Error Recovery in Production Systems: A Petri Net Based Intelligent System Approach

Nicholas G. Odrey
Lehigh University

Report No. 07T-006

Technical Report 07T-006

Error Recovery In Production Systems: A Petri Net Based Intelligent System Approach

Nicholas G. Odrey
Department of Industrial and Systems Engineering, Lehigh University, USA

1. Introduction

Leading-edge companies require flexible, reliable and robust systems with capabilities to adapt quickly to changes and/or disturbances. In order to be adaptable a flexible manufacturing systems must possess the ability to (i) reconfigure the existing shop floor and (ii) automatically recover from expected and unexpected errors. One of the major problems in flexible manufacturing systems is how to effectively recover from such anticipated and unanticipated faults. Traditional techniques have addressed the error recovery problem from the point of view of defining a set of actions for a pre-specified set of errors. The main disadvantage of this approach is that not only a huge amount of coding is required but also that two undesirable situations still may occur: (i) some errors may not occur in a prespecified set during the lifetime of the system and (ii) there may be errors that cannot be anticipated. Pre-enumerating a large number of error occurrences will not guarantee that the system will not encounter a new error situation. Our intent here is to show the genesis of work into intelligent control of discrete event dynamic systems to overcome (ii) as exemplified by a Petri Net based model for large scale production systems. Petri Nets have been successfully used for modeling and controlling the dynamics of flexible manufacturing systems (Hilton & Proth, 1989; Zhou & DiCesare, 1993). Generally, in a Petri net, the operations required on a part are modeled with combinations of places and transitions. The movement of tokens throughout the net models the execution of the required operations. The content of this chapter is multi-faceted. Topics include Petri Net modeling, state space representation and associated solution techniques, hierarchical decomposition and control, hybrid modeling, multiple agent systems, and, in general, issues pertaining to our work on intelligent control of manufacturing systems.

Our focus here is on the characteristics of physical error occurrences which impose difficult challenges to discrete event control. The majority of our effort has been on workstation/cell control within the hierarchical system originally proposed by the National Institute of Standards and Technology (NIST) e.g. (Albus, 1997). The controller must first handle simultaneously production and recovery activities, and second, treat unexpected errors in real-time to avoid a dramatic decrease in the performance of the system. In the following sections we follow the modeling approach previously presented by (Odrey& Ma, 1995) which had its origins in the work of (Liu, 1993). This previous work included modeling, optimization, and control within the framework of hierarchical systems. In particular, the research was focused on efforts towards the foundations of a multilevel multi-layer hierarchical system for manufacturing control. The Petri Net formalism can handle the complexities of the highly detailed activities of a manufacturing workstation such as parallel machines, buffers of finite capacity, dual resources (multiple resources required simultaneously on one operation), alternative routings, and material handling devices to name a few. Details on the mathematical structure and definitions pertaining to Petri nets can be found in numerous sources e.g., (Zhou & Dicesare, 1993; Murata, 1989). The reader is referred to this literature for detailed underlying mathematical models. A further thrust of our work has been to enhance a multilevel multi-layer model by the incorporation of intelligent agents with the purpose of adding flexibility and agility. Thus, one objective of our effort is to determine whether it is possible to integrate Petri Nets constructs with object-oriented formalisms and have an "all in one" modeling and implementation tool for intelligent agent-based manufacturing systems. Several researchers have attempted to combine these techniques. One of the first approaches was Object Oriented Petri Nets (Lee and Park, 1993).

More recent work pertains to addressing the issue of monitoring, diagnostics, and error recovery within the context of a hierarchical multi-agent system (Odrey & Mejia, 2003). The system consists of production, mediator, and error recovery agents. Production agents contain both planner and control agents to optimize tasks and direct material flow, respectively. Here we address the error recovery agent within a hierarchical system at the workstation level in more detail. It is assumed that raw sensory information has been processed and is available. When an error is detected, the control agent requests the

action of a recovery agent through a mediator agent. In return, the recovery agent devises a plan to bring the system out of the error state. Such an error recovery plan consists of a trajectory having the detailed recovery steps that are incorporated into the logic of the control agent. In the context of Petri Nets, a recovery trajectory corresponds to a Petri subnet which models the sequence of steps required to reinstate the system back to a normal state. After being generated, the recovery subnet is incorporated into the workstation activities net (the Petri Net of the multi-agent system environment). In this research, we follow the designation of others (Zhou & DiCesare, 1993) and denote the incorporation of a recovery subnet into the activities net as net augmentation. The terms "original net" or "activities net" refer to the Petri Net representing the workstation activities (within a multi-agent environment) during the normal operation of the system. The net augmentation brings several problems that require careful handling to avoid undesirable situations such as the occurrence of state explosions or deadlocks. Intelligent agents seem to be a promising approach to deal with the unpredictable nature of errors due to their inherent ability to react to unexpected situations. Research on intelligent agents in the context of manufacturing have been mostly concentrated on the "production activities" e.g. scheduling, planning, processing and material handling (Gou, et al. 1998; Sousa & Ramos, 1999; Sun, et al. 1999) However the activities related to exception handling such as diagnostics and error recovery have received little attention. Our research aims to provide some evidence as to how the performance of a manufacturing system can be improved by using intelligent agents modeled with Petri Nets.

1.1 Statement of the Problem

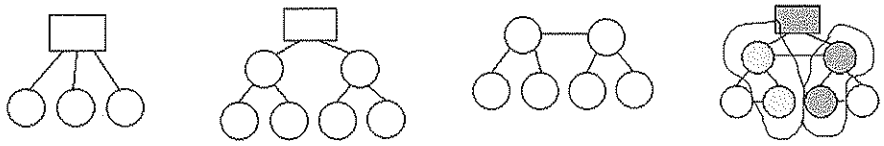
The focus in this chapter is on physical error occurrences and is directed towards supporting effective procedures for error recovery in an attempt to arrive at a reconfigurable, adaptive, and "intelligent "manufacturing system. As such, a hybridization of Petri Nets and intelligent agents seem to be a promising approach to deal with the unpredictable nature of errors due to their inherent ability to react to unexpected situations. Within this context, we investigate system learning with a hybrid Petri net-neural net structure. The following sections of this chapter first discuss the background on architectures for reconfigurable and adaptable manufacturing control. Subsequent discussions will be based on the genesis of work at Lehigh University on Petri nets from initial modeling and solution approaches to more recent work on embedding intelligent agents with Petri Nets. A hybrid nets consisting of a Petri Net with a Neural Net approach for the purpose of intelligent control is also discussed.

2. Architectures

Even though our focus in this chapter is on Petri Net modeling and error recovery , we would be remiss to not mention the underlying architecture of the systems being investigated, While some performance tests (Brennan, 2000; Van Brussel, et al.,1998) suggest that intelligent agent architectures for manufacturing systems outperform other architectures, the lack of standards on design methodologies, communication protocols and task distribution among the agents makes difficult their introduction to real-life applications. Opposed to intelligent agent-based architectures, hierarchical architectures have been conceived with the standardization issues in mind. A hierarchical architecture groups the elements of the manufacturing system into hierarchical levels, e.g. enterprise, factory, shop, cell, manufacturing workstation and equipment levels, with the purpose of coping with complexity. The major drawback of hierarchical architectures is that their structure is overly rigid and consequently difficult to adapt to unanticipated disturbances (Van Brussel, et al., 1998). To increase the functionality of the system, components at the same level may be linked. The purpose was to loosen the strict master-slave relationship of the proper hierarchical form. This resulted in the so termed, modified hierarchical form. Higher flexibility was reported with this architecture; however some problems arose in the communication links between entities of the same level mostly caused by the lack of development of the technology available at that time (Dilts et al, 1991).

To overcome the difficulties of the hierarchical architectures a heterarchical (distributed) form was proposed (Duffie et al, 1988). In this architecture a single entity did not exist at the top level as in the hierarchical scheme. In this architecture there existed a number of parts or components which "negotiate" the utilization of scarce resources. As such, a feedback signal did not have to go one level up in the hierarchy to find a response and a corrective action. A system failure in the context of this architecture meant "lack of communication" between two entities. As one communication link failed other resources were capable of establishing the linkage. There was not a single information source as the information was distributed throughout the system. Ideally the system would have been very flexible and adaptable as new elements (software or hardware) could have been "attached" to the existing ones without major disruptions. The heterarchical control architectures coped very well with disturbances and reacted quickly to changes but the lack of hierarchy led to unpredictability in the system. Consequently global optimization was almost impossible because there was neither global information nor a higher-level entity that controlled the overall performance of the system. Responses to perturbations that could be assimilated to "quick fixes" or expediting could have caused further disturbances. Further developments led to the concept of holonic manufacturing (Van Brussel et al, 1999; Valckernaers et al, 1994). The Holonic paradigm considers three primary (basic) types of agents: Order agents, product agents and resource agents, each with different goals and functionality. The basic agents are

assisted by other specialized agents namely staff agents which take the role of higher-level controllers in a hierarchy (Van Brussel et al, 1999). These staff agents are at fact in a higher level of the hierarchy but their role is only to provide expert advice to the basic agents instead of enforcing rules. To tackle with complexity and to avoid a large number of low-level agents trying to interact, agents are grouped and classified into categories. An agent is dual entity that is both a part and an autonomous entity. Related agents form aggregated agents as in a hierarchical structure but that structure differs from the traditional approach which aims for a fixed structure. The holonic hierarchy is loosely connected. This means that the configuration of the system can be changed to adapt to new conditions (Bongaerts, 1999). The ease of adaptation implies a high degree of compatibility and ex-changeability between the software and hardware elements of the system. The following figure depicts the structure of different control architectures. Notice that in the Holonic model, the modules can be reconnected and form new hierarchies. The basic elemental structure of the discussed architectures is sketched in Figure 1.



a) Hierarchical b) Heterarchcal c) Holonic

Figure 1. Basic Control Architectures

The architecture adopted in our research consists of a multi-agent system inspired by the holonic architecture developed in Europe (Van Brussel et al, 1999) and the elementary loop function (ELF) modified from the work at NIST for intelligent systems (Meystel & Albus 2002; Albus & Barbera, 2005). It has been noted that the ELF architecture is common to most intelligent systems (Meystel & Messina, 2000). In essence we are attempting to capture and implement the flexibility, adaptability, and reconfigurability required for an environment (production systems modeled via Petri nets) subject to various disturbances. A later section provides more detail as to the status of this work.

3. Workstation Modeling

3.1 Workstation Modeling with Alternative Routing

Earlier research on Petri Net modeling and analysis at Lehigh University was focused on a hierarchical structure for automated planning and control of a cellular-based shop. (Liu et al, 1997; Odrey and Ma, 1995) The adopted architecture was a hierarchical structure that followed a model developed by Saleh (1988) that was based on the hierarchical model of the National Institute of Standards and Technology (NIST) (Jones and McLean, 1986). Saleh's model incorporated both multi-levels and multi-layers. Multi-levels were designed to partition the complex structure of the shop into smaller decision and control units such as shop, cell, workstation and equipment levels. In this research we developed three different layers of control, namely the optimization, regulation and adaptation layers. The purpose was to develop a near-optimal steady state schedule along with the corresponding regulatory actions in the event of disturbances.

Following Saleh's work, Liu (1993) constructed a Timed Colored Petri Net (TCPN) model for a manufacturing cell. A three attribute coloring scheme was used and is described later. One example of a cell contained two workstations; the first workstation was a material handling device and the other described a loading/unloading station. This is shown in Figure 2 on the next page. For brevity, only a partial description of all places is given. The objectives here were (i) the construction of a PN model with rerouting capabilities, and (ii) the development a state-space representation to predict and optimize the dynamics of this system. To model a flexible manufacturing cell a machine oriented approach was undertaken and was based on modular constructs. This approach provided a construct such that a sudden addition or reduction of system resources (e.g., machines) required a minimal restructuring of the workflow within the production system. It should be noted that it can still take a great amount of effort for modeling of a PN based system. The TCPN cell model in this earlier research was determined by the system capacity of the cell and the production workload. The system capacity included the number of workstation types, the number of parallel resources in a workstation, and the material handling system (MHS). The production workload included job types, the processing times, and the routing of jobs.

From a Petri net viewpoint the system capacity dictated the configuration of the cell model whereas the production workload determined the number of job tokens and operational circuits in the workstation subnets. Figure 2 depicts a TCPN for the system but note that the recovery from machine breakdowns is not included in this figure. Two job types were modeled in the cell. The two workstation subnets and the load/unload (L/UL) subnet are connected in parallel through the MHS subnet. The parallel subconnections subnets fulfilled a requirement of a random direction material flow. The interface between cell entities are the two sets of places {P4, P7, and P15} and {P27, P25, and P26} which represent the input queues and output queues to the L/UL station and workstations W1 and W2, respectively. In this model the number of tokens in each closed-loop subnet represented the total availability of a particular resource in a cell entity. For example, two tokens in place P9 represent two identical machine resources in workstation W1, whereas a single token in places P6 and P12 represent a single space for the input and output buffer, respectively, of workstation W1. In a TCPN cell model, token colors are useful for both visual identification and mathematical representation. Consider place p1 in Figure 2. Two job types identified by their different token colors (one black dot and a white circle pattern). In the case of parallel resources, i.e., two parallel machine tokens in P9, distinctive colors would be used for individual resource identification.

In this modeling approach, a three-attribute coloring scheme (part number, workstation number, resource number), was used to differentiate token colors. Part number (pt#) represents the job number; Workstation numbers (wks#) indicates the workstation where a part is currently being processed or is to be processed; a resource number refers to either a buffer number (b#) or a machine number (m#) in a particular workstation, an equipment number (e#) in a load/unload station, or a device number (d#) for material handling systems. These resource attributes provide a tracking record for the resource assignment decisions. Hence, a token color, (i, j, m), indicates that the token is the i^{th} job which uses the m^{th} resource in the j^{th} workstation. The coloring scheme is embedded in the matrix representation of the TCPN cell model used in the system dynamic equations.

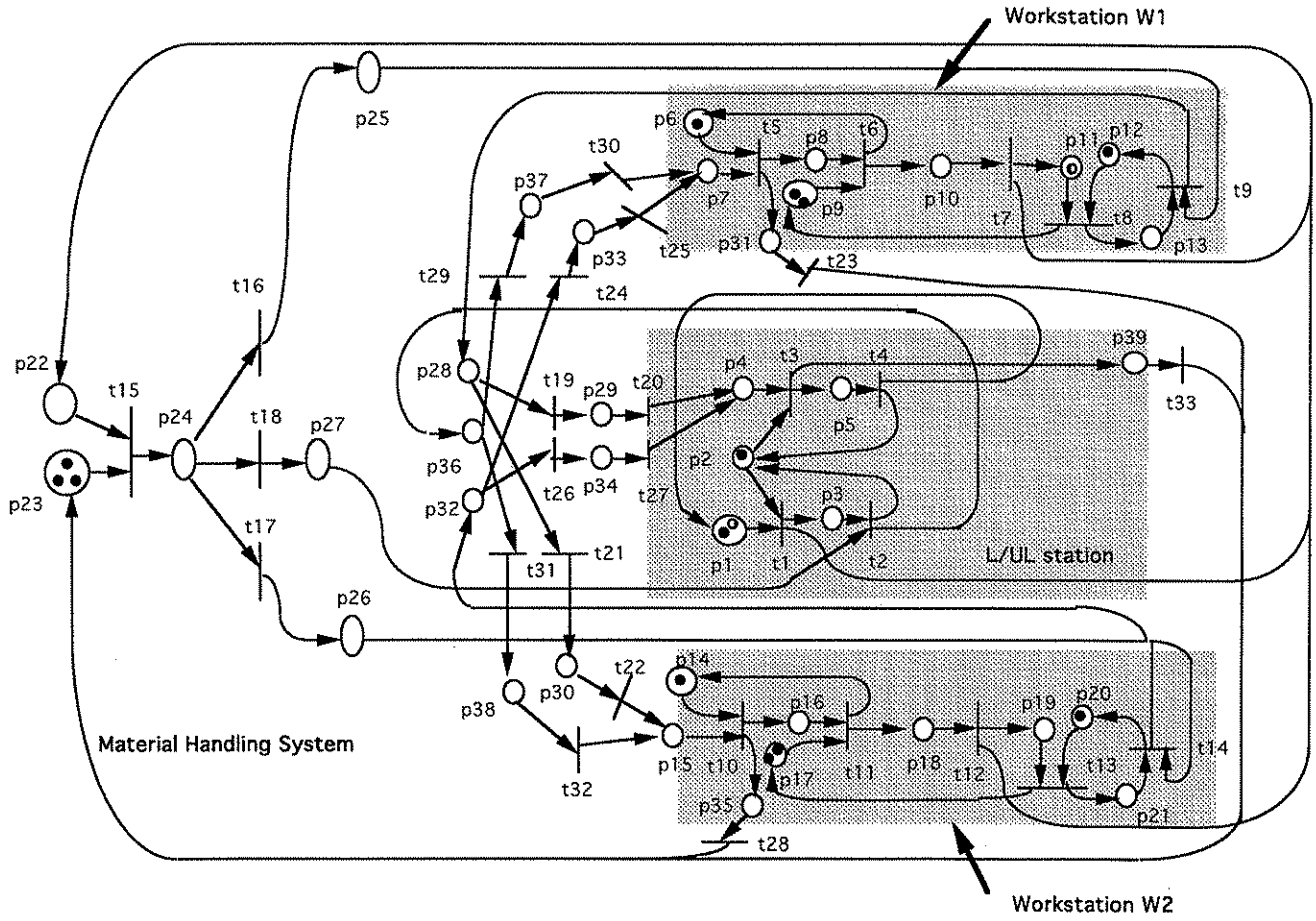


Figure 2: Time Colored Petri Net for Two Workstations a load/Unload station, and a Material Handling System (Liu, et al. 1997; Liu, 1992)

In this research, the modular construct was a convenient restructuring method proved adaptable to changes in the production environment. The possible system configuration changes were categorized into two types: changes in a physical entity or changes affecting jobs. In the event of adding or deleting a physical entity (e.g., a workstation), the workstation subnet was connected or disconnected to/from the MHS subnet. In this earlier work, if machine breakdowns occurred the corresponding machine resource token was simply stopped from circulating in the subnet until recovery. For entity disruption, the overall model structure remained relatively the same. Any changes affecting jobs consisted of a cancellation of jobs or changes in the job routing information. Job routing changes involved the deletion of operation circuits from previous stations and the addition of operation circuits to the new stations. Furthermore, in this research, for each physical entity considered in a cell there existed a 1-to-1 representation in the TCPN model. As such, each operation performed had a corresponding processing time associated with the operation circuit in the processing workstation and each system resource corresponding token representation. Parallel resources were represented by multiple resource tokens of the same color in the cell model. The total number of token types represented the total number of that resource types available in the system. This TCPN development methodology provided a safe, bounded, and live model.

Naturally, an important consideration was the representation of a disruption (error) occurring and a possible rerouting strategy. This was approached by noting that machine breakdowns can be satisfied by regarding a machine breakdown as an external input (the firing of a transition in a Petri net model). This additional structure provided an immediate transfer of tokens from a place (which represents processing) without waiting for the elapse of processing time. Figure 3 depicts a TCPN workstation which can be used to represent an alternative routing logic for machine breakdowns.

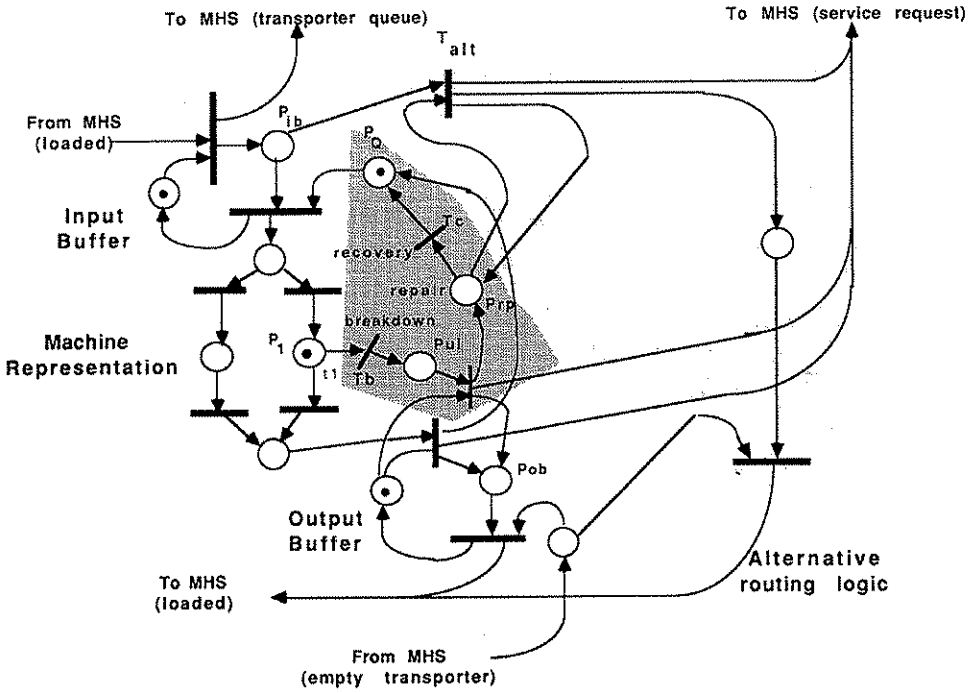


Figure 3: A Workstation Petri Net Representation with an Alternative Routing Logic (Liu, 1993)

Firing transition Tb represents the fact that a machine breakdown has occurred. The token in place P1 is released to Pul. In such an instance the remaining processing time (t1) is set to zero. This unload place may have a queue and waits for an output buffer to unload the part from the breakdown machine. In this representation 3 tokens are generated once an output buffer is generated. A machine token passes to a repair process Prp whereas a token in place Pr signals a service request for the material handling system (MHS). Simultaneously a job token is outputted to place Pob (place signifying an output buffer). Other transitions noted in Figure 3 consist of Talt (initiate re-rote mechanism to alternative machines) and Tc (to indicate recovery of machine from the breakdown). The firing of transition Tc causes the machine token to be returned to the common queue (place P_Q) and stops the firing of the alternate machine transition Talt. At the time this scheme was developed to overcome drawbacks associated with 1) an inhibitor arc approach (Teng & Black, 1990) and, 2) a timed Petri net

representation by (Barad & Sipper, 1998). An inhibitor arc approach cannot provide a systematic mathematical representation in the event of changes in transition firing rules. The work here was a modification of the latter TPN approach.

3.2 Workstation Analysis

The state space representation used to analyze the workstation Petri nets was a modification of the traditional state equation (Murata, 1989) with the incorporation of equations for the remaining processing times of every timed place. The conventional state space representation can be written as:

$$M(k+1) = M(k) + L\,u(k) \tag{1}$$

where $M(k)$ is the marking of the Petri Nets in time k, L is the incidence matrix and $u(k)$ is the vector of transition firings. The reader is referred to Murata (1989) and Al-Jaar and Desrochers (1995) for details on this equation.

The state space representation developed by Liu (1993) considers operational, precondition, post-condition and resource places. Only operational places (those where actions are carried out) have associated processing times. The other places, as their name suggest, represent conditions (e.g. idle, ready) (Liu et al, 1997). The modified structure contains two different "marking" vectors: the first marking vector ($M_p(k)$) is the conventional marking vector (Murata, 1989) that accounts for the number of tokens in each place; the second one ($M_r(k)$) is the remaining processing time vector i.e. a vector containing the remaining time for the next transition firing for each place.
The state space equation is stated as follows (the dimensions of these matrices are omitted for simplicity):

$$X(k+1) = A(k)\,X(k) + B(k)u(k) \tag{2}$$

$$X(k) = \begin{vmatrix} M_p(k) \\ M_r(k) \end{vmatrix} \tag{3}$$

u(k) is a control vector that determines which transitions fire at time k. Define uj(k) as the jth position of u at time k. uj(k) = { 1 if transition j fires, 0 if it does not } Mp(k) is the marking vector at after k transition firings; Mr(k) is the remaining processing time vector after k transition firings; A(k) is the system matrix and it is partitioned as follows:

$$A(k) = \begin{vmatrix} [I] & [0] \\ -\tau(k)[P] & [I] \end{vmatrix} \tag{4}$$

[0] Zero matrix;
[I] Identity matrix
τ(k) Time for the next transition firing.
[P] Diagonal matrix that serves to distinguish operational places from resource, precondition and post-condition places.
Pii = {1 if place pi is an operational place; 0 otherwise}
Pij = 0 when i≠j

B(k) is the distribution matrix that transforms the control action u(k) into token evolution i.e. addition and removal of tokens when firing a transition represented in vector u(k).

$$B(k) = \begin{vmatrix} [L] \\ [W]\,[L]^{+} \end{vmatrix} \tag{5}$$

[W] = Processing time matrix for operational places.
[L]= Incidence matrix [L] =[L]+ - [L]-
[L]+ = Incidence output matrix that accounts for the addition of tokens in output places.
[L]- = Incidence input matrix that accounts for the removal of tokens from input places.

The dimension of these matrices is determined by the number of places, transitions and colors in the system. For a detailed discussion and explanation see (Liu, 1993; Liu et al., 1997). This representation was the basis for an optimal control formulation for scheduling optimization. A near-optimal solution was found by using forward dynamic programming on the sequence of states (markings) generated by the state equations.

3.3 Petri Net Decomposition

In the process of establishing a hierarchical Petri net-based workstation model, issues can be categorized into different classes where each class occurs at different levels of the hierarchy. At the Petri net modeling level two decision classes were identified, namely, generation of conflict-free sequences and the determination of process steps sequences. In order to facilitate the decision-making and performance evaluation processes, a hierarchical system of state equations for the Petri nets based model was studied. The general form of the hierarchical state equations have previously been state in equations 2 through 5. An example of the net decomposition for an assembly workstation is indicated in Figure 4. For the top level TCPN model

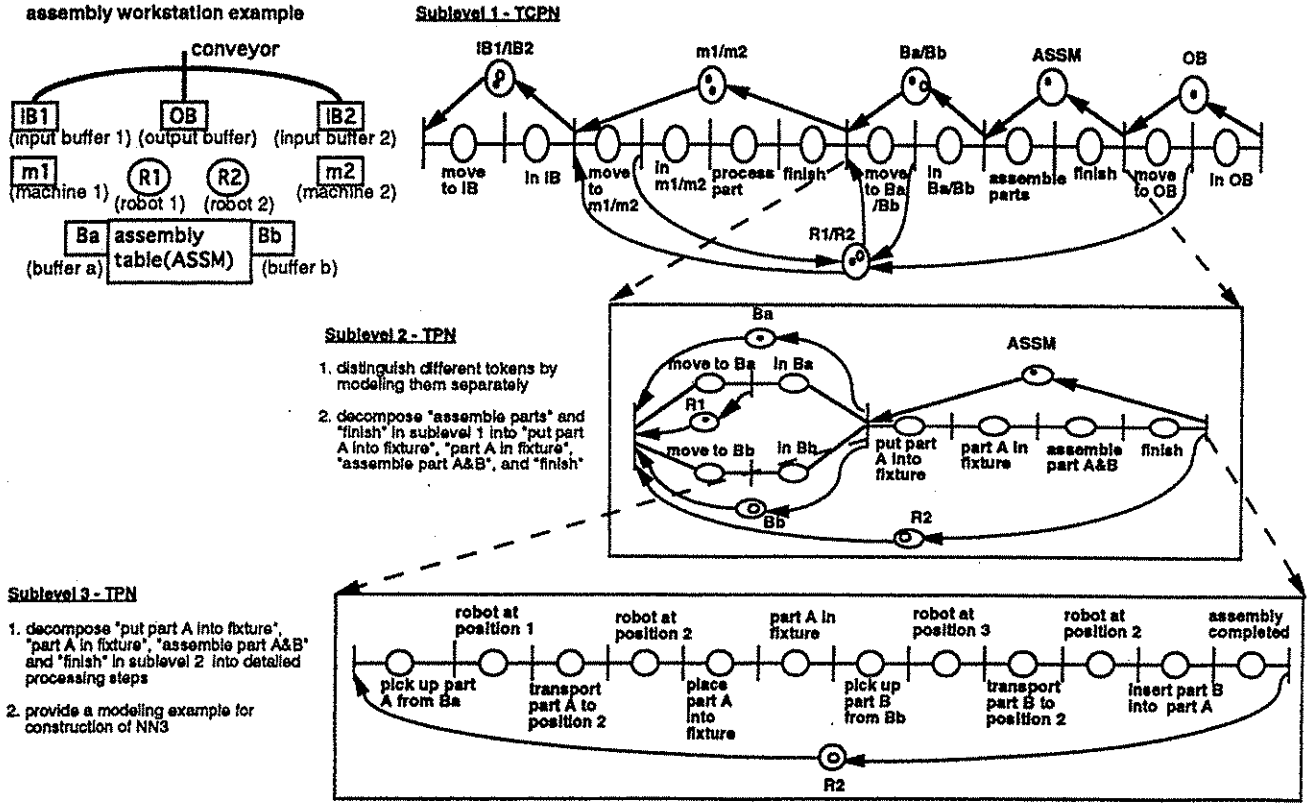


Figure 4: An example of decomposition of a multi-layer Petri net model for an assembly station (Ma & Odrey, 1996)

(termed sublevel 1), the state dimension depends on three values: (1) the sum of all colors on tokens associated with places which represent the process of manufacturing individual parts, (2) the sum of all colors on tokens associated with places which represent the process of handling assembled final products, and (3) the sum of all colors on tokens associated with the resource places. When decomposing the TCPN model to a sublevel 2 TPN model the system can be viewed as a two-level hierarchical Petri net with one discolored TPN at the upper level and several subnets, which are also modeled by TPNs, at the lower level. Between upper and lower levels, interface places are added that serve as connectors between two levels. For a state space representation, the discolored TPN at the upper level and each detailed subnet at the lower level can be individually represented using TPN state equations. Thus, the system state equations for the sublevel 2 TPN workstation model are obtained by combining all the TPNs and augmented to incorporate the interface places, i.e. all the vectors/matrices in the subnets are become the subvectors/submatrices in the sublevel 2 TPN workstation state equation. For example, the distribution matrix for the sublevel 2 TPN model would have the form of the matrix given below. L^i is a

distribution submatrix of TPNi .The bottom row denotes the distribution submatrices of the interface places and the input/output transitions associated with TPNi. Details of this work can be found in (Odrey & Ma, 2001). This multi-level, multi-layer Petri net framework establishes layers to provide the linkage between high-level abstract information for discrete systems and

$$L = \begin{bmatrix} L^1 & [0] & [0] & \cdots & [0] \\ [0] & L^2 & [0] & \cdots & [0] \\ [0] & [0] & L^3 & \cdots & [0] \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ [0] & [0] & [0] & \cdots & L^J \\ L_c^1 & L_c^2 & L_c^3 & \cdots & L_c^J \end{bmatrix} \tag{6}$$

low-level numeric data for continuous systems. Different nets are used to represent different levels of complexity. Three functional distinct subnets which are the basic building blocks for the Petri net workstation model were proposed to represent higher level abstract commands such as "move," "process," and "assemble". These subnets allow basic routing information to be incorporated in the model through a bottom-up approach in a systematic manner. The process task can then be decomposed into a Petri net representation of process steps which follow a feature-based process plan. Alternative sequences and resources are incorporated in the process task model to provide flexible operation instructions. Dynamic state space equations correspond to each sub-level in the hierarchical Petri net graphical representation. These state equations are used in current research to evaluate various control strategies and performance workstation operations in a unifying way.

4. Intelligent System Approaches Using Petri Nets

4.1 Intelligent Agent Approaches

Current efforts are directed towards the aspects of error recovery associated with intelligent agent-based manufacturing systems and has been motivated by the work done at Lehigh University. As noted above, previous work included modeling and optimization and control of hierarchical systems. Our focus is to enhance this multilevel multi-layer model with the incorporation of intelligent agents with the purpose of adding flexibility and agility. This on-going effort investigates (i) architecture reconfigurations with enhanced capabilities of flexibility and adaptability, (ii) the adoption of adequate modeling techniques and their mathematical representation (in particular, modifications to the previous Timed Colored Petri Net models developed), (iii) modeling the aforementioned intelligent agents with Petri Nets, and (iv) model testing.

Our motivation has its origins in the research mentioned in the previous sections in addition to models incorporating intelligent agents for manufacturing operations which appeared in the eighties and nineties as an alternative to the shortcoming of hierarchical and heterarchical architectures. Some of these additional approaches include Bionic Manufacturing (Okino, 1993), Fractal methods (Warnecke, 1993), the MetaMorph Architecture (Wang et al, 1998; Maturana et al, 1998). These approaches preserve a hierarchy that controls the autonomy of individual agents, but unlike the hierarchical architectures, the relation-ship between low and high level controllers (agents) does not follow the master-slave scheme. The low level agents have a high degree of autonomy as in the heterarchical approach but still have "loose" links with higher-level agents. An intelligent agent based approach attempts to preserve the advantages of both hierarchical and heterarchical approaches but at the same time avoids their drawbacks. The architectures mentioned present differences primarily in the definitions of the intelligent agents, the degree of reactivity versus long-term planning, the degree of adaptation and reconfiguration, and the communication methods between agents. For example, in the Holonic, Bionic, MetaMorph and Fractal approaches the intelligent agents are loosely connected and their structure can evolve over time; the RCS resembles a hierarchical architecture whose structure is primarily fixed. In the Holonic, MetaMorph and RCS approaches the system has a set of fixed predefined goals. In the Fractal approach the agents negotiate their goals (Tharumarajah et al, 1996). Bionic architectures (Okino, 1993) do not set long-term goals but seek essentially adaptation to the environment. In the Holonic manufacturing approach parts, computers and resources are considered as intelligent agents. The other approaches regard schedulers, planners, controllers and resources as agents, but exclude parts.

It should be noted that the concept of Intelligent Agents was built around the Object-Oriented Programming (OOP) paradigm (Tharumarajah et al, 1996). The underlying principle of OOP is the encapsulation of attributes and methods into code units called classes. The code embedded in a class defines its internal actions and the relationships with other classes (Wyns and Langer, 1998). In the intelligent agent approach, each agent becomes an object with clearly defined functionality

and attributes. Thus these concepts of OOP such as instantiation, inheritance, and polymorphism can be applied directly to the theory of intelligent agents (Venkatesh and Zhou, 1998). To date OOP platforms are the preferred choice for control software development (Gou et al, 1998). Some of its advantages over conventional programming include reusability, portability and expandability. OOP seems to be the natural approach to implement the control software for intelligent agent-based architectures (Gou et al, 1998). Venkatesh and Zhou (1998) have pointed out need for integration of control and simulation and modeling software to expedite the system development. In other words, the control software should not be exclusively dedicated to issue commands to the components of the manufacturing systems but to optimize the system performance. It should also be noted that all agents are objects but not all objects are agents. Agents are autonomous entities that have choices and control on their behavior; objects may be totally obedient (Jennings, 2000).

4.2 Multi-Agent Systems with Embedded Petri Nets

Our more recent work presents an architecture for control of flexible manufacturing systems which is a synthesis of hierarchical and intelligent agent-based systems (Odrey & Mejia, 2003). The approach undertaken provides responsive and adaptive capabilities for error recovery in the control of large scale discrete event production systems. A major advantage of this is the ability to reconfigure the system. The communication links between agents can be re-directed in order to form temporary clusters of agents without modifying the internal structure of the agent. At the same time, having the hierarchical structure greatly facilitates the organization of new groups of agents. In our approach, agents possess the freedom to move within their hierarchical level but cannot move out to another level. The approach, based on Petri Net constructs is expected to improve the performance of agent-based systems because (i) it decentralizes the control activity for complex and unusual failure scenarios (ii) provides basic autonomy to resource agents (iii) follows a proved design hierarchical design methodology and, (iv) defines clearly the responsibilities of control and resource agents. A thrust of this effort was to determine whether it is possible to integrate Petri Nets constructs with object-oriented formalisms and have an "all in one" modeling and implementation tool for intelligent agent-based manufacturing systems.

At the time of this investigation the major focus was on the diagnostics and error recovery activities in the context of intelligent agent-based architectures for semi-automated or autonomous manufacturing systems. Our approach addressed the issue of combining the discipline of hierarchical systems with the agility of multi-agent systems. We adopted in-part the holonic paradigm (Van Brussel et al, 1999) for description of the three primary (basic) types of agents: Order agents, product agents and resource agents, each with different goals and functionality. The basic agents are assisted by other specialized agents namely staff agents which take the role of higher-level controllers in a hierarchy. In particular, the focus was on the construction of a re-configurable system having production agents, error recovery agents, and a classifier/coordinator/ mediator agent structure connecting production and recovery agent hierarchies. In addition, the relationship to the previous work at Lehigh University pertaining to a multi-level, multi-layer hierarchy control was established. This latter hierarchy, based on Petri net constructs, serves, in one sense, as a retrieval based resource for process planning and generation of re - cover plans to the production and recovery agents within the proposed multi-agent system. An objective of this effort was to provide a test-bed for comparison of purely hierarchical systems, non-hierarchical but highly re-configurable multi-agent systems, and a hybrid combination which was the focus of the investigation presented here. Our primary efforts are on a hierarchical intelligent agent-based system linked to a structure of agents dedicated exclusively to diagnosis and error recovery tasks. Our work has focused primarily on error recovery strategies at the workstation level in an intelligent-agent based system and is still on-going.

Unlike the traditional structure (Albus, 1997) in which the control function is exerted top down, our approach provides the agents basic control capabilities that allow them to react to common and local disturbances. In addition, specialized control and recovery agents assist these production agents on complex diagnostics and recovery tasks. This approach is expected to combine the discipline of hierarchical systems, but with the inherent ability to react as would be congruent with intelligent agent-based systems. Here we adapt the intelligent agent principles to hierarchical control models. The most significant difference lies in the definitions of a workstation controller and a workstation agent. In a broader scope, a workstation agent comprises a workstation controller, a number of resources (conveyors, machines, tools, fixtures, etc.) and their respective controllers [Van Brussel, 1998]. The workstation agent acts as a single decision unit when negotiating with higher-level agents. At the same time a workstation agent is considered as a system when controlling and coordinating its components (equipment agents and error recovery agents).

4.2.1 System structure

Our approach is based on prior work on hierarchical architectures as outlined in previous sections. As such our model shares a number of features with the prior work, namely, hierarchical decomposition of activities, sensor strategies, methods for diagnosis and error recovery, and modeling techniques. The reader is referred to (Odrey & Mejia, 2003) for a more detailed

explanation of this section. A sketch of the integrated control architecture is shown in Figure 5. The architecture is partitioned into three segments. A mediator agents structure is positioned between and separates a production agents architecture and a recovery agents architecture. Each of these structures follows a hierarchy and communication can be at and among different levels within the hierarchy. To-date, we distinguish between cell level and workstation level production agents which communicate through mediator agents. In the schema adopted if an error occurs at the shop floor and the workstation agent cannot produce a satisfactory recovery plan by itself, such an agent requests the actions of the workstation mediator agent. The workstation agent provides all the available information pertaining to the error which should include sensor readings, location, priority, etc. The mediator agent classifies the error and matches the error with a recovery agent at the same hierarchical level. The recovery agent attempts to produce a recovery plan and if it succeeds the plan is communicated back to the mediator. At the same time, if the error exceeds a pre-determined time threshold, the workstation agent sends a message to the cell agent (higher level) informing of the abnormality. The cell agent takes this new input and determines whether or not rescheduling pending jobs is necessary. In order to keep the system running, the workstation agent adopts a temporary measure e.g., dispatching rules. At this point, this is the maximum the workstation agent can do since it lacks of the information and methods to perform global optimization. When a new schedule, generated by the cell agent, is available, the workstation agent attempts to adapt the new plan to the current conditions. In this way, each agent contributes independently to the overall optimization of the system.

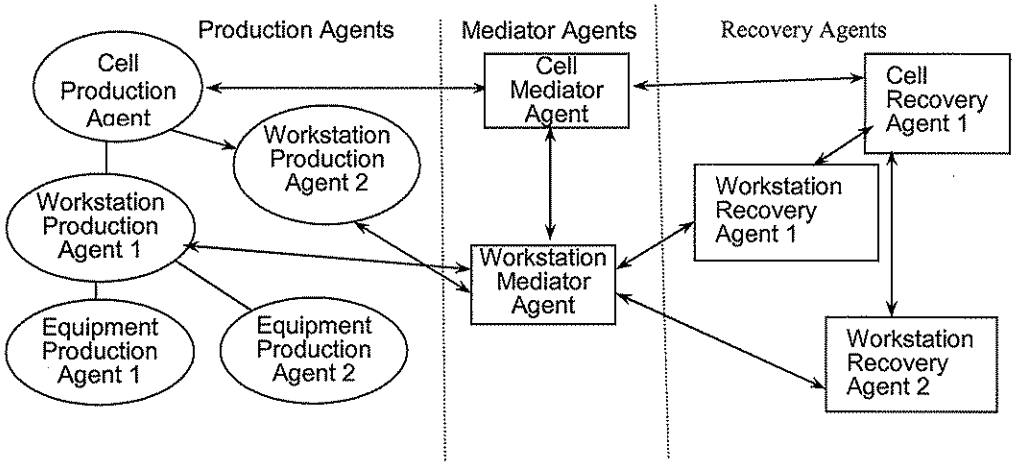


Figure 5 : Error recovery agents within an intelligent agent hierarchical architecture (Odrey& Mejia, 2003)

The workstation agent requires additional techniques to optimize the realization of the process plan of all the current jobs that have been allocated to it. In our approach, the workstation agent itself constructs a Petri Net model of the sequence of coordinated activities for all current jobs using a multi-level multi-layer Petri Net approach [14]. In this approach the sequence of activities at the workstation level and the required resources are modeled using several "layers" which represent degrees of modeling abstraction (from generic activities to highly specific tasks). As noted in the previous section the highest layer is modeled with a Timed Colored Petri Net (TCPN). The TCPN layer is then "unfolded" in several layers with different degrees of detail. Lower levels are represented by Timed Petri Nets and Ordinary Petri Nets. For each of these nets in order to track the system status state equations can be developed. These equations serve to determine the flow of tokens and the remaining process times for each operation place provided by a sequence of transition firings.

The BRIC (Block-like Representation of Interactive Components) was chosen as our initial modeling tool in that adoption serves very well to develop control software in that it provides the most important features of OOP (Object Oriented Programming). Additionally, BRIC provides a graphical representation of the behavior of a multi-agent system. In the BRIC approach each agent is modeled by a Petri subnet that comprises an internal net representing the agent's methods and a set of "communication" places. Agents are linked together by through external transitions and arcs. Tokens flowing between communication places serve as message between agents. The complex data structure is embedded in the colored token coloring scheme. For example, a token in an input message interpreted as a work order could include several different labels such as sender id, job priority, job constraints , etc. Conventional token rules of Colored Petri Nets (CPN) apply to the communication places. A token can go from a conventional place to a communication place and vice-versa. For further details the reader is referred to (Odrey & Mejia, 2003). It should be noted that the agent interaction/communication structure is an on-going investigation. Other techniques are currently being investigated.

4.2.2 Mediator agent

Mediator agents are the link that connects the production structure with the recovery structure. Their function is to facilitate the communication between production and recovery agents. The primary functions of mediator agents are: 1) filtering/ processing sensory information from production agents, 2) classifying errors and performing preliminary diagnostics based on feedback information, 3) matching errors that occur on the shop floor with error recovery agents, and 4) communicate recovery plans to production agents. A BRIC model of the structure of a mediator agent is shown in Figure 6. Places are as defined. In this schema, a mediator agent first receives a request (P11) and classifies the error (P12) according to a set of corrective preliminary actions. We adopt here the approach of our previous work (Ma, 2000) in which error classification was performed using a Petri Net embedded in a neural network linked to an expert system. Next, a matching module embedded in the recovery agent attempts to match the error with recovery agents capable of generating a recovery plan for the error that occurred (P13). The issue of matching errors with recovery agents is a subject of further research. If a suitable recovery agent is found, the mediator sends it a request for recovery (P14). A token in P15 represents that a recovery plan (or a failure to generating a plan) has been received. The mediator agent evaluates the received plan (P16) and communicates it to the corresponding production agent (P19). Provisions are made should the mediator agents require aid from other mediators (places P17 and P18).

P11: Receiving recovery request message
P12: classifying errors
P13: Matching classified error with recovery agents
P14: Sending request messages
P15: Receiving responses from recover agents
P16: Evaluating responses
P17: Receiving responses
P18: Sending messages to mediators
P19: Communicating recovery plans

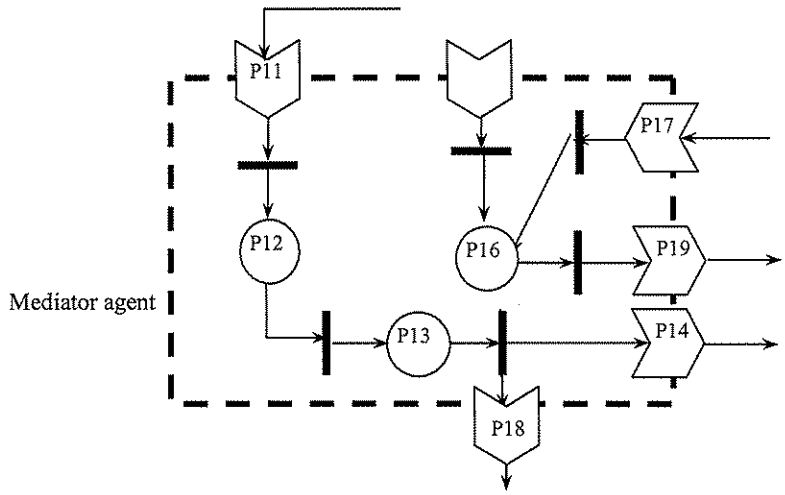


Figure 6. A BRIC model of Workstation Mediator Agent (adapted from Odrey & Mejia, 2003)

4.2.3 Error recovery agents

The recovery agents at the workstation level are responsible of three major tasks: (i) screening recovery requests sent by mediators, (ii) performing in-depth diagnosis, and (iii) generate recovery plans for expected and unexpected errors. The BRIC model of a workstation recovery agent and place definitions are shown in Figure 7. Once an error is classified a token is placed in P20 and further diagnosis is performed when a marking reaches P21. When a root cause is known and classified, a plan can be generated (P22) and sent to the appropriate agent via P23 and P24.

Our current efforts here focus on developing an automated reasoning technique for generating recovery plans. The recovery plan generation primarily depends on whether or not the error has been anticipated. Anticipated and unanticipated errors require two different strategies: In the case of anticipated errors, a recovery plan is generated by matching the error with a recovery task in a lookup table (Odrey & Ma, 1995). Unexpected errors require more complicated (deep) reasoning that implies finding and matching error patterns with gross recovery plans or searching alternative paths to return or advance the system to an error-free state. Previous work at Lehigh University (Ma, 2000) was concentrated on generation of gross recovery plans using Neural Networks. The last stage of modeling our proposed architecture consists of linking the agents to form a Petri Net model of the control structure and can be found in (Odrey & Mejia, 2003).

5. Error Recovery Approaches

Error recovery is the set of actions that must be performed in order to return the system to its normal state (Odrey and Ma, 1995; Seabra-Lopes and Camarinha-Matos, 1996). The key concept is that there should exist at least one sequence of actions to bring the system to its normal operation. The purpose of error recovery is to find the best actions that minimally disrupt the system while down-time is minimized. Our work presented here follows 2 approaches: 1) the first section used an

augmented Petri Net approach and 2) a subsequent section was an attempt to provide a hybrid net by joining Neural Nets with Petri Nets. This was done for a workstation level controller with in a hierarchical system following the work done at NIST. Both of these approaches are discussed in subsequent sections.

P20: Evaluating classified error
P21:Diagnosing errors (in-depth Diagnosis)
P22: Generating recovery plan
P23: Sending recovery plan to mediator agents
P24: Receiving recovery requests messages

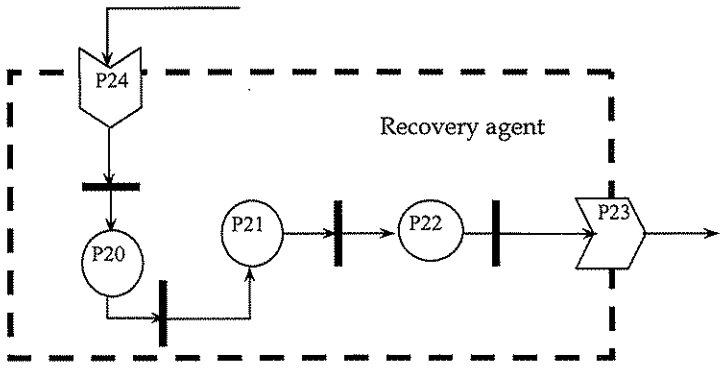


Figure 7. Workstation Recovery Agent Structure (Odrey & Mejia, 2003)

5.1 Definitions for diagnostics and error recovery

An *error* occurs when the observed behavior conflicts with the desired behavior of the system (Odrey and Ma, 1995; Seabra-Lopes and Camarinha-Matos, 1996). Similarly, (Chang et al. 1991) defined that an error occurs when a resource reaches an undesired state. (Kokkinaki & Valavani, 1996) define errors as manifestations of faults. A *fault* is the cause of an error (Chang et al, 1991). As long as the error is not detected or does not produce a failure, it remains latent. A *failure* occurs when a resource does not deliver a service. For example a worn gear in an automated fixture prevented a part to be accurately positioned on a machine tool. Because of this, the part could not be correctly machined and resulted in a bad assembly. The worn gear is the fault that generated the errors and failures. The error is a positional error (the undesired state) and the failure is the wrong assembly (a service that could not be delivered). *Diagnostics* is the activity in which the source fault(s) is (are) determined and isolated (Odrey and Ma, 1995). When a failure is detected, the operation that failed is not necessarily the source of the failure. A source fault is propagated through the system generating errors and failures. Diagnostics involves backtracking the failed operations to the source fault. The failure propagation tree is the tool that serves the backtracking actions by linking operations until the one that failed is found (Chang et al, 1991). In our research incorporating a multi-agent approach faults are considered as inconsistencies in the behavior or status of an agent or inconsistent interactions between agents and between agents and the environment. The environment is everything outside the boundaries of the intelligent agents. For example, a broken gear that produces paralysis in the machine spindle is an abnormal behavior of a resource agent; an out-of-tolerance part is an abnormal state of a part agent; failure to grasp a part is an inconsistent interaction between the robot agent and the part agent and blocking a robot agent by an external entity is an undesired interaction between the robot agent and the environment. When faults occur the workstation controller agents and the low level agents that depend on the workstation controller, namely machine and part, investigate the reasons of the failure. The low level agents investigate their own internal failures and the workstation controller investigates its own internal faults and the interactions between the part and machine agents and between those two and the environment. For now, the work has been focused on Petri net approaches.

5.2 Augmented Petri net approach for error recovery

The approach taken here was based on integrating Petri subnet models within a general Petri net model for a manufacturing system environment, and, in particular, a workstation controller. In essence, the error recovery plan consists of a trajectory (Petri subnet) having the detailed recovery steps that are then incorporated into the workstation control logic. The logic was based on a Timed Petri Net (TPN) model of the total production system. The Petri subset models consist of a sequence of steps required to reinstate the system back to a normal state. Once generated, the recovery subnet is incorporated into the Petri net model of the original expected (error free) model. The workstation controller is the entity responsible for the coordination, execution and regulation of the activities at the physical workstation. The workstation controller receives a higher level command, generally form a higher level controller that issues a set of operations to be performed by the workstation with desired start and finish times. The workstation controller decomposes such a command into a lower level set of coordinated activities. In addition to executing activities, the workstation controller should also provide a reactive and

adaptive response to errors and other disturbances (Odrey and Ma, 1995). In this work we followed the modeling approach discussed in previous sections. The following discussion is a summary of (Odrey and Mejia, 2005).

5.2.1 Relationship to previous work

The characteristics of physical error occurrence impose difficult challenges to the workstation controller. The controller must first handle simultaneously production and recovery activities, and second, errors that appear unexpectedly must be treated in real-time to avoid a sudden decrease of performance. Examples of automated reasoning systems for error recovery procedures, such as neural nets include the work of (Seabra-Lopes et al., 1996; Kokkinati and Valavanis, 1996) and our work discussed in section 5.3. As previously discussed (section 4), work addressing the issue of monitoring, diagnostics, and error recovery within the context of a hierarchical multi-agent system consisted of production, mediator, and error recovery agents. Production agents contain both planner (scheduler) and control agents. In this section we address the error recovery agent within the hierarchical system at the workstation level in more detail. It is assumed that raw sensory information has been processed and is available. When an error is detected, the control agent diagnoses the error and requests the action of a recovery agent via mediator agents discussed in section 4.2.2. In return, the recovery agent devises a plan to bring the system out of the error state. Such an error recovery plan consists of a trajectory having the detailed recovery steps that are incorporated into the control agent logic. A forward trajectory is the most desirable, but at the same time it is the most difficult to implement with automated reasoning systems (Fielding, et al., 1987). In the context of Petri Nets, a recovery trajectory corresponds to a Petri subnet which models the sequence of recovery steps required to reinstate the system back to a normal state. A schematic of error recovery trajectories is given in Figure 8 as follows:

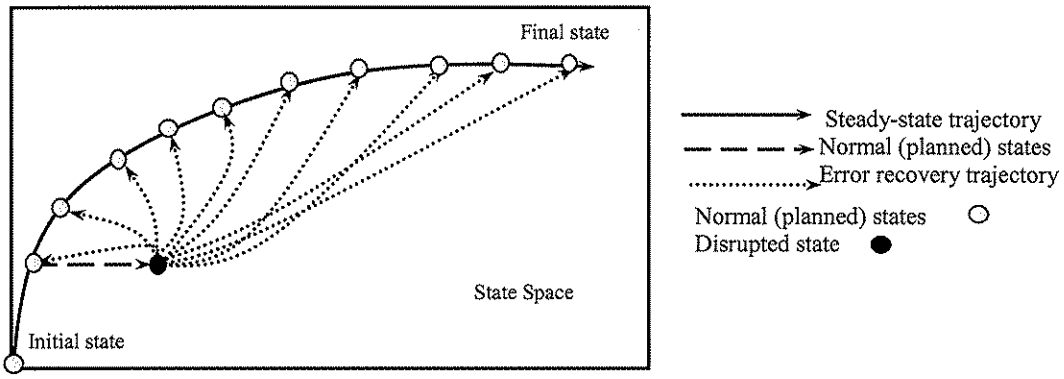


Figure 8. Error recovery trajectories from a disrupted state (Odrey and Mejia, 2005)

Figure 8 illustrates a view of the issue of "match-up" state in a manufacturing system and shows a desired "trajectory" constructed out of normal states, a disrupted state and the possible transient trajectories (dotted lines) to return to the original trajectory. The disrupted state is reached involuntarily. After being generated, the recovery subnet is incorporated into the workstation activities net (the Petri Net of the multi-agent system environment). In this research, we followed the designation of others (Zhou and DiCesare, 1993), and denoted the incorporation of a recovery subnet into the activities net as net augmentation. Zhou and DiCesare developed a formal description of these three possible trajectories in terms of Petri net constructs, namely input conditioning, backward error recovery, and forward error recovery. This prior work on error recovery strategies was intended to model the specifics of low level control typified by the equipment level of a hierarchical control system. The terms "original net" or "activities net" refer to the Petri Net representing the workstation activities (within a multi-agent environment) during the normal operation of the system. In the work presented here, the three recovery trajectories are applied to the workstation level within a hierarchical model. The enormous number of errors and the corresponding ways to recover that can occur at the physical workstation implies unlimited possibilities for constructing recovery subnets. The important issue is that any error and the corresponding recovery steps can be modeled with any of the three strategies mentioned above. Without loss of generality, this research limited the types of errors handled by the control agent to errors resulting from physical interactions between parts and resources (e.g. machines and material handling devices). The reason for this assumption was to facilitate the simulation of generic recovery subnets. Backward recovery suggests that a faulty state can become a normal state if an early stage in the original trajectory can be reached. The forward recovery trajectory consists of reaching a later state which is reachable from where the error occurred.

5.2.2 State equations and recovery subnets

The state space mathematical description was briefly described in section 3.2.In general that work consisted of a cell level timed, colored Petri nets (TCPN) state space representation for systems with parallel machining capability. This TCPN state representation extended Murata's generalized Petri net (GPN) state equations by modifying the token marking state equations to accommodate different type of tokens. In addition, a new set of state equations was developed to describe time-dependent evolution of a TCPN model. As a result, the system states of a cell level TCPN model were defined by two vectors:

System marking vector (M^p): This vector indicates the current token positions. A token type may consist of a job token, a machine token, or a combined job-machine token.

Remaining processing times vector (M^r) : This vector denotes how long until a specific job, machine, or job-machine token in an operation place can be released (i.e. an operation is completed)

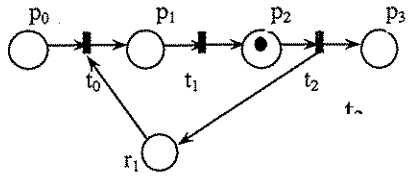
The TCPN workstation state equations provide a mathematical evaluation of the workstation performance at a higher level. After evaluation, a decomposed Timed Petri net (TPN) can then be constructed according to the evaluation results along with more detailed workstation operations. This was illustrated in section 3.3. As previously noted, subnets are viewed as alternative paths to the discolored TPN. The alternative path approach taken here is more flexible than a substitution approach in the sense that changes in subnets can be made without changing the configuration of the discolored TPN. The TPN workstation state equations provide a mathematical evaluation of the workstation performance at a lower level where primitive activities are coordinated to achieve desired task assignments.

In the event of disruptions, the original activity plan devised off-line by the workstation controller may require adjustments. The question that arises is how to re-construct the activity plan. A first alternative would be to build a completely new plan to execute the pending jobs. The other extreme would be waiting until the disturbance is fixed and continuing with the original plan. This would be partially constructing a new plan to a point where the original plan can be resumed. In terms of the Petri Nets this corresponds to find a marking (state) in the original plan reachable from the disrupted state and the question to be answered is the selection of a marking that should be reached. From there, a number of possibilities exist to return to the original plan. Details on performance optimization are given in a companion paper (Mejia & Odrey. 2004).

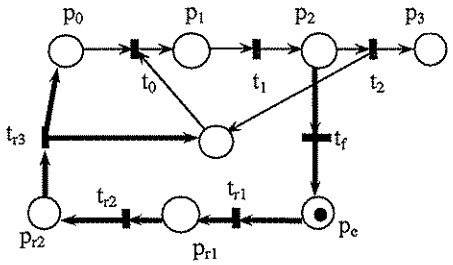
In terms of the Petri Nets, an error occurs when a transition fires outside a predetermined time frame. When a transition fires earlier or later (if the transition fires at all) than expected, an alarm is triggered and an error state is produced. After the error is acknowledged and diagnosed, a recovery plan is generated. This is accomplished by linking an error recovery subnet to the activity net. This linking produces an augmentation of the original net. At this stage the controller must devise a plan to reach the final marking M_f based on the status of the augmented net. Reaching the final marking M_f is accomplished by constructing a plan to reach some pre-defined intermediate marking M_{int} from previously determined List markings and then firing the pre-determined sequence of transitions from such an intermediate marking to the final marking. If a path to the intermediate marking can be found, then the original execution policy (sequence of transition firings) can be employed from the desired intermediate marking M_{int} to reach the final marking M_f. The issue of selecting the appropriate intermediate marking can be found in companion article (Mejia and Odrey, 2004). Our focus at this juncture is to demonstrate the construction of recovery subnets.

5.2.3 Construction of recovery subnets for error recovery

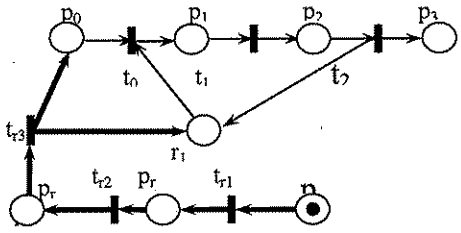
Perhaps the most complete descriptions of error recovery trajectories were developed by (Zhou and DiCesare, 1993). They proposed three possible trajectories. These consisted of input conditioning, forward error recovery, and backward error recovery. Input conditioning notes that an abnormal state can transform into a normal state after other actions are finished or some conditions are met. Forward error recovery attempts to reach a state reachable from the state where the error occurred. Backward error recovery suggests that a faulty state can become a normal state if an earlier stage in the trajectory can be reached. Obviously, not all trajectories are applicable in all cases due to logical or operational constraints. An example demonstrating backward error recovery is presented here but note that a similar approach can be applied to the other types of trajectories. Figure 9 illustrates the events during an error occurrence and the corresponding recovery in terms of Petri Net constructs. Figure (9a) represents the Petri Net during the normal operation. Places are defined in Figure10. The error is represented by the addition of a new transition t_f and a place p_e representing the error state in (9b). Firing t_f removes the residing token in p_2, resets the remaining process time corresponding to the place p_2, and puts a token in the new place p_e. The error recovery subnet and procedure are discussed in more detail in the following section.



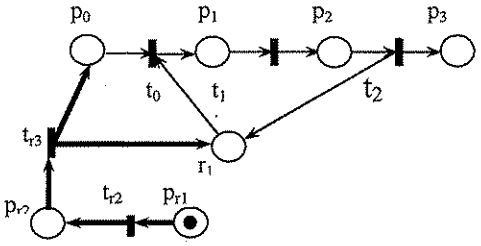
(a) Petri Net of during normal operation. A part is being processed by resource r_1



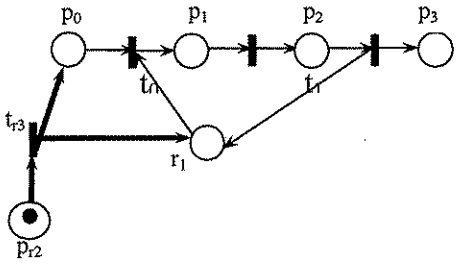
(b) Incorporation of an error/error recovery net. The error/error recovery net is shown with thicker lines.



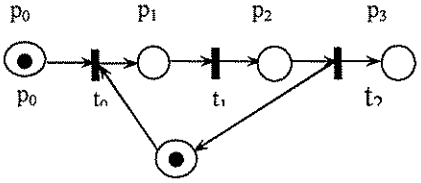
(c) Firing and deletion of t_f and the arc $I(t_f, p_e)$



(d) Firing and deletion of tr1, the place p_e and the corresponding arcs.



(e) Firing and deletion of t_{r2}, the place p_{r1}, and the corresponding



(f) Firing and deletion of t_{r3}, the place p_{r2}, and the corresponding arcs.

Remarks:
p_e represents an error state.
t_f is the transition that represents the initiation of the failure
p_0 to p_3 represent arbitrary operational places;
p_{r1} and p_{r2} represent recovery steps
t_{r1} to t_{r3} represent the start and end of the recovery step
t_0 to t_2 are changes of events in the original net

Figure 9. Construction and Deletion of Recovery Paths (from Odrey and Mejia, 2005).

5.2.4 Incorporating a recovery subnet into the original Petri net

The incorporation of the recovery subnet into the original net by the recovery agent is the first step. In the preceding example (see Figure 9), such a subnet trajectory consists of two places (p_{r1} and p_{r2}) and three transitions (t_{r1} to t_{r3}). Place p_{r1} represents the recovery action "find part" and place p_{r2} the recovery action "pick up part". Transitions t_{r1} to t_{r3} represent the change of states of these two recovery actions. With the recovery trajectory incorporated into the original net, the workstation control agent is required to execute the recovery actions. In (9.b), returning to the normal state requires the firing of transitions t_{r1},t_{r2} and t_{r3}. After firing t_{r3} the scheduled transition firings in the original net resume. The augmented net now contains an Operational Elementary Circuit (OEC) = {p_2, t_f, p_e, t_{r1}, p_{r1}, t_{r2}, p_{r2}, t_{r3}, p_0, t_0, p_1, t_1, p_2} that has only operational (timed) places.

One difficulty that arises is the potential that the operational elementary circuits constructed can result in infinite reachability graphs which make a search strategy difficult. Our approach to overcome this problem consisted of a sequential methodology which eliminates arcs and transitions from the combined original net and error/error recovery subnet. Every time that a transition on the recovery subnet fires, such a transition, its input places (except those places belonging to the original net) and the connecting arcs are eliminated from the augmented net. As noted in Figure 9, the elementary circuit which would be created during the generation of the recovery subnet will only be partially constructed. For example, in (9b), as soon as the transition t_f fires, the transition t_f and the arc I (p_2, t_f) are removed from the net. Subfigures (9c) to (9f) illustrate the sequence of firings and elimination of transitions, places and arcs from the net. The original net is restored when the last transition (t_{r3}) of the error recovery subnet has been fired. After firing t_{r3}, the part token returns to the original net and the resource token to the resource place. The workstation control agent records the elements (places, transitions and arcs) that belong to the original net and recovery subnets, respectively. A record is kept by the workstation controller such that for every time that a transition of the augmented net fires the controller searches for such a transition on the agenda. If the transition is found, it means that the transition belongs to a recovery subnet and all the transition input places and all its input and output arcs are deleted from the recovery agenda and from the augmented net (with the exception of arcs and places belonging only to the recovery subnet and not to the original net).

The next step relates to resuming the normal activities after an error is recovered. In terms of Petri Nets this implies finding a non-error state where the activities net and the recovery subnet are linked. The desired non- error state may not the same as the state prior to the occurrence of the error. For example, the state (marking) in subfigure (9f) is not the same as the state shown in subfigure (9a). The example described illustrates a possible trajectory (backward trajectory) which "started" (according to the arc directions) at p_2. Defining the non-error state is the task of the recovery agent and depends primarily on the characteristics of the error and its recovery. In the event of an input-conditioning strategy, the corresponding net originates and terminates at the same place (Zhou and DiCesare, 1993). Our investigations assume that any part token that goes through either a backward or a forward recovery trajectory is placed in a storage buffers after an error is fixed. Figure 10 illustrates an example for backward error recovery.

Description of places and transitions
p_0: part available
p_1: part in buffer 1
p_2: part being moved to resource 1
p_3: part being processed by resource 1
p_4: part processed
r1: resource 1 available
b1: buffer 1 available
t_{r1} and t_{r2}: Recovery transitions

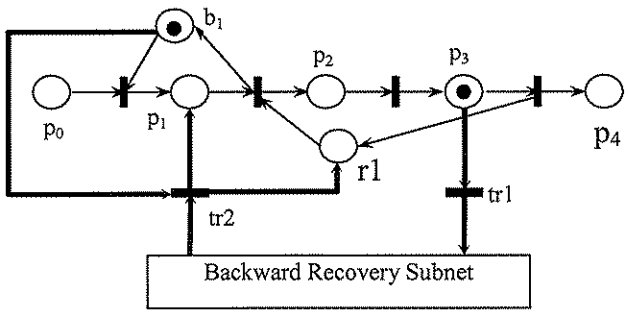
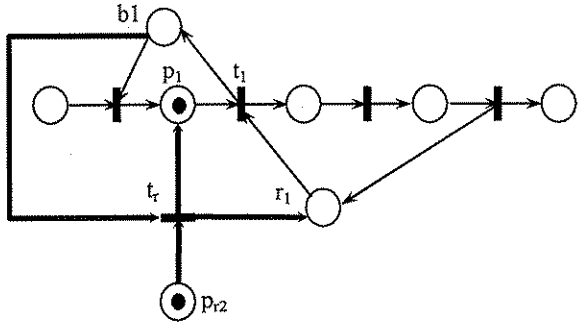


Figure 10: Example of backward recovery trajectory with buffer

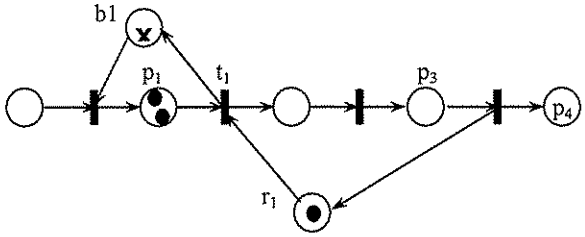
5.2.5 Handling resources and deadlocks

The work presented here assumes that, when an error occurs, all resources involved in the operation that failed and the part that was being process or manipulated become temporarily unavailable. Consider an example where two recovery actions are required to overcome an error. This could correspond to a situation of a robot dropping a part. To recover the part the part must first be found and then a command for the robot to "pick up part" must be given. Vision systems have been used for the first action of finding the part. It should be noted that during the execution of recovery actions both the resource and the part remain unavailable for other tasks. This differs from our previous work (Liu, 1993) which considered machine breakdowns in which only the machine that failed remains unavailable during the failure and repair period. The actual manipulation of a part during the failure states is considered in the logic of a workstation control agent. If the selected trajectory is an input conditioning subnet, the resources that intervened in the operation that failed remain unavailable until the operation is successfully completed. For backward and forward recovery the procedure is more complex in that all resources required to execute the operation that failed may need to be released at some point (to be determined by the recovery agent) in the recovery trajectory. Another issue is the possible occurrence of deadlocks in net augmentation. The policy adopted was to maneuver out of such deadlock states by temporarily allowing a buffer overflow. An example of maneuvering out of the deadlock situation using a Petri Net model is given in Figure 11. In the Petri net illustrated, the transition tr will be allowed to fire even if no tokens are available at place b1 (i.e, the buffer b1 is full). In that case, the ***place***

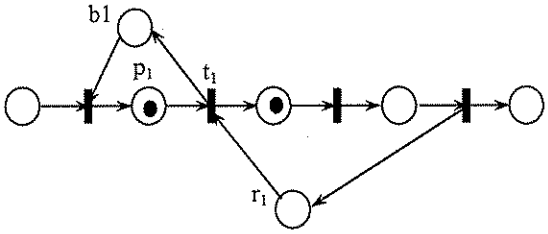
p1, representing the "parts in buffer" condition, would accept a token overflow (two tokens instead of one) only for the case of tokens coming from recovery subnets. The advantage of this policy is that clears the deadlock situation in an efficient way that addtionally can be automatically generated in computer code. It should be note that if this policy is not feasible in a real system due to buffer limitations, human intervention may be required.



(a) Deadlocked net before firing t_r



(b) Firing and deletion of t_r and the corresponding arcs and places. Overflow of tokens occurs at the buffer to avoid a deadlock. X represents a negative token



(c) Firing of t1 restores the original buffer capacity

Figure 11. Deadlock Avoidance by Allowing Temporary Buffer Overflow (Odrey and Mejia, 2005)

Another issue considered was the situation where firing t1 twice would put two tokens in place b1 and the original buffer capacity would be permanently doubled. In a Petri net this overflow condition was modeled with negative tokens. Negative tokens for Petri Nets have previouusly been proposed for automated reasoning (Murata and Yamaguchi, 1991).To compensate for an overflow situation our procedure was as follows: when a token coming from a recovery net arrives to a buffer, one token is substracted from the buffer place (in this case, the place b1 that represents the buffer availability) even though the buffer place has no available tokens. If the buffer place has no tokens available then a buffer place will contain a "negative" token representing the temporary buffer overflow. In the approach taken negative tokens indicated that a pre-condition of an action was not met but still the action was executed. The overflow is cleared when transitions, which are input to the buffer place, are fired as many times as ther are negative tokens that reside in the buffer place. The storage buffer remains unavailable for other incoming parts from the original net until both the overflow is corrected and one slot of the buffer becomes empty. In terms of the Petri net of Figure 10, the buffer will be available again only when there is at least one token in the "buffer" place b1.

5.3 A combined Neural Net-Petri net approach for diagnostics

In an attempt to investigate an "intelligent" manufacturing workstation controller an approach integrating Petri net models and neural network techniques for preliminary diagnosis was undertaken. Within the context of hierarchical control, the focus was on modeling the dynamics of a flexible automated workstation with the capability of error recovery. The work-station studied had multiple machines as well as robots and was capable of performing machining or assembly operations. To fully utilize the flexibility provided of the workstation, a dynamic modeling and control scheme was developed which incorporated processing flexibility and long-term learning capability. The main objectives were (i) to model the dynamics of the workstation and (ii) to provide diagnostics and error recovery capabilities in the event of anticipated and unanticipated

faults. A multi-layer structure was used to decompose complex activities into simpler activities that could be handled by a workstation controller. At the highest layer a TCPN represented generic activities of the workstation. Different color tokens served to model different types of machines, robots, parts and buffers that are involved in the system operation. This TCPN model is based on modules which model very broad workstation activities such as "move", "process" or "assemble". A processing sequence is built by linking some these modules following the process plan. Then the resources needed to execute these activities are linked. Figure 3 shows an example of the move and assemble modules. If changes are required, the designer only needs to re-assemble the activity modules.

Our goal was to provide responsive and adaptive re-actions to variation and disruption from a given process plan or assembly sequence. Specifically, three subproblems were in this research : (1) a workstation model was constructed which allowed a top-down synthesis and integration of various control functions. The proposed workstation model had several levels of abstraction which decomposes operation commands requested by a higher cell level into a sequence of coordinated processing steps. These processing steps were obtained through a hierarchical decomposition process where the corresponding resource allocations and operations synchronization problems are resolved. The motion control function is incorporated at the lowest level of the hierarchy which has adequate intelligence to deal with uncertainties in real-time, (2) a model-based monitoring scheme was developed which includes three functions : collecting necessary information for determining the current state of the actual system, checking the feasibility of performing the current set of scheduled operations, and detecting any faulty situation that might occur while performing these scheduled operations. A Petri net-based watch-dog approach was integrated with a neural network to perform these monitoring functions, and (3) an error recovery mechanism was proposed which determines feasible recovery actions, evaluated possible impacts of alternative recovery plans, and integrates a recovery plan into the workstation model (Ma, 2000; Ma & Odrey, 1996) . Our focus here is on the integration of Petri Net based models and neural network techniques for preliminary diagnostics.

Diagnostics determines the fault or faults responsible for a set of symptoms. A diagnosis may require a complete knowledge of the physical structure of the present devices and their functionality (deep knowledge) and a short series of pre-established actions (shallow knowledge) for pre-defined faults. The diagnostics activity, as structured by Ma (2000), can be divided into two main types: (i) Preliminary diagnostics and (ii) deep reasoning. The neural network architecture for preliminary diagnostics is shown in Figure 12. Preliminary diagnostics is the first subtask of the diagnostic subfunction and is used to facilitate the diagnostic process. The approach taken here contains three different neural networks as shown in Figure 12. Neural net 1, termed NN I, generates the expected system status by converting a Petri net representation into a neural network structure for real-time control. The second neural net NN2 implements a sensor fusion and/or logical sensors concept (Henderson & Shilorat, 1984) to provide NN3 with the actual system status such that a sensory-based control system can be realized. NN3 is a multilayer feedforward neural network for classifying data obtained from NN1 and NN2 into different categories for preliminary diagnostics. Preliminary diagnostics provided a scheme to reduce efforts for further diagnostics by classifying conditions for recovery into four categories: (i) shut down the system, (ii) continue operation, (iii) call operator or (iv) invoke proper operation. The purpose of the deep reasoning module was to isolate the failure(s) and report to the error recovery module. Ma (2000) investigated a neural network model for preliminary diagnostics using an input-output technique for shallow knowledge. A Petri Net embedded in a neural network was used to classify errors. These errors were linked to a rule-based expert system containing pre-defined preliminary corrective actions (Ma and Odrey, 1996). The neural network was trained and tested with examples drawn from combinations of PN states and sensory data. Deep reasoning was not considered in Ma's work and is a subject of on-going research.

A top-down Petri net decomposition approach was performed to construct a hierarchical PN model for the given work-station example. High level Petri nets such as TCPN and TPN are included to enhance the modeling capability and the hierarchical concept provided the necessary task decomposition. The first (highest) sublevel was a timed-colored Petri net (TCPN) which is a general PN with two additional parameters: 1) a time factor to represent the operation time for each operational place, and 2) color tokens to distinguish between parts. This is decomposed into the second sublevel which is a timed Petri net (TPN) where color tokens are not required because different parts (color tokens) are modeled separately. The third decomposition (sublevel) of the model further decomposes the operations at the assembly table into detailed processing steps such as "pick up", "transport", and "place". This final decomposition allows the Petri net to be more easily analyzed.

The approach taken in this research embedded a Petri net model in a neural network structure and was termed Petri Neural Nets (PNN). The purpose of a PNN is to facilitate the process of obtaining state evolution information (the expected system status) by taking advantage of the parallel computational structure provided by neural networks and utilizing the T -gate threshold logic concept proposed by (Ramamoorthy & Huang, 1989). The state evolution of a system modeled by Petri nets can be expressed using the following matrix equation:

$$M(K+1) = M(K) + U^T(K)A, \quad K=1,2,\ldots \tag{7}$$

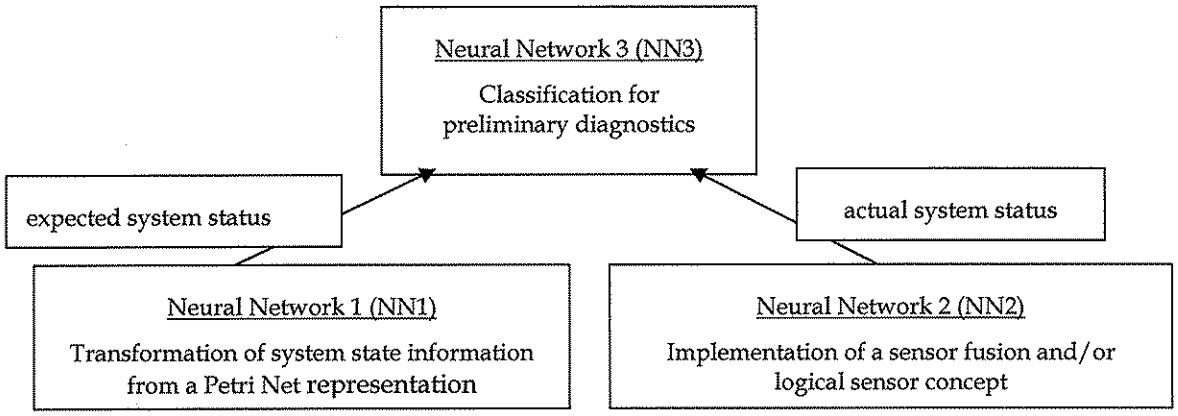


Figure 12: Neural Network architecture for preliminary diagnosis

M(K) is a (lxm) row vector representing the system marking at the Kth stage. U(K) is a (n x 1) column vector containing exactly one nonzero entry "I" in the position corresponding to the transition to be fired at the Kth firing. The matrix A is a (nxm) transition-to-place incidence matrix. A schematic of the NN1 architecture is indicated by Figure 13.

Based on the state equation, a three-layered PNN with an embedded T -gate threshold logic which simulated the state evolution of a general PN from M(K) to M(K+I) was developed as follows for the different layers: 1) an input vector I_k = [I_1, ...,I_m] (m = number of places) is set equal to M(K). The expected output vector O_i (i= 1,...,m) is M(K+I). The second layer of the PNN contains three vectors: (i) V_J (i=1,2 ,... , m) representing M(K), (ii) G_r (r= 1,...n) where n = number of transitions representing UT(K) which is determined by execution rules for Petri nets, and 3) H_h (h=l, ..., m) which represents UT(K)A. For a decision-free PN, the execution rules can be implemented using AND T -gate threshold logic. The T -gate threshold logic is a neural network with fixed weights and can be used to implement a rule-based expert system for time-critical applications as noted by (Ramamoorthy and Huang, 1989). The weights in the PNN are hard weights and are assigned according to specified rules. Details can be found for theses weights and the output function for each layer in (Ma & Odrey, 1996).

The purpose of preliminary diagnostics was to classify operation conditions occurring in the workstation into several categories, each one associated with a preliminary action. The input vector of NN3 is portioned into two sets of nodes. The

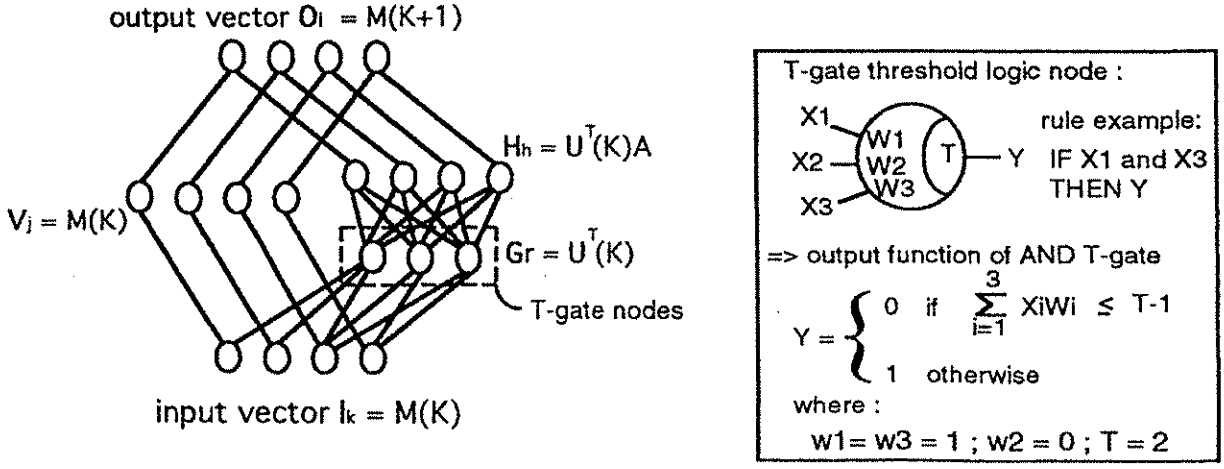


Figure 13: NN1 Neural Network architecture incorporating T-gate threshold logic gates (Ma & Odrey, 1996)

first set represents the expected system status and is obtained from the output of NNI (i.e. M(K.+I) of the corresponding sublevel-TPN model). The second set of nodes [S1, S2, ... Sn] represent categories of sensor information which are obtained from NN2. The output vector of NN3 represents the four preliminary actions: shutdown (O1), call operator (O2), continue operation (O3), and invoke further diagnostics (O4). The value of these output are either "0" representing not activated, or "1" representing activated. An outline of the system is given in Figure 13. Training and testing data are obtained using diagnostic rules based on common knowledge about the system. In general, the actual operation status of a system at any instant is the set of readings of all the sensor outputs. However, the actual system status information given by the sensor outputs is not sufficient for determining preliminary actions. Both the actual system status and the expected system status are required. The determination of a preliminary action for operations can thus be stated for the example of Figure 14 as follows:

IF "the expected system status" = [p1,p2.p3,p4,p5] AND "the actual system status" = [s1.s2.s3.s4]
THEN "preliminary action" = Oi (i = 1,2,3,4)

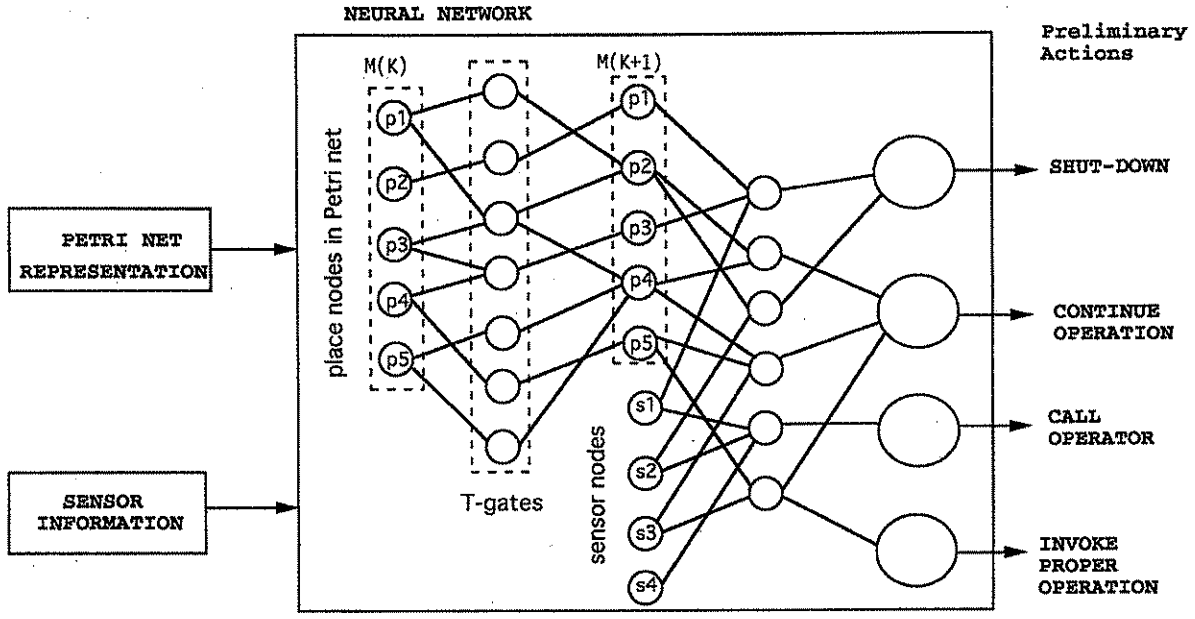


Figure 14. Generation of preliminary actions in a neural network incorporating T-gate threshold logic

Based on a sublevel TPN model, NN1 generates different outputs corresponding to possible expected system status M(K). Different fault scenarios were used as the basis for simulation of actual system status and for generating diagnostic rules. Details of the simulation and results can be found in (Ma and Odrey, 1996). In general a neural network for preliminary diagnostics was investigated. For NN3 (classification for preliminary diagnostics) different 3-layer perceptron networks with different hidden nodes were simulated and it was found that a 19-15-4 perceptron network gave the lowest percent classification. Note that this work did not construct the NN2 network and only simulated data was used to test the proposed neural network NN3. We plan to continue this approach which incorporates a hybrid neural - Petri net in future research.

5.3.1 Advanced diagnostics and error recovery

Preliminary diagnostics, as noted in the previous section, provides a scheme to reduce efforts for further diagnostics by classifying conditions to be diagnosed into four categories, each one associated with a preliminary action. The preliminary actions separate the diagnostic conditions which require knowledge about the physical structure of the devices and/or their functional descriptions (i.e., deep knowledge). from the conditions which need only a short series of inferences but fast responses (i.e. shallow knowledge). Shallow knowledge which usually appears in the form of direct input-output association can store patterns of predefined instructions from designers and/or experts was considered more desirable at the preliminary diagnostics stage in this research.

5.3.2 Further diagnostics

Further (advanced) diagnostics is initiated to consider two possible situations: either a preplanned error(s) has occurred or an unanticipated error(s) has occurred. Regardless of error type, a recovery plan is needed to construct a recovery trajectory to bring the system back to a normal condition (nominal trajectory). For preplanned errors, the corresponding error causes and/ or sources can be established in a failure reason data structure. With such a database structure, one can then obtain the failure reasons associated with a particular operation. In this research, an integrated approach which utilizes both knowledge-based systems and neural networks is proposed for unanticipated errors. Neural networks are used to provide additional information about unanticipated situations through learning. The same neural network used in the preplanned error is used to get as much information as possible about unanticipated errors. The research effort is directed toward using preplanned errors as training data and a multilayer, feedforward network as the initial test structure. A knowledge-based system then takes this information as inputs to automated processes. The modeling process is based on the feasibility of using Petri nets with negative tokens (Murata and Yamaguchi, 1990). Our current efforts focus on developing an automated reasoning technique which can draw conclusions from unknown errors in a workstation environment. To develop an automated reasoning scheme, a corresponding Petri net is established from information gathered by the neural net approach to model the reasoning. A schematic of the general framework for error recovery is given in Figure 15.

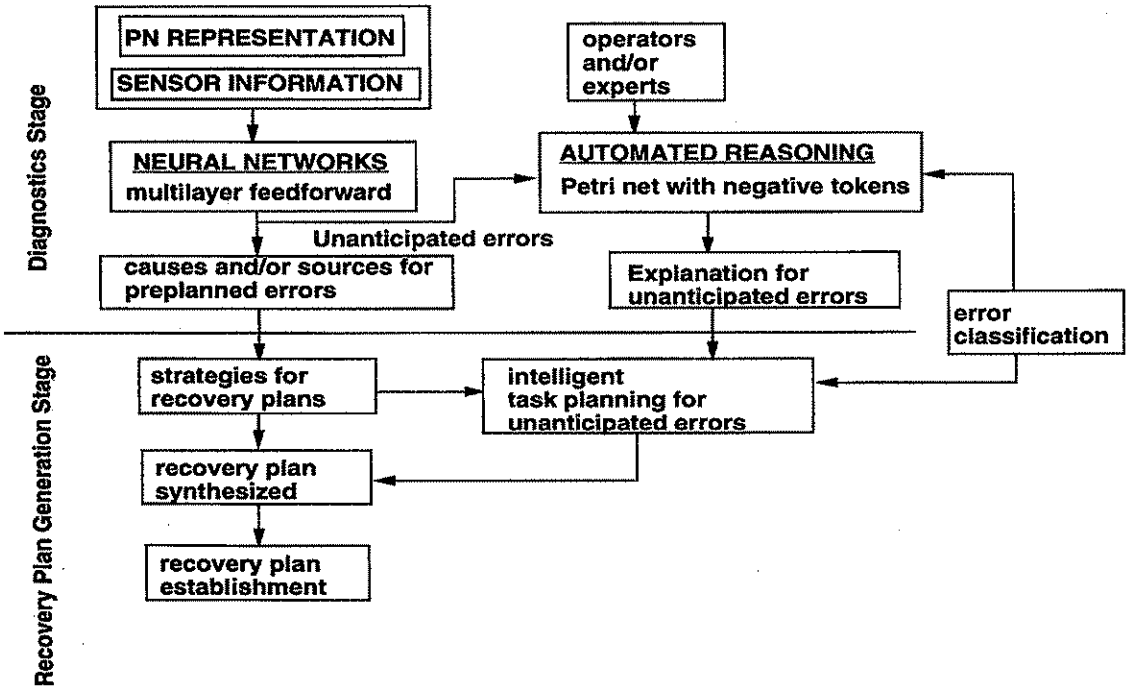


Figure 15. A general framework for error recovery in a Petri net based system

5.3.3 Error Recovery Strategies

After diagnostics, the workstation controller needs to generate a recovery plan to return the system back to a normal state and to continue the remaining tasks. The generation of recovery plans involves determining recovery strategies, constructing recovery activities, synthesizing a recovery sequence, and establishing a recovery plan. To determine recovery strategies, general and specific rules may be selected as constraints in the generation of recovery plans. In particular, preplanned errors and unanticipated errors usually have different sets of rules to be followed. In the case of preplanned errors, the construction of recovery activities can be easily done by recalling from computer memory. For unanticipated errors, however, an intelligent task planning system is required, and at least one feasible set of recovery activities needs to be constructed. In the approach taken recovery activities are synthesized with the planned activities to form a sequence of coordinated primitive activities. Finally, a complete recovery plan is established which includes not only the recovery actions but also other information or commands. In the research

done to-date the most important issues in the generation of recovery plans was to develop an intelligent task planning system and to synthesize Petri nets corresponding to the recovery activities and to the planned activities. The purpose of an intelligent task planning system is to select and sequence processing steps that will change the current state of the system into a desired system state.

A Petri net based processing step representation to establish error recovery trajectories through a neural network based learning mechanism was undertaken. The processing steps modeled by Petri nets were categorized into two classes, namely, an action-class and a condition-class. Processing steps such as "move", " process", and "assemble" that execute a task and usually have time associated with them are considered as an action-class. The condition-class processing steps represent the preconditions and/or post-condition of an action-class processing step. Examples of condition-class processing steps include "part in IB" and "part finished processing". Every action-class processing step is followed by condition-class processing steps. Similarly, a condition-class processing step can trigger one or more action-class processing steps. Based on the relationship between action-class and condition-class processing steps, two sets of problems are defined:

P1: Action-Condition Problem (ACP), i.e. given an action-class processing step, find a (pre) condition-class processing step
P2: Condition-Action Problem (CAP), i.e. given a (post) condition-class processing step, determine an optimal action-class processing step

The recovery plan generation problem then involves solving ACP and CAP iteratively which then generates a sequence of processing steps until a desired system state is reached. When the error recovery module is initiated by the monitoring and diagnostics module, the expected system state is compare with the actual system state to obtain the discrepancy (error) of the system. If the error state is at an action-class processing step, the ACP problem is solved (through the Action Neural Network) and the result is compared with the normal trajectory to see if any of the normal state can be reached. If not, the error recovery routine continues by feeding the results from the ACP problem into the CAP problem which is solved through the Condition Neural Network. The ACP and CAP problems are invoked iteratively until a state in the normal trajectory can be reached. Similarly, if the error state is at a condition-class processing step, the CAP problem is invoked first and the results are fed into the ACP problem, if necessary.

To solve ACP and CAP problems, it was necessary to consider the interactions between action-class processing steps and condition-class processing steps. In a workstation environment, many different processing steps can be constructed. It would be difficult to consider all the interactions among all the processing steps. The basic elements for constructing a processing step, however, are limited and thus manageable. We term these individual steps as primitive elements. Our approach consisted of action-class processing steps being composed of three different elements: the action element, the object element, and the location element. For example, in the "move part A to m1 using robot 1" processing step, the action element is "move", the object elements are "part A" and "robot 1", and the location element is "m1". Similarly, the condition-class processing steps have the object element, the location element, and the status element. For example, the processing step, "part A finished at m1", has "part A" as an object element, "m1" as the location element, and the status element is "finished". Various action-class and condition -class elements can be constructed. In an industrial setting such steps could be constructed from basic Method-Time-Measurement (MTM) data already available. Each processing step is represented in terms of different elements using binary vector representations. An action-class processing step, "move part A to machine 1 with robot 1", can then be represented as an action -class vector PSA

$$PSA = [1\,0\,0\,0\,0\,0\,0\,1\,0\,0\,0\,0\,1\,0\,0\,0\,0\,0\,1\,0]$$

where the "1" designation refers to the primitive element considered and "0" is interpreted as an element not considered from the action-class set. Similarly, a vector PSC can be defined to represent a condition-class processing step. An example of a condition-class processing step, "part A at machine 1", could be represented by a vector PSC as follows:

$$PSC = [1\,0\,0\,0\,0\,1\,0\,0\,0\,0\,0\,0\,0]$$

The elements within the vector are interpreted as active or inactive. This representation has the advantages of being able to represent many combinations of actions, objects, locations, and status. In addition, the vector-based representation allows one to apply neural network techniques that provide learning capability in the generation of recovery plans for unanticipated errors. In this research, in order to capture the relationship among processing steps and to generate error recovery plans, a Boltzmann machine neural network was investigated.

5.3.4 Boltzmann Machine Neural Network structure

The Boltzmann Machine is a particular class of neural networks that consists of a network of simple computing elements. The states of the neurons are binary, i.e. 0 and 1. The neurons in the network are connected by synapses with different (real) weights, which represent a local quantitative measure for the desirability that the two connected neurons are on. Similar to backpropagation neural networks, Boltzmann machines can be trained on test data to associate input and output values. In addition, one can use Boltzmann in optimization problems where the state of an individual neuron is iteratively adjusted to achieve minimal cost objective. The ability of doing both association and optimization makes Boltzmann machines very appealing in the application of workstation recovery plan generation. In this research, the Boltzmann machine is used at two different stages, namely a learning stage and an optimization stage. At the learning stage, the objective is to capture the relationships among various elements of the processing steps through weights adjustment. The relationships among various elements of the processing steps should be the same throughout the operations. Therefore, the learning stage is performed off-line. Once the relationships (weights) are established, the desired output is found at the second stage, on-line, through solving an optimization problem. In this research, in order to capture the relationship among process steps ant to generate error recovery plans, a Boltzman machine neural network was used. Details of this investigation are beyond the scope of this chapter and are currently being submitted for publication. Details can also be found in (Ma, 2000)

6. Conclusions

The work presented above essentially summarizes past and on-going work within the Industrial & Systems Engineering department at Lehigh University on "smart" systems. The research undertaken indicates a variable architecture and approach for such systems. Extensions to this work will incorporate stochastic implications, communications and negotiation strategies between agents, and further work on control nets and strategies. Hybrid nets such as the Petri -Neural Net are of particular interest. The techniques integrated into this work in the future will be directed toward development of robust, reconfigurable, adaptable large scale systems. Applications are currently in production and logistic systems. Other applications are being pursued.

Acknowledgments

The author would like to thank the students who have contributed to this work over the years. In particular, the work in this chapter is based on the work of Drs. Cheng-Sheng Liu, Christina Ma, and Gonzalo Mejia. The author would also like to thank Ms Julie Drzymalski for helping in proof reading this manuscript and providing helpful suggestions in its formulation. Her graduate dissertation is extending to supply chains and enterprise level problems.

References

Albus, J. (1997). The NIST Real-time Control System (RCS): an approach to intelligent systems research. *Journal of Expert Theory in Artificial Intelligence.* Vol. 9, No 2-3, pp. 157-174.

Barad, M. & Sipper, D. (1988). Flexibility in Manufacturing Systems: Definition and Petri Net Modeling, *International Journal. of Prod. Research,* Vol. 26, No.2, pp. 237-248.

Brennan, R. (2000). Performance Comparison and Analysis of Reactive and Planning-based Control Architectures for Manufacturing. *Robotics and Computer Integrated Manufacturing.* Vol. 16 , No. 2-3, pp.191-200.

Duffie, N. Chitturi, R. Mou, J. (1988). Fault Tolerant Heterarchical Control of Heterogeneous Manufacturing System Entities. *Journal of Manufacturing Systems.* Vol. 7, No. 4. pp. 315-327.

Fielding, P. J., DiCesare, F., Goldbogen, Geof., Desrochers, A.(1987) Intelligent automated error recovery in manufacturing workstations. *Proceedings of IEEE International Symposium on Intelligent Control* 18, pp. 280-285, Philadelphia, PA, 1987, IEEE, Piscataway, NJ.

Gou, L. Luh, P. Kyoya, Y. (1998). Holonic manufacturing scheduling: architecture, cooperation mechanism, and implementation. *Computers in Industry.* Vol 37, No. 3, pp. 231-231.

Henderson, T., Shilcrat,E. (1984), Logical Sensor Systems, *Journal of Robotic Systems,* Vol.1, No.2, pp. 169-193..

Hillion, H. Proth, J.M. Performance Evaluation of Job-Shop Systems Using Event Graphs (1989). *IEEE Transactions on Automatic Control,* Vol 34, No 1, pp. 3-9.

Jennings, N.R. (2000) On agent-based software engineering, *Artificial Intelligence,* Vol. 117, No. 2, pp. 277-296.

Liu C. S. (1992). Planning and Control of Flexible Manufacturing Cells with Alternative Routing Strategies. *Ph.D. Dissertation.* Department of Industrial Engineering, Lehigh University.

Liu, C, Ma, Y. Odrey, N. (1997) Hierarchical Petri Net Modeling for System Dynamics and Control of Manufacturing Systems. *Proceedings of the FAIM Conference,* pp.169-182, Middlesbrough, UK, June 1997, Begell House, NY.

Ma, Yi-Hui (2000) Flexible Manufacturing workstation with Error Recovery Capability, *Ph.D.Dissertation*, Dept. Of Industrial Engineering, Lehigh University

Ma, Yi-Hui, Odrey, Nicholas G.. (1996), On the application of Neural Networks to a Petri net -based intelligent workstation controller for manufacturing, *Proceedings of theArtificial Neural Networks in Engineering (ANNIE '96) conference*, pp. 829-836, Vol. 6, St. Louis, MO, November, 1996 ASME Press, NY.

Maturana, F. Shen, W. Norrie, D. (1999). MetaMorph: An adaptive agent-based architecture for intelligent manufacturing. *International Journal of Production Research*, Vol. 37, No.10, pp. 2159-2173.

Mejia & Odrey(2005), An approach using Petri Nets and improved heuristic search for manufacturing systems scheduling, *Journal of Manufacturing Systems*, Vol. 2, No. 2, pp. 79-92.

Mejia, G. , Odrey, N. (2004) Real Time Control and Error Recovery of Flexible Manufacturing Workstations: An Approach Based on Petri Nets, *Proceedings of the 14th International Conference on Flexible Automation and Intelligent Manufacturing*, pp. 824-831, Toronto, CN, June, 2004, Begell House, NY.

Meystel & Albus, (2002), *Intelligent Systems: Architecture, Design , and Control*, John Wiley &Sons, Inc. , New York,

Meystel, A and Messina, E. (2000) The Challenge of Intelligent Systems, *Proc. Of 15th Int'l Sym. On Intelligent Control, pp.* 211-216, Rio Patras, Greece, July, 2000, IEEE, Piscataway, NJ.

Murata, T. (1989) Petri nets: properties, analysis, and applications. *Proc. of IEEE.* Vol.7, No. 4, pp.541-580

Odrey & Mejia (2003), A reconfigurable multi-agent system architecture for error recovery in production systems. *Robotics & Computer Integrated Manufacturing*, Vol. 19 No. 1-2, pp. 35-43.

Odrey & Mejia(2005), An augmented Petri Net approach for error recovery in manufacturing systems control, *Robotics & Computer-Integrated Manufacturing*, Vol. 21, pp. 346-354.

Odrey, N. Ma Y-H. (2001). A Multilevel, Multi-Layer Petri Net Based Approach for manufacturing Systems Control. Proceedings of the 11th International FAIM Conference. pp. 218-228, Dublin, Ireland. July 2001., Begell House, NY.

Odrey, N. Ma, Y. Intelligent Workstation Control: An Approach to Error Recovery in Manufacturing Operations. *Proceedings of the 5th International FAIM Conference*, pp. 124-141, Stuttgart, Germany, 1995, Begell House, NY.

Odrey, N.G., Mejia. N,(2005), An augmented Petri net approach fro error recovery in manufacturing systems control. *Robotics and Computer Integrated Manufacturing*, Vol. 21, pp. 346-35.

Okino, N. (1993) A Prototype of Bionic Manufacturing Systems in Flexible Manufacturing Systems, Past, Present, Future. Publisher, J Peklenik,. Slovenia.

Sousa, P. Ramos (1999), C. A Distributed Architecture And Negotiation Protocol For Scheduling In Manufacturing Systems *Computers in Industry*. Vol. 38, No. 2, 1999, pp. 103-113.

Sun, J. Xue, D. Norrie, D (1999). An Intelligent Production System Scheduling Mechanism Considering Design and Manufacturing Constraints. *Proceedings of the Third International Conference on Industrial Automation*, pp. 2411-2414. Montreal. June, 1999

Teng, T.E. & Black, J.T., (1990), Cellular Manufacturing System Modeling: the Petri Net Approach, *Journal of Manufacturing Systems.*, Vol.9 No.1, pp. 45-54.

Tharumarajah, A. Wells, A. J. Nemes, L. (1996) Comparison of the bionic, fractal and holonic manufacturing system concepts. *International Journal of Computer Integrated Manufacturing*. Vol. 9, No.3, pp. 217-226.

Valckernaers, P. Bonneville, F. Van Brussel, H. Bongaerts, L. Wyns, J. (1994). Results of the Holonic Control System Benchmark at KULeuven. *Proceedings of Rensselaer's 4th International Conference on Computer Integrated Manufacturing and Automation Technology (CIMAT)*, pp. 128-133, Troy, NY 1994, IEEE, Piscataway, NJ.

Van Brussel, H. Wyns, J. Valckernaers, H. Bongaerts, L. Peeters, P.(1998) Reference Architecture For Holonic Manufacturing Systems: PROSA. *Computers in Industry* Vol 37, pp. 255-274.

Van Brussel, H. Bongaerts, L. Wyns, J. Valckernaers, P. Van Ginderachter, T.(1999). A conceptual framework for Holonic Manufacturing: Identification of manufacturing holons. *Journal of Manufacturing Systems*, Vol. 18, No. 1, pp. 35-52.

Venkatesh, K.; Zhou, M.(1998). Object-oriented design of FMS control software based on object modeling technique diagrams and Petri nets. *Journal of Manufacturing Systems*. Vol. 17, No. 2, pp.118-136.

Wang, L. Balasubramanian, S. Norrie, D. Brennan, R. (1998). Agent-based Control System for Next Generation Manufacturing. *Proceedings of the 1998 IEEE ISIC/CIRA/ISAS Joint Conference*. Gaithersburg, MD., 1998, IEEE, Piscataway, NJ.

Warnecke, H.(1993), *The fractal factory*. Springer-Verlag, NY.

Zhou, M. DiCesare, F. (1993) *Petri Net Synthesis for Discrete Event Control of Manufacturing Systems*. Kluwer Academic Publishers. USA.