Proc. 9th CIM-Europe Annual Conference, 12 - 14 May 1993,
Amsterdam, pp. 170 - 179, IOS Press.

Physical Error Recovery for NC-Machines

Ursula Schmidt
Institute for Parallel and Distributed High Performance Systems, University of Stuttgart
Breitwiesenstr. 20 - 22, 7000 Stuttgart 80, Germany

Abstract. The use of transactions in flexible manufacturing systems is expected to result in a number of advantages like increased software reliability and minimized restart times in the case of system or machine failures. Here, an architecture of a system for handling processes on machine tools under transactional protection is briefly presented. Special attention must be drawn to the component which determines the recovery measure for NC-machines in a Flexible Manufacturing System, the Physical Recovery Manager. Various affecting factors are described as well as a possible strategy for evaluating the available information. A prototypical Physical Recovery Manager within a testing environment for manufacturing operations is currently being implemented.

1. Introduction

The project "transactional concepts in manufacturing" at the University of Stuttgart explores how manufacturing processes can be performed more reliably with the help of transactional concepts.

Classical transactions were developed for areas like banking or flight-reservation systems, where data have to be correct and persistent under all circumstances, i.e. where many users and processes, respectively, concurrently access and update data and where system breakdowns must not leave the system in an incorrect state. Transactions are groups of operations, which are executed by the transaction processing system (TPS) *atomically*, i.e either the group is performed as a whole or, in the case of failures or if initiated by the user, none of the actions leaves any relevant trace in the system. This is performed by undoing all effects of the failed transaction's operations (rollback), whichever reason the failure may have.

Transactions guarantee data consistency, results are durable and the applications can work in a logical single user mode. These features are provided by the low-level functions of the TPS. Thus, transactions completely remove the burden of failure handling from the application programmer.

These properties motivate the use of transactions in manufacturing, too:

- when using transactions, failure handling is kept off the application code which in turn will be considerably simplified and thus be more reliable.
- the restart times of a machine or the system will be minimized by guaranteeing consistency of the system state. The defined failure semantics of the transactional model make many of the otherwise required checks superfluous.
- In the long run, transactions may alleviate the integration of manufacturing into the general business' processes, where transactions have already gained acceptance.

Unfortunately, classical transactions cannot simply be applied to manufacturing processes. The problem stems not from concurrency control which in database systems relies on advanced locking mechanisms; a transformation to FMS seems not too hard, as FMS have implemented their own concurrency control in control systems. The point is that the low-level recovery mechanisms guarantee atomicity by using backward recovery in the case of failures. Rolling back data is easy as long as there is enough information about the original values to restore them. Rolling back a physical operation is often enough impossible. However, the expected improvements motivate us to relax the hard requirement of atomicity as far as necessary und save as many advantages as possible.

In this paper, we want to

- show that transactions are an appropriate tool to model manufacturing processes,
- briefly present an extended transaction processing architecture and introduce the necessary transactional terminology,
- describe the requirements to machines and the underlying system and will
- put the emphasis on the determination of recovery measures for physical operations.

2. Modelling manufacturing processes with transactions

To demonstrate the feasability of transactions in a manufacturing environment, we consider the use of NC-machines in flexible manfucturing systems. First, we have to introduce the construct of action chains.

Action chains are used in control systems (see for example [6]) to keep track of the next physical action which has to be performed on physical objects such as work pieces, filing buckets (which must be emptied in regular intervals), and tool pallets (in order to exchange tools between manufacturing machines). Action chains for work pieces are of special interest. They are generated by the control system from the work schedule for that part, which mainly contains the action to be performed and the capacity entity (i.e. a machine, cell, or manual work place) where this step is to be performed. A very simple schedule for a work piece could look as follows:

step	action	capacity entity
1	clamp work piece into fixture	work place
2	mill	machine 1 or machine
3	release work piece from fixture	work place

An action chain is built from this work schedule by adding the necessary transport jobs, but still leaves some alternatives open. The corresponding action chain for the work schedule above then might have the following form:

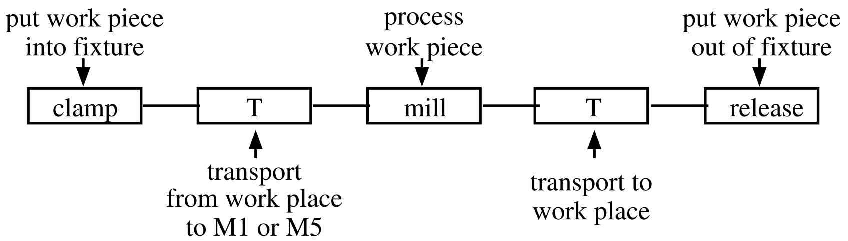


Figure 1 Example of an action chain

An action chain is processed step by step, and scheduling decisions are made immediately before the corresponding step. If, for example, machine 1 is overloaded or out of order, the job is assigned to machine 5.

These action chains seem to be ideally suited for being mapped onto transaction chains: There are distinguished steps depending on other steps, several action chains (or to be more precise: tasks that execute these action chains) compete for the same machines, the whole chain can be treated like a logical unit and so on.

Further, the requests to the machines or databases can be automatically generated by the control system, when transforming the work plan to an action chain. The requests generated for a step semantically belong together - this allows for letting the control system set the transaction demarcations automatically. Here, we do not care which operations are requested to perform a step - there could be potentially many operations like loading a program or writing the result of the manufacturing operation into some variable of a database. We simply require that these operations are performed by so-called Resource Managers (RMs). An RM is "a subsystem that manages some transactional objects" [4]. On such objects, for all operations belonging to the same transaction, atomic execution is guaranteed. Classical RMs like databases ensure this by undoing all respective operations. There could be more than one RM involved when performing the requests for a certain step, and this leads to distributed transactions.

3. An Architecture for distributed transactions

When working for a distributed transaction, the participating resource managers must coordinate their work - if one of the RMs fails and undoes its work, the other RMs must undo their work, too, to ensure atomic execution of the whole transaction. This procedure is called *aborting* a transaction. The job of telling the RMs when a transaction is to be aborted is done by the Transaction Manager (TM) in the following manner:

At the beginning, the application informs the TM that it wants to start a transaction. Then, it communicates with the RMs to make them perform the desired operations under the scope of this transaction. Finally, it informs the TM about the end of the transaction. If one of the RMs fails, the TM tells the remaining RMs to abort the transaction, i.e. to undo the work they had performed so far on behalf of that particular transaction (see figure 2, without the shaded box, for the architecture). Such a failure could occur at the end of the transaction, when the RMs try to make the results durable or it might occur when an RM fails during processing, for example, if it crashes. When carrying out the termination of a transaction, TM and RMs participate in the 2PC - Protocol [1] for correct behaviour.

Now, if we want to let NC-machines participate in transactions, they must be represented by special RMs for physical operations, Physical Resource Manager (PRMs), which can participate in the 2PC-protocol. A PRM consists of the PRM-core and the NC-machine itself. The core represents the machine in that it receives all requests for the machine, keeps track of which request was performed for which transaction and initiates all necessary recovery operations. However, this makes a new system component necessary.

4. An Extended Architecture for integrating physical operations

NC-machines provide services which cannot be undone - for them, atomicity can no longer be achieved by means of rollback. Therefore, we have to try forward recovery, when a request for such a service fails. This means that the failed operation must be brought to a successful end[1]. However, should recovery fail irrevocably, the transaction must be aborted: The participating RMs undo their part of the work, and the PRMs undo the respective non-physical operations, put the underlying machine into the reference position and unload the work piece. Then, it can at least be assured that the machines and databases are left in a proper, consistent state. Further, if the failure did not put the machine into inoperable mode, the PRM is ready to service requests for other transactions.

As the determination of the proper recovery measures depends heavily on application and environment specific characteristics, a separate component is needed, the Physical Recovery Manager (PRecM), to find the correct operations.

If the failure can be handled locally, i.e. on the failed machine, the PRecM simply tells the machine what to do next. If other machines have to be involved, for example if a substitute machine must be used to continue processing of the workpiece, the PRecM extends the particular work piece's action chain.

The following picture shows how the general architecture for distributed transactions must be extended to allow for forward recovery. Additionally, the PRecM needs access to the applications data structure for the action chain.

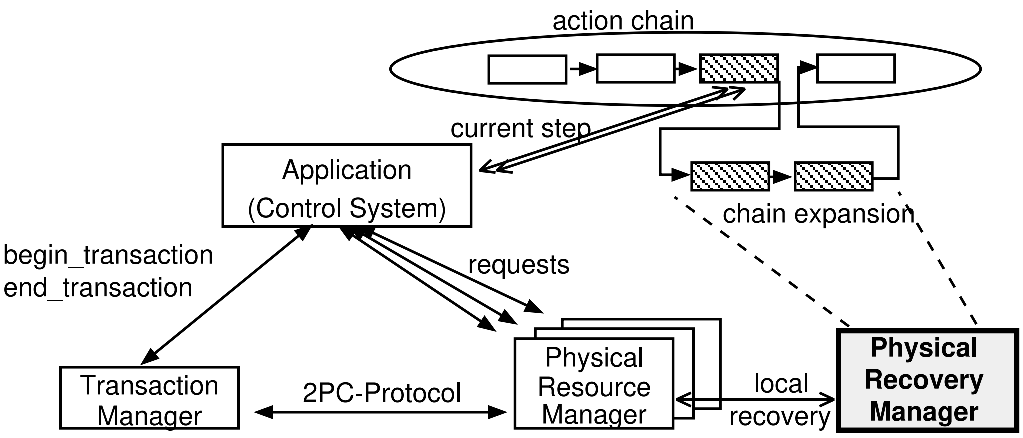


Figure 2 Extended Architecture of a Transaction Processing System

Before explaining the PRecM's behavior in detail, we want to list the requirements to our environment.

5. Requirements and Assumptions

The *control system* automatically transforms work plans to action chains and action chains to chains of transactions.

The mechanism for *processing chains of transactions* is reliable (one could use the script-processing component of the ConTract-Model [10]).

For a *machine* which underlies a PRM we assume the following:

- In the case of a failure during a physical operation the machine stops and reports failure class and reason, if available, to the supervisory PRM-core.

1 How a PRM deals with failures of nonphysical operations is described in [9]. In short, we try to redo a transaction to the point of interruption and, then, continue with regular operation. The intention is to minimize the number of aborts.

- All requests to a machine are received by the PRM-core. This means, the control system now has to communicate with the PRM-core instead of talking directly to the machine.
- Diagnosis is performed either by the machine itself or by an appropriate function or module of the control system.
- Maintenance is initiated and performed by the control system or a machinist.
- The decision how to repair a failed machine and to perform these actions is done by a machinist. For some repair operations like refilling oil or exchange of a broken tool, regular maintenance functions can be engaged. Then, the PRecM is informed whether repair was successful[2].

To sum up, we do neither care how failure classes are determined nor how a machine can be made recognize a failure and stop or how diagnosis, maintenance and repair of machines can be performed. We assume that these things are done by the control system or a human being.

6. Recovery for NC-machines

The crucial problem when trying to perform forward recovery is to find the right recovery operations. As mentioned above, many factors have impacts on the process of making a decision. It must be considered both whether recovery is technically possible at all and whether this economically makes sense. To determine the appropriate actions, the PRecM uses facts and rules:

6.1. Facts

Facts can be characteristics and state of machine, work piece, application, and failed operation. They are known in advance (describing static properties), reported when the failure occurs, or must be explicitely asked for. The semantics of operations require special attention: Our approach is to assign, possibly several, recovery classes to each operation to allow for determination of recovery operations without knowing which kind of processing a failed operation exactly involved. We distinguish the categories resumability, restartability, and compensability:

An operation is *resumable*, if it can be continued after a failure without undesired effects on the work piece. Milling, for example, could be a resumable operation. The operation is *restartable,* if its restart will lead to the desired result. If it is *compensable*, then there is a way to bring the work piece back to its initial state and a following restart will do. Typical examples are found in transportation jobs. Finally, an operation is *interruptable*, if at least one of these characteristics applies. Welding, for example, cannot be interrupted as it is neither resumable, nor restartable or compensable.

Generally, interruptability tells us whether there exists a potential recovery measure for an operation. Assigining an operation to recovery classes must be done by the step programmer as well as giving the respective test and compensation operations of the operation. Additionally, the categories of interruptability can be differentiated into the respective general and dependent category. For a *generally* interruptable operation (i.e. a generally resumable, restartable or compensable one) the respective recovery measure can be performed independently from the progress of the failed operation. *Dependent* interruptability involves a

[2] If the PRecM is a sophisticated one, it could, before repair is started, additionally be informed about the expected time of repair, to decide whether this is acceptable for recovery.

check how far the operation had progressed before the failure occured. Therefore, information about dependent interruptability can be evaluated only if the outcome of the operation can be checked - then the operation, to be precise, its outcome, is said to be *testable*.

6.2. Testability

Testability is not really an operation's characteristic - manufacturing operations do always modify a work piece, and each kind of modification can be found out - but rather a system characteristic. The problem is that the process of finding out the (partial) result can modify the work piece in an undesired way. Take as an, admittedly extreme, example, checking a work piece's welded joint for quality control. Here, the welded joint is torn apart - one might find that it sufficed the requirements, but the work piece must be thrown away. In other cases, measuring the result is no problem, in principle, but the required gauges and equipment might not be present in the manufacturing system, for example, because they are too expensive.

For our purposes, testability is achieved, if for the required information one of the following conditions holds:

- the information is available or can be obtained by reading and deriving some variables. Regarding a transport robot, for example, the position of the work piece could be derived from the robots' axes' positions.
- it can be obtained by explicitly starting some checking operations. Some manufacturing centers offer integrated gauges, which can automatically be exchanged and used. Other machines can be combined with image processing systems.
- it can be provided by a human being, i.e. a machinist or operator. This is the most general, and the most unspecified, case.

Testability is a complex issue and, therefore, a critical point. In general, the need for testability should be avoided by rather using generally interruptable operations than dependently interruptable ones. For example, requests with absolute parameters like "move to position a" should be preferred to requests like "move 3 inches left", for which the real position is needed to determine the remaining transport distance.

Of course, we must consider more factors than the mere operation's characteristics for recovery: For a machine, some obviously necessary information is its state (is it still operable and available?), but additionally one must know the precision it provides. Usually, an application allows certain tolerances for a manufacturing operation, say, 3 units deviation from the required measure might be acceptable (one needs this information, too!). If the machine provides only a precision of 2 units, recovery may not restart the operation - the second try would already exceed the tolerances.

For the work piece, its state must be known (do we have to throw it away?). Additionally, there is a need for information about the type of failure. The type of failure implicitly provides information about the states of machine and work piece. If, for example, a failure occurs which allows resume of the current program, it is safe to assume that the machine is operable and the work piece is not damaged. After a power failure, however, both machine and work piece must be checked for integrity. Here it is assumed, that the PRecM knows the characteristics of a failure class (for a classification of failures into classes according to the impacts on the continuation of processing see manuals like [5]).

For global recovery, the PRecM must know whether there are substitute machines in the system and which state they are in.

6.3. Rules

Rules incorporate general heuristic strategies to evaluate the available information and find an acceptable solution. Rules could be like the following:

1) keep the failure hidden: try to perform local recovery first (do not change the action chain, try to avoid transport jobs to substitute machines).

2) save as much work as possible: first, check whether there is a possibility to resume. If there is none, try restart. Only if restart is not possible either, try compensation and restart.

3) avoid testing: try general interruptability before dependent interruptability.

Some of these objectives contradict each other, as operations can belong to more than one recovery class. For example, for a generally restartable and dependently resumable operation, rule 2 demands resume, which would involve testing, of the operation, whereas rule 3 demands restart to avoid testing. A similar argument can be found for local and global recovery measures.

Figure 3 shows a possible evaluation strategy of the available information with rule 3 having precedence over rule 2. It is based on the classification of the operations into semantic recovery classes[3]. This strategy can be used to look first for a local recovery measure. To cover rule 1, the same strategy can be used to find a global recovery measure if no local measure can be found. However, it must be modified in a way such that the check for resumability drops: a program resume is possible only on the machine on which the failure took place. On another machine, the program can only be restarted[4].

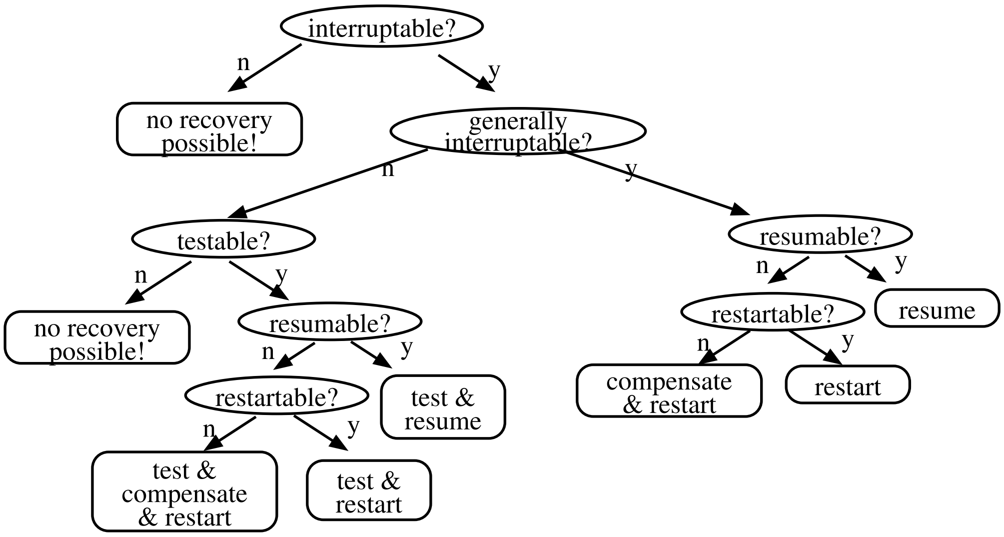


Figure 3 A decision tree for finding recovery measures

Other strategies could be used to control recovery more in detail. Examples could be :

- minimize the number of transport jobs
- do not try recovery for work pieces of a certain type
- do not exceed a given cost or time limit
- try all means of recovery for valuable work pieces

3 For the sake of simplicity, we have left out the additionally required checks for the states of the machine, of the work piece and so on.

4 Restart of a program on another machine does not necessarily mean (re)starting the same program on the new machine, as this machine might have another operating system or understand another programming language. Here, it is sufficient to use a program with semantically identical contents. The information which program performs the desired actions on which machine can be found in the work plan.

7. Implementational Issues and Problems

Experiences from the implementation of a prototypical PRecM show two main obstacles. First, it is hard to find a generally applicable schema for the varying characteristics of all the types of the NC-machines for which the PRecM might be used. When designing a PRecM, one should take care of exploiting as much as possible of the available information.

Second, with the complexity of the FMS and of the operations (represented by programs) to be controlled, the PRecM must be made more complex, too, possibly regarding measures for failures which occur neglectably often, or one must accept that potentially possible recovery measures cannot be found. Here, a happy medium must be found.

For complex operations, we handle the problem of testability in the following way: If there is information provided about the characteristics of the single NC-statements which build the NC-program and if the number of the interrupted statement can be obtained from the NC-machine's control, we know which operation had been interrupted and we can use the Recovery Manager on it. Of course, this requires a careful categorization for the single statements, for example, when deciding whether the whole program can be restarted if it was interrupted at the NC-statement in question.

An example for an especially critical characteristic is resumability. In order to perform correct recovery, one must know the actual machine tools controls behavior when resuming (does it really resume or does it just start the program with the last program statement?). Generally, the semantics of all categories must be thoroughly understood in order to prevent misbehavior during recovery. Additionally, a disadvantegeous style of programming might prevent recoverability for an actually recoverable operation.

8. Examples

Here, we want to illustrate the principles of our architecture and of the PRecM with two examples, one for local, and one for global recovery. The examples show the flow of control between the involved system components. Both cases base on the same action chain and assume a failure occurring at the step which performs the turning operation.

8.1. Local Recovery

Here, we assume a simple failure after which operation can (locally) be resumed.

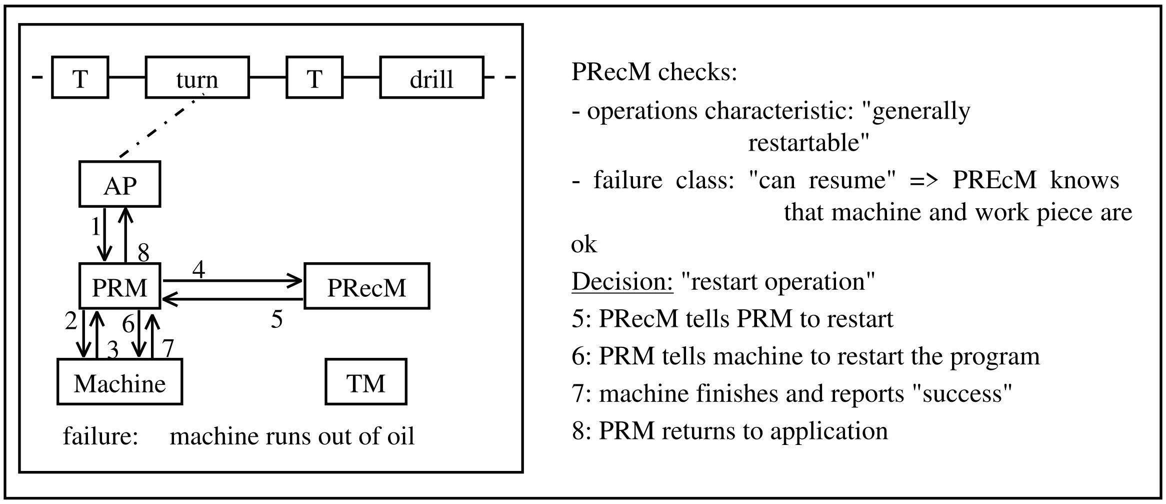


Figure 4 Local Recovery

In figure 4, as well as in figure 5, steps 1-4 show the interactions between the components which result in an activation of the Physical Recovery Manager (PRecM).

1: The application (control system) calls the PRM to perform a turning operation

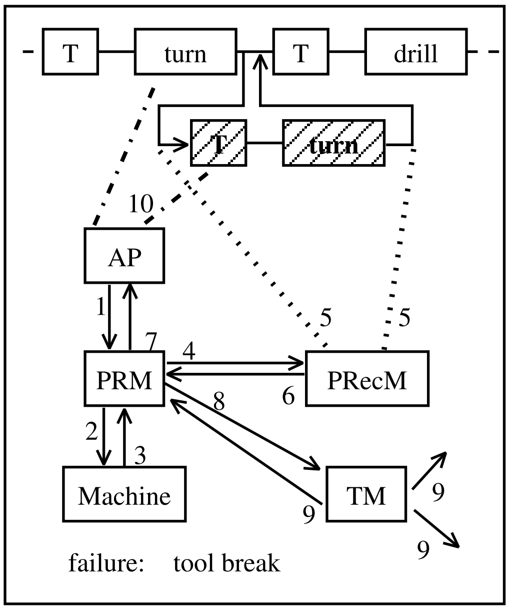
2: The PRM orders the machine to turn

3: The machine fails during turning, stops and returns the failure class (note that it is the machinist’s job to refill the oil).

4: The PRM calls the PRecM, which first tries to perform local recovery

8.2. Global Recovery

To illustrate global recovery, we assume a failure which leaves the work piece intact, but makes further use of the machine impossible. An example for such a failure could be a tool break, which did neither harm machine nor work piece, but there is no substitute tool available for that particular machine, i.e. repair or maintenance of the machine is not possible Here, global recovery must be tried.



PRecM finds:
- operation: generally restartable
- failure: tool break (=> check machine and work piece)
- machine: inoperable, repair not possible
- work piece: ok

<u>Decision:</u> restart on other machine (PRM)

5: PRecM - expands action chain with transport and restart steps

6: - returns "no local recovery" to PRM

7: PRM - makes AP stop processing the current step

8: - asks TM to abort the transaction

9: TM issues "abort"-requests to all involved RMs

10: PRecM lets AP process recovery step in the chain, (an implementation must synchronize this with 7 and 9).

Figure 5 Global Recovery

9. Related work

In transaction processing, there have been many considerations to integrate operations which have effects on the real world (as they are called real actions) into transactions. One of the first proposals can be found in [3]. It suggests that the execution of real actions should be deferred until it is for sure that the nonphysical actions can be performed successfully. However, this requires semantic independence between the physical and the nonphysical operations. A different approach groups operations into semantic classes for forward recovery [8]. This approach served as a starting point for our classification; however, we used classes more appropriate for manufacturing operations and added an additional distinguishing feature: generality versus dependence on the operation’s outcome for the recovery measures.

In the area of manufacturing, we did not find much about systematic error recovery. For autonomous robots, however, several approaches exist (see for example [7],[2]) which mainly rely on exception handling. Here, the robot's programmer must consider each possible kind of failure and provide the right recovery actions.

10. Conclusions

We have shown how transactions can be used in the area of manufacturing to simplify the programmer's job who does no longer have to take care of failure handling. This is made possible by providing a component, the Physical Recovery Manager, which concentrates the algorithmic part of failure handling. Additionally, it allows for flexibly reacting to changes of the system or of general strategies. Further, transactions guarantee data consistency by executing transactions atomically - this minimizes restart times if part of the system or the whole system fails.

The feasability of our approach to existing systems is debatable. Many control systems are not structured clearly enough to be integrated with a TPS for manufacturing operations. Additionally, not many of the existing machines provide the required functionality. For a large FMS, collecting and establishing information for the PRecM might take a lot of time and money. For small FMSs, however, with modern machines and a clearly structured control system, our approach is promising. When developing a Physical Recovery Manager for such a system, it is recommended to start with a simple and robust version which covers the most probable failure reasons and recovery measures and is also tailored to the involved machines' characteristics. Outgoing from the experiences gained from this starting version, one should implement the strategies most suited for the particular system and applications.

Prototypical PRM and PRecM are under development and will show the validity of our considerations. The concept presented here is a first step towards integrating manufacturing processes with transaction processing systems.

References

[1] Bernstein, Phil & Hsu, Meichun & Mann, Bruce: Implementing Recoverable Requests Using Queues. In : Proc. ACM SIGMOD, 1990.

[2] Gini, Maria: Automatic Error Detection and Recovery. In: Rembold, Ulrich (Ed.): Robot Technology and Applications, Marcel Dekker, 1990, pp. 445-483.

[3] Gray, Jim: The Transaction Concept: Virtues and Limitations. In: Proc. 7th Int. Conf. on VLDB, Cannes, 1981.

[4] Gray, Jim & Reuter, Andreas: Transaction Processing - Concepts and Techniques. Morgan Kauffman Publishers, 1992.

[5] Index-Werke: Druckschrift: Störungsbeseitigung Index GB42, GB65, GB100.

[6] ISW (Institut für Steuerungstechnik der Werkzeugmaschinen, Universität Stuttgart): Das allgemeine Referenzmodell für ALSYS, Studie, 1991.

[7] Meijer, G.R. & Hertzberger, L.O.: Exception handling for robot manufacturing process control. In: Proc. 4th CIM Europe Conf., 18-20 May 1988, pp. 213-222.

[8] Pausch, Randy: Adding Input and Output to the Transactional Model. CMU, Ph.D.Thesis, 1988.

[9] Schmidt, Ursula: Transaktionskonzepte in der Fertigung. In: Tagungsband BTW '93, März 1993, Springer.

[10] Wächter, Helmut & Reuter, Andreas: "The ConTract Model". In: A.K. Elmagarmid (Ed.): Transaction Models for Advanced Database Applications. Morgan Kauffmann Publishers, 1991.