An Augmented Petri Net Approach for Error Recovery in Manufacturing Systems Control

Nicholas G. Odrey
Lehigh University

Gonzalo Mejía
Universidad de Los Andes

Report No. 04T-021

An Augmented Petri Net Approach for Error Recovery in Manufacturing Systems Control

*Nicholas G. Odrey and **Gonzalo Mejía

* Department of Industrial and Systems Engineering Lehigh University Bethlehem, PA, 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 implications of error recovery tasks from the perspective of control are also discussed. 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 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.

Keywords: Petri Nets, Error recovery, Flexible manufacturing systems, Manufacturing systems control

1. Introduction

In this paper we focus on the workstation level of a hierarchical manufacturing system. A workstation is typically a set of parallel machines linked by material handling devices that perform one or more manufacturing and assembly operations. 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 [1].

Petri Nets have been successfully used for modeling and control the dynamics of flexible manufacturing systems. Several modeling approaches based on Petri Nets that include those of [1][2] and [3] have been proposed. Generally, 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. In this paper we follow the modeling approach previously presented by [1] and [4]. 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

* Corresponding autor: email: ngo0@lehigh.edu. Phone Number: (610) 758-4036 FAX (610) 758-4886

routings, and material handling devices to name a few.

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 dramatic decrease of performance. The error recovery problem has been extensively studied: Examples of automated reasoning systems for error recovery procedures, such as expert systems and neural networks have been proposed over the years and include [7], [8] and [9].

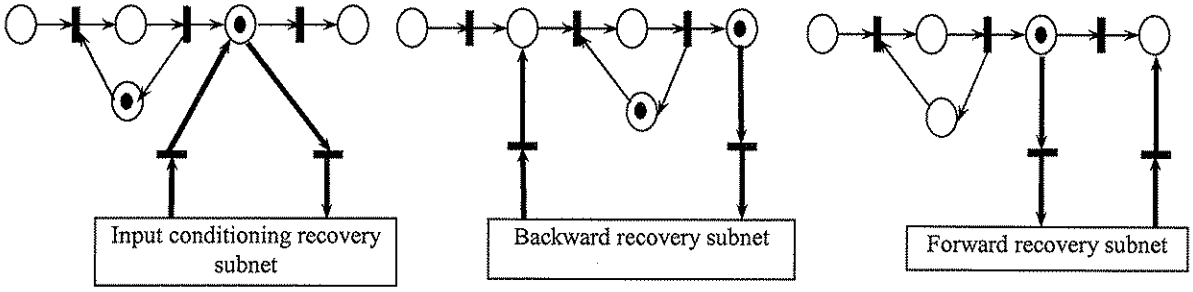
Previous work pertained to addressing the issue of monitoring, diagnostics, and error recovery within the context of a hierarchical multi-agent system [5]. The system consists of production, mediator, and error recovery agents. Production agents contain both planner (scheduler) and control agents. Here 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 diagnostics 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. 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 [3], 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 deadlocks.

2. Background

The construction of error recovery Petri subnets and similar representations is a topic which has received considerable attention in the literature. an abnormal state can become a normal state after other actions are finished or some conditions are met. 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 A forward trajectory is the most desirable but at the same time, it is the most difficult to implement with automated reasoning systems [6].

For example, [10] presented a typical Petri Net representation for machines breakdowns and alternative routings. Extensions to such model representations to handle a more complicated logic including requests for recovery actions and temporary storage in buffers were also accomplished [11]. This work was extended [12] with a representation to handle a more complicated logic that included requests for recovery actions and temporary storage in buffers.

Perhaps the most complete description of error recovery trajectories was developed by [3] who proposed three possible error recovery trajectories: 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. 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. Zhou and DiCesare [3] developed a formal description of these three possible trajectories in terms of Petri Net constructs. See figure 1 for an example. The input conditioning example shows a trajectory that "returns" to the state where the error occurred (figure 1.a). The backward recovery trajectory aims to reach a state visited prior to error occurrence (figure 1.b). Finally, the forward recovery trajectory aims to reach a state reachable from the state where the error occurred. Figure 1.c illustrates a forward recovery trajectory.



(a) Example of input conditioning

(b) Example of backward error recovery.

(c) Example of forward error recovery

Figure 1. Error Recovery Trajectories (following Zhou and DiCesare [3])

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 [6]. A forward trajectory is the most desirable but at the same time, it is the most difficult to implement with automated reasoning systems as pointed out by [6]. The three recovery strategies developed by [3] 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 are applied to the workstation level. The main contributions of this paper are (i) the incorporation and deletion of such types of trajectories in real time and (ii) the handling of resources in error recovery situations. In particular, the handling of resources implies a difficult task since error recovery may bring undesirable situations such as deadlocks and irreversible states.

The control logic workstation of the controller must be able to handle both production and recovery 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".

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. 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
- 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 [3] 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 [13]. 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. Real Time Control of Flexible Manufacturing Workstations

A mathematical representation of a General Petri Net (GPN) was first reported by Murata [14] which described the state evolution of a system as a result of firing enabled transitions. Approaches for a state space representation of a timed Petri net have been reported by Sifakis [15], Chretienne [16], and Liu, Ma, and Odrey [12]. The state equa- tions developed by Sifakis were expressed within a continuous time domain and were used for performance evaluation. Chretienne [16] introduced the concept of controlled executions to accommodate the addition of time delays. Several classes of controlled executions including finite, complete, periodic, and limited executions were identified to describe the corresponding system evolution. Both approaches are limited to applications with a single machine assumption. Liu et.al.[12] developed a cell level timed, colored Petri nets (TCPN) state space representation for systems with parallel machining capability. His 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. Subnets are viewed as alternative paths to the discolored TPN. This approach is similar to bottom-up synthesis methods in the sense that subnets are connected/related to a discolored TPN through several transitions. The difference is that instead of fusing transitions together, external places are created to link transitions together. Although the total number of transitions and places generated in this approach is increased, the resulting graphical representation is not as complicated as the substitution approach due to the preservation of modular structure that distinguishes each subnet from the discolored TPN. In addition, the alternative-path approach is more flexible than the substitution approach in the sense that changes in subnets can be made without changing the configuration of the discolored TPN.

The system state equations for TPN workstation model may be represented by the following model. The reader is referred to [4] and [9] for detailed explanations of these equations.

$$\begin{bmatrix} M^p(k+1) \\ M^r(k+1) \end{bmatrix} = \begin{bmatrix} [I] & [0] \\ -\delta(k)\, Q & [I] \end{bmatrix} \begin{bmatrix} M^p(k) \\ M^r(k) \end{bmatrix} + \begin{bmatrix} C \\ TC^+ \end{bmatrix} U(k)$$

Where

- $M^p(k)$ is the system marking vector at stage k.
- $M^r(k)$ is the remaining time vector at stage k
- $U(k)$ is the control vector at stage k
- $\delta(k)$ is a scalar representing the time elapse between the k-th and the $(k+1)$-th transition firing
- C is the incidence matrix
- Q represents the operation places matrix (Q_{ii} =1 for timed places (operational); 0 otherwise)
- T is a diagonal processing time matrix for TPNs (T_{ii} = time delay associated to place i)
- $[I]$ is the identity matrix and [0] is a 0 matrix

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. Process plans, assembly plans and error recovery can be modeled by subnets and be evaluated independently through individual TPN state equations.

Our overall interest is in optimizing the performance of a manufacturing workstation in the context of dynamic scenarios where errors occur. In the event of disruptions, the original activity plan devised off-line by the workstation controller may require adjustments. The question that arises is which 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. In general, the literature (e.g. [17]) has shown that a completely new plan provides the best performance. However, the completely new plan may be costly in terms of computational time. The other extreme which resumes the original plan as soon as the disturbance is fixed does not require major calculations but may deteriorate the overall performance of the system [17]. An intermediate measure between these two extremes is also possible. 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 marking that should be reached. Figure 2 illustrates a view of the issue of "match-up" state in a manufacturing system. Figure 2 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. From there, a number of possibilities exist to return to the original plan. The best plan implies the trajectory with minimum cost to reach the final state. Notice that the minimum cost plan to the final state may include a combination of transient and steady state trajectories. The match-up point is the point in time where the transient and the steady state schedules are compatible [18]. Details on performance optimization are given in a companion paper [19].

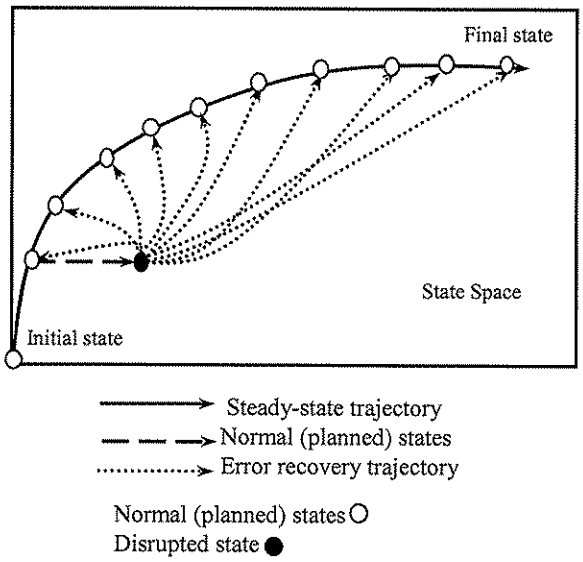


Figure 2. Error Recovery trajectories from a disrupted state

The issue of the match-up approach from the perspective of the Petri Net terminology uses the following definitions:

Definition of Final Marking (M_f): It is the state represents the state where all operations in all operations have been completed.

Definition of Intermediate Marking (M_{int}): An intermediate marking ($M_{int} \neq M_f$) is a state of the Petri Net reachable from the initial marking M_0 after following a pre-planned sequence of transition firings.

Definition of the List of Markings (Listmarkings): Listmarkings is an ordered list that contains M_0, all intermediate states M_{int} and the final marking M_f. Any transition on Listmarkings (except M_0) is reachable from the previous one on the list by firing one transition. Listmarkings corresponds to the sequence of states of the Petri Net generated offline by the workstation controller that lead from the initial to the final marking. This sequence of states corresponds to the activity plan at the workstation.

Definition of Error Marking (M_e): M_e is a marking of the Petri Net that contains one or more tokens in places representing error states and the corresponding error recovery subnets if available. An example is given in figure 6.2c.

Definition of Match-up Marking (M_u): M_u is the intermediate marking selected to be the end of the transient activity plan.

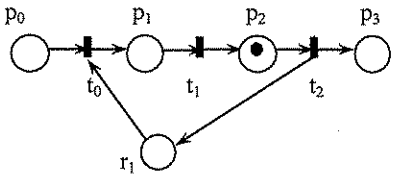
In terms of the Petri Nets, an error occurs when a transition fires outside a predetermined time frame [19]. When a transition fires earlier or later (if the transition fires at all) than expected, an alarm is triggered and an error state is produced. After the error is acknowledged and diagnosed, a recovery plan is generated. This is accomplished by linking an error recovery subnet to the activity net as shown in figure 1. Linking the error recovery subnet to the activities net 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 Listmarkings 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 is the subject of the companion article [19]. Our focus here is to demonstrate the construction of recovery subnets prior to detailed mathematical analysis.

4. Construction of Recovery Subnets

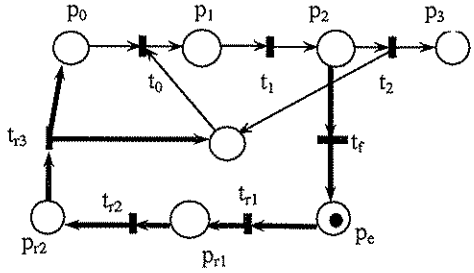
For the construction of recovery subnets, a number of important issues must be considered. An example demonstrating backward error recovery is presented here but note that a similar approach can be applied to the other types of trajectories. Figure 3 illustrates the events during an error occurrence and the corresponding recovery in terms of Petri Net constructs. Figure (3.a) represents the Petri Net during the normal operation. In figure (3.b), an error occurs in the 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 by [15] 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 figure (3.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) = $\{p_2, t_f, p_e, t_{r1}, p_{r1}, t_{r2}, p_{r2}, t_{r3}, p_0, t_0, p_1, t_1, p_2\}$. This is an elementary circuit that has only operational (timed) places.

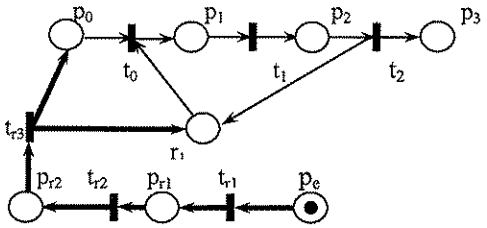
OECs describing the error recovery plans may result in infinite reachability graphs since tokens can infinitely loop around the OECs. This can be troublesome in a search strategy The strategy adopted in this research to overcome this drawback consists of the following: Every



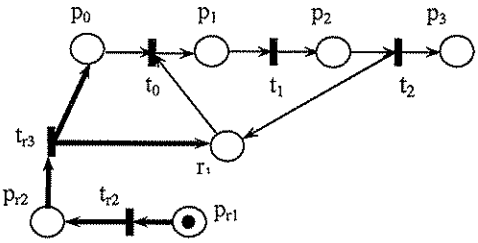
(a) Petri Net of during normal operation. A part is being processed by a resource r_1



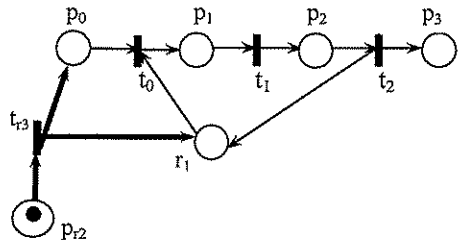
(b) Incorporation of an error/error recovery net. The error/error recovery net is shown with thicker lines.



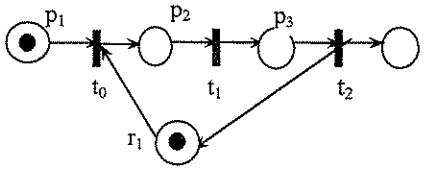
(c) Firing and deletion of t_f and the arc $I(t_f, p_e)$



(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 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 3. Construction and Deletion of Recovery Paths

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. 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) of figure 3 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/error recovery subnet

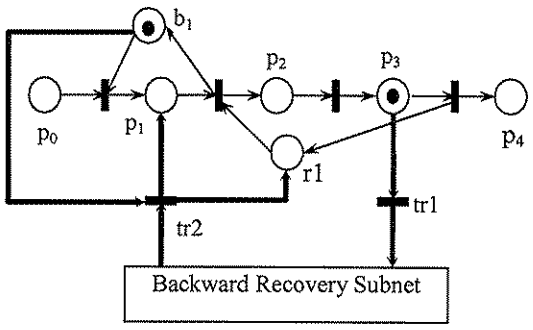
has been fired. After firing t_{r3}, the part token returns to the original net and the resource token to the resource place. The workstation control agent records which elements (places, transitions and arcs) belong to the original net and which elements correspond to the recovery subnets. This record allows that 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).

4.1 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 3(f) is not the same as the state shown in figure 3(a). The example described in Figure 3 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. In the event of an input-conditioning strategy, the corresponding net originates and terminates at the same place [3]. This research assumes that a 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 4 illustrates the application of a dynamic backward recovery trajectory to this research.

4.2 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 [11] which consider machine breakdowns in which only the machine that failed remains 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 may need to be released at some point (to be determined by the recovery agent) in the recovery trajectory.

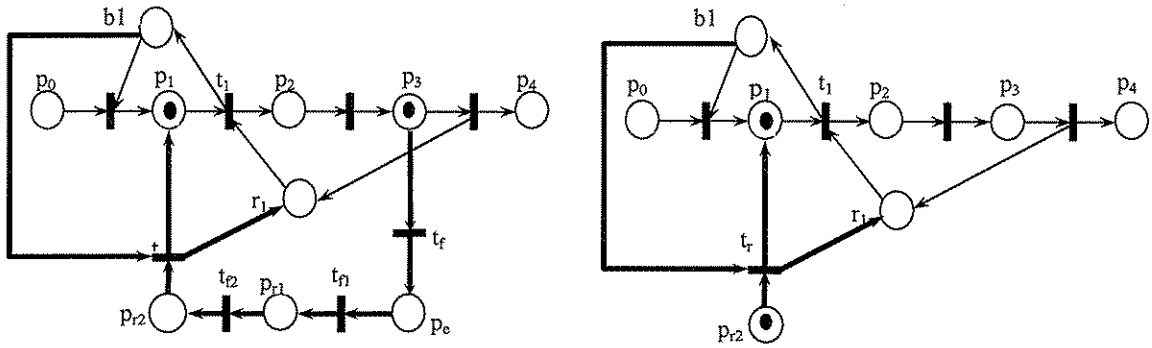


Description of places and transitions
p_0: part available
p_1: part in buffer 1
p_2: part being moved to resource 1
p_3: part being processed by resource 1
p_4: part processed
r1: resource 1 available
b1: buffer 1 available
t_{r1} and t_{r2}: Recovery transitions

Figure 4: Example of Backward Recovery Trajectory with Buffer

4.3 Issues on Deadlocks with Augmented Nets

In the case of net augmentation, the deadlocks may 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 a previously visited 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 5(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 5(b). Notice that neither transitions t_1 nor t_r can be enabled and the net is deadlocked. A circular wait (deadlock) situation is 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 aforementioned characteristics of the error recovery nets.



(a) Potential deadlock

(b) Deadlocked net

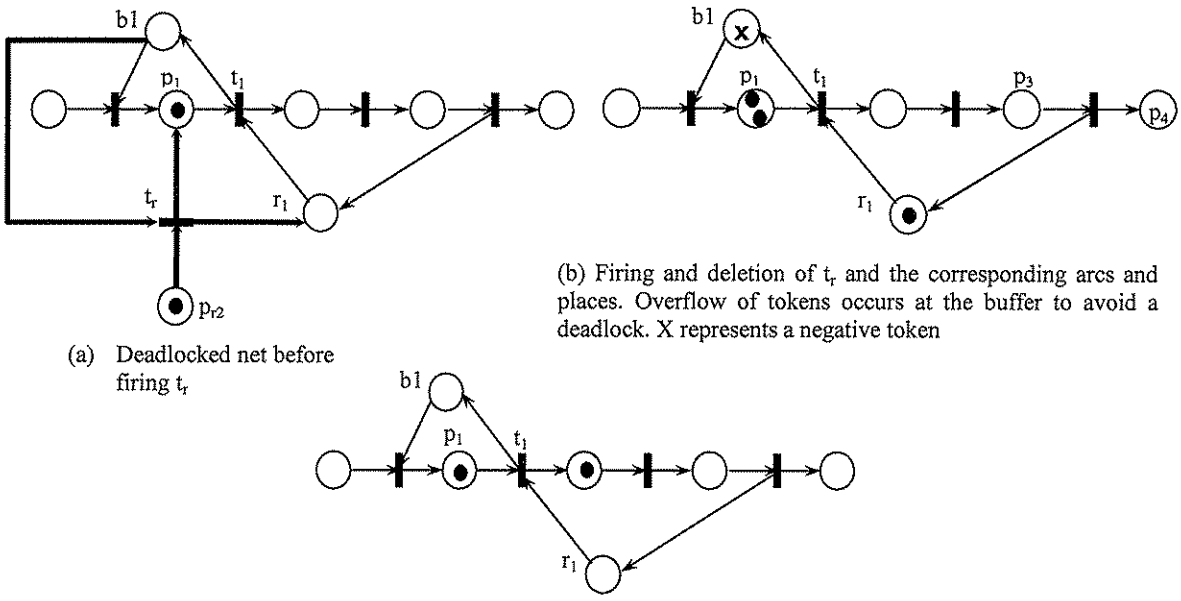
Remark. When the token in place p_e reaches the place p_{r2}, a deadlock will occur. Place description in figure 4.

Figure 5: Example of Deadlock Wait in an Error Recovery Situation

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 6 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 6 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 6 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 proposed for automated reasoning [20]. 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 6, the buffer will be available again only when there is at least one token in the "buffer" place b1.



(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 6. Deadlock Avoidance by Allowing Temporary Buffer Overflow

5. 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. As noted, the dynamic incorportion and deletion of recovery nets is given in a concurrent paper [19]. 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. Ma, Y. Intelligent Workstation Control: An Approach to Error Recovery in Manufacturing Operations. Proceedings of the 5th International FAIM Conference, June 28-30, Stuttgart, Germany 1995; 1: 124-41.

[2] Hillion, H. Proth, J.M. Performance Evaluation of Job-Shop Systems Using Event Graphs. IEEE Transactions on Automatic Control 1989; 34(1): 3-9.

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

[4] Mejia G. Odrey, N. Petri Net Models and Heuristic Search for Scheduling of Manufacturing Systems: A Comparative Study. Proceedings of the 17th ICPR Conference, Blacksburg (USA) 2003; In CD ROM.

[5] Odrey,N .Mejía G. (2003). A Reconfigurable Multi-Agent System Architecture for Error Recovery in Production Systems. Robotics and Computer and Integrated Manufacturing 2003; 19 (1-2): 35-43.

[6] Fielding, Paul J., DiCesare, Frank Goldbogen, Geof. Desrochers, Alan. Intelligent Automated Error Recovery in Manufacturing Workstations. Proceedings of the IEEE International Symposium on Intelligent Control 1987; p. 280-285.

[7] Seabra-Lopes, L. Camarinha-Matos, L M. Towards Intelligent Execution Supervision for Flexible Assembly Systems. Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, Beijing, China: 1996; 2:1225-1230.

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

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

[10] Barad, M., Sipper, D., Flexibility in Manufacturing Systems: Definitions and Management of Manufacturing Systems, New York: John Wiley and Sons; 1996.

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

[12] Liu, C, Ma, Y. Odrey, N. Hierarchical Petri Net Modeling for System Dynamics and Control of Manufacturing System, Proceedings of the 7th FAIM Conference, Middlesbrough, UK: 1997; p 220-231

[13] Albus, J., RCS: A Reference Model Architecture for Intelligent Control. IEEE Journal on Computer Architectures for Intelligent Machines 1192; 46-59.

[14] Murata,T., Petri Nets: Properties, Analysis, and Applications, Proceedings of the IEEE April 1989; 77(4): 541-580.

[15] Sifikas, J., Use of Petri Nets for Performance Evaluation, in Measuring, Modeling and Evaluating Computer Systems, H. Beilner and E. Gelenbe (Eds.), North-Holland Pub. Co.; 1977

[16] Chretienne, P., Controlled Execution of Timed Petri Nets, Technology and Science of Informatics 1984; 19-26.

[17] Sabuncuoglu, I. and Bayiz, M. Analysis of Reactive Scheduling Problems in a Job Shop Environment. European Journal of Operational Research 2000; 126(3): 567-86.

[18] Akturk, S. Gorgulu, E. Match-up Scheduling under a Machine Breakdown. European Journal of Operational Research 1999; 112 (1): 80-96.

[19] Mejia, G., Odrey, N., 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, June 12-14, Toronto, Canada: 2004; 1:824 – 831.

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