Transactions in Manufacturing Applications

Andreas Reuter
Ursula Schmidt

Stuttgart University
Institute for Parallel and Distributed
High Performance Systems

Abstract

This paper considers the use of transaction processing systems in manufacturing applications. It is part of the ConTract project at Stuttgart University, which explores techniques to extend transaction technology into areas of long-lived, interacting applications. We will focus on two aspects: First, we will describe a transaction-oriented invocation protocol between a control computer and a physical device. Second, we present an extended transaction manager, the ConTract manager, which takes advantage of the features of this invocation protocol to perform recovery for transactions containing real actions.

1 Introduction

In manufacturing there is an interesting discrepancy: The "classical" manufacturing technology, i.e. the procedures for manufacturing work pieces, is very sophisticated. There are extremely durable tools and materials, precision is rapidly improving and the manufacturing machines become faster, more efficient and more versatile. On the other hand, the methods for controlling these machines and manufacturing processes are much less sophisticated, judged by the state of the art in Computer Science. Communication standards for manufacturing systems have only recently been developed and up to now only few implementations exist. They standardize only the form of the requests to be exchanged and do not say much about the behavior of components in error situations.

On the other hand, manufacturing systems need to be integrated into the overall enterprise model, which requires data sharing, a uniform execution model, and means for distributed control. This integration brings about the necessity to deal with long-lived activities and - at the level of manufacturing devices - with real actions. Real actions can have effects on the external world which cannot be undone. We want to show how these processes can be integrated into the existing model of ConTracts, which currently is being developed at our institute. As a base mechanism we introduce a transaction oriented

invocation protocol, which is an extension of MMS (Manufacturing Message Specification), a standard for communication in industrial environments [ISO 9506]. Based on this protocol we propose a mechanism for dealing with various failures, at both the computer and machine device level in a systematic way.

This paper is organized as follows:

After explaining the manufacturing environment in chapter 2, chapter 3 introduces the way we use the transaction concept and its integration into the ConTracts model. Chapter 4 gives an overview of an existing standard for communication in manufacturing, which in chapter 5 will be modified to meet our requirements. In chapter 6 the measures to perform recovery for transactions including real actions are shown.

2 Automated Manufacturing Systems

In this chapter we want to introduce the essential of automated manufacturing systems. Our special interest is on flexible manufacturing, which in the extreme means the "just in time" principle. This principle has two characteristics: Orders are processed as soon as they arrive and the number of work pieces requested by these orders usually is very small, potentially one. Therefore, it must be possible to perform short-term rescheduling of planned operation sequences in order to meet criteria like high throughput or short processing times. This is in contrast to the principle of production lines, which is well known from car production. Here, the steps to be executed and operations to be performed are - smaller variations notwithstanding - known in advance and do not change over time.

In flexible manufacturing, neither the sequences of work pieces are known in advance, nor is the possible number of similar work pieces which have to be produced on specific machines. This results in an event driven processing mode, and it cannot be assured that a given sequence of work pieces on one machine will be preserved on the next machine.

Before describing typical processes, we will introduce some of the standard terminology.

2.1 A Control Hierarchy

In the area of flexible manufacturing, there exist some very frequently used terms (Werner und Kolb, 1988).

A *Flexible Manufacturing Cell* (FMC) consists of several manufacturing machines of the same kind, which are able to replace each other and which are linked by a shared tool and work piece transportation. This requires flexible material flow chaining, flexible work piece supply, and integrated computer control.

A *Flexible Manufacturing System* (FMS) is built from cooperating Flexible Manufacturing Cells, single machines and a central control instance, the control system. This control system runs the software that manages an FMS.

FMSs and FMCs are embedded in a control hierarchy:

The top level is represented by the *Production Planning System* (PPS). It determines deadlines for the customer orders and the production capacities (underlying FMCs),

where the orders shall be processed. The next level is the control level. Here, simulation and planning for the various FMCs takes place. At the cell controller level (which is identical to the control level, if the FMS consists of a single FMC), the scheduling of tools and fixtures is done. This level controls the machines by sending requests to the corresponding machine controllers. At the machine controller level, the control orders are transformed into operations of the physical resources. This is done by sending signals to the mechanical and electrical components representing the actor/sensor level.

Note that from a functional point of view, a numerically controlled manufacturing device consists of two blocks. One block is the machine controller, which can be regarded as some kind of computer, whatever its capabilities may be. The second block contains the actors and sensors.

In summary we can say that *control systems* comprise the control and cell controller level, and a manufacturing device has an internal control hierarchy.

Now we want to have a closer look at the tasks a control system has to perform.

2.2 Functions of a Control System

Besides the interactions with the PPS (receiving orders and reporting results or problems), a control system obviously has to manage and coordinate all actions at the lower hierarchy levels:

- Dispatching the manufacturing tools, fixtures or NC programs, which means downloading of control programs or initiating transport jobs for tools to a specific machine.
- Controlling the manufacturing process and keeping track of the actual processing state.
- Exception handling in case a production machine fails, which in serious cases can lead to a request to the PPS to reschedule all processes.
- Monitoring of the machines to take care of the necessary maintenance.
- General data capturing for statistics and quality control.

Figure 1 shows the actions the control system has to perform to process an order received from the PPS[1]:

[1] For simplicity we use an order consisting of a single processing step. Of course , multiple steps are possible.

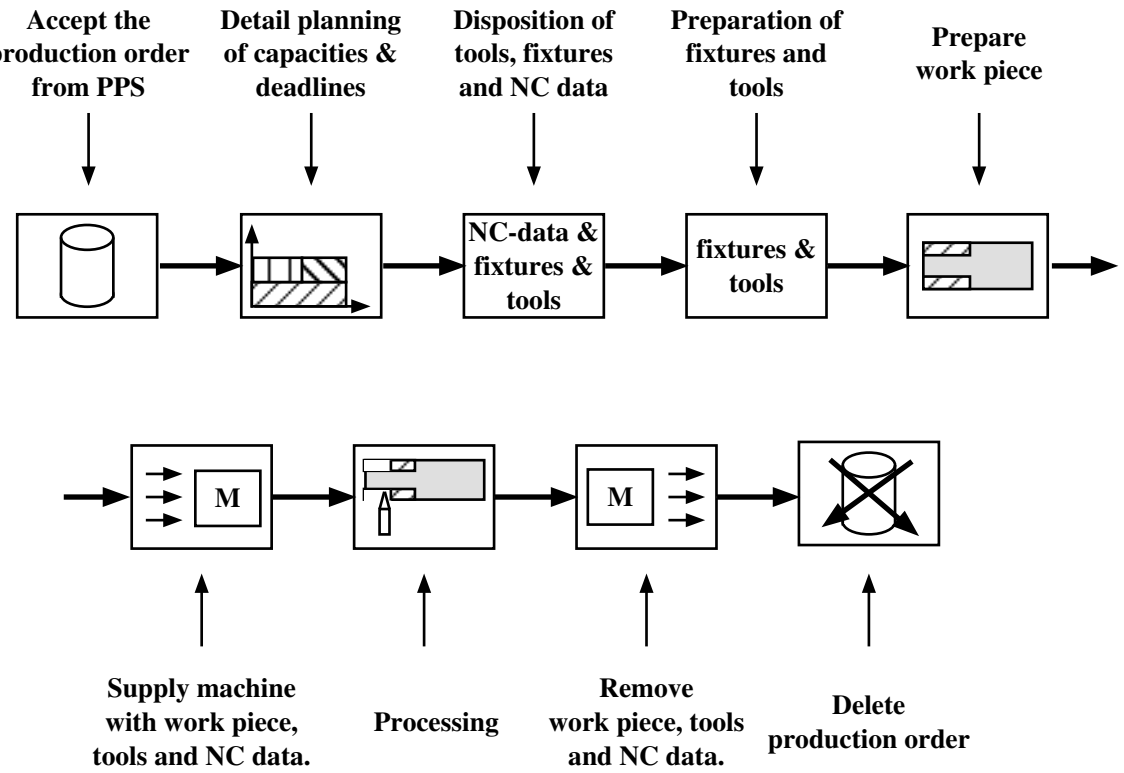

Figure 1: Processing steps of a control system

Preparing a work piece or tool includes the physical transport of the object to its destination.

Let us now consider the behavior of a manufacturing device in case of failures.

2.3 Failures on Machine Level

When manufacturing a work piece or loading NC data, a manufacturing machine can undergo various failures, which have different impacts on the operation. In case of extreme danger, operation is stopped immediately, without regard of possible damages of the machine, tools or work pieces. Other failures merely result in a warning being displayed.

Failure Classes

In case of failures which are not caused by a power failure, the positions of the various mechanical machine components are tested and written to a special "save area". This function shall enable a subsequent machine diagnosis.

Figure 2 shows the different failure levels of a metal-turning lathe [Index]:

class	impacts on			after fault elimination
	num. control	tool progress	spindle	
1	failure display	---	---	(not required)
2	failure display	---	---	start of next program
3	failure display	stop	(deferred) stop	continuing
4	failure display	stop	immediate stop	restart of program

Figure 2: Failure classes of a metal-turning lathe

These classes help determining appropriate measures for failure handling.

Classes 1 and 2 have no influence on the actual processing. Examples for class 1 are an overflow of the synchronization buffer or the blocking of a function key. A class 2 failure could occur when the machine's coolant is running out or the temperature in the control board exceeds a critical value.

A class 3 failure can occur, for example, if the machine's protection hood is not closed. After the failure's elimination, the machine, in principle, can continue where it had stopped. A class 4 failure, however, is a serious one. It results in an emergency stop (for example, if the turning chuck was loose), and after its elimination a check must be performed to ensure that no serious damages had happened. If this check is negative or cannot be performed, the processing must be aborted.

Power Failure

We find another situation when the machine's breakdown is caused by a power failure. On the controller's side, no data will be written to the save area and the processing state of the NC program is lost. On the machine's side, the hydraulics fail and the reference points (which are necessary to perform correct operation) get lost. So, after elimination of the failure, the machine and the actual program have to be initialized anew and restarted.

The only data which is recoverable is that written to a certain area which has an additional power supply, the emergency battery. This data may be machine related like constraints for the allowed number of revolutions or may be variables which are explicitly specified by the user. This property will be used in later sections.

The problem to be investigated in this paper is which kind of low-level control mechanism can be used for structuring applications like these, involving physical operations and controlling manufacturing devices. The next chapter explains the idea of action chains, which are the kernel construct for the execution of this task. Then we show where and how the transaction concept comes in.

3 Action Chains and Transactions

First, let us explain the fundamental differences between a database application, where the transaction concept was originally conceived, and a manufacturing application.

3.1 Basic Considerations

Though manufacturing seems to be a promising application area for transactions, there are differences which prevent us from simply adopting the concepts proven in conventional transaction applications.

Success of Operations: One of the fundamental assumptions in DBMSs is that write operations, which are reported as successfully completed, did in fact complete successfully and, furthermore, were correct. Memory failures are either automatically detected and corrected or the operation fails. In manufacturing, the knowledge of the machine controller about the machine's state and the actual machine state can be incongruent. This may be caused by electrical failures at the interfaces or by an undetected sensor failure. If a sensor reports the wrong distance, for example, this is not easy to detect by checksums.

Type of Operation and Rollback: While in databases the modification of data ***is*** the operation per se, in manufacturing, data serves only as a model of physical events and processes.

In databases, copies of changed data represent their previous values, enabling us to restore the previous state, if necessary.

The state of a physical object, however, may be irrevocably changed by certain operations. This implies that in manufacturing the "rollback" of the data means no problem, but consistency with reality is violated, if the impacts of the associated physical actions are not "rolled back". Because this is in general not possible, all backward directed measures (compensation of physical operations and rollback of associated data) should be avoided or at least handled very carefully.

Interfaces: In databases access to the data is exclusively controlled by the DBMS. In manufacturing, however, we have "open" processes and systems. For example, a human can randomly interfere with the processes, by removing work pieces manually from conveyor belts. We will not consider this aspect.

3.2 Action Chains

Action chains are a construct used in control systems (see for example [ISW, 91]) 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 built 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 work schedule could look as follows:

step	action	capacity entity
1	clamp work piece into fixture	work place
2	mill	machine 1 or machine 2
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:

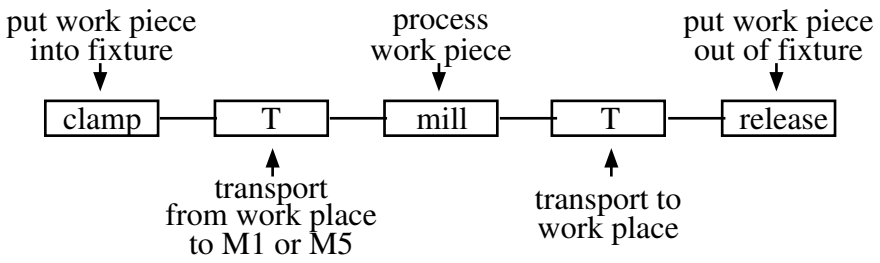

Figure 3: Example of an action chain

An action chain is processed step by step, and the 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 such as sagas [Garcia-Molina and Salem, 87]: 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 is a logical unit and so on. Let us now consider some features of it in more detail.

3.3 Real Actions and Transactions

As mentioned earlier, there exist classes of actions which have impacts on the real world. These actions are called *real actions* [Gray, 81]. We restrict our investigations to the operations of the manufacturing world. We refer to them as *physical operations* or *physical actions*. Some of these physical actions have impacts that can neither be undone nor compensated.

Therefore, we have to reckon with the possibility that transactions, which contain physical operations, may abort irrevocably and cannot be redone. The inverse situation is that incomplete transactions cannot be rolled back.

Execution of Physical Actions

One of the first suggestions to solve the problem was to execute these physical actions after the data operations, actually at commit time [Gray, 81]. But then, one would have to guarantee that these physical actions can always be performed successfully. In manufacturing, this guarantee cannot be given. On the other hand, if the physical action is performed first and then the appropriate data operations (for example on the data representing the system image), a computer failure before committing the data operations leads to inconsistencies: Though the impact of the physical operations is visible in the real world, the system does not know about it. In other words: the results of a not (yet) committed transaction would be partially visible, which contradicts the principle of atomicity.

Now, there remains only one alternative: the real action must be executed as the last operation in a transaction, in fact immediately **before** the commit or abort decision[1]. This decision then depends on the outcome of the physical operation and only on its outcome.

3.4 The ConTract Model

Though at first sight *sagas* seem to be the appropriate means to model the processing of action chains, they do not help us when modeling manufacturing processes which are not strictly sequential. As an example, consider assembly, where parallel manufacturing processes must be executed before starting the actual assembly process.

An extension of the transaction model which allows to describe more general invocation patterns is the ConTract model [Reuter, 89], [Wächter and Reuter, 91]. It provides parallelism, explicit flow control, and a flexible way of handling failures.

For our purposes, each step of an action chain can be modeled by a ConTract step, and the few global data can be kept in the ConTract context. Our approach is to model each of these ConTracts steps by exactly one transaction.

Modeling Parallel Processes

Take as an example a ConTract modeling the concurrent production of two work pieces by parallel execution of two transactions (one for each work piece) and a subsequent transaction for the assembly. If one of the two components cannot be produced, the following assembly cannot be done or, if the assembly fails, the two work pieces do still exist.

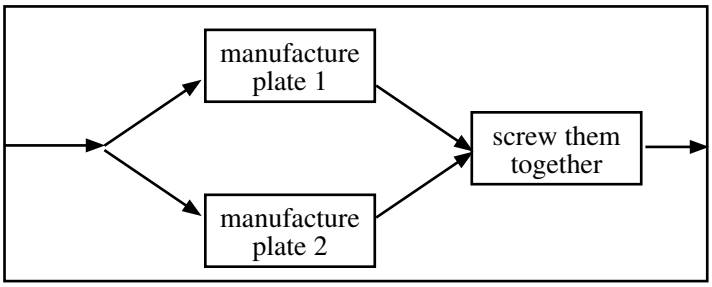

Figure 4: An example ConTract

A ConTract consists of steps connected by a control flow graph. In our application domain, at least one step will perform physical actions as described above. Now what does that mean for the definition of a ConTract, which says that it has to undergo compensation if one of its steps aborts?

Recovery

During regular processing, the transaction manager will keep track of the operations. In case of failures, it usually informs the ConTract manager which then performs the recovery measures as they are specified in the script. However, the failure handling of a step containing (or consisting purely of) physical actions depends on the actual situation. Therefore, the otherwise static script has to be modified dynamically. For that

[1] Please note that for this type of transaction an abort is quite different from a regular abort as it is known from ACID transactions. Here, an abort leaves the transaction in a terminated state without any possibility of resuming or reexecuting.

purpose we introduce a recovery manager. Figure 5 shows the interactions of the involved managers:

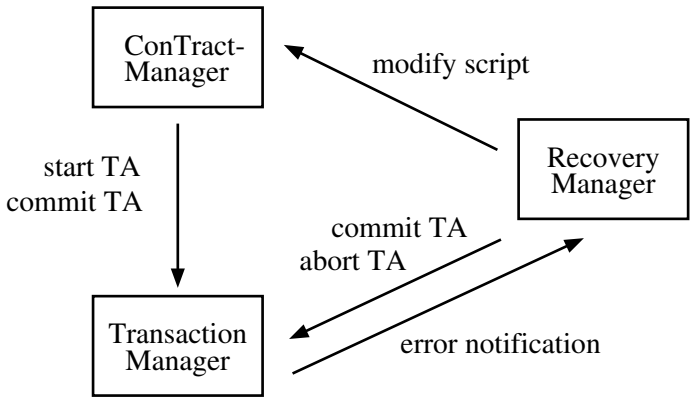

Figure 5: Interactions with the recovery manager

The transaction manager provides low-level transaction services. It notifies the recovery manager of failures involving physical actions. The recovery manager decides how to recover from these failures and initializes the modification of the ConTract script. The ConTract manager executes this modified script to perform the recovery.

In the following, we want to show how a recovery manager can be constructed which determines the proper recovery measures in case of failures. As basis, we expound a communication protocol, which is built upon an existing communication standard, ensuring a reliable communication and protection against most network and computer failures.

The next chapter will sketch the principles of the underlying communication standard, which has to be modified to meet the purposes sketched above.

4 A Short Description of MMS

In this section we want to give a brief overview of the principles of MMS [ISO 9506-1], [ISO 9506-2]. MMS is an application layer protocol of MAP (Manufacturing Automation Protocol), which was initiated in the early eighties to allow standardized communication in industrial environments based on the ISO/OSI reference model. MMS provides its users with standardized interfaces for communication, describes the behavior of manufacturing devices and has a client-server structure. The *MMS provider* implements the whole underlying protocol stack, whose services can be accessed only by the defined services. This interface is called the service access point (SAP).

4.1 Client-Server Relationship

A MMS service request has the following form (Figure 6): The requestor (or client) sends the request to the server, which is to perform the desired operations. From the view of MMS both server and client are applications. The server is notified by its

provider of the request by a service indication, performs the necessary actions and sends the result back to the client[1]. The result can be negative, if it was not possible to perform the request (for example, if a specified resource was not found). The application program has to take care of this.

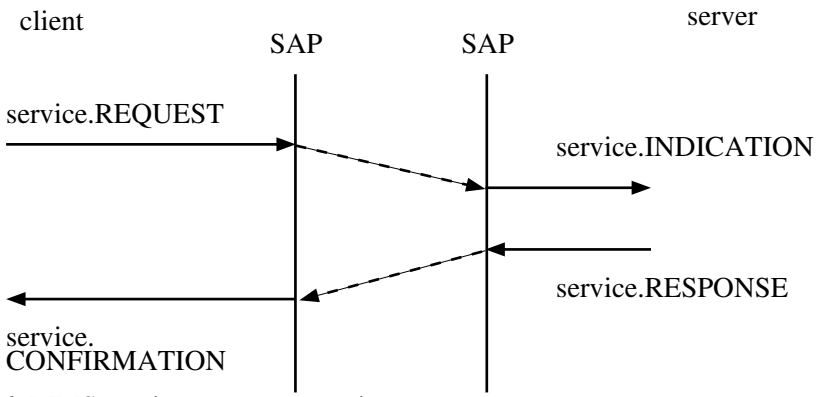

Figure 6: MMS service request execution

4.2 The Virtual Manufacturing Device

MMS only describes the features of the server. The client's behavior results from the use of the defined services. For the server, objects and operations on these objects (the services) are defined. Objects and operations together describe the Virtual Manufacturing Device, the VMD, which abstracts from a real device. This means that a real device can be accessed only via the VMD. The physical implementation is left to the vendor. This includes aspects like exception handling or behavior in case of network failures.

MMS offers eight service groups, which are shown in Figure 7 and refer to 16 object types. Besides the VMD the object types "domain" and "program invocation" have special importance. Domains represent a set of resources within a VMD, while a program invocation corresponds to an executable program, possibly consisting of several domains like operating system and subprograms. A program invocation can be associated with various states, for example "running" or "stopped".

[1] This is the description of a confirmed service. In MMS, three unconfirmed services exists, which can be neglected in our context.

.

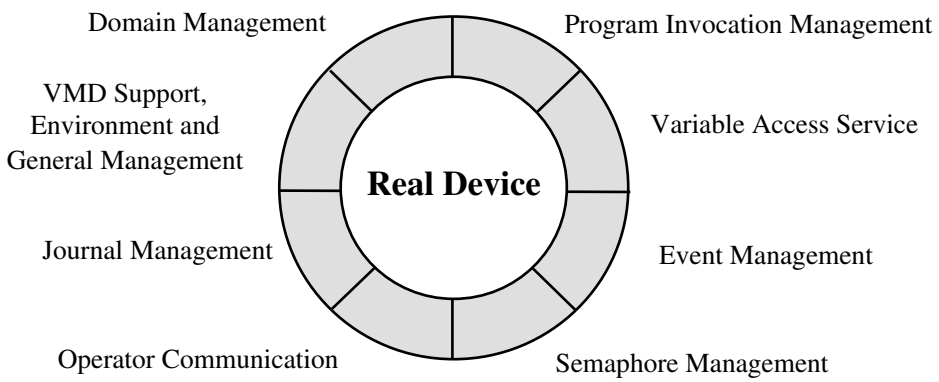

Figure 7: MMS service groups

4.3 Associations

MMS is a connection oriented protocol: Its services can be processed only in reference to an explicit binding between two MMS applications, which is called an association.

When establishing an association, client and server negotiate an application context: the client suggests an abstract syntax (machine independent data type description) and one or more transfer syntaxes (unique representation of data values for transmission). The server accepts possibly the abstract syntax and chooses a particular transfer syntax (if not, a negative result will be transmitted to the client). Abstract and transfer syntax result in the presentation context. When establishing the association, the supported object types and operations are negotiated (not every VMD has the same features and capabilities).

4.4 Physical Actions in MMS

In MMS, physical actions can be performed only by running a program invocation, i.e. executing an NC program, which controls the operations on the manufacturing device. A program invocation is started by a client calling the "RunProgramInvocation" service. If the start was successful, the client will receive a positive result. But this result refers to the successful start of the program and not to its successful completion. If the client wants to know when the program stops, it can specify a certain parameter when creating the program invocation. Thereupon, the server will inform the client of the termination of the program invocation, but not of the type of the termination. The client has to find out whether the program completed successfully or was killed respectively stopped by another MMS request.

5 Fault Tolerance in MMS

In this chapter we show how to modify MMS such that both safe communication and well defined processing results in error situations can be achieved. Before explaining these modifications, we first list the necessary requirements and assumptions.

5.1 Requirements

As stated earlier, MMS describes only the behavior of the VMD in the absence of failures. To enhance its capabilities, some additional properties of the peers and the network must be assumed.

Loss of Connection

We assume that the communication service provider informs the participants in case the connection between them is lost. A connection breaks if either the underlying network or one (or both) of the peers fail. If the loss is due to a network failure, both parties shall be informed, a loss due to the failure of one peer causes the other one to be informed. This functionality is part of the ISO/OSI reference model and can be taken for granted.

Stable Storage

To achieve tolerance against failures, some data has to survive these failures. For example, incoming requests must be queued, and the actual processing state of the requests from the various associations has to be recoverable. Therefore, both client and server need some stable storage to implement a log, which could be realized by using the buffered storage area of a device controller as described in section 2.3.

Recoverable Program State

Since most physical actions cannot be undone, failure recovery must be forward directed. For the client's side we assume, that the state of a program is recoverable.

However, no general statements are possible about the behavior of a server in case its controller breaks down. Consider the case when a program invocation had been created. Will it still exist after the server has recovered from failure? It is not realistic to demand a VMD to perform persistent actions. But as long as no physical actions are involved, the server in principle can perform "atomic" actions in the sense that it either performs the request successfully or sends a negative result back to the client. The VMD's implementor is responsible for the realization of this feature. An example is described in [Sperling, 90].

Duplicate Elimination

Our model relies on the ability of both client and server to recognize duplicate requests. Therefore we need a mechanism for allocating request identifiers and a duplicate detection mechanism. Because these mechanisms are not new and can be found, for example, in the lower layers of the network model, we simply assume their existence.

5.2 The Enhanced MMS Protocol

The actions which have to be performed by the client and the server are shown in Figure 8. The modifications to standard MMS are mainly write operations to stable storage.

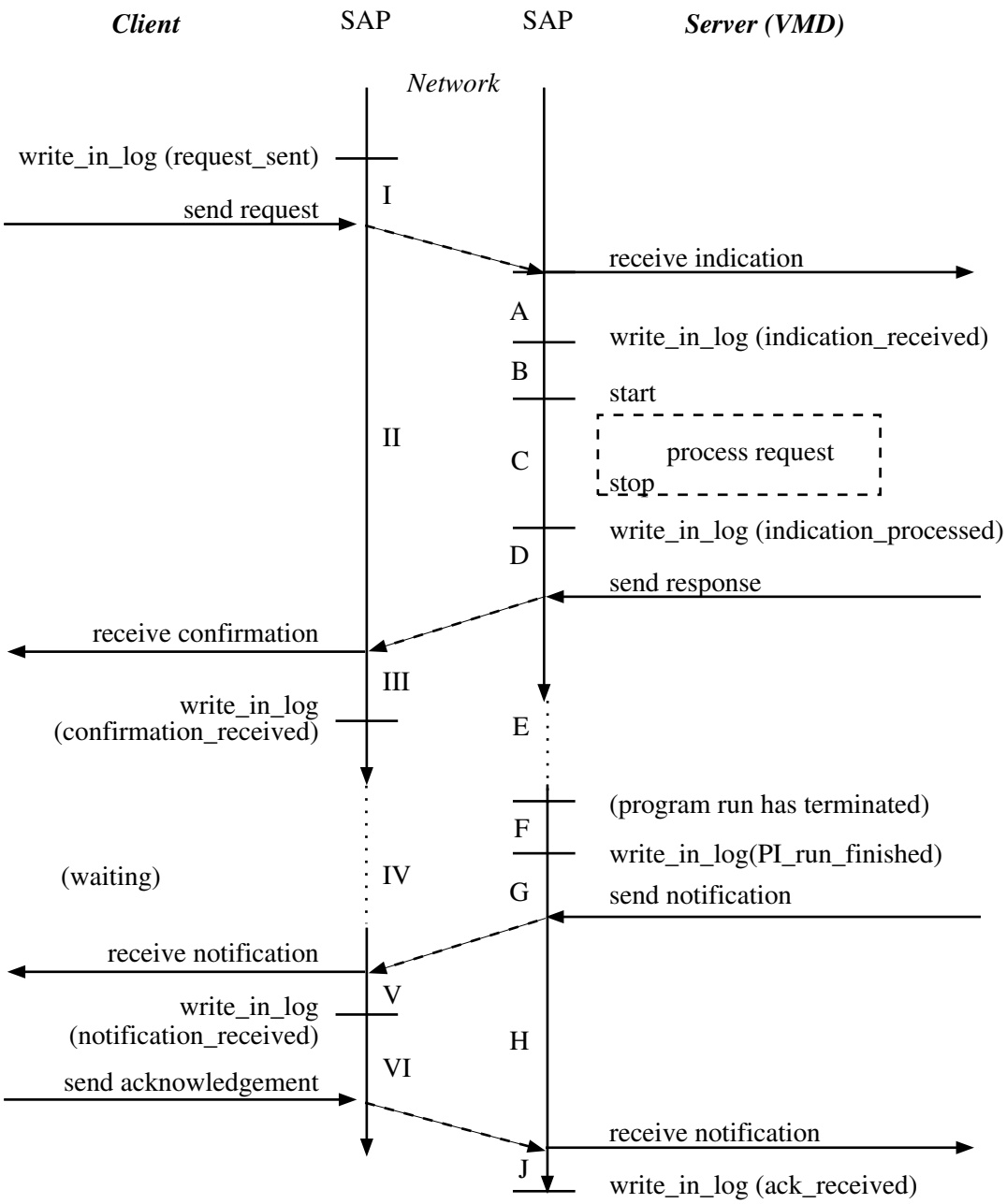

Figure 6: Enhanced MMS service request execution

Each action or event will be logged.

The log entries consist of

- the type of request (for example "LoadDomain", "StartProgramInvocation",..),
- the state of processing ("indication_received", "response_sent", "processing_started",...),
- its parameters (names, identifiers),
- the request identifier (for duplicate elimination),
- the association (to keep track of the corresponding communication partner and the negotiated type of connection).

To keep things simple, Figure 6 showed only the request's processing state, leaving out the contents of the log entries. They are used to determine the actual phase of the request's processing. In the various cases of failure, these log entries will be used according to the following description:

Behavior of the Client

Independent of the type of failure, i.e. whether the client failed itself or is informed about a loss of connection (LOC). after recovery respectively reestablishment of the association the client will perform the following actions:

If the failure occured during phases I, II or III, the client will send the request again. This covers the case that the client does not know whether it has sent the request already as well as the case that the server has not yet received the indication. If the server had already received the indication, it recognizes it as a duplicate and ignores it.

Did the failure occur in time interval IV resp. V, the client undertakes no additional actions: it will wait for a message from the server (the server will be informed about the client's failure in form of an LOC during time intervals G, H or J). After a failure in time interval VI the acknowledgement will be sent again.

Behavior of the Server

Phases B and C notwithstanding, the actions of the server are independent of the failure type as well.

In Phase A, the server does not (yet) know that it had received an indication. Therefore, it takes note of the failure and waits for a request from the client. (If the server or the network fails in phase A, the client will be notified of this LOC in time phase II and,will send the request again). After a failure in phases E or F, it will send the response again. After a failure in phases G, H or J, it will send the notification again.

In phases B or C, the actions of the server depend on the failure type. If the server is notified of an LOC during phases B or C, it will simply continue with it's processing. During this phase it does not need interaction with the client.

Now consider what happens if the server fails during phases B or C. After recovery, the server finds that it has received an indication for a request and possibly started processing, but does not know, whether it has already finished or not. But, physical operations notwithstanding, the processing of requests can be considered "atomic". This means in MMS context that operations are either successfully completed or, in case of a failure, all partially results will be undone (ISO 9506-2). So, if the processing has already started, no effects of the failed processing are visible. That is why the server, after fail-

ure elimination, can continue by starting (possibly restarting) the processing of these request.

There is another situation, if the request was a "StartProgramInvocation". If the failure there was caused by a power failure and further processing is possible at all, the program has to be restarted and the machine must be initialized (see section 2.3). So recovery is not possible, the processing is aborted and the server will be reset. In case of all other failures, the server will send the machine's state and possibly the failure class as a parameter of the negative result response to the client.

So, the client will always receive a response, be it positive or negative. If this result is negative and caused by a machine failure, the client in turn has to inform the recovery manager about the failure.

There is one case we have not explicitly mentioned: This is the case that the server fails before it has logged the termination of the program invocation, i.e. during time interval F. However, if the failure was caused by a power failure, the processing anyway cannot be continued. In all other cases, after failure elimination, the server has to test the program invocation state, whether it has terminated or not. If the program invocation has terminated, the server will continue with writing this to stable storage, otherwise it will wait for the program invocation to terminate.

5.3 Remarks

One could wonder why we are using this approach to modify MMS, although there are services and service elements in the ISO/OSI reference model providing similar features. Take as an example the synchronization services offered by the presentation layer. The answer is simple: MMS does not use them.

An analysis that is beyond the scope of this paper also showed that services like RTSE (Reliable Transfer Service Element) and ROSE (Remote Operation Service Element) cannot feasibly combined with MMS, so we chose the approach outlined above.

In the long run, however, use of ISO/OSI standards should be achieved. Currently two application layer service elements, CCR (Commitment, Concurrency and Recovery) and TP (Transaction Processing), are being developed. In combination with MMS they possibly might provide the features as described in chapter 5. See [Bever, 89] or [Rose, 89] on how to combine several service elements.

6 Recovery Measures for Physical Action Transactions

This chapter shows which measures the ConTract manager has to perform when a failure occurs. Failures of the client are not very hard to handle, for it does not perform real actions. The interesting case is a server failure during the processing of an MMS request. The client's transaction manager will invoke the recovery manager to modify the script. (see Figure 5). The ConTract manager must try to bring the transaction to a successful end. If this is not possible, the user will be informed and is requested to cancel the ConTract.

The appropriate measures depend on:

- the kind of failure and machine state
- the type of the operation, when the failure occured
- the system state (availability and actual load of the machines)
- external factors like scheduling policies, cost strategies, and so on.

Before explaining possible strategies for recovery, we have to distinguish the impacts and manageability of the various operations. They will be categorized in different classes.

6.1 Taxonomy of Physical Operations

Our taxonomy is a variation of the one established by Randy Pausch [Pausch, 88]:

An action is **idempotent**, if "it can be executed multiple times with no additional effect". Examples for this type are all operations, which are kind of "absolute". For example, the operation "move robot arm to position xyz" is idempotent, while the operation "move it 10 inches up" is not.

An action is **compensable**, if another operation can be specified, "that undoes the effects of the original operation".

These definitions do not consider the fact that physical actions are not atomic, but rather can be interrupted during their execution. We assume additionally that interrupts do not change these features, i.e. an idempotent operation, which was started, but not yet completed, can be redone without additional effects. For compensability we assume that the compensation of an interrupted action has the same effect as the compensation of a completed action.

The following table shows some examples:

class	compensable	idempotent	examples
A	Y	Y	transportation jobs, movement of a robot arm
B	Y	N	screwing
C	N	Y	drilling a hole (in most cases)
D	N	N	welding or cutting an plate

This classification is not complete and can vary, depending on external conditions. For example, drilling a hole could be considered idempotent: if restarting a drilling process, the first processing phase will be "drilling the air". However, as the drill-tool is vibrating, the part of the hole which is already drilled will be widened a little bit. So, if high precision is required, a drill operation may not be idempotent.

Now we need a last criterion, testability.

An action is **testable**, if "the transaction processing system can determine, if the operation occured". Determining this feature is tricky, because it depends on what the system can determine and what not. Of course, each action can somehow be made testable, for example by using sophisticated diagnosis systems, asking a machine operator of starting extended measurement procedures. This in turn depends on the efforts one is willing to spend on testability. If the part to be manufactured is very cheap, it may be more effective to throw it away and start with a new one than to perform complicated tests and checks.

The strategy we suggest is to avoid testing if possible and to prefer forward recovery by means of redo and compensation.

6.2 Behavior of the Recovery Manager

We can now establish rules for the recovery manager to coordinate recovery of a manufacturing system. Again, we have to consider the request "StartProgram-Invocation", for this is the only request causing real actions, some of which have irrevocable impacts. All other operations in MMS are either idempotent (for example "Read-Variable") or compensable, depending on implementation aspects (for example "LoadDomain-Content"). They are handled internally by the server. The only failures the ConTract manager is notified of are those occurring during the run of a program invocation. As was shown above, the sequence of actions to be taken during error recovery depend on the properties of the physical action(s) involved and on the impact they have had on their environment. So we assume the recovery manager has access to the following set of information:

- The type and parameters of the operation the server was performing when it failed.
- Compensating action or actions, if there are any[1].
- Class of the machine failure (if the failure class is not known, we have to assume the worst case, i.e. class 4).

For failures of class 3, the recovery measure is simply sending a "ResumeProgramInvocation" request to the server, for the program run can be continued, as soon as the machine failure has been eliminated.

For failures of class 4, however, a security check must be performed. If there are no damages, the client performs the following measures, depending on the operation types.

- Idempotent operations are executed again.
- Operations, which are not idempotent but compensable are compensated and then redone.

For both types of operations it is not necessary to test if the physical action was already completed. However, we find another situation for the remaining operations:

- Operations, which are neither idempotent nor compensable can possibly be continued. This continuation must be initiated and monitored manually, for it is not possible to provide continuation measures for every possible exception case.

[1] For the moment, we do not consider where the classification of the operations and the determination of the related compensating actions comes from and who does it. We simply assume its existence.

Figure 9 illustrates these rules.

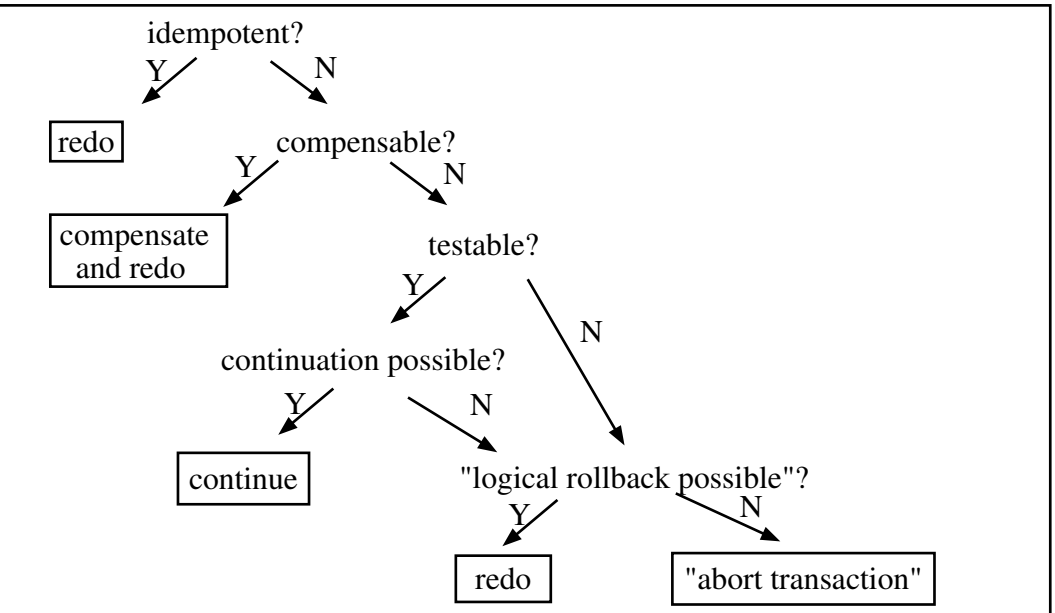

Figure 9: Recovery Measures

What remains to be explained is the case of "logical rollback" and how to abort a transaction.

Logical Rollback

Though the effects of various physical operations on a work piece cannot be compensated, from a logical point of view a compensation can be achieved: The work piece will be replaced by a new one and all physical operations will be repeated on the new object. This strategy is not always advised. If the actual step was one of the last in a very long processing sequence, compensating all "data" operations which were performed up to then can be very expensive. It could be more effective to set the actual ConTract in an defined "aborted" end state and start a new, identical one.

More Extensive Recovery Measures

There is one case we have not yet considered. It is the case that the machine is seriously damaged and stops working for good. The work piece then has to be transported to an alternative machine, where the processing can be continued using the recovery measures as shown in figure 8. The recovery for this case additionally includes a transport job. Note, that in the extreme this could result in a ConTract being continued on another machine and another work piece, without the user (control system) noticing it.

Aborting and Suspending a Transaction

The abort of a transaction can become necessary, if a physical action did fail irrevocably. The suspension of a transaction can become necessary, if during the preparing operations for a manufacturing operation a high priority order has to preempt the actual

process from the server. In case of a suspension the physical operation has not yet started.

In both cases the server should be left in a "cleaned up" state, enabling further processing of other requests.

Let us consider a simple example transaction, consisting of a sequence of MMS requests[1]:

LoadDomainContent(A)

LoadDomainContent(B)

CreateProgramInvocation(PI_from_A_and_B)

StartProgramInvocation(PI)

This could be the loading and starting of a NC program operating on a physical object.

The first three operations are operations on data and can be compensated. This should take place, whether the physical operation, initiated by the "StartProgramInvocation" request, was performed or not. So the Contract Manager has to initiate the compensating actions for each of the performed data operations. Besides, if the transaction had to be aborted, because the physical operation could not be performed or completed, this fact must be notified somewhere, possibly for further investigations or statistics.

6.3 Remarks

In this paper we assumed that the interrupt of a physical operation has no influence on its idempotence or compensability. Of course, there are operations for which this assumption is not valid. Additionally, an investigation of interruptability or a modified definition of the term idempotence is needed.

Another difficulty arises from the fact that quite often various sequential operations are executed within a single NC program. Take as an example a program, which executes the drilling of different holes in a work piece and performes the necessary tool changes. Our taxonomy treats these processes as a single operation, but how shall recovery be performed on them? It could be hard to determine whether these operations as a whole are idempotent or how they could be compensated.

7 Summary and Outlook

In this paper we have modified an existing communication protocol for manufacturing devices such that transaction-like means for fault tolerance and synchronization can be supported. Failures at machine level will be handled by the ConTract manager, which uses information about failure and operation classes for forward recovery.

What remains to be done is a detailed investigation of the terms "idempotence", "compensability" and "testability" and how they are affected by technological and economical constraints.

[1] The request LoadDomainContent is a request of MMS Ease (an enhanced MMS implementation), which causes a lot of requests and data to be exchanged between the server and a third party. From the user's view, however, this service has atomic properties.

At the modeling level, more knowledge is needed about the interdependencies of the various processes that belong to a specific production order. These interdependencies might have considerable impact on the appropriate way of modeling of production processes.

8 References

Bever, Martin: OSI Application Layer - Entwicklungsstand und Tendenzen. In: Tutorium Kommunikation in verteilten Systemen, 20. - 21.2.89, Universität Stuttgart.

Garcia-Molina, Hector and Salem, Kenneth: Sagas. In: Proc. ACM SIGMOD, San Francisco, 1987.

Gray, Jim: The Transaction Concept: Virtues and Limitations. In: Proceedings of the 7th International Conference on VLDB, Cannes, 1981.

Index-Werke: Druckschrift: Störungsbeseitigung Index GB42, GB65, GB100.

ISO 9506: Manufacturing Message Specification (MMS). Part I: Service Definition

ISO 9506: Manufacturing Message Specification (MMS). Part II: Protocol Definition

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

Pausch, Randy: Adding Input and Output to the Transactional Model. PhD Thesis, Carnegie Mellon University, 1988.

Reuter, Andreas: ConTracts: A Means for Extending Control Beyond Transaction Boundaries. In: Proc. 3rd Int. Workshop on High Performance Transaction Systems, Asilomar, 1989.

Rose, Marshall T.: The ISO Development Environment: User's Manual. The Wollongong Group, 5.0 edition, 1989.

Sperling, Wolfgang: Untersuchung der Leistungsfähigkeit von MMS (ISO 9506) für den Entwurf verteilter Fertigungssysteme und Entwicklung eines integrierten MMS-Diensterbringers in einer numerischen Steuerung. Diplomarbeit, Universität Stuttgart, Februar 1990.

Wächter, Helmut and Reuter, Andreas: The ConTract Model. In: Elmagarmid, A.K.: Transaction Models for Advanced Database Applications, Morgan Kaufmann Publishers, 1991.

Werner und Kolb: Flexible Fertigungssysteme in der Praxis. Anwendersymposium "FFS in der Praxis", Berlin, 1.-2.12.1988.