Real-Time Control and Error Recovery of Flexible ManufacturingWorkstations: An Approach Based on Petri Nets

Gonzalo Mejía
Universidad de Los Andes
Bogotá, Columbia

Nicholas G. Odrey
Lehigh University

Report No. 04T-013

Real Time Control and Error Recovery of Flexible Manufacturing Workstations: An Approach Based on Petri Nets

Gonzalo Mejía*, and **Nicholas Odrey

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

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

ABSTRACT

This paper introduces an intelligent flexible workstation controller integrating Petri Net models, heuristic search methods and error recovery capability. The activities of the workstation controller are modeled via a re-configurable Petri Net formalism that allows the incorporation of recovery tasks into its logic. The workstation controller follows a production plan generated off-line during its normal operation and appropriately reacts when the original plan is disturbed due to the presence of errors. The off-line production plan is generated via a heuristic Beam A Search (BAS) algorithm presented in previous work. The implications of the error recovery tasks from the perspective of control are also discussed. When errors occur and the system cannot follow the original schedule, the controller attempts to reach the original production plan in a later stage in such a way that the effect of the disruption is minimized. This is accomplished with a modified version of the BAS algorithm adapted for on-line scheduling. This modified BAS algorithm generates a partial plan of activities that "matches-up" with the original production plan. Simulation tests were conducted in order to study the behavior of the algorithm under different scenarios. The results are and analyzed discussed. We conclude that our approach can be a valuable tool for real time control of flexible manufacturing systems.*

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 from a higher level controller (i.e. a cell controller) that corresponds a set of operations to be performed on the workstation with the desired begin and finish times. The workstation controller decomposes such a command into a lower level set of coordinated activities [1]. 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. Typically a Petri Net representation of a manufacturing workstation includes the routing of the to-be-processed parts and the required resources to process such parts. 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].

In simple workstation configurations, the coordination of activities is not difficult as the number of possible states could be easily handled by exhaustive enumeration. As the complexity of the workstations increase more robust methods are required for sequencing and coordination of such activities. 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.

In order to optimize the execution of the workstation activities, we introduced a heuristic search algorithm [4]. This method partially searches the net state space of a Petri Net model of a manufacturing workstation to find a near-optimal execution plan. In the Petri Net terminology this corresponds to finding a valid sequence of transition firings whose execution would minimize a criterion. This algorithm was named as Beam A* Search (BAS). In addition, the sequence of transition firings found with BAS is guaranteed to be deadlock free. Here the criterion is to maximize production rates or minimize makespan.

Unfortunately, the Petri Net representation of the workstation activities does not have provisions for errors and other disturbances as these errors are generally not explicitly included in the logic of the controller. Thus, the controller would have difficulties when handling such abnormal error states. A temporary solution for these situations is to explicitly model the error states via Petri subnets [5][6]. Figure 1 shows an example of such subnets. These are nets, linked to the original net (the net which models the pre-planned activities and the required resources), representing states such as "machine unavailable" or "machine repair"[5]. This solution, however, is not feasible in real workstation controllers because (i) the number of possible errors can be enormous, and (ii) an error not planned would prevent the controller to work properly. In this paper errors are only modeled with Petri subnets when they occur and eliminated from the original net when recovered. In the remainder of the documents such subnets are denoted as error recovery subnets.

The efficient handling of errors imposes a difficult task to the workstation controller as the original activity plan (off-line plan) must be adapted to the error situation. When the original plan is disrupted, the workstation controller must adopt a remedial measure to minimize the effect of the error. Thus, the workstation controller must devise a contingency plan to re-sequence the jobs on the physical workstation. In this paper we propose an approach in which the controller attempts to "match-up" the contingency plan with the original production plan. The primary goal of this paper is investigate the issues related to the re-planning activity in order to obtain both good performance in terms of total flowtime and efficiency measured in computational time. The remainder of the paper is organized as follows: Section 2 discusses the issues related to modeling error recovery with Petri Nets; Section 3 introduces our approach for real time control; Section 4 presents the results and analysis of our simulation tests. Finally in section 5 our insight for further research is discussed.

2. BACKGROUND

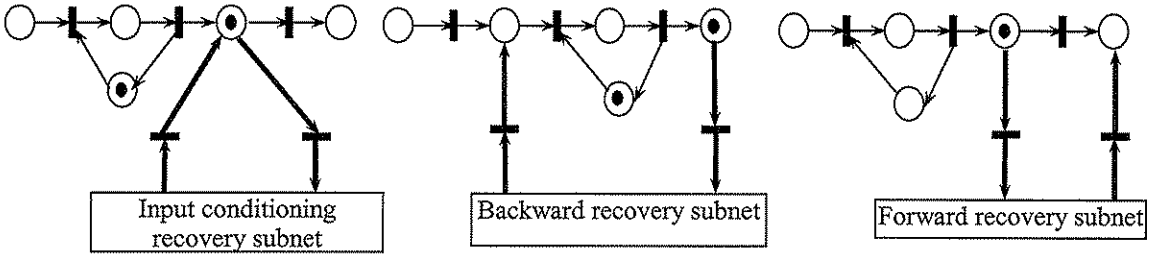
The construction of error recovery Petri subnets and similar representations is a topic which received considerable attention in the literature. For example, [6] presented a typical Petri Net representation for machines breakdowns and alternative routings. Liu [7] extended such a representation in order 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 of the three types of recovery paths. 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 the state where the error occurred (figure 1.b). Finally, the forward recovery trajectory aims to reach a state reachable from the state where the error occurred (figure 1.c).

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 [8]. 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 [8]. 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 also applied to the workstation level. 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 the maintainability of the net properties.

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".



(a) Example of input conditioning (b) Example of backward (c) Example of forward

Figure 1. Error Recovery Trajectories modeled with Petri Nets

3. REAL TIME CONTROL AND OPTIMIZATION OF FLEXIBLE MANUFACTURING WORKSTATIONS

This section investigates the issue of 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. [9]) 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 [9]. 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 [10].

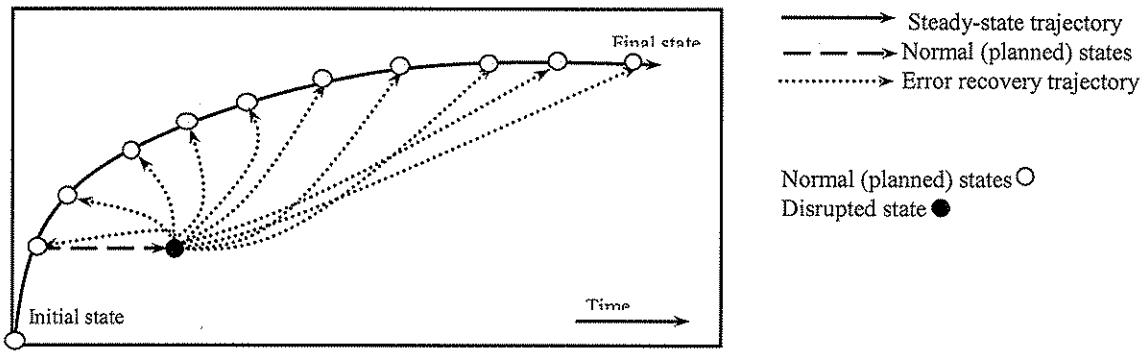


Figure 2. Error Recovery trajectories from a disrupted state

3.1 MATCH-UP STRATEGIES USING PETRI NETS AND HEURISTIC SEARCH

This section investigates the issue of the match-up approach from the perspective of the Petri Net terminology. The following are definitions.

Definition of Finak 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 [11]. 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 a 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 Petri Net model. 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 will be discussed in section 4.

3.2 THE ADAPTIVE BEAM A SEARCH ALGORITHM*

The logic of the workstation controller must translate the match-up requirements to methods (algorithms) that apply to the Petri Net model. This means that the workstation controller must possess the methods to find a sequence of transitions to reach any feasible intermediate marking M_{int} (on Listmarkings) reachable from an error marking M_e. In the approach proposed in this research, the workstation controller employs a modified version of the BAS algorithm presented earlier [4] to find intermediate markings. The BAS algorithm is a modified version of the popular A* Search algorithm applied to Petri Nets [12]. BAS expands only the most promising nodes of the net reachability graph by improving on the deficiencies of A* Search. The main features of BAS are:

- Limited expansion of nodes within the levels of the reachability graph
- A non-delay scheme
- Improved lower bounds by using dispatching rules

Essentially BAS aims to trim down the reachability graph with two strategies: Limited expansion and node filtering. BAS expands only a maximum predefined number of nodes at each level of the reachability graph. When such a number is reached, the search moves onto the next level. In addition, a non-delay scheme filters out non-promising nodes by guaranteeing that a resource is never idle when it can be processing/handling a part. At the same time BAS calculates lower bounds by looking at feasible sequences of transition firings generated with dispatching rules. Recall that A* search is a branch and bound like algorithm which requires lower bounds in order to direct the node expansion.

The modified algorithm version of BAS is termed here as Adaptive A* Search Algorithm (ABAS). (ABAS) uses essentially the same features of BAS. The main difference is that the ABAS algorithm is primarily structured to find paths to desired intermediate markings, instead of paths to the final marking. In the case of the BAS, the task of

finding a path to the final marking M_f is not difficult because the algorithm forces the search deeper in the reachability graph until the final marking is eventually found. The final marking is at the lowest possible depth and a simple depth-first strategy would find the final marking. This is true since the final marking is reachable from any marking in the reachability graph provided there are no deadlocks. Finding a path between any two intermediate markings would require a breadth-first strategy or similar strategy and there will be no guarantee that such a path exists.

The proposed solution for finding intermediate markings is to construct a path to (i) a desired intermediate marking M_{int} whenever such a marking generated, (ii) any marking in the original trajectory reachable* from M_{int}, or (iii) to the final marking. The initial conditions for the ABAS algorithm are the error marking M_e, the desired intermediate marking M_{int}, the final marking M_f and the list of pre-planned markings (Listmarkings) that constitute the original plan devised offline.

Notice that in the case of the net augmented with error recovery paths, M_{int}, M_f and M_e are augmented vectors. These augmented vectors are re-dimensioned to the original dimensions once the recovery trajectories lose their tokens (i.e. the errors are recovered).

4. COMPUTATIONAL EXPERIENCE

This section investigates the selection the match-up marking from the list of pre-defined markings (Listmarkings) when the normal operation of the system is resumed after the recovery of an error. Selecting an intermediate marking is related to choosing the number of operations to be re-scheduled. It can be noted that targeting intermediate markings located near the end of Listmarkings, will involve the re-planning of more operations. The primary idea of the experiments is to detect differences in performance as the number of operations to be re-planned is varied.

4.1 ASSUMPTIONS

- There are a number of jobs assigned to the workstation which are available at time 0. A job consists of a batch of parts. The setup time for each job is 0. No new jobs arrive to the workstation and no jobs are canceled.
- There is a probability of error for each operation of a job. This is, either an operation fails with probability p_f or it is successfully completed with probability 1- p_f. The probability that an error occurs in an operation is independent of the previous error occurrences. Unlike other approaches that involve machine breakdowns (e.g. León et al, 1994), the generation of errors here is event driven and not time driven.
- All errors can be recovered. The issue of how to devise the recovery plan is out of the scope of this paper. The input to this model is the recovery time which is the time required to fix the error. The recovery times are randomly generated with a uniform probability distribution.
- All manufacturing, inspection, and transport operations have the same probability of failure. This assumption can be easily relaxed for real life studies. The frequency of error occurrence refers to the probability of failure of an operation. The probability of failure for each operation was set to 10%.

The recovery steps are modeled with a subnet containing one timed place that represents the downtime. The downtime includes the time to diagnose and classify the error and to generate a recovery plan. The recovery steps can lead to an input conditioning, backward recovery or forward recovery strategies, depending on the type of error. For the purpose of the simulation only, we assume the workstation controller selects any of the recovery strategies with the same probability.

4.2 SIMULATION STUDIES

The performance evaluation was conducted through 12 case studies that correspond to 12 workstation models. The Petri Nets workstation models were randomly generated with the following characteristics:

- Each workstation processes five jobs and each job comprises 10 parts (i.e. batch size is 10)
- There may be parallel machines for the operations performed at the workstation.

* A marking M is reachable from M_{int} if M is located after M_{int} in the ordered list Listmarkings.

- The number of operations performed at each workstation varies between 1 and 4.
- All jobs follow the same route.
- There are buffers located between machines with limited capacity (capacity = 10 parts)
- There may be one or more resouces working on a particular operation.

The following flowchart illustrates the execution of the simulation:

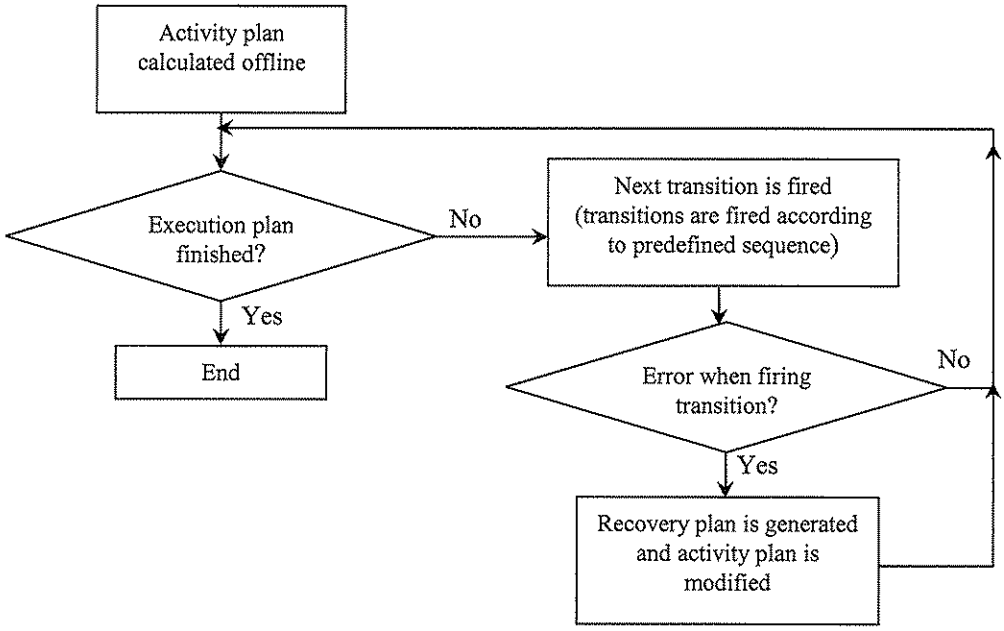
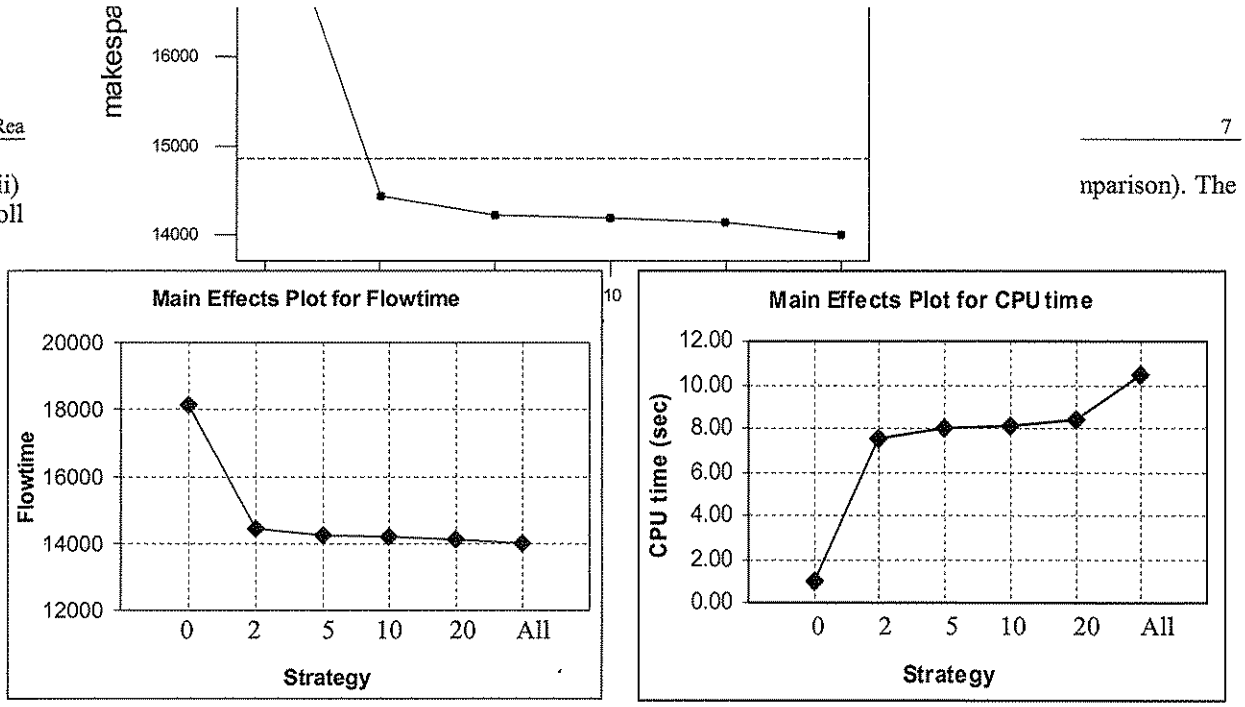


Figure 3. Flowchart of the simulation tests

The experiments were conducted by running 10 replicates for a particular problem. The re-scheduling strategy was defined by the match-up marking M_u. The match-up marking is selected from Listmarkings and can be any marking reachable from the marking state prior to the error occurrence, including such a marking itself and the final marking M_f. The farther away the selected match-up marking is located down the list Listmarking, the greater number of operations that *can* be re-scheduled The notation established here is as follows: The strategy selected depends on the number of transition firings required to reach the selected match-up marking on Listmarkings from the last non-error state (this is the last reached marking before the occurrence of the error). For example, if the strategy 0 is selected, then the match-up marking is same as the last non-error state. Strategy 2 means that the match-up marking can be reached by firing two transitions from the last non-error marking. Strategy "all" indicates that the match-up marking is the final marking. There may be cases in which a match-up marking on Listmarkings is not reachable from the last non-error state. This happens because some recovery actions put the system back on a normal state not defined in the original activity plan. In this case, the ABAS algorithm searches for any marking located on Listmarkings in the path from the selected match-up marking to the final marking. The Petri Net models were constructed that the final marking is reachable from any marking M reachable from the initial marking M_0.

The analysis of the results was carried out with an ANOVA experimental design having the final flow time and the required CPU time (sec) for the calculations as the responses and "strategy" and "problem" as factors. Flow time was the simulation clock time for each run. CPU time was the total computer time in seconds for each run. Notice that the match-up algorithm takes varying CPU times to be executed depending on the selected strategy. This is the main reason why the simulation times change from run to run. For both responses, "Strategy" is the factor of interest and "problem" is a blocking factor. "Strategy" is a fixed effect and "problem" is a random factor. Six levels were chosen for the factor "strategy" and twelve levels for the factor "problem". Ten replicates were run for each combination strategy-problem. The levels of strategies were: Strategy 0, strategy 2, strategy 5, strategy 10, strategy 20 and strategy "all". The significance level α was set to 0.05.

The ANOVA analysis revealed that the strategies are significant at the 0.05 level for both responses (flowtime and CPU times). Further analysis was carried out to (i) find the means for the factor "strategy" on both responses and



(a) (b)

Figure 4. Main Effects Plots for Flow Time and CPU Time vs. Strategy

Table 1 presents the results of pairwise comparisons for the factor strategy for both Flow time and CPU time. A "Yes" on the tables is interpreted means that the strategies of the corresponding row and column are significantly different ($\alpha = 0.05$). The pairwise comparisons were carried out with the Tukey test (See [13] for details.)

Table 1. Statistical significance between strategies

Flow time	0	2	5	10	20	All
0		Yes	Yes	Yes	Yes	Yes
2			No	No	No	No
5				No	No	No
10					No	No
20						No

CPU Time	0	2	5	10	20	All
0		Yes	Yes	Yes	Yes	Yes
2			No	No	No	Yes
5				No	No	Yes
10					No	Yes
20						Yes

4.3 ANALYSIS

The results presented the figure 4a for total flow time show that better performance is achieved with higher strategy numbers. Recall that higher strategy numbers indicate a higher number of operations that are subject to re-schedule and consequently indicate a higher level of planning. However, the gains resulting of higher number strategies were very small and only in the case of strategy 0 compared against the other strategies, there were no differences at the desired significance level. The second analysis concerns finding differences between the means of the strategies being the CPU time the response for the experiment. Figure 4.b shows the means for the factor CPU.

The Tukey comparisons show that the only strategies that were significantly different from the others were the strategies 0 and "All". The overall conclusion of the experiments is that small gains in performance (measured as total flow time) were achieved with significant increase in computational effort. At the workstation level, the planning horizon is short [1] and therefore a quick response is critical. Hence, the selection of the right level of planning for the problems tested here seems to correspond to a low-number strategy.

5. CONCLUSIONS

This paper has presented the methods to react to errors occurring at the shop floor employed by a workstation controller the context of a hierarchical architecture. The methods developed in this research for the controller were:

- The incorporation and deletion of error recovery trajectories into the logic of the controller.
- The re-planning capability that can adapt the original execution plan to the conditions in the shop floor without requiring a full plan overhaul.
- The translation of these methods into a Petri Net language.

The strategy followed in this research corresponds to the "match-up" approach adapted to the Petri Net logic. In terms of the Petri Nets employed here, match-up consists of finding a path from an error state to an arbitrary state termed here as the "match-up" marking. In this paper, a new Petri Net based algorithm (ABAS) was derived with the purpose of finding an arbitrary match-up marking in the logic of a Petri Net based controller. Simulation experiments were carried out in order to investigate the performance of the algorithm in terms of simulation total time and CPU time. The analysis of the experiment was performed with an ANOVA factorial experimental design having the factors "strategies" and "problem". Further research will be focused on adapting the algorithm to situations related to other disruptions such as order cancellation, machine breakdowns and the arrival of new jobs.

REFERENCES

1. *Odrey, N. Ma, Y. Intelligent Workstation Control: An Approach to Error Recovery in Manufacturing Operations. Proceedings of the 5th International FAIM Conference, Vol 1, pp124-141.1995*
2. *Hillion, H. Proth, J.M. Performance Evaluation of Job-Shop Systems Using Event Graphs. IEEE Transactions on Automatic Control, Vol 34, N 1, pp 3-9. 1989.*
3. *Zhou, M. DiCesare, F. Petri Net Synthesis for Discrete Event Control of Manufacturing Systems. Kluwer Academic Publishers. USA. 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). In CD ROM. 2003*
5. *Proth, J.M Xie, X. Petri Nets: A Tool for Design and Management of manufacturing Systems. John Wiley and Sons. New York. 1996.*
6. *Barad, M. Sipper,D. Flexibility in Manufacturing Systems: Definition and Petri Net Modeling. International Journal of Production Research. Vol 26, N 2, pp 237-248. 1988*
7. *Liu, C, Ma, Y. Odrey, N. Hierarchical Petri Net Modeling for System Dynamics and Control of Manufacturing Systems. Proceedings of the 7th FAIM Conference, Middlesbrough, UK 1997.*
8. *Fielding, Paul J. DiCesare, Frank. Goldbogen, Geof. Desrochers, Alan. Intelligent automated error recovery in manufacturing workstations. Proceedings - IEEE International Symposium on Intelligent Control, pp280-285. 1987.*
9. *Sabuncuoglu, I. and Bayiz, M. Analysis of Reactive Scheduling Problems in a Job Shop Environment. European Journal of Operational Research. Vol 126, N 3, pp 567-586. 2000.*
10. *Akturk, S. Gorgulu, E. Match-up Scheduling under a Machine Breakdown. European Journal of Operational Research. Vol 112 N 1, pp 80-96. 1999.*
11. *Ma, H.Y. Flexible Manufacturing Workstation Control with Error Recovery Capability. PhD. Dissertation. Lehigh University. Department of Industrial Engineering.*
12. *Lee, D. Y. DiCesare, F. Scheduling Flexible Manufacturing Systems using Petri Nets and Heuristic Search. IEEE Transactions on Robotics and Automation, Vol 10 N 2,pp 123-131. 1994.*
13. *Montgomery, D. Design and Analysis of Experiments. 4th edition. John Wiley and Sons. New York. 1997.*