14

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,

Source: Petri Net,Theory and Applications, Book edited by: Vedran Kordic, ISBN 978-3-902613-12-7, pp. 534, February 2008, I-Tech Education and Publishing, Vienna, Austria

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

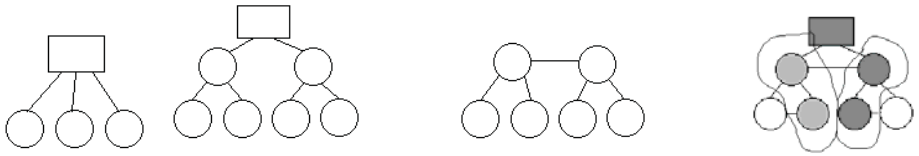
The focus in this chapter is on physical error occurrences and is directed towards supporting effective procedures for error recovery in an attempt to arrive at a reconfigurable, adaptive, and "intelligent "manufacturing system. As such, a hybridization of Petri Nets and intelligent agents seem to be 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

Fig. 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.

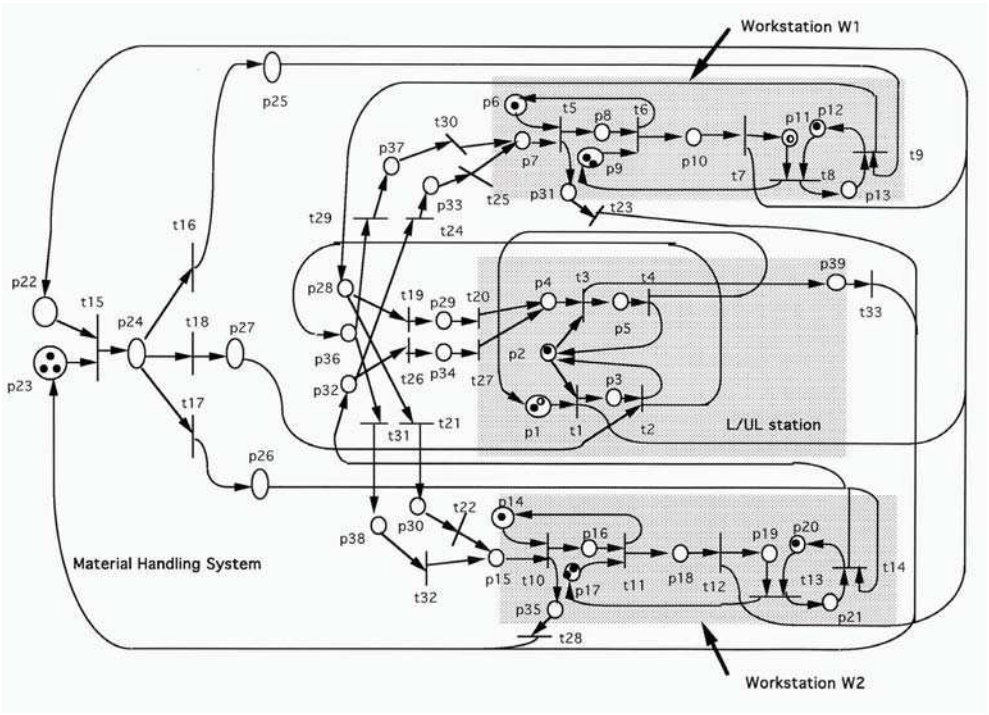


Fig. 2. Time Colored Petri Net for Two Workstations a load/Unload station, and a Material Handling System (Liu, et al. 1997; Liu, 1992)

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.

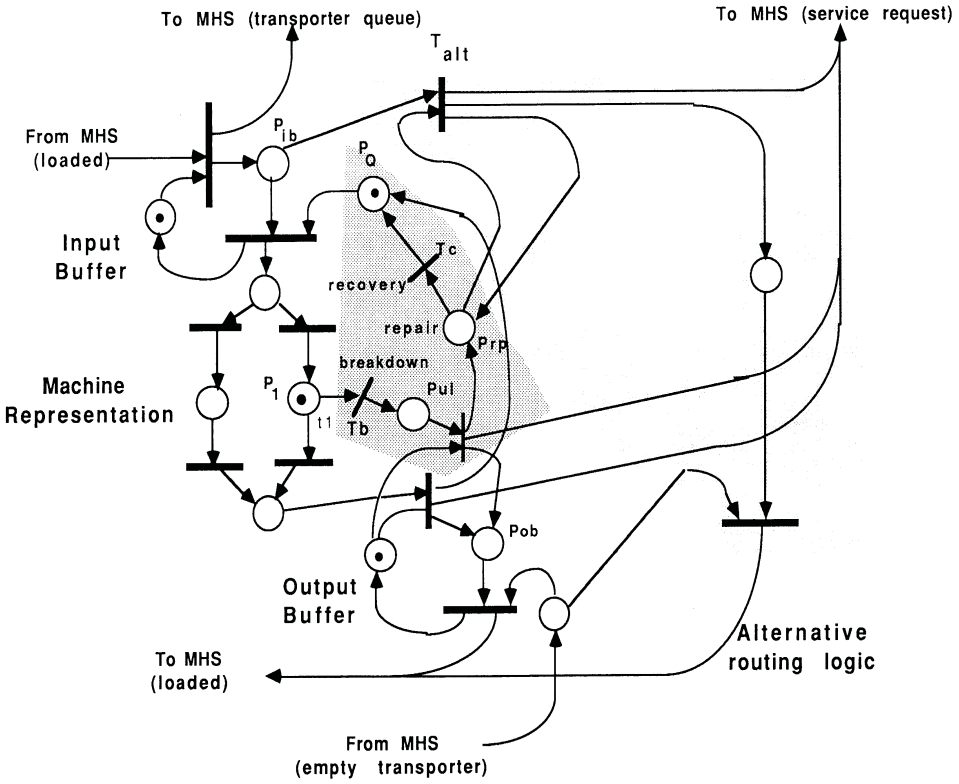


Fig. 3. A Workstation Petri Net Representation with an Alternative Routing Logic (Liu, 1993)

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:

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

where $\boldsymbol{M}(k)$ is the marking of the Petri Nets in time k, $\boldsymbol{L}$ is the incidence matrix and $\boldsymbol{u}(k)$ 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 ($\boldsymbol{M}_p(k)$) is the conventional marking vector (Murata, 1989) that accounts for the number of tokens in each place; the second one ($\boldsymbol{M}_r(k)$) 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):

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

$$\boldsymbol{X}(k)=\begin{vmatrix} \boldsymbol{M}_p(k) \\ \boldsymbol{M}_r(k) \end{vmatrix} \tag{3}$$

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:

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

[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).

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

[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.

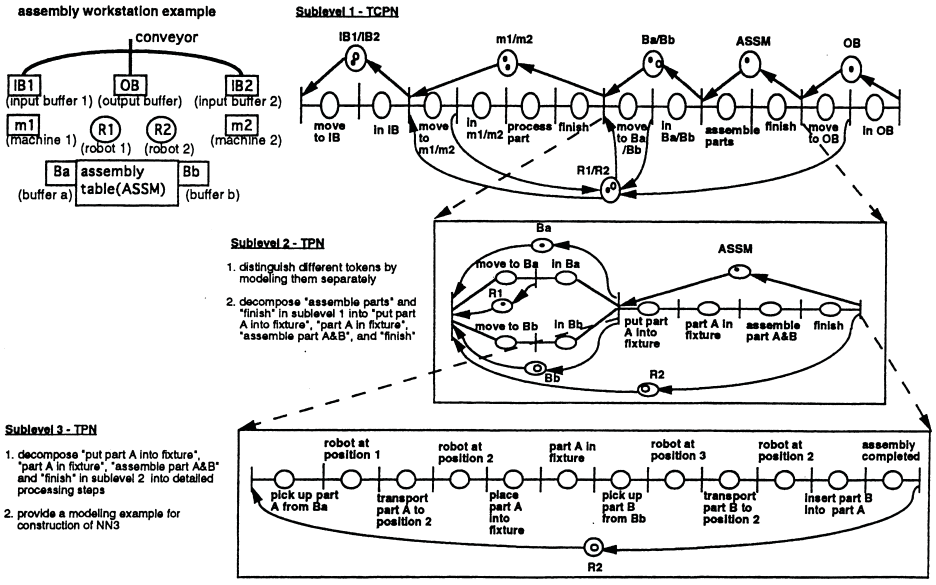


Fig. 4. An example of decomposition of a multi-layer Petri net model for an assembly station (Ma & Odrey, 1996)

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 (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 in-corporation 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 model-ing 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.

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.

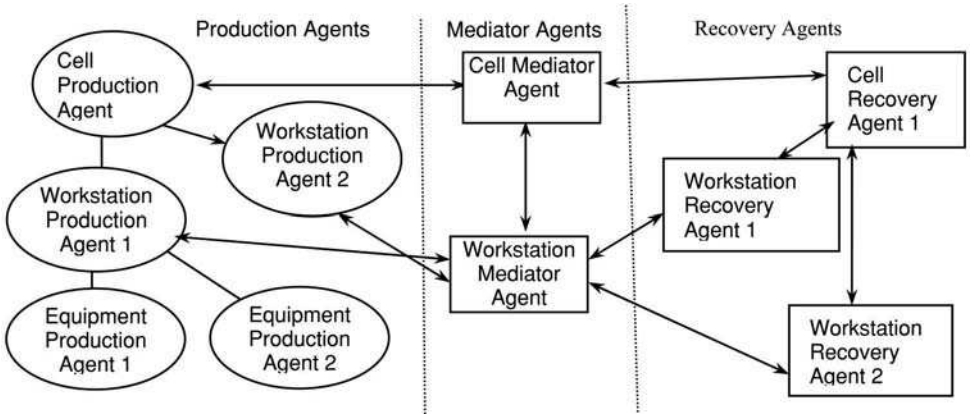


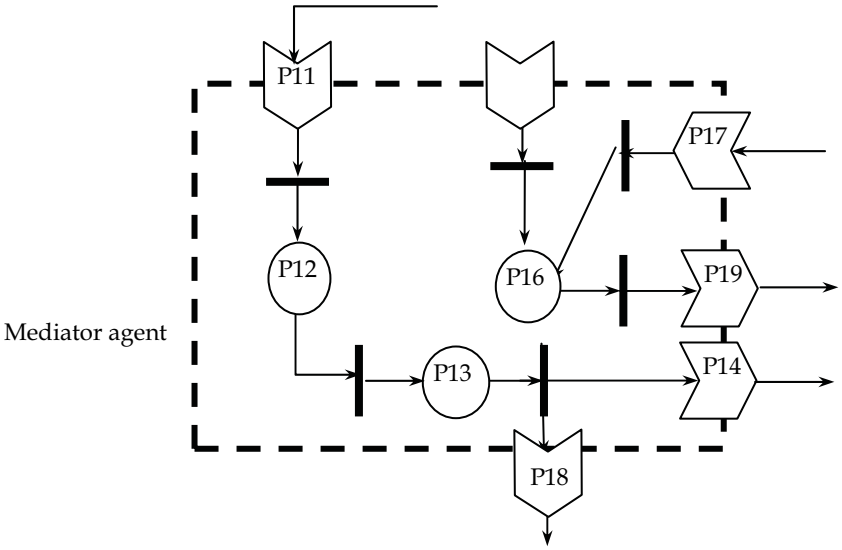
Fig. 5. Error recovery agents within an intelligent agent hierarchical architecture (Odrey& Mejia, 2003)

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).



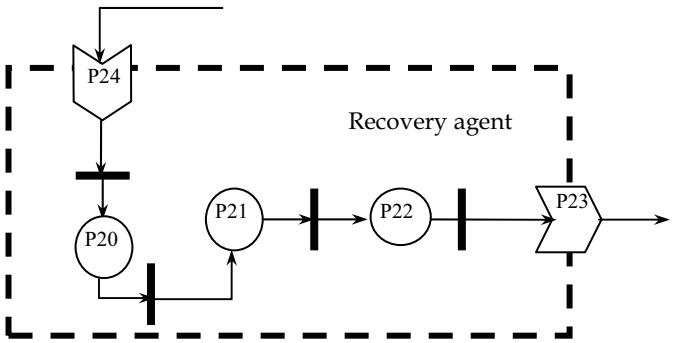
P11: Receiving recovery request message
P12: classifying errors
P13: Matching classified error with recovery agents
P14: Sending request messages
P15: Receiving responses from recover agents
P16: Evaluating responses
P17: Receiving responses
P18: Sending messages to mediators
P19: Communicating recovery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

Fig. 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).



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

Fig. 7. Workstation Recovery Agent Structure (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.

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 re-source 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