Error Recovery for Real-Time Manufacturing Control Via Augmented Petri Nets

Nicholas G. Odrey
Lehigh University

Gonzalo Mejía
Universidad de Los Andes

Report No. 04T-012

Error Recovery for Real-Time Manufacturing Control Via Augmented Petri Nets

*Nicholas Odrey and **Gonzalo Mejía

* Department of Industrial Engineering
Lehigh University
Bethlehem, PA, 18015, USA

**Department of Industrial Engineering
Universidad de Los Andes
Bogotá, Colombia

ABSTRACT

The construction of error recovery Petri subnets and similar representations have received considerable attention in the literature. Previous work has presented a multi-agent system representing various levels of control in a reconfigurable architecture. Agents pertaining to production, mediation, and error recovery within such an architecture were considered. Our focus here is on the workstation level of a hierarchy where the workstation has the capability for recovery from physical errors. The approach is based on integrating Petri subnet models within a general Petri net model for a manufacturing system environment. 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 is based on a Timed Colored Petri Net 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. Petri net augmentations pertaining to various issues are discussed in detail throughout the paper. Issues include the implication of generated error recovery trajectories in the production activities, linking of production activity net and the error recovery subnet, potential deadlocks, the role of resources, and part handling

1. INTRODUCTION

The characteristics of physical error occurrence impose difficult challenges to workstation control. The control must first handle simultaneously production and recovery activities, and second, errors that appear unexpectedly must be treated in real-time to avoid a dramatic decrease in the performance of the system. Previous work pertained to addressing the issue of monitoring, diagnostics, and error recovery within the context of a hierarchical multi-agent system [1]. The system consists of production, mediator, and error recovery agents. Production agents contain both planner and control agents. Here 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. For complex systems the diagnostic task may be performed by a mediator agent. When an error is detected, the control agent diagnoses the error and requests the action of a recovery 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 control agent logic. The construction of error recovery Petri subnets and similar representations is a topic which received considerable attention in the literature during the late eighties and early nineties. Petri Nets have been used previously to model recovery activities for machine breakdowns and alternative routings [2]. Extensions to such model representations to handle a more complicated logic including requests for recovery actions and temporary storage in buffers has also been accomplished [3].

*Corresponding author: Tel.: (610) 758-4036; Fax: (610) 758-4886; E-mail: ngo0@lehigh.edu

More recently a controlled automata logic to represent normal and error states in a manufacturing system has been developed [4]. 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 [5], 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.

Perhaps the most complete description of error recovery trajectories that has been developed [5] proposes three possible error recovery trajectories, namely, input conditioning, backward error recovery and forward error recovery. The concept of input conditioning is that an abnormal state can become a normal state after other actions are finished or some conditions are met [5]. 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 in the original trajectory after satisfying some operational constraints.

The input conditioning example shows a trajectory that "returns" to the state where the error occurred The backward recovery trajectory reaches a state visited prior to the state where the error occurred. Hence, the state where the error occurred is reachable from the state that represents the recovery. Finally, the forward recovery trajectory reaches a state that is reachable from the state where the error occurred. Obviously, not all trajectories are applicable in all cases due to operational or logical constraints. For instance, if the operation "process part" fails, the state "part processed" cannot be reached unless the operation "process part" is re-attempted and satisfactorily completed. Backward recovery trajectories can only be applied to reversible processes. A forward trajectory is the most desirable but at the same time, it is the most difficult to implement with automated reasoning systems [6]. Examples of automated reasoning systems for error recovery procedures, such as expert systems and neural networks have been proposed over the years include [7], [8] and [9].

2. Background

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 in this research is that any error and the corresponding recovery steps can be modeled with any of the three strategies mentioned above, namely input conditioning, and backward or forward error recovery. Without loss of generality, this research limits 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 is to facilitate the simulation of generic recovery subnets. An example of errors handled by the control agent is the incorrect positioning of a part in a fixture. This error represents the failed physical interaction part-fixture (the fixture is the resource).

Although not explicitly modeled in this research, the following examples are typical errors corresponding to interactions part-resource.

Parts missing in input buffer	Incorrect inspection of parts
Part missing in machine	Part positioning in machine or inspection table
Incorrect or defective parts in buffer	Part jammed in machine
Gripper slippage	Part requiring additional processing
Incorrect fixturing of parts	

Prior work recovery strategies [5] were intended to model the specifics of low level control typified by the equipment level of a hierarchical control system. In the research presented here, the three recovery trajectories can be applied to the workstation level within a hierarchical model as developed at NIST [10]. The application of these trajectories implies that:

(i) the level of task decomposition is the same for the recovery and production commands, and

(ii) a careful handling of allocation and release of resources is performed in order to guarantee maintainability of the net properties.

Since in this research the execution of both production and recovery activities is assigned to a workstation control agent, the control logic of such an agent must be able to handle both activities. For this reason the degree of detail in the command structure and the time horizons of both recovery and production must be compatible. For example, if the workstation production activities include the action "unload part", a possible recovery action at this level would be "re-attempt part unloading". Likewise the sequence of tasks for the control agent may be "process part A", "unload part A", "re-attempt part A unloading", and "process part B". Here the advantages of Petri Net modeling can be fully appreciated.

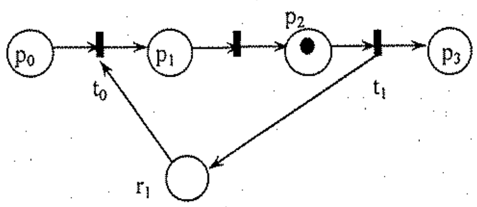
3. CONSTRUCTION OF RECOVERY SUBNETS

An example of backward error recovery is presented here but note that a similar approach can be applied to the other types of trajectories. For the construction of recovery subnets, a number of important issues must be considered. Figure 1 illustrates the events during an error occurrence and the corresponding recovery in terms of Petri Net constructs. Subfigure (a) represents the Petri Net during the normal operation. In (b), an error occurs in an operation "move part" represented by the operational place p_2. The error is represented by the addition of a new transition t_f and a place p_e. A similar approach was previously employed [3] to handle machine breakdowns. The transition t_f represents the start of the event "error occurs" and p_e represents the error state. 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 next step pertains to the incorporation of the recovery subnet: In the example, such a trajectory consists of two places (p_{r1} and p_{r2}) and three transitions (t_{r1} to t_{r3}). Here. p_{r1} represents the recovery action "find part" and p_{r2} the recovery action "pick up part". The transitions t_{r1} to t_{r3} represent the change of states of these two recovery actions. Having the recovery trajectory incorporated into the original net by the recovery agent, the workstation control agent is required to execute the recovery actions. In (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. Notice that the augmented net now contains an Operational Elementary Circuit (OEC) = $\{p_2, t_f, p_e, t_{r1}, p_{r1}, t_{r2}, p_{r2}, t_{r3}, p_0, t_1, p_2\}$.This is an elementary circuit that has only operational places

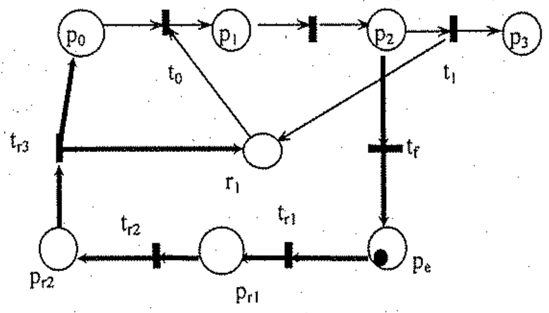
OEC nets describing the sets of sequences of activities results in infinite reachability graphs since tokens can infinitely loop around the OECs creating new states. The strategy adopted in this research to overcome this drawback is that every time that a transition on the recovery subnet fires, the fired transition, its input places (except those places belonging to the original net) and the connecting arcs are deleted from the augmented net. Thus, the elementary circuit which would be created during the generation of the recovery subnet will only be partially constructed. For example, in (b), 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 (c) to (f) illustrate the sequence of firings and deletion of transitions, places and arcs from the net. The original net is restored when the last transition (t_{r3}) of the error/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 keeps a record of which elements (places, transitions and arcs) belong to the original net and which elements correspond to the recovery subnets. This record allows the incorporation, deletion, and update of the recovery nets. In a typical scenario, a number of normal and error recovery activities are likely to occur simultaneously. For the purpose of tracking the recovery nets, every time a recovery agent creates a new recovery subnet, such a subnet is stored in a "recovery agenda". Every time that a transition of the augmented net fires the workstation control agent 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).

3.1 LINKING THE ACTIVITY AND RECOVERY NETS

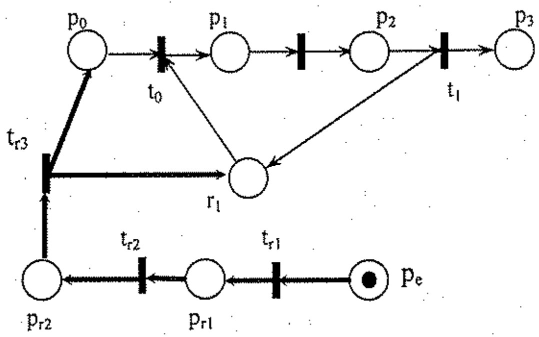
The next step relates to resuming the normal activities after an error is recovered. In terms of Petri Nets this implies finding a non-error state where the activities net and the recovery subnet are linked. The desired non- error state may not the same as the state prior to the occurrence of the error. For example, the state (marking) in figure 1(f) is not the same as the state shown in figure 1(a). The example described in figure 1 illustrates a possible trajectory (backward trajectory) which "started" (according to the arc directions) in 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. Except for the input conditioning case, this research adapts a previous model [3] in which a job is transferred from the broken-down machine to a storage buffer. In the event of an input-conditioning strategy, the corresponding net originates and terminates at the same place [5]. This research assumes that a part token that goes through either a backward or a



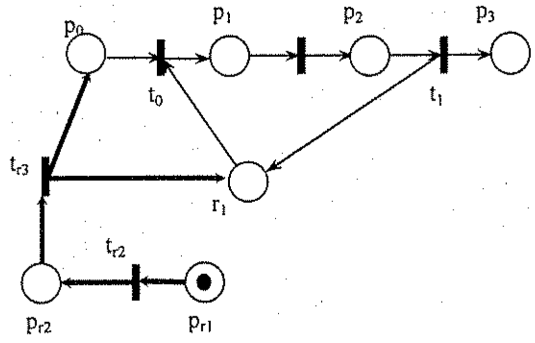
(a) Petri Net of during normal operation. A part is being processed by a resource r_1



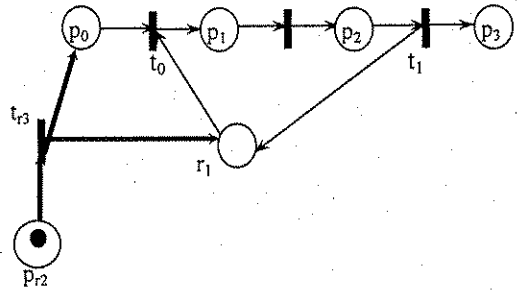
(b) Incorporation of an error/error recovery net. The error/error recovery net is shown with thicker lines.



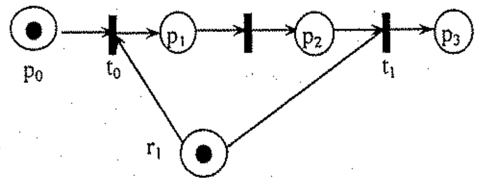
(c) Firing and deletion of t_f and the arc I (t_f)



(d) Firing and deletion of t_{r1}, the place p_e, and the corresponding arcs.



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



(f) Firing and deletion of t_{r3}, the place p_{r2}, and the corresponding arcs.
The original net has been restored

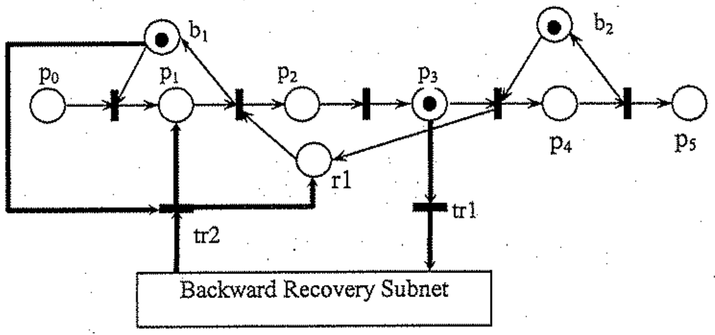
Remarks:
p_e represents an error state.
p_0 to p_3 represent arbitrary operational places; t_0 to t_2 are changes of events in the original net
p_{r1} and p_{r2} represent recovery steps
t_f is the transition that represents the initiation of the failure.
t_{r1} to t_{r3} represent the start and end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

Figure 1: Construction and Deletion of Recovery Paths

forward recovery trajectory is placed in a storage buffers after an error is fixed. Figure 2 illustrates the application of backward recovery trajectories in this research.

3.2 Handling of Resources in Recovery Trajectories

An important issue is the handling of resource tokens. This research assumes that, when an error occurs, all resources involved in the operation that failed as well as the part that was being manipulated become temporarily unavailable. For example, assume that two recovery actions, "find part" and "pick up part", are required to overcome the error "robot dropped part" occurred in the course of the pre-planned activity "move part". During the execution of such recovery actions both the robot and the part remain unavailable for other tasks. This is a major difference compared to approaches which consider machine breakdowns in which only the machine that failed remain unavailable during the failure and repair period. At the workstation level, the actual manipulation of the part during the failure states is considered in the logic of the 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. The cases of backward and forward recovery are more complex: All resources required to execute the operation that failed need to be released at some point (to be determined by the recovery agent) in the recovery trajectory.



p_0: part available	p_5: part processed
p_1: part in buffer 1	r1: resource 1 available
p_2: part being moved to resource 1	b1: buffer 1 available
p_3: part being processed by resource 1	b2: buffer 2 available
p_4: part in buffer 2	tr: Recovery transition

Figure 2: Example of Backward Recovery Trajectory with Buffers

3.3 Issues on Deadlocks With Augmented Nets

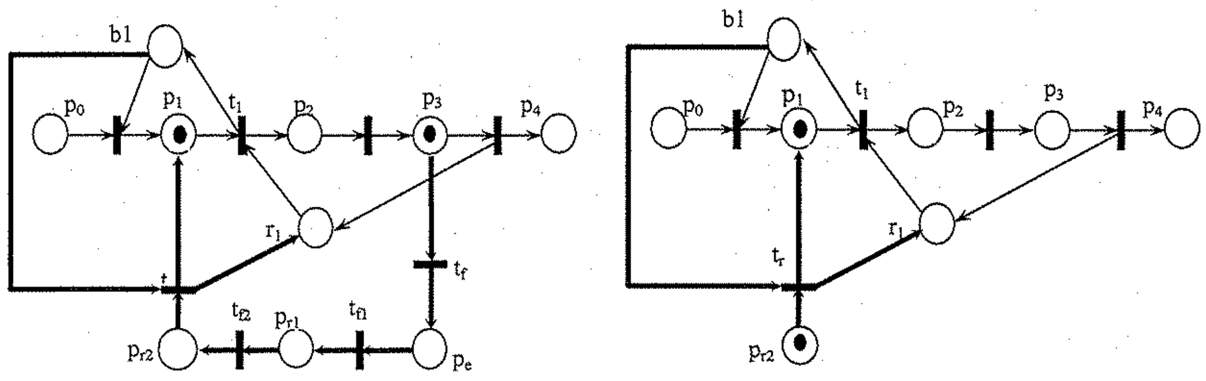
In the case of net augmentation, the following situation might occur: Consider the case of a part transported by a robot and that the operation "part moved by robot" fails because the robot dropped the part. Suppose that the error recovery agent finds a trajectory that returns the part to the previous buffer but the buffer is full. Simultaneously, parts located at the buffer are waiting for the robot to become available for transportation. This situation is depicted in figure 3(a) which shows the incorporation of a backward recovery net into the activities net. The activities of the recovery net are represented by the places p_{r1} (pick up part) and p_{r2} (drop part in buffer). As in previous figures, the thicker arcs highlight the recovery net. After the execution of the activity "pick-up part", the transitions t_f, t_{f1} and t_{f2} and the places p_e and p_{r1} are eliminated from the net. The resulting net is shown in figure 3(b). Notice that neither transitions t_1 nor t_r can be enabled and the net is deadlocked. A circular wait and a deadlock situation are encountered and the control agent must adopt a policy to maneuver out of the undesired state. During the execution of the normal (production) activities, deadlocks can be avoided with an adequate control policy (i.e. sequence of transition firings). For the error recovery activities, the deadlock prevention or avoidance may be more difficult due to the characteristics of the error recovery nets.

Deadlocks might be unavoidable and provisions must be taken to handle such undesirable situations. The policy adopted in this research to maneuver out of such deadlock states consists of allowing temporarily a buffer overflow.

Figure 4 illustrates an example of maneuvering out of the deadlock situation using a Petri Net model. In the Petri net, the transition t_r in figure 4 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. If this policy is not feasible in real systems due to buffer limitations, human intervention may be required.

This deadlock maneuvering brings another undesirable situation: Consider figure 4 where firing t1 twice would put two tokens in place b1 and the original buffer capacity would be permanently doubled. To compensate for this situation, the following measure was taken in this research: 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. Negative tokens for Petri Nets have been proposedfor automated reasoning[11]. In this research, the concept of negative tokens indicates 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 negative tokens 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 4, the buffer will be available again only when there is at least one token in place b1. In terms of the physical system, a buffer only accepts overflows when parts are transported from situations that involve recovery actions. Since the handling or processing capability of resources cannot be easily expanded (e.g. a robot cannot handle more than one part at a time), the only way to create the required extra capacity is to expand the storage capability of the buffers.



(a) Potential deadlock

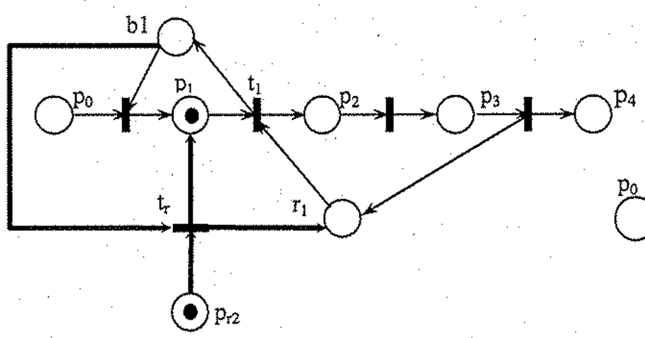
(b)Deadlocked net

Place description:
p_0: part ready
p_1: parts in buffer
p_2: part located at platform
b1: buffer available
p_{r1}: looking up part
Transitions represent the start and the end of the events

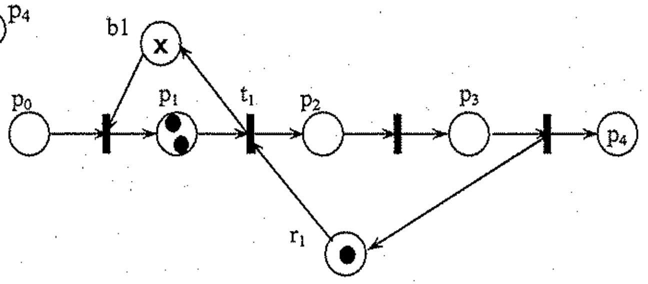
p_3: part moved by robot
p_4: part movement completed
r_1: robot available
p_e: error state
p_{r2}: picking up part
t_r: transition on the recovery subnet
t_l: transition on the activities subnet

Remark. When the token in place p_e reaches the place p_{r2}, a deadlock will occur.

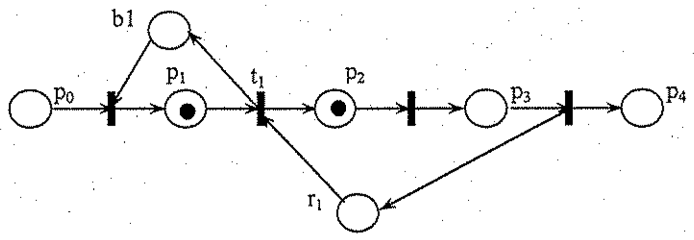
Figure 3: Example of Deadlock Wait in an Error Recovery Situation



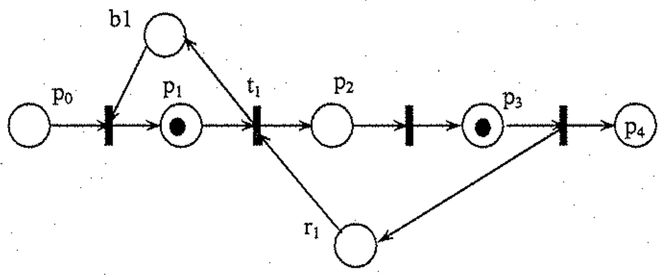
(a) Deadlocked net before firing t_r



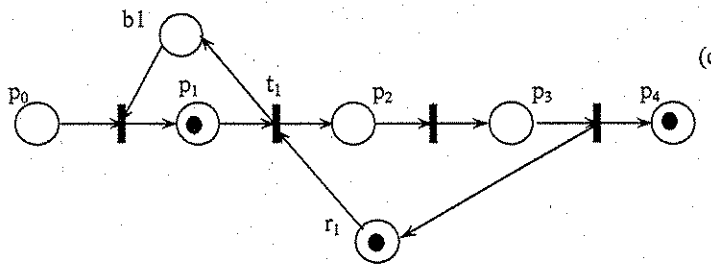
(b) Firing and deletion of t_r and the corresponding arcs and places. Overflow of tokens occurs at the buffer to avoid a deadlock. X represents a negative token



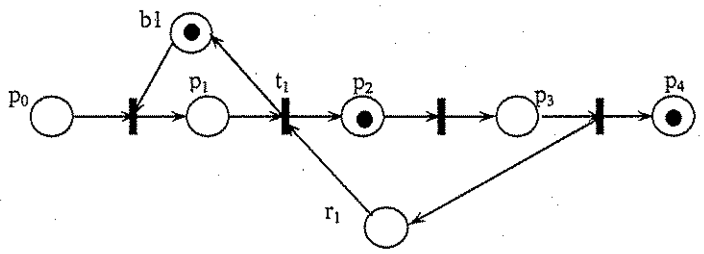
(c) Firing of t1 restores the original buffer capacity



(d) Execution of the activity represented by p_3



(e) Activity p_3 completed and release of resource r1



(f) Acquiring resource r1 and releasing resource b1

Figure 4: Deadlock Avoidance by Allowing Temporary Buffer Overflow

4. Conclusions

This paper has discussed the issues of incorporating recovery trajectories into the control logic of a workstation control agent. A contribution is the real-time error treatment which involves the addition and deletion of recovery paths from the control logic. In terms of Petri Nets, the recovery activities are modeled with a Petri subnet that is attached to the existing activities net. In this research, three types of recovery actions, namely input conditioning, backward recovery and forward error recovery were investigated from the perspective of the workstation level in a hierarchical intelligent based architecture. Since the recovery actions were previously developed for very low levels of control (i.e. equipment level), modifications have been proposed to the three types of recovery action features that characterize the workstation level. Such features are the preservation of the level of detail workstation commands and the handling of resource allocation during the execution of recovery actions. The incorporation of recovery subnets at the workstation level brings two undesirable situations:

1. The reachability graph becomes infinite with the incorporation of elementary circuits that model the recovery activities.
2. The new augmented net may bring deadlock situations that cannot be prevented or avoided.

Since the augmented net (recovery and activities net) can produce unavoidable deadlocks, a strategy must be designed to maneuver out of the deadlock situation. The strategy proposed here to counteract the deadlock situation is to allow temporary overflows at the storage buffers. In terms of the Petri Nets this overflow is modeled with the concept of negative tokens. This simple but effective strategy solves the problem and avoids the verification of structural properties. The dynamic incorportion and deletion of recovery nets is given in a concurrent paper [12]. The primary idea is that given a Petri Net and an initial marking, it is possible to reach the desired unaugmented final marking when errors and their recovery nets are randomly generated. This involves the generation of execution plans that include both recovery and normal operation activities.

REFERENCES

[1] Odrey, N.G., Mejia, G.(2002), A Re-configurable Architecture for Error Recovery in a Multi-Agent Production System, Proceedings of the 12th International Conference on Flexible Automation and Intelligent Manufacturing, Dresden, Germany, pp.1115-1125, 2002

[2] Barad, M. Sipper,D. (1988). Flexibility in Manufacturing Systems: Definition and Petri Net Modeling. International Journal of Production Research. 26 (2) 237-248.

[3] Liu C. S. 1993. Planning and Control of Flexible Manufacturing Cells with Alternative Routing Strategies. Ph.D. Dissertation. Department of Industrial Engineering, Lehigh University.

[4] Brandin, B. 1996. Error-recovering supervisory control of automated manufacturing systems. Integrated Computer-aided manufacturing, 3 (4) 255-267

[5] Zhou, M. DiCesare, F. 1993. Petri Net Synthesis for Discrete Event Control of Manufacturing Systems. Kluwer Academic Publishers. USA.

[6] Fielding, Paul J., DiCesare, Frank Goldbogen, Geof. Desrochers, Alan. 1987. Intelligent automated error recovery in manufacturing workstations. Proceedings - IEEE International Symposium on Intelligent Control, 280-285

[7] Seabra-Lopes, L. Camarinha-Matos, L M. 1996. Towards intelligent execution supervision for flexible assembly systems. Proceedings of the IEEE International Conference on Systems, Man and Cybernetics. 2,pp1225-1230.

[8] Kokkinaki, A I. Valavanis, K P. 1996. Error specification, monitoring and recovery in computer-integrated manufacturing: an analytic approach. IEE Proceedings: Control Theory & Applications, 143 (6) 499-508.

[9] Ma, H.Y. (2000). "Flexible Manufacturing Workstation Control with Error Recovery Capability". PhD. Dissertation. Lehigh University. Department of Industrial Engineering.

[10] Albus, J. 1992. RCS: A Reference Model Architecture for Intelligent Control. IEEE Journal on Computer Architectures for Intelligent Machines, 46-59

[11] Murata, T.Yamaguchi, H. (1991). A Petri Net with Negative Tokens and its Application in Automated Reasoning. Proceedings of the 33rd Midwest Symposium on Circuits and Systems. Chicago, USA. 2 762-5.

[12] Mejia, G. , Odrey, N. (2004), Real Time Control and Error Recovery of Flexible Manufacturing Workstations: An Approach Based on Petri Nets, submitted to 14th Internationl Conference on Flexible Automation and Intelligent Manufacturing FAIM 2004, Toronto, Canada, (unpublished)