a color information attached to it. Each place in the Petri Net model is associated with a type, which determines the kind of color it may contain. A detailed explanation of Colored Petri Net is given in Appendix B.

1.2 Deadlock

Concurrent flow of multiple parts competing for limited amounts of resources in a manufacturing system may lead to deadlock situations which hinders automation of manufacturing systems. Deadlock is a situation, where parts in a set require a resource, which is currently used by another part in the same set. Due to this situation, the normal system operation is disrupted. Consider two machines M1 and M2 as shown in Figure 1.1, where the input and output buffers of both machines are occupied and the machines are busy. If the job on machine M1 requires M2 and the job on M2 requires M1, then the system is said to be in deadlock (neither of the parts can move). This type of deadlocked state is called circular wait state, where a job is waiting for a resource being held by another job, while occupying a resource required for the completion of the other job. This state can propagate and lead to a chain reaction where the whole system is in a circular wait state. It has been shown in the literature [32] that the sufficient conditions for a deadlock are:

1. Mutual Exclusion. The resource are shared by the parts in the system for processing.

2. Hold and Wait. All parts hold the resource until the next resource for the part is available.

3. No Preemption. Parts cannot be forced to leave the resource until the process is over.

4. Circular wait. Here a set of parts forms a circular chain, where each part waits for a resource that the next part in the chain holds, this is shown in Figure 1.2.

The FMS considered in this work cannot avoid the first three conditions given above. Therefore deadlock resolution can be achieved by addressing the last condition.

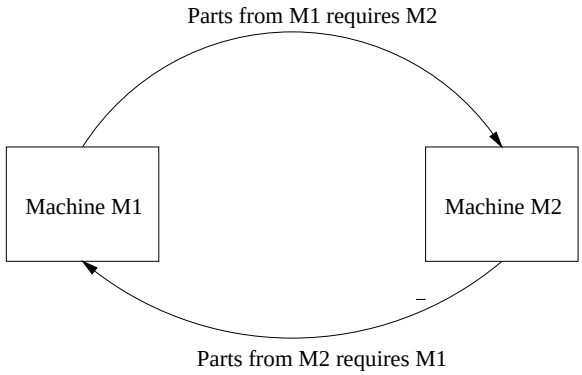

Figure 1.1. Deadlock

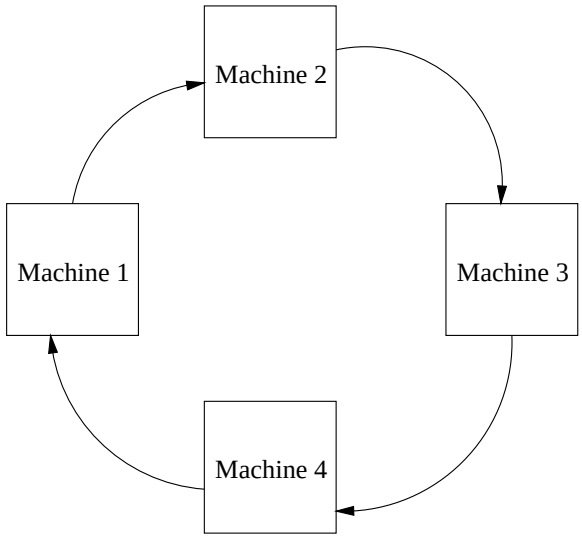

Figure 1.2. Circular Wait

The following methods are proposed to avoid such a condition.

1. Deadlock Prevention. Static rules are applied to control the resources in the FMS such that the system never reaches a deadlocked state.

2. Deadlock Detection and Recovery. The system is allowed to reach a deadlock state after which the deadlock is detected and recovered using deadlock recovery procedures.

3. Deadlock Avoidance. The allocation of parts to the resources is done in real time based on the current state of the system avoiding any possible deadlocked state by looking into the future.

In the case of deadlock prevention, static rules are used to prevent deadlock and the flexibility of the system is reduced. Viswanadham, et. al. [35], show that the prevention policy can be used only for the smaller system, since, the policy works on the reachability graph of Petri Net. In the case of detection/recovery, special hardware system or buffer systems are needed to recover the system from deadlock. Deadlock avoidance works in a real time environment by considering the current state of the system. This technique works by allowing a subset of safe states in the systems to be reached, but finding all safe states in a system is proved to be a NP-Hard problem. A safe state is defined as a state which will not eventually reach deadlock.

1.3 Practical Perspective on Deadlock Avoidance Controller

Consider an FMS and a set of parts that has to be processed, under specified criteria such as due date completion, shortest makespan. The specification of the parts and the machines (used in the FMS) are stored in a database. The specification of a part gives information about the operations that have to be performed on the part. The specification of machine describes the operations that can be performed on it. Considering that the parts are sent in a pre determined sequence into the FMS, the deadlock avoidance policy acts as a controller and controls the movement of the parts in the system. Control is achieved by checking whether the resulting state of the FMS is safe, when a part is moved.

The controller determines the safety of the resulting state by building a graph using the routing plan (from the specification obtained in the database) of the parts according to a look ahead policy. The controller makes sure that the FMS never reaches a deadlocked state. However the problem with such a method arises due to the computational complexity involved in the look ahead policy. Once the look ahead policy exceeds two steps, the state space becomes computationally intractable. Since the controller can consider only one or two step look ahead, the FMS may eventually reach deadlock. This kind of deadlock situation is referred to as impending part flow deadlock.

Deadlock avoidance policies work by enabling only a subset of possible events in a given state of the system in order to prevent the system from reaching deadlock or unsafe state. This is shown in Figure 1.3. A deadlock avoidance policy that allows for all the safe states of the system to be visited is said to be maximally permissive. Maximally permissive policies are desirable since these policies do not further restrict the cell other than to avoid deadlock. Under the supervision of a maximally permissive deadlock avoidance control policy any desired deadlock free schedule is executable, allowing for maximum utilization of the resources for varying objectives. On the other hand, under more restrictive policies, these allocation sequences may not be executable, adversely affecting the performance of the system. A maximally permissive approach to deadlock avoidance becomes computationally intractable as the capacity of the system increases. This difficulty has channeled a lot of the research effort in this area into deadlock avoidance policies that are based on structural properties of the models and are polynomial in complexity, which compromise maximal permissiveness for tractable computational complexity.

This research designs a polynomially complex deadlock avoidance controller (based on the Neighborhood deadlock avoidance policy) for a system modeled using Colored Petri Nets. Different from the approaches in the literature that measures restrictiveness based on the state space allowed by controller, the restrictiveness of a deadlock resolution policy is measured based on the systems performance. In this chapter we discussed FMS and

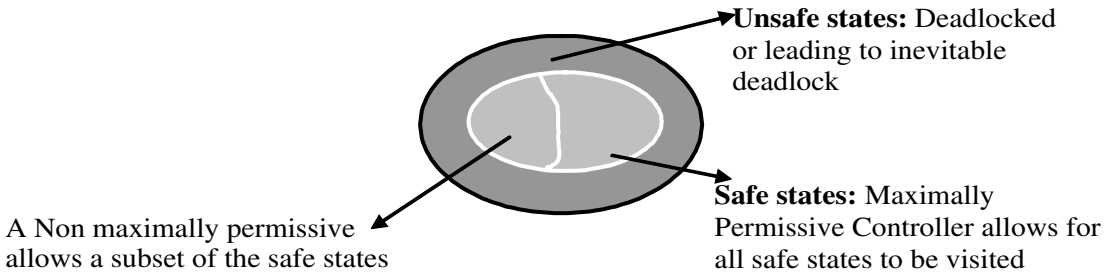

Figure 1.3. Maximally Permissive Control Policies

the formalism used to model such systems and their controllers. We also discussed the conditions under which these systems can be in a state of deadlock.

1.4 Summary of Remaining Chapters

The rest of the thesis is organized as follows, Chapter 2 is the literature review, which discusses the existing work on deadlock resolution techniques. Chapter 3 discusses the research objectives and the Colored Petri Net architecture and the controller based on the Neighborhood Deadlock Avoidance Policy. Chapter 4 describes the software implementation of the FMS and controller models that will be used to study the effectiveness of the Neighborhood Deadlock Avoidance Policy (NHDAP). The performance of the deadlock avoidance policy is tested and the results are discussed in Chapter 5. Chapter 6 deals with the conclusions and future work.

CHAPTER 2

LITERATURE REVIEW

This chapter summarizes the previous research efforts in the area of automated flexible manufacturing control systems grouped by deadlock resolution approaches.

2.1 Deadlock Prevention

Petri Net modeling of FMS's as Simple Sequential Process with Resource (S^3PR) nets has been introduced by Ezpeleta, et. al. [8]. The Petri Net model is based on the arrangement of machines, rather than on the working process of the parts. In order to avoid the deadlock, a deadlock prevention method based on the concept of siphons has been introduced. It has been shown, that, for an FMS to be deadlock free, the Petri Net structure should be free of unmarked siphons. In their work Ezpeleta, et. al. [8], prevent the unmarked siphons by adding new control places, which results in a deadlock free Petri Net.

Abdallah and Barkaoui [3] provide another structural control policy for the same S^3PR net for removing the unmarked siphons by adding control places. The resultant structure is not guaranteed to be deadlock free. So, the initial marking of the net is modified, resulting in independent siphons, which are not empty. The prevention policy in [3] has been compared to a detection and recovery policy in [11] by Barkaoui, et. al. [17].

Yiseng, et. al. [15], improved the deadlock prevention method in [8], by implementing a Mixed Integer Programming (MIP) technique for finding the unmarked siphons in the net. This method proves more efficient in finding the unmarked siphons than the method introduced by Ezpelta, et. al. [8]. Yiseng, et. al. [15], use two main stages in order to remove the unmarked siphons from the model, namely, siphon control and augmented

siphon control. The siphon control adds a control place to the original net, to avoid the unmarked siphons and the augmented siphon control makes sure that the newly added place doesn't introduce a new unmarked siphon.

The policy proposed by Yiseng, et. al [15], was extended to another class of Petri Nets called ES^3PR net [14]. Here, the deadlock is detected by the siphons and prevented by the control policy based on Yiseng, et. al [15].

Feng and Xiao-Lan [6] prove that the availability of an empty siphon (potential deadlock) is the necessary condition for a Petri Net to be deadlock free. In order to find the empty siphons a mixed integer programming approach is used.

The Petri Net architecture defined by Ezpelta, et. al. [8], has been subsequently extended to a Colored Petri Net in [7]. Here, the concept of the coloring policy was introduced to identify the process plans of the parts in the system. Each part is given a color based on the process plan and the last machine visited by the part. With this information, one can identify the process plan of the part in a system. For the Petri Net architecture in [1], a new type of coloring policy is introduced by Yalcin and Boucher [40]. In the coloring policy proposed by Ezpelta, et. al. [7], the process plans of the part should be pre defined. Based on these process plans, the colors of the parts are defined. In Yalcin and Boucher [40], the color of a token indicates only the next operation to be performed on the part type and not the process plan as Ezpelta, et. al. [7]. This coloring concept is explained in detail in Section 3.1.

A set of two diagraphs, working procedure diagraph and transition diagraph are used to represent the manufacturing system by Fanti,et.al. [12]. The working procedure diagraph represents the sequence of each part in the system and the transition diagraph represents the parts in each resource and the parts requesting access to the resources. The transition diagraph is updated dynamically when a part leaves or acquires a resource in real time. A condition called deadlock of a system can be identified using a diagraph and controlled by applying control laws, called restriction policies. The restriction policies enable or disable certain transitions based on the current system state and are applied on the dynamically

changing diagraph. Based on this restriction policy, a deadlock avoidance policy has been developed by Fanti, et. al. [9]. This policy was applied to an extended Colored Petri Net model proposed by Ezpelta, et. al. [8]. Fanti, et. al. [11], concluded that, when the resources are being heavily utilized, the prevention method is a more favorable method than the detection or recovery method.

Abdallah and ElMaraghy [1] modeled an FMS by S^4R nets and the deadlock resolution techniques they used were based on the siphon controls in the Petri Net. The prevention method was used to prevent the unmarked siphons in the net and the avoidance method was used to minimize the occurrence of the unmarked siphons.

2.2 Deadlock Detection and Recovery

Wysk, et. al. [37], modeled the Flexible manufacturing cell with the transporting system by the graph theoretic tool. An algorithm based on the structure of the system was proposed to detect the deadlock. Hyuenbo, et. al. [5], modeled FMS by Digraph and the deadlock is detected by the properties of the diagraph. The recovery or the avoidance method was used to avoid the deadlock based on the factors such as part processing time and buffer space. The detection and recovery method proposed by Wysk, et. al. [38], has been extended to accommodate multiple resources by Fanti, et. al. [10]. The FMS here was modeled by the diagraph.

2.3 Deadlock Avoidance

Viswanadham, et. al. [35], show that the deadlock avoidance methods can be used in real time, unlike the deadlock prevention policy. Here, it was shown that the deadlock prevention can be used only for small systems, where we can find the reachability graph and for the larger systems, deadlock avoidance policy is used. Banasazak and Krogh [2] proposed a Deadlock Avoidance Algorithm (DAA) for a class of Petri Net models, which imposes restriction policies on the resource allocation. This type of Petri Net modeling of

FMS has been extended to Colored Petri Nets by Ezpeleta and Colom [7] and a deadlock avoidance policy is proposed for the particular kind of Petri Net.

Wu [36] suggested a Petri Net model called, Colored Resource Oriented Petri Net (CROPN), which takes care of concurrent resource contention and production processes necessary for deadlock control. But this model can be applied, only, when the part types and their respective process plans are known. The CROPN initially drawn would not be able to take care of the situation wherein new types of parts arrive or the process plan of certain parts suddenly change. Hence to model this new situation, a new CROPN should be drawn and the control policy should be applied. This kind of modeling cannot take care of the dynamics of the situation. Moreover, it gets more cumbersome and tedious because, whenever, there is a change of state a new Petri Net model needs to be drawn. Therefore the problem here is to model such a kind of dynamic real time situations with efficacy, using Petri Net and dictate a control policy such that the system is deadlock free.

Lawley, Reveliotis, and Ferreira [18] introduced the concept of Resource Upstream Neighborhood (RUN) deadlock avoidance policy. According to them the deadlock avoidance policy should posses the following characteristics [33]

- Correctness. The DAP should posses a correct deadlock free operation.
- Scalability. The policy should be computationally scalable.
- Operational Flexibility. The policy should not reduce the operational flexibility since this decreases the performance of the resources.
- Configurability. The policy should adapt easily as the system changes.

As described in Chapter 1, the authors have also classified Resource Allocation Systems into four types called 1) the single unit resource allocation system (RAS); 2) the single-type RAS; 3) the conjunctive RAS; and 4) the disjunctive/conjunctive RAS(C/D-RAS). The flexibility issue is taken care only in C/D-RAS. The RUN policy is based on the concept that the machines with higher capacities can act as a buffer for the parts in the system. The time required to find the safe state by this policy is a polynomial function of system

size (i.e. the number of resource types and the distinct route stages of the process running through the system) [32, 31]. First the RUN policy is applied to the SU-RAS [18, 24, 30] which is an automated manufacturing system and it is modelled as an FSA. Which is then extended to C-RAS [25] where the system considered is an FMS and modelled as a Petri Net. In C/D RAS [27, 28, 29], the considered FMS is modelled by a Petri Net called S^3PGR^2 net, since this type of Petri Net has ability to model the flexibility involved in the FMS. The RUN policy is also referred as a NHDAP. In order to make the policy less restrictive, an IP formulation[21] is proposed. This formulation reduces the unit entries in the neighborhood matrix, which represent the resources allocated by the NHDAP for each parts processing stage. This IP formulation becomes complex as the neighborhood matrix grows. These are explained in detail on the following chapters.

Deadlock prevention techniques are based on static rules and hence cannot be applied to real time applications. Deadlock detection and recovery require special hardware in some cases and hence adds to the complexity of the recovery process. Deadlock avoidance techniques are based on the state of current system conditions and hence can be applied to real time applications. The major drawback of the models used in deadlock avoidance techniques is the necessity for remodeling the complete system, for simple changes in the process plan of the part types. Yalcin and Boucher [40] using a Petri Net coloring policy have overcome the above drawback, but the model lacks controller to prevent deadlocks. The objective of this work is to extend the CPN architecture proposed by the authors [40] with a real time deadlock avoidance policy.

CHAPTER 3

PROPOSED RESEARCH

3.1 Research Objectives

The proposed research includes several extensions to the CPN architecture proposed by Yalcin and Boucher [40]. These are:

1. The architecture introduced by the authors only models the physical manufacturing system and part movements within the system. The formalism does not include any means to deal with the deadlock problem prevalent in limited capacity resource allocation systems. In this research a deadlock avoidance policy is implemented that will be used to control resource allocation decisions to avoid deadlock.

2. The proposed deadlock avoidance policy is implemented as an external controller that controls the firing of an enabled transition based on the current markings of the system model.

3. The Petri Net architecture proposed in [40] is extended to accept more than one resource of same type for a processing stage of a part.

4. The system performance measure (makespan) is collected. This is done to compare the NHDAP with the maximally permissive policy.

The rest of the chapter is organized as follows, Section 3.2 summarizes the CPN architecture for modeling FMS in [40], Section 3.3 explains the deadlock avoidance policy addressing extension 1. Several motivating examples are included at the end of this chapter to illustrate the application of the deadlock avoidance policy to the Colored Petri Net model. The basics of the Petri Net and the Colored Petri Net are being described in Appendices A and B.

3.2 Colored Petri Net Architecture

A brief description of the Colored Petri Net model described by Yalcin and Boucher [40] is given in this section. The Petri Net architecture which describes the physical location of the machines, robots and the connections between them is an ordinary Petri Net. An example of this type of Petri Net is shown in Figure 3.1.

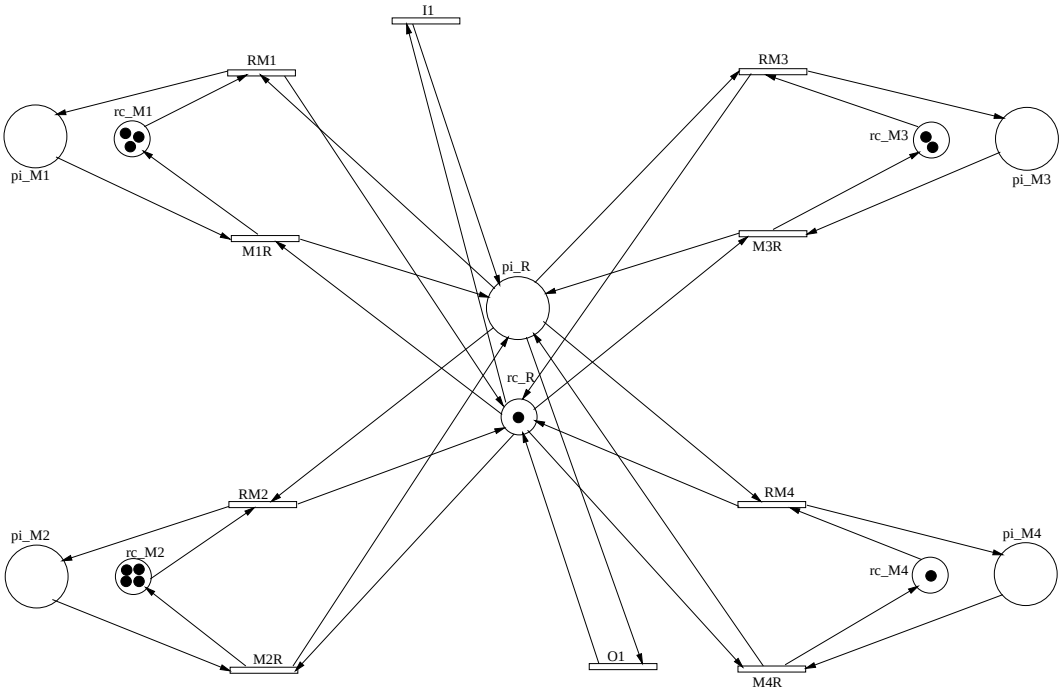

Figure 3.1. Petri Net Modeling of the CPN Architecture

In the Petri Net model each part entering into the system and the number of remaining resources are represented by tokens. The places in the Petri Net represent the resources and the transitions represent the flow of parts between the resources. There are two types of places, rc_X represents the remaining capacities of a resource X and pi_X represents the usage of a resource X. For example the place rc_M1 has three tokens, which indicates the remaining capacity of machine M1 and the place pi_M1 represents the usage of the resource. The marking shown in Figure 3.1 is the state where all the resources are idle. There are three types of transition in this Petri Net model, representing the loading, unloading and

the flow of parts in the system. In Figure 3.1 I1 and O1 represent the input and output transitions. The transition RM1 models the flow of parts between the robot and Machine M1, and M1R models the reverse flow. The notation is followed for all the transitions in the Petri Net. This ordinary Petri Net model for the manufacturing system is formally defined as ,

S=($P_R \cup P_K, T_F \cup T_I \cup T_O$, Pre, Post, m_o)

where,

P_R - set of places modeling the process of the part by the resource, pi_r

P_K - set of places modeling the remaining resources,rc_R

T_F - set of places modeling the flow of parts among the resources

T_I - set of places modeling the loading of parts in the system

T_O - set of places modeling the Unloading of parts from the system

Pre, Post - functions for the movement of tokens in the Petri Net.

m_o - initial marking of a petrinet

Note that the ordinary Petri Net model does not include any information regarding the operations performed on the machines or the process plan of the part. The following definitions facilitate the description of operations performed by the machines in the cell.

Definition 1: The set C represents all possible combinations of the operations that can be performed in the cell. Each member of set C is an n digit binary number, where n is the number of operations performed in the cell. For the automated flexible manufacturing cell with 4 operations, C=0000, 0001, 0010,..., 1111.

It is assumed that each machine can undertake a subset of all the possible operations performed in the manufacturing system. This information is represented by a function that associates each operation place of a resource (p $\in P_R$) with a binary number that is 1 for each operation that can be performed by the machine and 0 otherwise.

Definition 2: Let vect()$P_w \rightarrow$ C be the function that maps each machine in the system to the operations it can perform.

An example of such a function for the cell in Figure 3.1 is shown in Table 3.1. Vect (pi_M1)= 0001 represents that the machine M1 can perform op1 and vect (pi_M4)= 1100 indicates that machine 4 can perform op3 and op4. Note that the operations are assigned to digits from right to left instead of left to right. This convention allows a more robust implementation of the architecture where the new operations added to the system do not require modifications to the vect ( ) values of those machines that do not perform the new operation. For example, if a 5^{th} operation (10000) is to be included in the cell and M4 does not perform this operation, the decimal equivalent of $01100_2 = 24_{10}$ is the same as $1100_2 = 24_{10}$. On the other hand, if the operations were to be assigned from left to right the decimal number representing the capabilities of M4 would have to be changed from 0011 (6_{10}) to 00110 (12_{10}). The only modification required to the vect ( ) function of the machines that are capable of performing the operation would be adding 2^K to the existing decimal equivalent of the vect ( ) function for the K^{th} new operation.

Table 3.1. Vect() Values for Processors

P_w	vect(pi_processor)
pi_M1	0001
pi_M2	0100
pi_M3	0010
pi_M4	1100

The parts are represented as tokens and the information on the process plan is stored by means of color. The color of a token has 2 components, first is the part identification number (unique to each part entering into the system) and the second is the set of possible next operations, determined from the process plan of the part. The set of all operations and precedence relationship between the operations to be performed on the part are represented by a vector and a matrix. The (1 $\times$ n) vector describes the operations required by the part type i denoted by S_i, where n is the total number of operations to be performed in the cell. The (n $\times$ n) matrix denoted by D_i describes the precedence relationship of the operations required by the part type i. The initial color of a token that represents a part is determined by,

$$C_{y.x} = (y.x, S_y - (S_y D_y)) \tag{3.1}$$

Where, x represents, the identification number and y the part type. The color of the token changes, when the token moves from one process place to another.

Example 1. Consider an example of 2 Jobs, Job A and Job B and a set of machines M={M1, M2, M3, M4}. Job A has to perform op1-op3-op2 and Job B has to perform op2-op4-op1. The S_i and D_i matrices for the parts are formed below following the formalism described by Yalcin and Boucher [40],

$$S_A = \left\{ \begin{array}{cccc} op4 & op3 & op2 & op1 \\ 0 & 1 & 1 & 1 \end{array} \right\}$$

and

$$D_A = \begin{array}{c|cccc} & op4 & op3 & op2 & op1 \\ op4 & 0 & 0 & 0 & 0 \\ op3 & 0 & 0 & 1 & 0 \\ op2 & 0 & 0 & 0 & 0 \\ op1 & 0 & 1 & 0 & 0 \end{array}$$

In S_A, the required operations for the job are represented as '1' and all the other columns are represented by '0'. In the precedence matrix D_A, '1' in the intersection of operation 2 column and operation 3 row, indicates that the operation 3 should be performed before operation 2. The initial color of Job A, with the identification 1 and the values of S_A, D_A found from the Equation 1 will be $C_{A.1} = (A.1, (0001))$. This indicates that the next operation required for Job A is operation 1. The color of the Job gets updated as the Job moves in the system. The same procedure is followed to find the initial color of Job B with an identification as 2,

$$S_B = \left(\begin{array}{cccc} 1 & 0 & 1 & 1 \end{array} \right)$$

and

$$D_B = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

and $C_{B.2} = (B.2, (0010))$. The vect()$P_w \rightarrow$ C function, used to map the machines and operations is given in Table 3.1.

The Colored Petri Net model is described by

S=($P_R \cup P_K, T_F \cup T_I \cup T_O$, Pre, Post, m_o, C, C_f)

where,

C is a set of colors represents all possible operations in the sytem

C_f is a color function defined from P_R into C (This is represented as vect()$P_w \rightarrow$ C function)

The three types of transitions represented in the Petri Net model is explained below,

Transitions representing unloading of a finished part

The transitions that unload the cell [T_O = O1] are only enabled by a token with color 0000 which corresponds to a part that has completed all its processing and is ready to leave the cell. Therefore,

$$\forall t \in T_0$$

$$C_t = (c|c \in Cand|c| = 0)$$

where, $|c|$ is the magnitude of c.

Transitions representing loading of a part on a machine

Color domains of transitions that load machines are assigned so that a machine will only be loaded with a part, if operation(s) currently required by a part can be performed by that machine. The vect ( ) function is used to identify the operations that can be performed in a particular machine. The color domains of transitions that load machines include colors that result in a value not equal to 0 when the token color is compared with the vect ( ) function of the machine.

Transitions representing loading of a part on a transporter

When assigning color domains to transitions that load transporters it is important to make sure that before a transporter is loaded with a part, it can either unload the part from the cell or load the part onto another machine which can process it. In the special case of single transporter, since all parts must be moved by the centralized transporter, the color domains of the transitions that load the transporter is the set C.

The color change of the token is explained as follows. Let us consider a token of color (A.1,(0001)) representing part type A with ID 1. Assume that the token is in place pi_R and ready to perform operation 1 [op-1]. In this case only the transition RM1 will be enabled, because from the Table 3.1, we can see that only Machine M1 can perform the operation 1. If the token moves from the place Pi_R to Pi_M1, a token will be returned to the place rc_R and a token will be removed from the place rc_M1 and deposited in the place Pi_M1. When a token moves from one place to another, the initial color of the part is updated and the next operation to be performed on the part is identified. These two things are being performed by Modify Color function as shown below,

Function Modify_Color (place, color, part_type_id)

$S_{part\_type\_id} = S_{part\_type\_id}$ - *(color* $\times$ *vect(place))*

Modify Color $= S_{part\_type\_id} - (S_{part\_type\_id} * D_{part\_type\_id})$

if$(|ModifyColor \times vect(place)| > 0)$

Modify Color = Modify Color (place, Modify Color, part_type_id)

EndIf

End Function

The Modify Color function is applied to Job A with ID 1 as explained above,

$$S_{part\_type\_id} = S_{part\_type\_id} \text{ - (color} \times \text{vect(place))}$$

$$S_{part\_type\_id} = (0111) \text{ - } ((0001) \times (0001))$$

$$=(0111) \text{ - } (0001) = (0110)$$

$$Modify\ Color = (0110)\ \text{-}\ \left[(0110) \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \right]$$
$$= (0110)\ \text{-}\ (0010)$$
$$= (0100)$$

Modify Color function finds the next operation to be performed on the part, from the new color of the token and the precedence relationship between the operations. Now the token color changed from (A.1,(0001)) to (A.1,(0100)).

3.3 Modeling the Deadlock Avoidance Controller

Limited capacity resource allocation systems require controllers that regulate resource allocation decisions considering the capacity of the resources and the future resource requirements of the entities (parts) in the system to avoid deadlocks. The architecture described in Section 3.2 does not incorporate any control decisions in routing of the parts in the system. One approach to incorporating the deadlock avoidance requirement is generating the reachability graph of the CPN model and disabling those events that lead to deadlocks [16]. However, this approach is not applicable for real-time control of large systems. The number of states in the reachability graph increases exponentially as the capacity of the systems increases. There are several polynomial complexity control algorithms described in the literature that are based on structural properties of the Petri Net models of the manufacturing system [36, 21, 29, 8, 2, 7]. These models require a process plan based Petri Net model of the underlying system and are not readily applicable to the architecture outlined in Section 3.2.

In this section, we describe the design of a controller that adopts the NHDAP developed in [21, 29, 18] to control a manufacturing system modeled based on the CPN architecture described in Section 3.2. In the process plan based models, each processing stage is modeled by a unique place in the system model. However, in the CPN architecture, processes

performed by the same machine are modeled using one place which leads to more than one processing stages corresponding to the same place. Colors of the tokens are utilized to differentiate between the processing stages corresponding to the same place in the CPN model. The underlying CPN is not a process plan based model, but a model that is based on the physical layout of the system as described in Section 3.2. In this manner, the control model is separated from the physical system model and exercises control over the cell as illustrated in Figure 3.2.

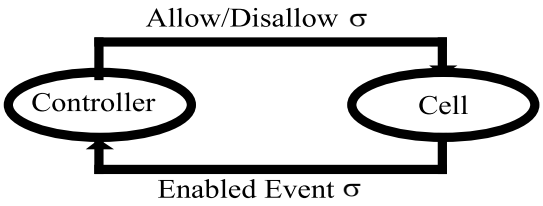

Figure 3.2. Cell Controller

According to the NHDAP, the number of parts in the neighborhood of a machine should not exceed the capacity of the machine. The number of parts in the neighborhood of a machine is determined by multiplying the system neighborhood matrix with the state vector. The system neighborhood matrix is a (machine $\times$ processing stage) matrix, describing the neighborhood of the processing stages. The system neighborhood matrix is found by combining all the neighborhood matrices of all the parts process plans. The state vector is a (processing stage $\times$ 1) vector representing the number of parts in a particular processing stage.

The formal definition of the neighborhood policy is a linear constraint,

$$NM \times S \leq C(M),$$

where NM = System Neighborhood Matrix, S = State Vector, C(M) = Capacity Vector. The system neighborhood matrix is determined off-line based on the processing require-

ments of each part. The state vector changes dynamically as parts move from one resource to another and is determined from the current marking of the CPN model in real time. The capacity vector reflects the total capacity of all the resources. The capacity of the system resources is assumed to be constant.

3.3.1 Generating the System Neighborhood Matrix

The System Neighborhood Matrix of each part type is described by the following formalism. Let $\rho_p = P(p) : P_R- > \{1, ...|P_R|\}$ be any partial resource ordering imposed on the processing places of each resource, where $|P_R|$ is the number of resources. Let p_m be the place occupied by the token representing a part of the part type for which the neighborhood matrix is created. Let

$$T_c = \{t|t \in (p_m)^{\cdot} \wedge c \cap C_f \neq 0\} \tag{3.2}$$

where C_t is the color domain of transition t as described in Table 2. $(p_m)^{\cdot}$ represents the set of transitions that input from p_m and c is the color of the token in p_m. In other words, T_c is the set of transitions process enabled by the token representing a part when it is in place p_m. Also, let

$$R_c = \{q|q \in (T_c)^{\cdot} \cap P_r\} \tag{3.3}$$

where $(T_c)^{\cdot}$ represents the set of places that transitions in T_c output to. The set of places R_c is the set of places that represent processing places of the machines where the part can be processed next. Further, let

$$L_c = \{q|q \in R_c \wedge \rho_q = \min_{v \in R_c} \rho_v\} \tag{3.4}$$

The set of places L_c is a subset of R_c such that the places in L_c have the minimum ρ value among the places in R_c. Finally, let

$$N_c = \{p_m\} \cup \{q|q \in \cup_{v \in L_c} N_v \wedge \rho_m \leq \rho_v\} \tag{3.5}$$

where N_c specifies the set of places that the part would traverse in the system in future and the current place. The set of places is determined based on each places ρ value through recursion as shown in the above formulae (the ρ value of the future places should be greater than or equal to the ρ value of the current place). Calculating the parameters associated with the above definitions is illustrated in Example 2.

Example 2. Consider Example 1 with 2 types of jobs, Job A, Job B and a set of resources M = {R,M1, M2, M3, M4}, where R is a material handler and their capacities are, C = {1,3,4,2,1} (found from the initial marking of the Petri Net Figure 3.1). The partial resource ordering is $\rho_{pi\_R} = 1, \rho_{pi\_M1} = 3, \rho_{pi\_M2} = 4, \rho_{pi\_M3} = 2, \rho_{pi\_M4} = 1$ based on the capacities of the resources. Let us consider a token of color (0100) representing a part of type A in place pi_R. The part's next required operation, op3, can be performed in M4 or M2. The process enabled transitions are

$$T_c = \{RM4, RM2\}$$

The places that represent the processing of this are,

$$R_c = \{Pi\_M4, Pi\_M2\}$$

By comparing the r value of the places pi_M2 and pi_M4,

$$L_c = \{Pi\_M4 | min(4,1) = 1\}$$

The set of places in N_c are calculated using recursion as indicated before. The N_c set also includes pi_M3, because the ρ value of the pi_M3 is equal to the current place pi_R. The set N_c includes,

$$N_c = \{Pi\_R, Pi\_M4, Pi\_M3\}$$

These results are used to populate column P_{A3} of the Neighborhood Matrix. The rows corresponding to the places in N_c in column P_{A3} have a value of "1" and "0" otherwise. The same process is repeated for each processing stage of each part type to generate the complete neighborhood matrix which is shown in Table 3.2.

Table 3.2. The Neighborhood Matrix for Example 2

	P_{A1}	P_{A2}	P_{A3}	P_{A4}	P_{A5}	P_{A6}	P_{A7}	P_{A8}	P_{B1}	P_{B2}	P_{B3}	P_{B4}	P_{B5}	P_{B6}	P_{B7}
R1	1	0	1	0	1	1	0	1	1	0	1	1	1	0	1
M1	1	1	0	0	0	0	0	0	1	1	1	1	0	1	0
M2	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
M3	0	0	0	0	1	1	1	0	1	1	0	0	0	0	0
M4	0	0	1	0	1	0	0	0	1	0	1	1	0	0	0

3.3.2 State Vector

The state vector represents the number of parts in the system and the corresponding processing stages the parts are in. This vector is dynamic and changes as the state of the system changes. In process plan based models, each processing stage is modeled by a unique place in the underlying system model and the state vector of the neighborhood deadlock avoidance algorithm is determined from the number of tokens in that place. On the other hand, in the CPN architecture, the processing places (P_R) are unique to resources; not processing stages. Therefore, two tokens in the same processing place may correspond to different processing stages. Furthermore, a token can have the same color in two different processing places when alternative routings and material handlers are considered. The different processing stages of the parts in the system are identified by considering the (color $\times$ place) combination of the tokens representing the parts.

To create a state vector, let us consider that there are 2 of part type B in the system, both are in the first process stage, i.e., Machine 3 . This is represented in state vector as follows,

$$s = \begin{Bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 2 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{Bmatrix}$$

According to NHDAP, System Neighborhood Matrix $\times$ state vector $\leq$ capacity as shown below,

$$\begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \end{pmatrix} \times s = \begin{pmatrix} 0 \\ 2 \\ 0 \\ 2 \\ 0 \end{pmatrix} \leq \begin{pmatrix} 1 \\ 3 \\ 4 \\ 2 \\ 1 \end{pmatrix}$$

The output vector, which is a (machine $\times$ 1) vector gives the number of parts being occupied in the machine starting from machine 1. The row value of the matrix corresponds to the machine number, if the row value is 1 then the machine number is 1. Here a value of 2 in the machine 1 place of the output vector indicates that 2 parts are being operated in that machine, since there are 2 parts of type B are being processed in Machine 1. The value of 2 in Machine 3 location represents that 2 places in Machine 3 are being reserved for

this process, these places are allocated according to Neighborhood Policy. Let us consider that a new Job A is being loaded in the robot, after finishing its first operation. Job A is now in process stage P13, the Job can either go to machine 2 or machine 4 to perform its next operation. If this Job moves to machine 4, it leads to a deadlock, because Job B in machine 3 requires machine 4 and the part type A in the machine 4 requires machine 3 as its next operation. The deadlock avoidance policy should not allow Job A to enter machine 4 to perform its next operation. It should instead allow the job to enter machine 2. This is shown below,

$$\begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \end{pmatrix} \times \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 2 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 0 \\ \mathit{3} \\ 1 \end{pmatrix} \leq \begin{pmatrix} 1 \\ 3 \\ 4 \\ \mathit{2} \\ 1 \end{pmatrix}$$

The condition System Neighborhood Matrix $\times$ state vector $\leq$ capacity is violated when a part type A is loaded onto the machine 4.

$$\begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \end{pmatrix} \times \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 2 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 2 \\ 1 \\ 2 \\ 0 \end{pmatrix} \leq \begin{pmatrix} 1 \\ 3 \\ 4 \\ 2 \\ 1 \end{pmatrix}$$

As shown above, the condition of NHDAP is not being violated when the part is loaded onto the machine 2.

In this chapter a design of a controller for the Petri Net architecture summarized in section 3.2 is described. We adopt and implement the NHDAP to control the system modeled using the CPN architecture. The design of the controller proposed in this research is not the process plan based which differs from previous approaches. Formal methodologies to extract the necessary information for the control policy from the underlying system model are described in Section 3.3. The simulation of the FMS, based on an object oriented design approach is described in the next chapter.

CHAPTER 4

CONTROLLER SOFTWARE DESIGN

An object oriented design approach for the simulation of the automated flexible manufacturing system is described. An object oriented approach facilitates the link between the analysis and specification of a real system and the design and implementation of an action model for that real system where the structure and behavior of entities are readily modeled as objects. An object, in an object-oriented paradigm, is a collection of data (attributes) together with all the operations (methods), which access or alter that data [4]. Some or all of these operations could be used to provide a uniform external interface to other parts of the system. Other objects in the system can interact with this object only through requests for the object to execute its operations. The ability of the object-oriented paradigm to provide a direct representation for real world objects comes with four unique features associated with it, namely, encapsulation, data abstraction, inheritance and dynamic binding [4, 34].

In this research C++, an object oriented extension to the C programming language is used to develop the controller software. This chapter provides a detailed description of the controller software design. The FMS and the controller are modeled as different objects.

4.1 Controller Implementation

This section describes the various classes that are used in simulating the controller and their interrelationship. The physical system considered by the controller is described first, the classes and their relationships are explained later.

4.1.1 Physical Systems

The physical system considered is a fully automated flexible manufacturing system that has a number of single or multi capacity machines with a centralized material handling device, that interconnect these machines. There is an input and an output buffer where parts can be loaded and unloaded to and from the cell via material handling device.

The controller (NHDAP) connected to the automated flexible manufacturing system coordinates the movements of the parts. The movement can be either from a machine to another machine, from a machine to buffer of from a buffer to a machine. The controller monitors the movement of the parts for deadlock situation with signals. The signal is given to allow or disallow a movement. Based on these signals provided by the controller, the automated flexible manufacturing system's state (location of the parts) changes.

4.1.2 Object Definition

The proposed object oriented design approach contains: 1. Part types, 2. Machines 3. Cell Model, 4. Controller. The modeling of the cell includes the creation of machines, robots, input and output buffers and their interactions with the parts. The supervisory control is used to control the cell. Each movement in the cell is notified to the controller, based on the signal provided by the controller (allow/disallow), the movement is allowed or disallowed.

4.1.3 Object Classes and Attributes

The physical characteristic of the manufacturing system is separated from the controller. In this manner, different control strategies can be applied to the manufacturing system for testing of deadlock situations. This type of separation can be easily implemented in the object oriented model by creating classes for the physical system and the control separately. Different combination of cell models and control strategies could be experimented by creating different instances of the corresponding classes and varying the values of their attributes.

The attributes and methods of all the classes used in the controller are shown in Figure 4.2 and the interactions between the classes are shown in Figure 4.1. A detailed explanation on each class is given below,

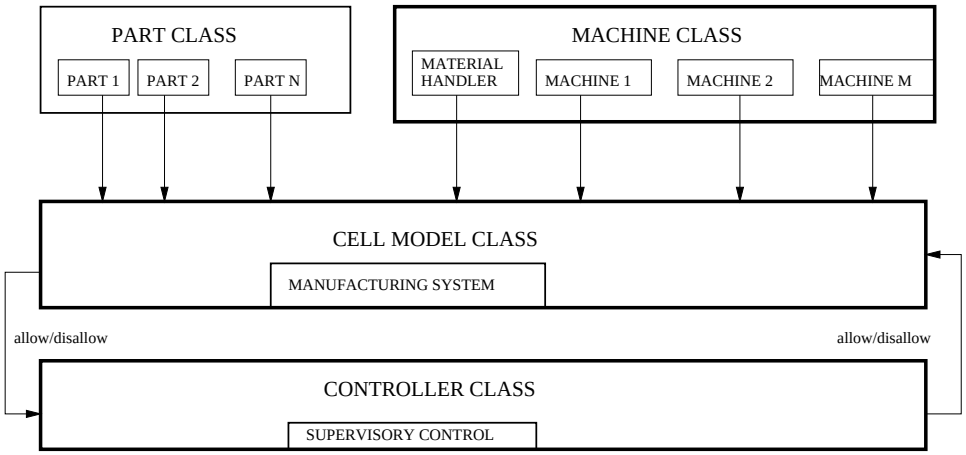

Figure 4.1. Relationship Between the Objects

- Part Type Class

  This class represents the parts manufactured in the cell, with the S vector and D matrix as attributes. The update color function, finds the color of the part from the S vector and D matrix and stores in an attribute named color. When the S vector of the part changes, as the part moves in the system, the update color function executes automatically.

- Machine Class

  This class represents the machines, the central material handler and the input and output buffers. At any time in the simulation, a machine keeps track of the number of parts present in it by the parts_present attribute. The job type of each job present in the machine can be identified by its identification number. A machine is in either busy or idle (processing a part or occupied by a part to be unloaded) state.

- Cell model Class:

  The cell class represents the machines, buffers, material handler and the parts that

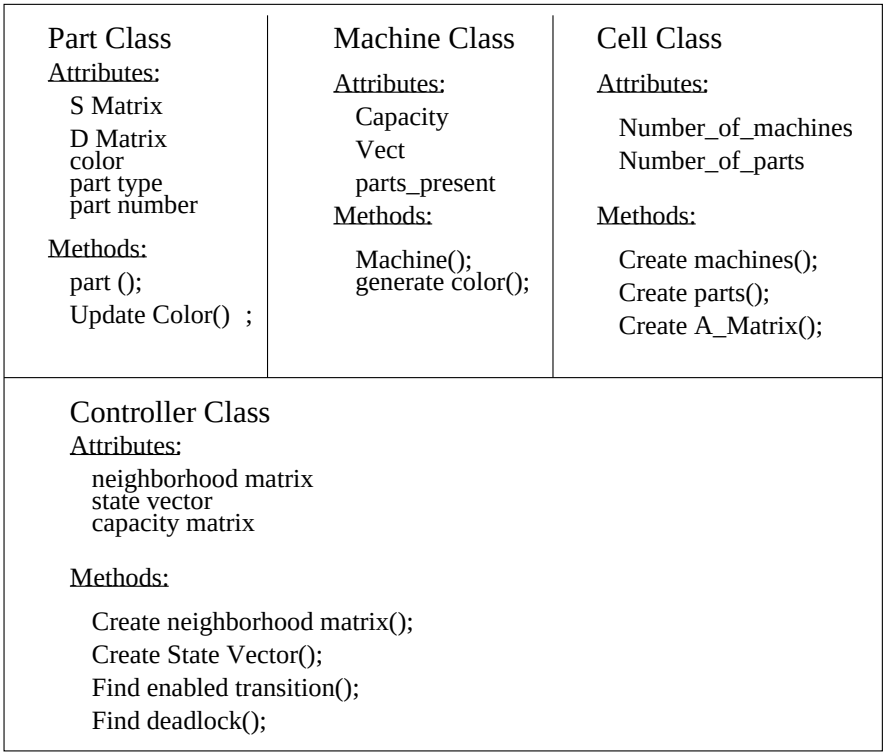

Figure 4.2. Classes Used in OO Model of FMS

are being considered in the FMS. The methods associated with this class includes create machines (), create parts (). This class creates an interaction between the machine, buffer and the material handler. This interaction is shown in the form of A matrix, corresponding to the incidence matrix of the Colored Petri Net model of the cell.

- Controller Class:

  The controller represents the logic used to prevent the system from reaching a deadlocked state. When there is a request to move a part from one machine to another or from a buffer to a machine, the controller class uses find deadlock () method to evaluate the presence of the deadlock. This method captures the current state of the FMS, in the form of state vector and uses the NHDAP to find the presence of deadlock. The neighborhood matrix used in the deadlock avoidance policy is created

by the create neighborhood matrix () and the state vector is created by create state vector() method.

4.2 System Simulation

This section describes the working procedure of the controller. It also explains the input parameters required to run the simulation of a manufacturing firm and the output produced by it.

4.2.1 Simulation Inputs

The input files to the simulation describe the sequence of the parts to be processed, the routing plan of the parts and the machine's capabilities. There are two types of files, mach.dat represents the number of machines being considered in the simulation and part.dat represents the sequence of parts present in the manufacturing cell. Each machine and part in turn has a separate input file, which represents their corresponding specifications. The machine specification file has two parameters, one is the vect() function (as shown in Table 3.1) of the machine and the other is the capacity of the machine. The part specification file has three parameters, first the S matrix of the part, second the D matrix and the third is the time required to perform each operation that the part requires.

4.2.2 Simulation Logic

The simulation logic is implemented in the object-oriented model through the updating of the attributes of the component objects and the interactions among these components are modeled through the message-passing facility in the object-oriented approach. The cell model monitors the movement of the parts and relays appropriate event signals to the controller class. The controller evaluates the request based on the current state of the system and sends back the decision to allow/disallow the request made by the cell model. The flow chart in the figure 4.4 outlines the control process.

At initialization, the CPN model of the cell is generated based on the machines technological capabilities. The neighborhood matrix for the part types is also generated offline at this stage. Parts to be processed during the simulation are loaded into the system by updating the marking of the CPN model. The cell model is used to determine the set of enabled transition. As the simulation starts with the arrival of parts to the input buffer, the transitions become enabled. As a transition becomes enabled, the deadlock avoidance policy is used to allow/disallow the transition. If the transition is allowed, it is fired in the cell model and the CPN model's marking and the state vector are updated. If the transition is disallowed, it is queued to be considered again when the state of the system changes after the execution of the next allowed transition. Once all the scheduled parts are processed and unloaded from the system, the simulation terminates. The installation procedure for running the simulation is given in Appendix D.

4.2.3 Simulation Results

The output of the simulation is captured in a data file, which includes event trace information and system performance information. The trace information includes a trace of events ordered by time for understanding the behavior of the system. The system performance includes the makespan of each part and the throughput.

In this chapter we have described an object oriented approach for the simulation of automated flexible manufacturing system and a controller to control the FMS. The modeling of the controller is separated from the physical system. In this manner various control policies can be tested on the same physical system. The results obtained by simulating the FMS considered in Section 3.2 with the NHDAP implemented as the controller is discussed in the next chapter.

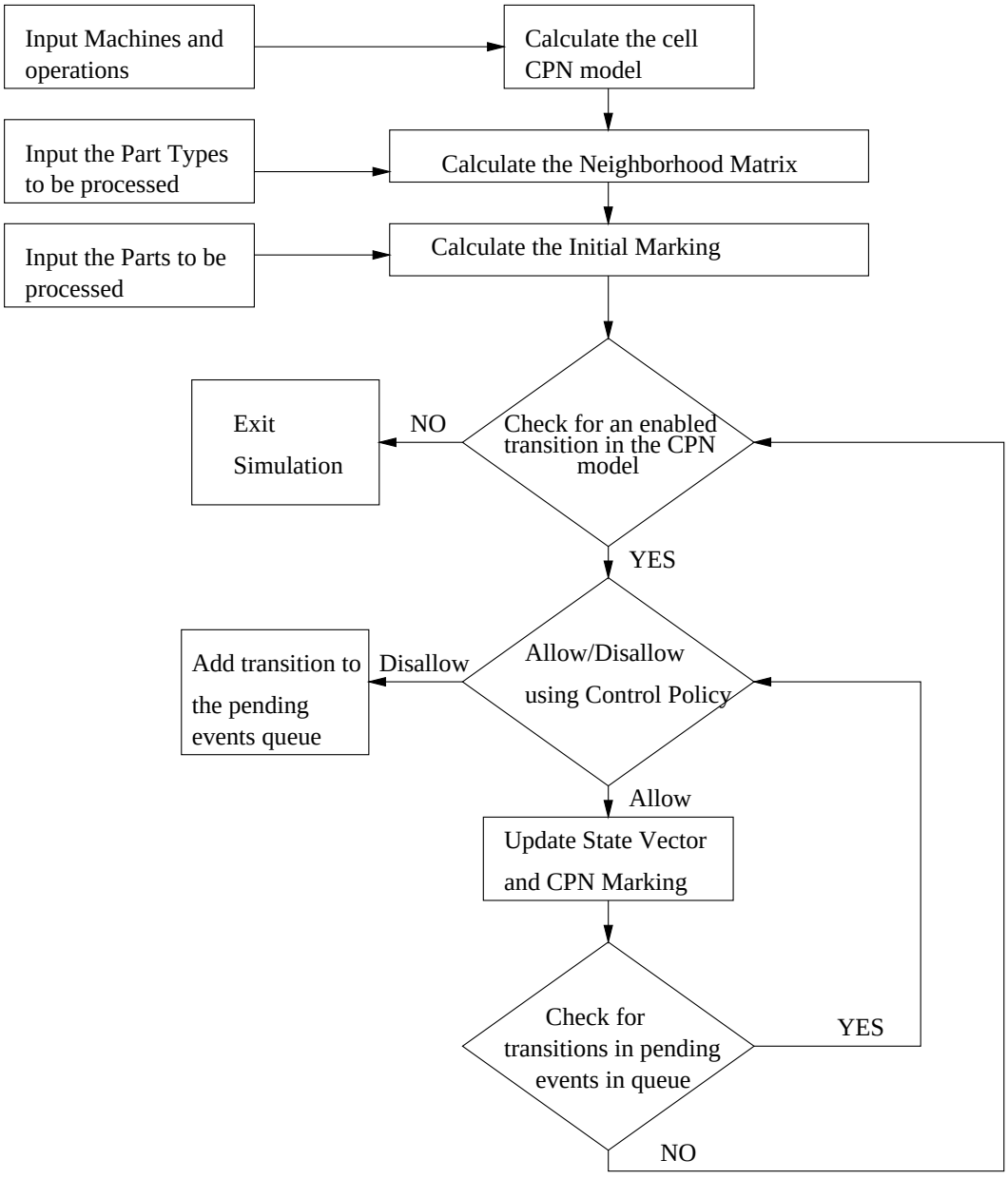

Figure 4.3. Flow Chart of the Simulation

CHAPTER 5

PERFORMANCE EVALUATION OF SYSTEMS CONTROLLED WITH THE NHDAP

This chapter compares the NHDAP to policies suggested by other researchers for testing the restrictiveness and the performance of the policy. The NHDAP is not maximally permissive policy but of polynomial complexity. This results in disabling some events even though they do not lead to deadlock. Since the NHDAP is not maximally permissive, it is necessary to establish the restrictiveness of the policy compared to one that is maximally permissive in terms of control policies effects on performance of the system. This comparison is shown in the first part of this chapter.

The second part of the chapter focuses on comparing the performance of the NHDAP in a constrained cell design. In this design the pool of part types are selected to balance the workload in the cell and there is a bottleneck machine that limits the performance of the system. This refers to the case where a cell has an expensive resource that is required by all processes and requires long processing time.

5.1 Experiment 1: Performance Comparison of the NHDAP with a Deadlock Avoidance Policy that Minimizes Makespan

In [18], an IP formulation is proposed to calculate optimal partial resource ordering (ρ_p values described in Section 3.2), which minimizes the capacity reserved by the NHDAP and is shown to increase the permissiveness of the NHDAP compared with random partial resource ordering. The objective of the IP formulation is to choose the ordering of the machines in a manner that results in reducing the unit entries of the neighborhood matrix and increases the allowable state space of the control policy. In [21], it is shown that an

optimal partial resource ordering leads to better system performance in terms of minimizing makespan when compared with random resource ordering under different dispatching rules such as FIFO, LPT, SPT. Our objective here is to compare the performance of a system controlled by the NHDAP with optimal performance measures using the controller design described in Chapter 3.

Figure 5.1 shows the formulation that minimizes makespan and avoids deadlock in an automated manufacturing cell based on [23] and [20]. Equation 5.1 ensures that the makespan is minimized. Equation 5.2 represents the processing time and the routing information of each job, and ensures that the difference between the completion time of the current operation and the immediately previous operation of a job is at least as long as the processing time of the current operation. Equations 5.3 and 5.4 restrict each machine to process one job at a time. Equations 5.5 and 5.6 ensure that a job leaves a machine only when it has found space on the next machine.

5.1.1 Experimental Design

We consider 10 test cases, first five cases have 3 machines and 4 part types each with 3 operations, and the last five cases have 4 machines and 3 part types each with 4 operations. The processing times of the operations are generated randomly. The processing plans of the test cases are listed in Appendix C. Using the formulation in Figure 5.1, deadlock-free schedules that minimize makespan are determined. Based on the part loading sequence of the optimal schedule, parts are released into the cell and processed to completion using the NHDAP. A First-Come-First-Served resource allocation policy is used to resolve resource allocation conflicts. In the implementation of the NHDAP, we considered two cases. The first case implements a partial ordering of the resources equal to the capacity of the resources. For the second case, we calculated the optimal partial resource ordering that maximizes the permissiveness of the NHDAP. It should be observed that the part loading sequence found from the IP formulation is based on the assumption that the number of parts that are to be processed is known and available in apriori. In systems where there

are random arrivals of parts into the system, calculating an optimal schedule every time a new part arrives to the system may not be possible during real time operational control due to the high computational requirements of the problem.

Table 5.1. Process Plan for Single Routing

	Integer programming		Neighborhood control policy	
Test Case	Makespan	Part dispatching	Optimal resource ordering	Random resource ordering
1	359	1-3-2-4	374	624
2	312	3-1-2-4	368	459
3	482	3-1-2-4	522	767
4	363	1-3-2-4	442	570
5	335	1-3-2-4	369	514
6	242	2-3-1	307	387
7	406	2-3-1	501	718
8	357	1-3-2	415	660
9	273	1-3-2	328	469
10	372	1-3-2	438	646

5.1.2 Experimental Results

The performance of the optimal resource ordering compared to the random resource ordering (based on the capacity of the resources) for the NHDAP is shown in the Table 5.1. The results indicates that the performance of the optimal resource ordering is better compared to the random resource ordering. The optimal resource ordering case results were within 4% to 23% of the optimal makespan values.

Table 5.1 also shows the results for the optimal solutions based on the Mixed Integer Programming given in Figure 5.1, with random resource ordering. Closer examination of the simulation outputs revealed that the neighborhood control policy allowed only one part at a time in the cell leading to makespans that were equal to the sum of the processing times of all parts. In the two cases of neighborhood deadlock avoidance with optimal and random resource ordering, the part dispatching sequence (the sequence in which the parts are release into the system) are the same as the dispatching sequence determined by the optimal solution using the mixed integer programming approach.

5.1.3 Effects of Part Dispatching

It is observed that the alternative part dispatching sequence results in shorter makespans compared with part dispatching sequence of the MIP formulation. These results are shown in the Table 5.2. It is noted that 4 of 10 cases resulted in a better makespan, when the part dispatching sequence changes. Note that the minimum makespan obtained using the NHDAP had the same part dispatching sequence for all the test cases in each of the two system setups. This indicates a possible correlation between the neighborhood matrix and the part dispatching sequence that may be utilized to improve system performance.

Table 5.2. Different Part Dispatching Rules

	Integer programming		Neighborhood control policy	Neighborhood control policy using alternative part dispatching sequence	
Test Case	Makespan	Part dispatching	Optimal resource ordering	Makespan	Part dispatching
1	359	1-3-2-4	374	-	-
2	312	3-1-2-4	368	345	1-3-2-4
3	482	3-1-2-4	522	517	1-3-2-4
4	363	1-3-2-4	442	-	-
5	335	1-3-2-4	369	-	-
6	242	2-3-1	307	272	1-3-2
7	406	2-3-1	501	492	1-3-2
8	357	1-3-2	415	-	-
9	273	1-3-2	328	-	-
10	372	1-3-2	438	-	-

5.2 Experiment 2: Effect of Capacity Representation in NHDAP

The objective of this experiment is to evaluate the different mathematical representations of manufacturing system capacities. The NHDAP is based on RUN, which states that the machines with higher capacities can act as buffer for the parts. We project that the performance of the NHDAP will improve if identical machines are represented as a single group with capacities equal to the sum of the capacity of the individual machines rather than individual machines with lower (single) capacity.

5.2.1 Design of the Manufacturing Cell

In this experiment we consider an automated flexible manufacturing cell, which has 6 machines and a robot for material handling as depicted in Figure 5.2. The operations and their processing times are shown in the same figure. As seen in the Figure 5.2, Machine M5 is identical to M2 and Machine M6 is identical to M3. We will assume that there are 5 part types and each part requires 4 different operations in the cell. The routing plan for each part type is shown in the Table 5.3. Note that the requirements for drilling and turning operations, which can be performed on M2 or M5 and M3 or M6 respectively, have been assigned to parts in a manner to balance the workload in the cell. For example P1 and P2 require M2 and M3 and P2 and P4 requires M5 and M6 for drilling and turning requirements.

Table 5.3. Process Plans

Part Type	op1	op2	op3	op4
p1	M1	M2	M4	M3
p2	M1	M6	M5	M4
p3	M2	M3	M1	M4
p4	M6	M5	M4	M1
p5	M4	M5	M1	M3

The IP formulation [18], used to determine the optimal resource ordering for the NHDAP is computationally complex in the system that we have considered. The number of parts considered in this experiment is 25 with 4 operations. The complexity arises, because the IP should capture the processing stages of each part ($25 \times 4 = 100$ constrains) and the relation with the other processing stages ($100 \times (100 - 1)$ constrains). Due to this complexity the random resource ordering is being used. Both the NHDAP and the maximally permissive policy follow a first come first serve resource allocation policy to resolve resource allocation conflicts.

5.2.2 Experimental Design

Twenty replications were run, by generating a pool of 25 parts (generated randomly from the 5 part types in Table 5.3). Since each sample used a different random number

seed, the 20 replications provided 20 unique sequences of 25 randomly selected parts. The average makespan found from the experiment is shown in Table 5.4. Note that due to the bottleneck machine, the theoretical lower bound of the makespan to process 25 parts is 25 $\times$ 40 = 1000 time units and the theoretical upper bound is 25 $\times$ 100 = 2500 time units.

5.2.3 Experimental Results

The Table 5.4 shows the performance of the FMS controlled by the maximally permissive deadlock avoidance policy [39] and NHDAP. It is observed that the NHDAP increases the makespan by 100%. From the results it is observed that the makespan obtained from the NHDAP is close to the theoretical upper bound makespan, which infers that only one or two parts are being allowed into the system at all times.

The FMS considered for this experiment has single capacity machines. The NHDAP which we used to test the result performs better when the system has multiple capacities compared to the system with single capacity machines. The reason for this is the neighbor deadlock avoidance policy is based on the concept of resource upstream neighborhood (RUN), which states that the machines with higher capacity can act as a buffer for the parts. That is, for each processing stage of a part, a buffer location is allocated by the deadlock avoidance policy, unless the machine processing the part is the higher capacity machine in the routing plan of the part. The single machine environment causes the NHDAP to allow one or two parts into the system at all states. It would be interesting to find out how the policy performance changes in the presence of multiple capacity machines. This experiment is discussed in the next section.

Table 5.4. Expermiental Results

Control Policies	Makespan
Maximally permissive	1074
NHDAP	2317

5.2.4 Alternate Representation of Manufacturing System Resource Capacity

Let us consider the same example, where the two drilling and turning machines are represented as a drilling and turning machine with capacity two. This representation is shown in Figure 5.3. Technically the capacity of the manufacturing system doesn't change, it is the mathematical representation that changes. The performance of the NHDAP is much better with this representation compared to the earlier scenario. This is shown in Table 5.5. It is noted that the average makespan of the NHDAP is half when the manufacturing systems mathematical model changes. Due to the presence of the resources with two capacities, the NHDAP allocates two parts in those machines neighborhood. This results in allowing more than one part into the system at all times. From this it can be concluded that the NHDAP performance is lower for a system with single capacity machine compared to a system with multiple capacity machine.

Table 5.5. Changing the Mathematical Representation

Policies	Machine Capacities	Makespan	Avg. Flow Time	Max. Flow Time	Throughput
Single Capacity	M1 = 1;M2 = 1; M3 = 1;M4 = 1; M5 = 1;M6 = 1	2317	298.23	520.8	0.01078
With changing Mathematical Representation	M1 = 1;M2 = 2; M3 = 2;M4 = 1	1164	253.28	498.2	0.02147
Maximally permissive	M1 = 1;M2 = 1; M3 = 1;M4 = 1; M5 = 1;M6 = 1	1074	235.48	439.5	0.023292

NHDAP predicts the resource requirements for a part and attempts to reserve these resources needed beforehand. Though there is a guarantee of avoiding deadlocks, there is also a tendency to be restrictive. This nature of the NHDAP forces the cell model to have a significant effect on the utilization of the resources. Capacity variations of a single resource type in isolation lower the restrictiveness of the policy, while variations of the resource capacities for multiple resources in relatively comparable amounts promotes the working of the policy. This leads to an observation that, instead of replicating machines, increasing their capacity would lead to a better performance for the NHDAP.

In this chapter, numerical results obtained by applying the developed NHDAP on two different test problems have been presented. The single unit resource allocation system

(SU-RAS) has been taken as the test bed to compare the developed deadlock avoidance policy solution with the optimal solution. The second experiment in this chapter compares the NHDAP's performance when the mathematical representation of the manufacturing system changes. The experiments reported in this chapter indicates that,

1. Optimal resource ordering leads to better system performance compared with random resource ordering.

2. The optimal partial resource ordering case results were within 4% to 23% of the optimal makespan values.

3. The system performance can be improved with changing the part dispatching sequence.

4. By changing the mathematical representation of the system, the performance of NHDAP can be improved.

5. Instead of replicating machines, maintaining their capacity equally would lead to a better utilization, when the NHDAP is used.

Minimize M

s.t

$M - x_{iK} \geq 0$ $\quad \forall i \in I$ (5.1)

$x_{ikj} - t_{ikj} \geq x_{ik-1l}$ $\quad \forall i \in I$ (5.2)

$x_{prj} - x_{qsj} + H(1 - y_{prqsj}) \geq t_{prj}$ $\quad \forall j \in J, (p,q) \in I$ (5.3)

$x_{qsj} - x_{prj} + H(y_{prqsj}) \geq t_{qsj}$ $\quad \forall j \in J, (p,q) \in I$

$x_{p(r-1)l} - x_{qsj} + H(1 - y_{prqsj}) \geq E$ $\quad \forall j \in J, \forall (p,q) \in I$ (5.4)

$x_{q(s-1)l} - x_{prj} + H(y_{prqsj}) \geq E$ $\quad \forall j \in J, \forall (p,q) \in I$

where

x_{iK}	completion time of last operation K of job i
x_{ijk}	completion time of job i operation k on machine j
t_{ijk}	processing time of job i operation k on machine j
y_{prqsj}	=1 if job p operation r follows job q operation s on machine j. 0 otherwise
H	large positive number
E	small positive number
I	set of all jobs
J	set of all machines
M	makespan

Figure 5.1. Mixed Integer Formulation for Minimimizing Makespan and Avoiding Deadlock

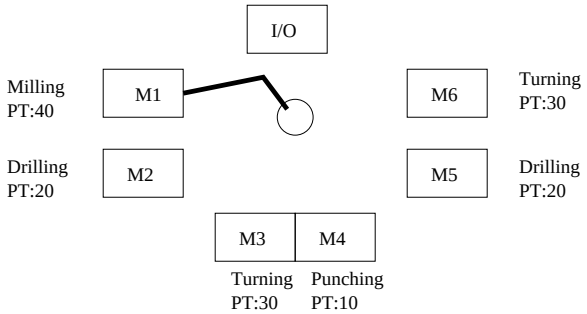

Figure 5.2. Automated Flexible Manufacturing Cell Considered in Experiment 2

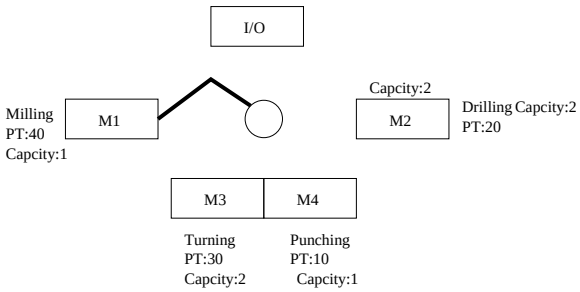

Figure 5.3. Changing the Mathematical Representation

CHAPTER 6

CONCLUSIONS AND FUTURE RESEARCH

6.1 Conclusions

The FMS must be reconfigured when the sequence of a part changes or a new part type is added. Yalcin and Boucher [40] overcome this problem by modeling the system based on physical layout and technological capabilities of the machines, rather than the process plans of the parts. The described model in [40] lacks a controller that controls part movements to avoid deadlock. Incorporating a controller for the model would permit application of the policy to FMS in a real time deadlock free control environment.

Finding a maximally permissive deadlock avoidance policy for a mixed capacity FMS is proved to be computationally intractable. This thesis attempts to evaluate the performance of a polynomially complex deadlock avoidance policy for a FMS. Results obtained using the polynomial time deadlock avoidance policy have been presented.

The Petri Net model in [40] has been extended to handle mixed capacity systems. The control to the extended architecture is implemented by adopting the NHDAP. The design of the controller differs from previous approaches in the fact that the underlying system model is not process plan based, but draws on the technological capabilities of the system resources. In this research the systems performance is measured based on the variations in makespan rather than the state space allowed by the NHDAP.

Two experiments are conducted to evaluate the performance of the NHDAP. The first experiment is a comparison between the performances of a system controlled by the NHDAP and the maximally permissive policy. The results indicate that the performance of the NHDAP with optimal partial resource ordering is comparable to the performance

of the maximally permissive policy. However the IP formulation used to determine the optimal resource ordering is computationally intractable for large systems. It is observed that part dispatching sequence have an impact on the system performance. This variations need to be investigated further. The second experiment is the performance comparison of NHDAP in different scenario's, where the representation of the capacities changes. It is concluded that NHDAP's performance increases when the multiple single capacity machines are represented as a group rather than an individual one's.

The controller described in this thesis can handle an extended Petri Net model of Yalcin and Boucher [40] using a single type resource allocation system with a centralized material handling system.

Finally an object oriented approach is used to design the tool for FMS simulation. This tool allows the implementation of system model independent of the control logic. Different control policies can be evaluated by the design engineer on the same system model and performance can be estimated. The object modeling of the simulation tool allows capturing dependencies between resources, which are difficult to identify in real time.

6.2 Future Research

In this section we identify the important research directions that improve and extend the modeling capability of the Colored Petri Net architecture and the object oriented simulator.

6.2.1 Machine Breakdowns and other Uncontrollable Events

The Petri Net model described in Section 3.2 works under the assumption that all the machines and the transporting devices are available at all times. It is important to determine the performance of the FMS and the deadlock avoidance policy in these situations.

6.2.2 Modeling the Material Handler Separately

The results indicate that the centralized material handler implementation reduces the performance of the deadlock avoidance policy. The alternative approach can be modeling the material-handling device separately from the machines. More details in this topic can be found in[22].

6.2.3 Improving the CPN Model

The Petri Net model defined in the Section 3.2 can model single type resource allocation systems. The model may be extended where more than one type of resource can be used for processing a single operation such as Conjunctive RAS or Conjunctive / Disjunctive RAS.

6.2.4 Improvements for the Software Implementation

The simulator described in Chapter 4 works under the assumption that there is only one type of transporting device available in the manufacturing system and the sequence of the parts that has to be processed is known aprior. The simulator can be extended to handle more than one type of transporting device and the dynamic sequencing of the parts.

REFERENCES

[1] Abdallah, I. B. and ElMaraghy, H. A. Deadlock prevention and avoidance in FMS: A Petri net based approach. *International Journal of Advanced Manufacturing Technology*, 10(4):704–715, September 1998.

[2] Banaszak, Z. A. and Krogh, B. H. Deadlock avoidance in flexible manufacturing systems with concurrently competing process flows. *IEEE Transactions on Robotics and Automation*, 6(6):724–734, December 1990.

[3] Barkaoui, K. and Ben Abdallah, I. A deadlock prevention method for a class of FMS. In *IEEE Transactions on Systems, Man and Cybernetics*, volume 5, pages 4119–4124, 1995.

[4] Booch, G. *Object-Oriented Analysis and Design with Applications.* Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA.

[5] Cho, H., Kumaran, T. K. and Wysk, R. A. Graph-theoretic deadlock detection and resolution for flexible manufacturing systems. *IEEE Transactions on Robotics and Automation*, 11(3):413–421, June 1995.

[6] Chu, F. and Xie, X. Deadlock analysis of Petri nets using siphons and mathematical programming. *IEEE Transactions on Robotics and Automation*, 13(6):793–804, December 1997.

[7] Ezpeleta, J. and Colom, J. M. Automatic synthesis of colored Petri nets for the control of FMS. *IEEE Transactions on Robotics and Automation*, 13(3):327–337, June 1997.

[8] Ezpeleta, J., Colom, J. M. and Martinez, J. Petri net-based deadlock prevention policy for flexible manufacturing systems. *IEEE Transactions on Robotics and Automation*, 11(2):173–184, April 1994.

[9] Fanti, M. P. and Turchiano, B. Deadlock Avoidance Policies Applied to Flexible Manufacturing Systems Modelled by Colored Petri Nets. In *IEEE International Conference on Emerging Technologies and Factory Automation*, volume 2, pages 1099–1106, 1999.

[10] Fanti, M. P., Turchiano, B. and Maione, B. Deadlock detection and recovery in flexible production systems with multiple capacity resources. 1:237–241, 1996.

[11] Fanti, M. P., Turchiano, B. and Maione, B. Diagraph Theoretic Approach for Deadlock Detection and Recovery in Flexible Production Systems. In *Studies in Informatics and Control*, volume 5, 1996.

[12] Fanti, M. P., Turchiano, B. and Maione, B. Event-based feedback control for deadlock avoidance in flexible production systems. *IEEE Transactions on Robotics and Automation*, 13(3):347–363, October 1997.

[13] Fanti, M. P., Turchiano, B. and Maione, B. Comparing digraph and Petri net approaches to deadlock avoidance in FMS. *IEEE Transactions on Systems, Man and Cybernetics*, 30(5):783–798, October 2000.

[14] Huang, Y., Jeng, M., Xie, X. and Chung, S. A Deadlock Prevention Policy For Flexible Manufacturing Systems Using Siphons. In *International Conference on Robotics and Automation*, volume 1, pages 541–546, 2001.

[15] Huang, Y., Jeng, M., Xie, X. and Chung, S. Deadlock Prevention Policy Based on Petri nets and Siphons. *International Journal of Production Research*, 39(2):283–305, January 2001.

[16] Jensen, K. . An introduction to the theoretical aspects of colored Petri nets. *Lecture Notes in Computer Science*, 803:230–272, June 1993.

[17] Kamel, B., Alloua, C. and Rabah, B. Performance of alternative strategies for dealing with deadlocks in FMS. 1:281–286, 1997.

[18] Mark, L. Flexible manufacturing system structural control and the Neighborhood Policy, Part.1 Correctness and scalability. *IIE Transactions*, 29(10):877–887, May 1997.

[19] Murata, T. Petri Nets:Propoerties, Analysis and Applications. *IEEE Transactions on Automatic Control*, 77(4):1344–1357, October 1989.

[20] Nabil Z. Nasr. *Scheduling of Job Shop Type Machining Systems with Alternative Machine Tool Routings*. PhD thesis, Rutgers University, 1990.

[21] Park, J. *Structural Analysis and Control of Resource Allocation Systems Using Petri Nets* . PhD thesis, Georgia Institute of Technology, 2000.

[22] Park, J., Reveliotis, S. A., Bodner, D. and McGinnis, L. F. A Distributed, Event-Driven Control Architecture for Flexibly Automated Manufacturing Systems. In *International Journal of Computer Integrated Manufacturing*, volume 15, pages 109–126, 2002.

[23] Ramaswamy, S. E. and Joshi, S. B. Deadlock-Free Schedules for Automated Manufacturing Workstations. *IEEE Transactions on Robotics and Automation*, 12(3):391–400, October 1996.

[24] Reveliotis, S. A and Ferreira, P. M. Deadlock avoidance policies for automated manufacturing cells. *IEEE Transactions on Robotics and Automation*, 12(6):845–857, December 1996.

[25] Reveliotis, S. A. and Ferreira, P. M. Deadlock avoidance policies for flexible manufacturing systems: the conjunctive case. In *IEEE International Conference on Robotics and Automation*, volume 1, pages 533 –538, 1996.

[26] Reveliotis, S. A. and Park, J. Deadlock avoidance policy for petri net modelling of flexible manufacturing systems with shared resources. *IEEE Transactions on Automatic Control*, 41(2):289–295, February 1996.

[27] Reveliotis, S. A. and Park, J. A polynomial-complexity deadlock avoidance policy for sequential resource allocation systems with multiple resource acquisitions and flexible routings. In *IEEE Conference on Decision and Control*, volume 3, pages 2663–2669, 2000.

[28] Reveliotis, S. A. and Park, J. Algebraic deadlock avoidance policies for conjunctive/disjunctive resource allocation systems. In *IEEE International Conference on Robotics and Automation*, volume 1, pages 70–76, 2001.

[29] Reveliotis, S. A. and Park, J. Deadlock avoidance in sequential resource allocation systems with multiple resource acquisitions and flexible routings. *IEEE Transactions on Automatic Control*, 46(10):1572–1583, October 2001.

[30] Reveliotis, S. A, Lawley, M. A and Ferreira, P. M. Deadlock avoidance policies for resource allocation systems with applications to FMS. In *IEEE Conference on Emerging Technologies and Factory Automation*, volume 1, pages 42 –48, 1996.

[31] Reveliotis, S. A, Lawley, M. A and Ferreira, P. M. On the complexity of optimal deadlock avoidance in flexible manufacturing systems. In *Proceedings of the American Control Conference*, volume 2, pages 1008–1012, 1997.

[32] Reveliotis, S. A, Lawley, M. A and Ferreira, P. M. Polynomial-Complexity Deadlock Avoidance Policies for Sequential Resource Allocation Systems. *IEEE Transactions on Automatic Control*, 42(10):1344–1357, October 1997.

[33] Reveliotis, S. A, Lawley, M. A and Ferreira, P. M. A correct and scalable deadlock avoidance policy for flexible manufacturing systems. *IEEE Transactions on Robotics and Automation*, 14(5):796–809, October 1998.

[34] Rumbaugh, J., Blaha, M., Lorensen, W., Eddy, F. and Premerlani, W. *Object-Oriented Modeling and Design.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[35] Viswanadham, N., Narahari, Y. and Johnson, T. L. Deadlock prevention and deadlock avoidance in flexible manufacturing systems using Petri net models. *IEEE Transactions on Robotics and Automation*, 6(6):713–723, December 1990.

[36] Wu, N. Necessary and sufficient conditions for deadlock-free operation in flexible manufacturing systems using a colored Petri net model. *IEEE Transactions on Systems, Man and Cybernetics*, 29(2):192–204, May 1999.

[37] Wysk, R. A., Yang, N. S. and Joshi, S. Detection of deadlocks in flexible manufacturing cells . *IEEE Transactions on Robotics and Automation*, 7(6):853–859, December 1991.

[38] Wysk, R. A., Yang, N. S. and Joshi, S. Resolution of Deadlock in Flexible Manufacturing Systems:Avoidance and Recovery Approaches. *Journal Of Manufacturing Systems*, 13(2):128–138, June 1994.

[39] Yalcin, A. *Architectures for automated manufacturing cells with routing flexibility.* PhD thesis, Department of Industrial and Systems Engineering, Rutgers University, 2000.

[40] Yalcin, A. and Boucher, T. O. An architecture for flexible manufacturing cells with alternate machining and alternate sequencing. *IEEE Transactions on Robotics and Automation*, 15(6):1126–1130, December 1999.

[41] Yalcin, A. and Boucher, T. O. Deadlock Avoidance In Flexible Manufacturing Systems Using Finite Automata. *IEEE Transactions on Robotics and Automation*, 16(4):424–429, August 2000.

[42] Yim, D. S., Kim, J. I. and Woo, H. S. Avoidance of deadlocks in flexible manufacturing systems using a capacity-designated directed graph. *International Journal of Production Research*, 35(9):2459–2475, September 1997.

[43] Zhou, M. C., McDermott, K. and Patel, P. A. Petri net Synthesis and Analysis of a Flexible Manufacturing System Cell. *IEEE Transactions on Systems, Man and Cybernetics*, 23(2):523–531, March 1993.

APPENDICES

Appendix A Petri Net

A Petri Net model consists of a set of places (P) and a set of transitions (T), in which the places correspond to the states of the model and the transitions represent the actions related to the states of the model. The places and transitions are connected by a set of directed arcs (A). An arc can only connect a transition with a place or vice-versa. An arc directed from a place to a transition is called an input-arc and an arc from a transition to a place is called an output-arc of the transition. A transition is said to be *enabled*, if there are enough tokens in each of the input places as specied by the arcs connecting the input places to the transition. An *enabled* transition can fire if the other conditions associated with the transition are satisfied. The initial state of the Petri Net is called its initial-marking. The current state of a Petri Net is the distribution of tokens to the places in the Petri Net. By firing the *enabled* transitions, the state(distribution of tokens in the places) of a Petri Net can be changed.A Petri Net is defined mathematically as,

$$N = (P, T, A, M_o) \tag{A.1}$$

where,

p = set of places

T = set of transitions

A = set of arcs

M_o = initial marking of the Petri Net.

The set of input transitions of a place in a Petri Net is represented by ${}^{\cdot}p$ and the output transitions of a place is represented by $p^{\cdot}$, in the same way the set of input places of a transition is represented by ${}^{\cdot}t$ and the output places of a transition is represented by $t^{\cdot}$. A Siphon as defined in [19], is a subset of places in Petri Net where ${}^{\cdot}S \subseteq S^{\cdot}$ i.e., every transition that has an output to S also has an input to S. A Trap [19] is also a subset of places in a Petri Net, where $S^{\cdot} \subseteq {}^{\cdot}S$, i.e., every transition having an input place in S has an output places in S. In order to prove the Petri Net to be free of deadlock it should be

proved that the Petri Net is live.A Petri Net is said to be deadlocked, when a marking is reached where the firing of a transition is impossible. Liveness guarantess a deadlock free Petri Net.If each marking of a Petri Net corresponds to a state in a system, then starting with the initial marking we can find all the possible markings(state) in the system. This is represented in form of a tree, called Reachability Tree. Consider a system shown in Fig.A.1, the corresponding reachability tree is shown in fig.A.2. For more detail on the theory of Petri Nets the reader is referred to [19].

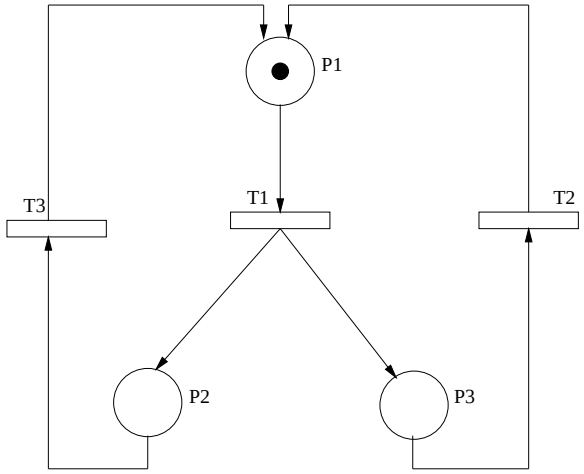

Figure A.1. Petri Net

Appendix A (Continued)

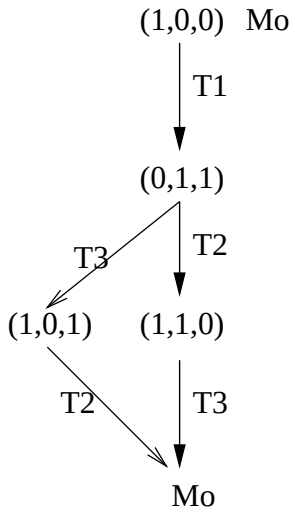

Figure A.2. Reachability Tree

Appendix B Colored Petri Net

In this section we will provide some definition and terminology regarding Colored Petri Net. The Colored Petri Nets, have type and color(C) information attached to places and tokens in the net. The color set determines the types, operations and functions that can be used in the net. It is assumed that the color sets have at least one element each. The arcs have expressions attached to them called *arc-expressions* (E). A transition in a CP-net is enabled if it is possible to bind the variables in such a way that the arc-expressions of all the input arcs evaluate when tokens are present at the corresponding input places. The transition can, in addition, have Boolean expressions called *Guard-expressions* (G), that are needed to evaluate to "true" in-order that the transition fires. The *node function* N maps each arc into a tuple where the first element is the source node and the second element is the destination node, the *color function* C maps the place p to a color set Σ, the *guardfunction* (G) maps the transition t to the boolean function and the *arcexpression function* (E) maps the arc, a to an expression. The *initialization function* (I) specifies the initial state of the Petri Net.A CP-net (CPN), can be defined mathematically as,

$$CPN = (\Sigma, P, T, A, N, C, G, E, I) \tag{B.1}$$

where,

(i) Σ is the color set,

(ii) P is a finite set of places,

(iii) T is a finite set of transitions,

(iv) A is a finite set of arcs such that, $P \cap T = P \cap A = T \cap A = \oslash$,

(v) N is a node function, defined from A into PxT $\cup$ TxP,

(vi) C is the color function defined from P into Σ,

Appendix B (Continued)

(vii) G the guard function,

(viii) E is an arc expression function,

(ix) I is the initialization function.

Appendix C Process Plan for Test Case

Table C.1. Test Case 1

	M1	M2	M3
J1	1(11)	2(15)	3(54)
J2	2(87)	1(97)	3(31)
J3	1(74)	2(36)	3(80)
J4	3(54)	2(76)	1(9)

Table C.2. Test Case 2

	M1	M2	M3
J1	1(23)	2(5)	3(14)
J2	2(38)	1(97)	3(11)
J3	1(43)	2(93)	3(49)
J4	3(36)	2(7)	1(43)

Table C.3. Test Case 3

	M1	M2	M3
J1	1(61)	2(31)	3(57)
J2	2(9)	1(97)	3(93)
J3	1(72)	2(61)	3(97)
J4	3(83)	2(25)	1(81)

Appendix C (Continued)

Table C.4. Test Case 4

	M1	M2	M3
J1	1(31)	2(57)	3(35)
J2	2(4)	1(24)	3(95)
J3	1(73)	2(62)	3(2)
J4	3(83)	2(16)	1(96)

Table C.5. Test Case 5

	M1	M2	M3
J1	1(36)	2(18)	3(30)
J2	2(92)	1(98)	3(8)
J3	1(74)	2(29)	3(15)
J4	3(17)	2(19)	1(78)

Table C.6. Test Case 6

	M1	M2	M3	M4
J1	2(5)	1(11)	3(36)	4(31)
J2	1(14)	2(43)	3(7)	4(57)
J3	3(38)	2(93)	4(43)	1(9)

Table C.7. Test Case 7

	M1	M2	M3	M4
J1	2(93)	1(89)	3(15)	4(31)
J2	1(72)	2(25)	3(54)	4(74)
J3	3(61)	2(81)	4(87)	1(36)

Table C.8. Test Case 8

	M1	M2	M3	M4
J1	2(54)	1(57)	3(95)	4(83)
J2	1(76)	2(35)	3(73)	4(16)
J3	3(9)	2(4)	4(62)	1(96)

Table C.9. Test Case 9

	M1	M2	M3	M4
J1	2(32)	1(57)	3(62)	4(10)
J2	1(75)	2(26)	3(1)	4(10)
J3	3(55)	2(26)	4(87)	1(28)

Table C.10. Test Case 10

	M1	M2	M3	M4
J1	2(55)	1(87)	3(33)	4(90)
J2	1(67)	2(42)	3(22)	4(60)
J3	3(39)	2(55)	4(30)	1(66)

Appendix D Externally Controlled Simulation Tool for FMS

This object-oriented based simulation software, developed in C++, models a Flexible Manufacturing System, with deadlock avoidance policy acts as an external controller. This controller acts in real time environment with the FMS to avoid deadlocks. The FMS and the supervisory controller are all modeled as objects. The software provides the user with the flexibility to observe the states of various parts and resources in the simulated plant at any time during the simulation and also accumulates into files, the desired performance measures at the end of the simulation. Software Requirements The following software is needed to run our simulation programs: UNIX operating system with GNU C++ compiler Installation

The software consist of the following c++ files,

1. Part Type .h
2. Machine .h
3. Cell model .h
4. Controller .h
5. Global .h
6. Header .h
7. Matrix .h
8. Petri Net .cpp
9. Rand .cpp
10. Run .sh

Running the Simulation

Step 1: Create input files to describe the FMS and the parts that are being considered
The input to the program is given thro files. Two types of files are being present, one to

represent the number of machines being considered in the system (mach.dat) and the other for the number of parts being present in the system (part.dat). Each machine and each part in turn have a separate input file, which represents their corresponding specification. The machine specification file should have two parameters, one is the vect() function of the machine and the other is the capacity of the machine. The part specification file should also have three parameters, first is the S matrix of the part, the second D matrix of the part and the last is the times required for each operation.

Step 2: Change the sequence of the parts

The simulation works by creating random sequence for the parts as specified by the user. The parts, present in the sequence should be inputted by user. This input is present in the Rand .cpp file. The user can change the part number in Rand .cpp file, so the program generates the random number accordingly.

Step 3: Change the simulation run

The simulation can be run for any number of times, for the same manufacturing system to test the efficiency. This information is stored in the Run .sh program. The default simulation run is 1. Each time when the simulation runs, the sequence of the parts are created randomly by rand .cpp program.

Step 4: Running the program

Run the file called, run.sh, by typing in the UNIX screen as "run".

Outputs

The results of the simulator are displayed in an output file, called output.dat. This file is based on the simulation runs. If the simulation is run for 2 times, then 2 output files called output1.dat, output2.dat will be present. The output1.dat gives the results from the first simulation run and the output2,dat gives the output from the second simulation run.