To be published in: Proc. PSAM II, March 20 - 25, San Diego, CA. Plenum.

RELIABLE PROCESS EXECUTION IN FMS BY EXTENDED TRANSACTIONS

Ursula Schmidt

University of Stuttgart
Institute for Parallel and Distributed High Performance Systems
Breitwiesenstraße 20 - 22
70565 Stuttgart
Germany

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.

When looking into the manufacturing area we find that there are great - and successful - efforts to improve safety and efficiency of the technology of manufacturing processes. Precision is increasing and machine tools become more and more versatile, fast and precise. However, we did not find a common pattern how failures of the technological process are handled. It lies in the responsibility of the programmers of the control system and the NC-programs to care of possible failures. There is no general model how failures are to be handled. Thus, after failures - failures of the machine or, even worse, of components of the control system - the system must be brought back manually to a safe and consistent state. Valuable time is wasted, machines stand still and the operator(s) has to make a lot of important decisions, possibly based on uncertain information.

Our goal is to provide a programming interface that removes failure handling from the application code of both control system and NC-programs. For this, we want to use the concept of transactions, which is well-known and proven in a lot of computer science applications. So, what are these transactions like?

Classical transactions[1] 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 resp. processes concurrently access and update data and where system breakdowns must not leave the system in an incorrect state. These transactions are groups of operations on databases, which are executed by the transaction processing system (TPS) *atomically*, i.e. either the group is performed as a whole or, in case of failures or a user-initiated rollback, none of the actions leaves any trace in the system, from the user's view. This is performed by undoing all effects of the failed transaction's operations, whichever reason the failure may have.

So, 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 us to extend the application areas of transactions:

- when using transactions, failure handling is kept off the application code which in turn will be considerably simplified and thus be more reliable.
- failure handling can be automated in a much higher degree.
- 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.
- for decisions an operator has to meet in case of urgency situations, he can rely on data which is guaranteed to be correct, i.e. which represents the system state as reported.

FAULT TOLERANCE WITH TRANSACTIONS

Unfortunately, classical transactions cannot simply be applied to processes including operations like manufacturing operations, which have effects on the real world. The point is that the low-level recovery mechanisms for transactions depend on the transactional property of atomicity, which in classical transactions is performed by rolling back data to their original state. This 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. These problems have been identified several times[1,2,4] and there was a suggestion to defer the execution of such a physical operation to the end of the enclosing transaction[1]. However, this assumption cannot be made in our environment as requests issued after the completion of a manufacturing operation might refer to results of this operation. An approach for exception handling and replanning in the field of autonomous robots is presented by Meijer[3].

The improvements expected by the use of transactions motivated us to try a new approach and to relax the hard requirement of atomicity as far as necessary while saving as many advantages as possible. For this, a classical transaction processing system must be extended to an extended transaction processing system, or ETPS, in two ways to allow for application dependent failure handling:

First, the executing components' (*Resource Managers*) behavior must be redefined to cover the characteristics of "real actions". Such a redefined component will be called *Physical Resource Manager*, short PRM. It manages the requests from an application to the NC-machines and differs from regular Resource Managers in performing autonomous recovery and in performing a transaction *reset* instead of a transaction rollback. A *reset* means that all nonphysical operations are rolled back (like a classical resource manager does), but for the "undo" of the physical operation there will be only a reset of the machine and unload of the work piece, for the PRM to be ready to process any new incoming transaction.

Second, the way of failure handling must be covered by a separate component, the *Physical Recovery Manager (PRecM)*. In contrast to classical transaction processing applications, failure handling for processes with direct impacts to the real world is application specific. Thus, the PRecM is the entity responsible for *physical recovery* and has to decide how to continue after a failure of a physical operation, trying to finish processing without doing harm to equipment and environment. Both components will be explained later in more detail.

A component which remains unchanged is the *Transaction Manager*. Its main contribution is the management of distributed transactions, i.e. the coordination of resource managers which perform work on behalf of the same transaction. If one resource manager fails irreparably during the execution of a transaction's request, the remaining resource managers involved must be informed so they can reset or rollback their results for this transaction, too, in order to achieve consistency.

The application per se is no integral part of this architecture. In manufacturing, it corresponds to the control system sending requests to the NC-machines resp. to the PRMs representing the machines. However, it must tell the ETPS, which requests should be performed as transaction, i.e. it must know where to set transaction demarcations. This will be briefly described in the next chapter.

The following figure shows the architecture of an extended transaction processing system, designed to cover the needs of a manufacturing environment.

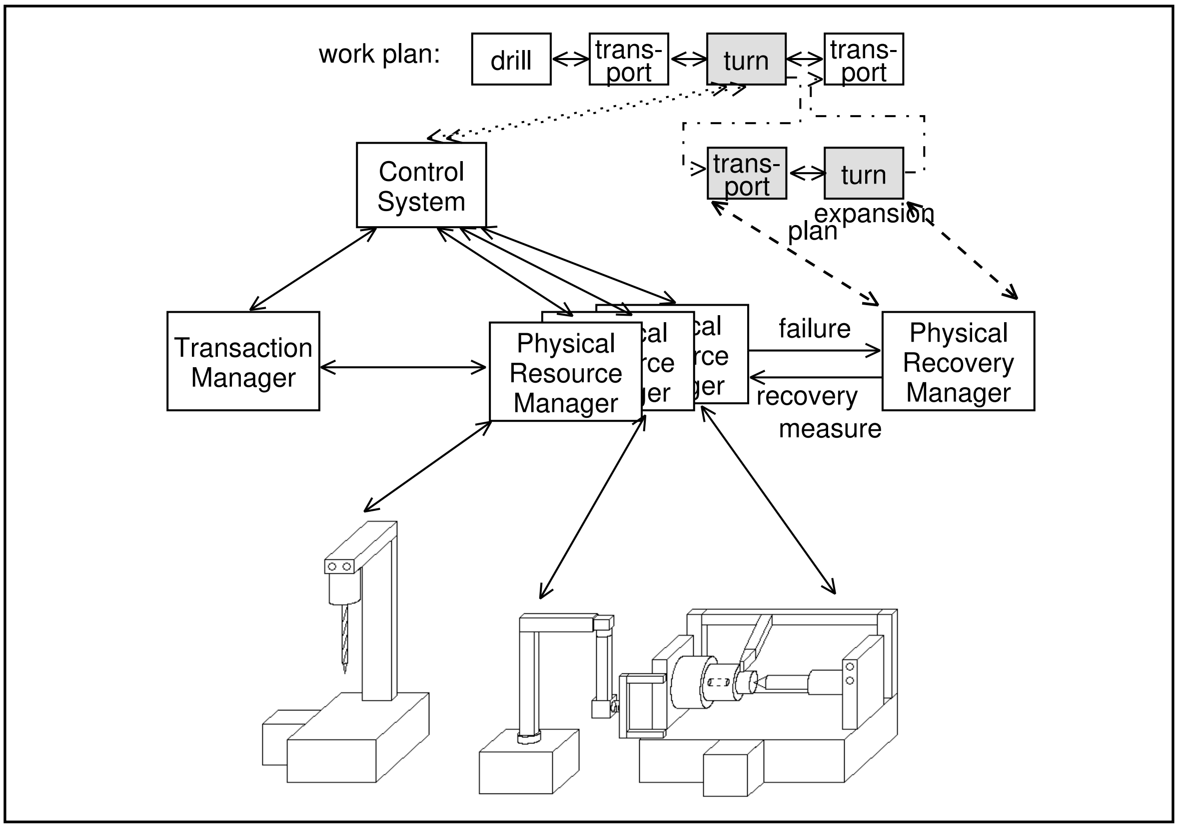


Figure 1. Extended transaction processing system for some sample manufacturing machines.

Here, two questions arise: What does a transaction in manufacturing look like and how are failures handled?

PROGRAMMING TRANSACTIONS

What we want to achieve with transactions in manufacturing is fault-tolerant processing of work pieces. So, where do these transactions come in? In flexible manufacturing systems, a work piece is manufactured by the control system processing the work plan. Besides performing the actual manufacturing operations, preparing and concluding operations like loading programs or providing fixtures are involved. For each physical object in the system - besides work pieces, there are tools, fixtures and pallets to manage - operations must be executed and synchronized with each other. During operation, the ETPS has to manage one separate work plan for each physical object in the system (figure 1 shows just one work plan for one work piece). We assume that the control system, which usually runs on a dedicated computer, creates so-called "action chains" for each object. An action chain is a sequence of steps to be performed for this object, with each step consisting of related operations. We want to have the nonphysical operations and the manufacturing operation, which comprise the step, performed as a logical unit, i.e. as a transaction. The manufacturing operation is determined from the work piece's work plan und will be performed by running the corresponding NC_ program on the specified machine. The related nonphysical operations will be added by the control system. We explain a transaction's characteristics with the sample step in figure 2:

```
STEP for PROG_1:

1:  upload_program_from_server(PROG_1)   /* load program into control computer       */
2:  download_program(PROG_1)             /* load program into NC-controller          */
3:  bind_program(PROG_1)                 /* something like linking                   */
4:  reserve_machine( )                   /* do this before starting the              */
                                         /* manufacturing operation                  */
5:  start_program(PROG_1)                /* start operation and wait for completion  */
6:  read_var(register1, value1)          /* register1 contains the actual            */
                                         /* tool wear caused by this operation       */
7:  read_var(register2, value2)          /* register2: consumption of coolant        */
8:  release_machine( )                   /* now, other programs can start            */
9:  write_to_DB(tool wear, value1)       /* save values of tool wear and coolant     */
10: write_to_DB(coolant, value2)         /* consumption for maintenance module       */
```

Figure 2. Sample step for the turning operation

You might have noticed that there is no evidence that this sample step performs a turning operation - unless you know that PROG_1 is a program performing this action. In fact, as failure handling is performed by the underlying system, the control system processing this step by sending its requests to the appropriate (Physical) Resource Managers does not need to care about the NC-program's contents*.

For the sake of simplicity, we used a pseudo-code notation. In reality, a step is coded using the native interfaces of the resource managers. For requests to databases, this will be SQL, for requests to Physical Resource Managers, we use MMS**. For the same reason, we have neglected issues like choosing an appropriate machine - scheduling should be performed by the control system - or establishing connections to the involved (Physical) Resource Managers. This sample step shows several characteristics of a transaction in manufacturing:

- *It contains exactly one manufacturing operation:* We want to make the transaction's outcome depend on the outcome of this operation. If there is more than one manufacturing operation, which one should be the determining one? Allowing transactions to contain more than one NC-program would make the PRM's failure handling algorithm too complex.
- *It is distributed*: operations are executed in various resource managers. In the sample step are involved: a file server containing NC- programs, an NC_machine performing the manufacturing operations, and a database which is used to store process values for evaluation by the control system's maintenance module.

* It should not do that in regular FMS, too. Otherwise it becomes pretty hard to make such a system flexible at all.

** MMS, Manufacturing Message Specification, is designed to allow for open communication in manufacturing and is the application protocol of the Manufacturing Automation Protocol (MAP) profile.

- *It consists of related requests:* The step's requests relate all to the carrying out of the same manufacturing operation - this is why they must be performed as a unit. This characteristic is used both for processing the work plan and for determining recovery measures: It allows to process steps of work plans like independent units of a modular construction kit. Moreover, only related requests should be composed into a transaction. Including unrelated actions would make the outcome of the whole manufacturing operation unnecessarily depend on irrelevant aspects.
- *It deals with temporary values:* A transaction which, after a program run, wants to read from registers which then contain process dependent values must keep other transactions from starting their programs until these values have been read. Otherwise, the registers might be overwritten during the run of the next program. This is what the "reserve_machine" and "release_machine" requests are for. One might want to solve this problem by not allowing interleaving processing of requests of different transactions. Such a *serialization,* however, would cause unnecessary machine idle times.
- *It is sequential:* That is just like manufacturing is - a work piece can be only at one location at a time. If there is any parallelism, it must be contained in the NC-program. For example, there are machines which can perform manufacturing operations on one work piece in parallel. If operations on different work pieces are to be processed in parallel, the corresponding action chains must be synchronized by the control system.
- And what is most important: *There is no failure handling in the application code* - not even hidden in the NC-program. This is what a transaction processing system does for you.

HANDLING FAILURES

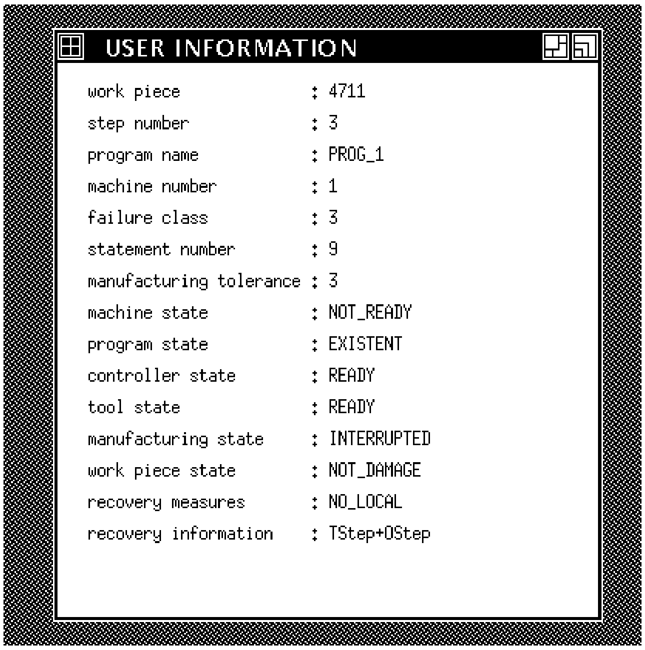
In general, failures are handled by the PRM, which might be supported by the PRecM. The PRM tries to hide failures from the application and to perform recovery on its own.

For a failure during a nonphysical operation on the PRM, in most cases it will simply undo and redo the operations performed so far, except for the physical operation, which will be skipped, if there was one already. Then, the PRM can simply continue processing transactions' requests. In order to be able to perform the proper undo- and redo- actions, the PRM must safely store the corresponding information. Additionally, to provide isolation for concurrent transactions, it keeps locks for the managed objects on the NC-machine's controller. (Of course, the operation of the PRM is more complex than it can be described here. A detailed description can be found in a separate paper[5]).

If the failure occurred during a physical operation, the PRM will call the PRecM for help, together with information about the actual system state. To determine the appropriate actions, the PRecM uses facts and rules: *Facts* can be characteristics of machine, tool, work piece, application, and failed operation. They are known in advance (describing static properties), are reported when the failure occurs, or must be explicitly requested. Special attention must be drawn to the features of physical operations. *Rules* incorporate general heuristic strategies to evaluate the available information and find an acceptable solution. These facts and rules must be given to the PRecM at installation time, but they can be selected or modified during operation as well.

If the failure can be handled locally, i.e. on the failed machine, the PRecM simply tells the PRM what to do next (i.e. to restart or to resume the failed program). Otherwise, it will tell the PRM to *reset* the transaction. Then, the control system will stop processing the current transaction. If other steps can be performed, for example, if there is a substitute machine to continue processing of the work piece, the PRecM extends the particular work piece's work plan (like shown in figure 1), and the control system can continue processing the work piece. If there is no way of failure handling, the PRecM inserts a so-called *abort step* into the work plan, which stops the processing of the work piece[6].

Figure 3 shows the information available to the PRecM and the measures it determined from that information. If it did not receive sufficient information from the calling PRM and could neither get it by interactively interrogating the operator, it assumed, to be on the safe side, the worst case: if, for example, there is no information about the machine's state, the PRecM has to assume that the machine is not operable any longer.



For this instance of failure, local recovery is not possible. Thus, the failed transaction must be reset. However, performing a transport of the work piece to another machine (TStep) and processing the original step (OStep) again will do.

Depending on the encoded rules, more or less strict safety rules can be chosen. It is, for example, possible to restrict failure handling for some actions to certain measures or to require that certain measures have always to be performed.

Figure 3. Recovery measures determined by the Physical Recovery Manager from available information

PROSPECT

A prototypical implementation is currently under development, with PRM-kernel and PRecM already finished. We have been performing functionality tests with a simulation environment and plan to integrate real machine tools, too. We are convinced, that transactions are a mighty means to increase data, and thus process, reliability. Combined with application adaptable failure handling, an operator could be assisted when deciding how to go on with processing a work piece. All in all, a higher degree of automation will be achieved.

REFERENCES

1. J. Gray, The Transaction Concept: Virtues and Limitations, *in*: Proc. 7th Int. Conf. on VLDB, Cannes (1981).
2. S. Jablonski, H. Wedekind and G. Zörntlein, Fehlerbehandlung in Flexiblen Fertigungssystemen (FFS), *Informatik Forsch. Entw.* 3:53-63 (1988).
3. G.R. Meijer. "Autonomous Shopfloor Systems - A study into Exception handling for Robot Control", Ph.D. thesis, University of Amsterdam (1991).
4. R. Pausch. "Adding Input and Output to the Transactional Model, CMU, Ph.D. thesis (1988).
5. U. Schmidt, Transaktionskonzepte in der Fertigung, *in*: Tagungsband Datenbanken in Büro, Technik und Wissenschaft (BTW), 3.-5. März, Braunschweig, S. 104 - 123 (1993).
6. U. Schmidt, Physical Error Recovery for NC-machines. *in*: Proc. 9th CIM-Europe Annual Conference, May 12-14, Amsterdam, pp. 170-179 (1993).