Data Management in Control Applications

A Proposal Based on Active Database Systems

© **Esa Falkenroth**

LINKÖPINGS UNIVERSITET

ISBN 91-7871-868-6 ISSN 0280-7971
PRINTED IN LINKÖPING, SWEDEN
BY LINKÖPING UNIVERSITY

To

My Beloved Wife

Anette

Abstract

ACTIVE DATABASE MANAGEMENT SYSTEMS can provide general solutions to data management problems in control applications. This thesis describes how traditional control algorithms and high-level operations in a control system can be combined by using an embedded active object-relational database management system as middleware. The embedded database stores information about the controlled environment and machinery. The control algorithms execute on a separate real-time server. Active rules in the database are used to interface the model of the environment, as stored in the database, and the control algorithms. To improve information access, the control system is tightly integrated with the database query processor.

A control-application language specifies high-level manufacturing operations which are compiled into queries and active rules in the database. The thesis describes how the generated active rules can be organized to solve problems with rule interaction, rule cycles, and cascading rule triggering. Efficiency issues are addressed. The thesis also presents practical experience of building the integrated control system and the requirements such systems put on an embedded ADBMS.

Preface

THE WORK presented in this thesis was conducted at the Department of Computer and Information Science at Linköping University. It is a part of the EERTS project (extending embeddable real-time systems), funded by NUTEK[1]. The motivation for the project was an interest in complex control applications that require database-like functionality. The research objectives were:

- Research in architectures, modeling and representation of complex flexible manufacturing systems with the focus on real-time data management.
- Extending database technology to support the realization of complex manufacturing control systems.
- Development of the ARAMIS[2] concept by investigating how the active and real-time database functionality can provide data management in control applications.

Among the issues raised by the project, this thesis focuses on the use of embedded active database systems in control systems.

1. Swedish National Board for Industrial and Technical Development.
2. ARAMIS was a joint project of ABB and Linköping University.

In particular , the thesis studies how active rules can support the integration of databases and control system. During the last two years this research area has become increasingly popular . In 1993, when this project was started, scarcely any work had been reported on the integration of active databases and control systems. T oday, several research groups are attempting to use this new technology in control applications.

This thesis should provide a valuable reading to people in the process of using active database systems. There is much to gain: databases in control applications can increase productivity , information access, consistency of information, ease of programming, and flexibility .

Parts of the thesis are based on a report published in the proceedings of the Second International Conference on Applications of Databases [Fal95].

Acknowledgements

First, I want to thank my supervisors: Dr. Anders Törne, Prof. Tore Risch, and Dr. Staffan Bonnier. I am very happy to be part of the RTSLAB team. Thanks for all the vivid discussions. I am also in debt to the members of EDSLAB who spent many hours teaching me the mystic internals of the AMOS database system. Anne Eskilsson, Ivan Rankin, and Lillemor Wallgren deserve many thanks: — Thank you!

Finally, I would like to express my gratitude to NUTEK, the Swedish National Board for Industrial and Technical Development, for the financial support. The local Linköping University funding from CENIIT, The Center for Industrial Information Technology is also gratefully acknowledged.

Esa Falkenroth

Contents

Preface

Acknowledgements

13 Introduction

25 Background

39 Related Work

49 Architecture & Language

75 Rules in Control

91 Requirements

95 Thesis summary

99 Future Work

101 References

Chapter I
Introduction

FUTURE COMPUTER-BASED CONTROL SYSTEMS will operate in complex environments that involve large volumes of data. This thesis studies how active database management systems (ADBMSs) can be used to handle the information in such systems. Previous approaches to data management have been based on data repositories or communication interfaces. To avoid concurrency problems, the access to shared data has been statically scheduled which becomes inflexible in large control systems with complex data dependencies.

Active database management systems (ADBMSs) provide new solutions to the data management problems of control applications. This thesis investigates how the information about the controlled environment and the manufacturing operations can be stored in an embedded ADBMS. A database-centered control system architecture is proposed where the active rules facility in the database integrates the embedded database with the control system. In particular, this work presents a method for organizing active rules in a manner that avoids problems with rule interference, rule cycles, and cascading rule triggering.

The goal of this work has been to make system development more flexible while retaining adequate performance. The goal has not been to improve the performance. Although efficient indexing methods in the database speed up data access, a slight decrease in the total performance is expected. More enticing is the general solution to many data management and complexity problems. A modern object-relational database management system with an active rule facility provides:

- A powerful query language
- A modeling language to organize data in high-level object-oriented models
- Efficient access to data using query optimizers and indexing techniques
- Persistence of data between executions
- Automatic maintenance of data integrity
- Ensuring safe concurrent access to data
- Separating the logical view of data from the physical organization
- Efficient change monitoring techniques using active rules

This chapter justifies the need for active and object-oriented databases in manufacturing control systems. The chapter starts with a very brief overview of the proposed database-centered control system architecture. In the following section, a review of previous techniques for information sharing in control systems is given. The last section presents the organization of the present thesis.

1.1 Database-centered architecture

This thesis addresses data management in control applications. The proposed approach is to embed an active object-relational database into a control system architecture, where the embedded database stores an abstract model of the controlled environment.

A control application language, tightly integrated with the query processor , specifies high-level manufacturing operations. The language operates on the abstract model stored in the database. Changes in the model trigger control algorithms in a separate real-time server , which in turn does the actual control of the physical environment. In this way our architecture combines cyclic control algorithms and an event-driven operation language with data management capabilities. The integrated architecture makes extensive use of the active rule facilities in the database, both in the execution of the control application language and to initiate appropriate control algorithms in the real-time kernel.

A detailed description of the architecture can be found in Chapter IV.

1.2 History of information sharing

Information sharing in control applications has developed rapidly during the last twenty years. Initially , traditional control systems were based on automatic control theory . These systems operated at an algorithmic level close to the controlled process [Åst84]. The data management problems were few since the data was local, *i.e*., it was not used outside the local control algorithm.

In the 70s, the introduction of programmable logic controllers (PLCs) allowed the control of more complicated processes to be automated [Årz91]. The introduction of powerful, general-purpose computers brought new possibilities to the exchange of information between systems.

At that time, the sharing of information was limited by the available communication technologies [Bar89]. Sharing information with more than one piece of equipment required, at best, multiple physical links and interfaces; at worst, manual transfer of information. This placed severe restrictions on the applications.

There were problems in exchanging large volumes of data and the systems could not guarantee the consistency of the information. Eventually, data repositories were being used for shared information, sensor values, process data, and interprocess communication. While this eliminated the need for multiple interfaces, it accentuated other problems of information access. When these applications grew in size and complexity, the sharing of information became difficult [Gra92].

The use of powerful computers had made it easy to share information between geographically distributed plants, but the issues of data integrity in data repository systems were not resolved. Often, the concurrency control was handled by static scheduling, which made it possible to deal with data dependencies and mutual exclusion in a static way. For large systems, this inflexible approach became very cumbersome [Tör96].

Over the years, even more of the focus was shifted from the hardware to the software. Control logic, sequence control, information presentation and alerts were implemented in software. At the same time, control of manufacturing processes continued to become increasingly complicated [Bor89] due to shrinking product lifetimes, short lead times, increasing process sophistication, and growing volumes of information in the manufacturing process.

To cope with the information handling, many research papers on control system architectures for manufacturing refers to "global databases" very briefly, without providing details of what was expected. Usually the database was assumed to be distributed and all non-private data should be accessible from all nodes in a distributed manufacturing control system. The National Institute of Standards and Technology, based in the USA, conducted a study of hierarchical control systems [Kra93]. None of the systems mentioned in the study had the sort of global database envisioned in the theoretical papers. What typically was labeled as a distributed database was usually more like a distributed object manager.

1.2.1 DATABASES IN CONTROL SYSTEMS

Database systems provide general solutions to data management problems, but they have traditionally been designed for business applications and not manufacturing control applications. However, it is clear that database technology can be useful in other application areas, such as manufacturing and computer -aided software engineering (CASE).

The lack of adequate information support for engineering applications has triggered research in database management systems [Pri89]. Concepts such as complex objects, long-lived transactions, extended relational models, object-relational models [Cat91] were developed as a result of this research. These concepts have significantly simplified the modeling of complex engineering applications.

For simple systems, relational database management systems (RDBMS) already provide good solutions for data management, whereas more complex systems require support for modeling of structures, assemblies, interconnections, and processing. Representing complex objects in a relational database is difficult since the relational model requires all data to be atomic [Elm89]. Therefore, the modeling of complex composite objects results in many tables that have to be joined [Cat91]. This makes the RDBMS notoriously slow when used to model complex manufacturing systems.

An assembly could be directly represented as a composite object in an object-relational database [Ors96]. The composite object could be composed of other composite objects, all through direct references instead of time-consuming join operations.

1.3 Motivating scenario

This section takes a closer look at the information in a typical workcell. The following motivating scenario [Lew94] with a production cell consisting of robots, cranes, machinery, conveyor belts, and sensory equipment will be used to illustrate problems and ideas in the thesis.

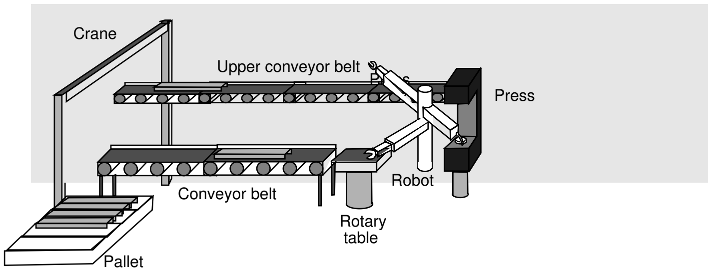

Figure 1.1: Metal processing plant

The production cell in Figure 1.1 contains a press that processes metal parts. The parts are placed in the press by a two-armed robot. Two conveyor belts and a crane transport the parts to and from the production cell. Processed parts are placed on a pallet. The production cell has several sensors providing information about the environment and the machines. Most sensors return a boolean valued result, such as "press is closed", but some sensors return real values, such as "the rotation angle of the robot base". A number of actuators control the cell.

Even in this simple scenario, there are many types of information. There is information about machines, tools and the actual manufacturing process (*i.e.,* process data). Furthermore, there is information about resources (such as inventory data), capacity, organization, production planning, status, produced amounts, and economic information. Matters are further complicated by the fact that there are many types of users with different needs.

Machine operators, management, and the control application itself need to access and manipulate information. In addition to this, each plant can produce many different products where each product consists of several components. Things get even more complex when a shop floor contains several production cells with buffers between them. The environment in which the control application operates can be very complex and heterogeneous with many diverse subsystems that need to exchange information. T ypically , the environment may contain sensors, actuators, file systems, communication sub-systems, and conventional databases containing for example inventory data.

All these components need to share information. An architecture that enables all the diverse components and software to work together , managing complexity and solving data management problems, would be of great value. Our attempt to tightly integrate a manufacturing control system with a database query processor and active rule facilities aims at a general approach to solve the data management problems using the next generation of database technology .

1.4 The thesis

This thesis addresses data management in control applications. It investigates how active databases can support complex manufacturing control systems. The focus of the thesis is on active rules and how they can be used to monitor the manufacturing operations and the environment.

1.4.1 The problem

The thesis presents efforts to solve the problems with data management in manufacturing control systems that operate in complex environments with large volumes of data involved. What are the typical data management problems and how do they complicate the system development process?

This section gives some examples of data management problems in control applications.

The implementation of control applications becomes difficult since much effort is required to ensure safe concurrent access to information. When several programs or users access the same information, it is hard to guarantee the consistency of information. Power failures also threatens the information consistency .

Another data management problem is the monitoring of system-wide integrity constraints. Much effort is required to ensure that no program violates the integrity constraints. Changes in system configuration or data structures are a difficult task. This generally require rewrites of large portions of a control system.

These are some examples of problems that can arise in control applications. This thesis proposes the use of embedded active database management systems (ADBMSs) in control systems to solve many of the data management and complexity problems that can arise in control applications.

Several interesting questions arise when attempting to integrate a database management system with a control system:

- How do we model the environment in the database?
- How do we integrate the control system with the query processor?
- Can declarative queries and active rules describe manufacturing operations?
- How can we avoid cascading rule triggering and rule cycles?
- How does the rule semantics affect the integration?
- How can we make the rule processing more efficient?
- How are transactions used in a control system architecture?
- How do we handle locking of external objects?
- How can conventional data and process data be combined?
- What are the requirements on the active database?

These questions are raised and answers are given in this thesis.

1.4.2 THE APPROACH

The proposed approach is to embed an active object-relational database in control systems that can be divided into a layered architecture.

The embedded database stores an abstract model of the controlled environment. A set of control algorithms, running on a separate real-time server, performs the actual closed loop control of machinery and equipment in the external environment. Active rules in the database constitute the interface between the control algorithms and the model of the environment, as stored in the database. In this way, our architecture combines traditional control algorithms and an event-driven operation language with data management capabilities.

1.4.3 HYPOTHESIS

The hypothesis is that ADBMSs can increase the flexibility of and consistency in the management of information in layered control systems.

1.4.4 CONTRIBUTIONS

The contributions are three-fold:

- An database-centered control system architecture
- A way to organize active rules that avoids rule cycles, cascading rule triggering, and rule interference
- A list of requirements on the embedded database.

The thesis shows how the database can be used to store an abstract model of the external environment. We discuss how the active rule facility in AMOS [Skö94][Ris93] is used to monitor the state of the environment as stored in the database and how the appropriate control algorithms are initiated in a separate real-time server. Examples are also presented of how the control application language based on the query language can be used as a high-level operation programming language.

The main contribution of the thesis is the way the active rules are organized to avoid rule interference and rule cycles and how dynamic rule grouping can be used to make rule processing more efficient.

To improve information access, a tight integration between control system and the database query processor is advocated. A specialized control application language, CAMOS(L) has been developed for this purpose. CAMOS(L) is used for describing high-level manufacturing operations and is tightly integrated with the database query language.

Since all information is stored in the database, the control system gains uniform access to combinations of conventional data, process data, execution states, sensor values, information from control algorithms and so forth. Therefore, the database presents a highly heterogeneous environment with many data sources as a homogeneous programming environment to the control application programmer . The database system provides the control system architecture with general solutions to data management, complexity and interoperability problems. The database query optimizer improves the performance of information access in the system.

The last chapter contains a description of how future control systems can benefit from the ADBMSs and what the requirements are on the embedded database. This type of advanced integrated control application puts new demands on database technology , such as: high performance, predictability , active rules, and a foreign function interface.

1.5 Organization of the thesis

Chapter I presents an introduction to the area of data management in control systems. The chapter starts with a very brief overview of the proposed database-centered control system architecture.

In the following section, a review of previous techniques for information sharing in control systems is given. The last section presents the organization of the present thesis.

A motivating scenario [Lew94] with a production cell consisting of robots, machinery, conveyor belts, and sensory equipment illustrates the problems (see Section 1.3).

Chapter II gives an overview of expert systems for process control, active database management systems, the AMOS system, and the ARAMIS control system architecture which has been the starting point for this thesis. Extended transaction models are discussed briefly.

Chapter III presents related systems that use databases to control processes, such as, BPFL, Chimera, TRIGS$_{flow}$, the SAMOS workflow system, and a CIM system based on Illustra. This chapter concludes with a brief account of the research efforts on rule groups.

Chapter IV presents the proposed CAMOS architecture and describes how the embedded database system is used in the integrated architecture.

Chapter V studies the use of active rules in control applications. In particular, the effects of coupling modes and rule ordering mechanisms are investigated.

Chapter VI contains a list of requirements on the embedded database. The chapter is divided into three parts: general database requirements, requirements on the active rule facility, and performance requirements.

Chapter VII summarizes the thesis.

Chapter VIII, Future Work, lists some unresolved issues. The long term goal of this work is also presented.

Chapter II
Background

This chapter serves as a background to related areas of research. Readers familiar with expert systems, active rules, transaction models, and workflow systems can go on to the next chapter .

2.1 Expert system control

Expert systems are useful in complex and ill-structured applications that lack efficient algorithmic solutions. The applications are decomposed into a set of rules without any ordering. Most expert systems consist of the following parts:

1. Rule base
2. Data bank
3. Inference engine
4. User interface

Rules, also called production rules, are the main knowledge representation method used in existing expert systems. Expert system shells such as OPS5 [For81], KEE [KEE085], and G2 [Årz91] all include rule management.

A typical syntax of the rules is:

```
if <condition> then <actions>
```

Most systems allow pattern matching to be used in the conditions. The data bank represents information about the domain of the application. Early systems used collections of lists [Årz88] as data banks. The Emycin class of systems used object-oriented data structures [W al88][For81][V in92].

2.1.1 ARCHITECTURES FOR PROCESS CONTROL

The earliest approaches to expert system control defined control operations using rules-based expert systems [Has86]. Since then, different architectural styles have been tested, such as blackboard architectures [Hay85], layered architectures, and recently , distributed multi-agent architectures [Efs92]. AI techniques such as rules, neural nets, and fuzzy logic have been applied in process control in the past few years.

The blackboard model

The blackboard model [Hay85][Eng88] is based on a shared data area where various system components can read and write information. This allows the components to reason and generate further conclusions. Later , the idea of blackboards has been extended with blackboard managers, schedulers, and blackboard servers, which control the data written to the blackboard [Efs92][Kri91]. The purpose of these managers is to control the ordering of access in data-intensive expert systems and when complex data dependencies exist between the components of the expert system.

Often the knowledge base of rules needs to span over large volumes of data. T o alleviate this problem, there has been considerable effort to couple expert system shells to database management systems. Some notable examples are KEE [KEE085], and KBMS [Cer86].

Most proposals suggest that a database should handle the data management while the expert system shell handles the rules. As we will see in Section 2.2.1, this approach has inherent problems with consistency .

Layered systems

Layered expert systems are systems where tasks are placed in different layers depending on the time-scale at which the events occur [Efs92]. Low-level control systems take data from sensors and apply actions to a plant. The supervisory system may also observe the sensor data, but takes longer time to reason about the information. The supervisory layer recommends actions on a longer time scale.

2.1.2 KNOWLEDGE-BASED REAL-TIME CONTROL SYSTEMS

Several consulting companies have presented expert system products aimed at process industries. Examples include Gensym Corp. (G2), the Carnegie group, Framentec, P A consultants, Cambridge Consultants, and Stone & W ebster Engineering Corp. [Årz88].

The IT4 *project*

One attempt to apply knowledge-based techniques on a real manufacturing problem was the IT4 project at Lund Institute of Technology [Årz91]. The goal was to design a distributed control system architecture that allowed knowledge-based techniques to be integrated with conventional programming techniques.

The IT4 project adopted an object-oriented database approach to structuring the knowledge and information in the system. At the time of implementation, the object-oriented database technology was not considered sufficiently mature so their system did not include any database in the actual implementation of their demonstrator . A problem with expert system control is the way expert systems encode heuristic knowledge in rule sets.

This makes it difficult to express sequences of operations. To overcome this problem Årzen and colleagues implemented Grafchart [Årz94] based on Grafcet (IEC848) which is also known as sequential function charts [IEC95]. Grafchart was implemented using the G2 system and allows parameterization the Grafcet Sequence Nets.

2.2 Active database management systems

This section provides a background to active database management systems (ADBMSs). Database management systems have traditionally been passive repositories for data. Operations are performed on the database: *insertions, deletions, and retrievals* but the database cannot actively changes in its own state. New applications such as air traffic control, network management, computer-integrated manufacturing (CIM), and program trading require database systems that can detect and react to certain situations. Active database technology shows its strength in these new scenarios.

An active database system [Dit95] is a database that monitors situations of interest and, when they occur, triggers an appropriate response in a timely manner [Day95].

Historically, the management of rules comes from the field of AI. Production-rule systems have been widely used in expert systems. What differentiates production-rule systems from active database systems is that active database systems are database systems. The active rules can affect large sets of items and the conditions can be stated over equally large sets of items. Since the active rule facilities are built into the database, the monitoring of data can be efficiently supported. To motivate the use of active databases, this section compares five different approaches to monitoring information in a database. Most approaches suffer from performance or consistency problems when complex conditions must be monitored in large databases.

The conclusion of this section is that active rules are an efficient monitoring mechanism for databases.

2.2.1 MOTIVATION FOR ACTIVE RULES

Consider an application where several programs are interfaced with a database. Furthermore, the application requires that some consistency constraints are maintained for the data in the database. This section presents five approaches to the monitoring of constraints in such applications:

- Application check
- After -the-fact queries
- Periodic queries
- Single interface program
- Active database

The rest of this section discusses each monitoring approach and explains why the active rule approach is the best for systems that require a combination of good performance and guaranteed consistency .

In the application check approach, every program that changes the database checks whether an update would violate any constraints before doing the update. This is a poor approach since each constraint must be checked in every program that may affect the constraint. T o ensure the constraint checking, the programs get cluttered with many constraint check routines in several places. Clearly , this makes it difficult to change a constraint. Database consistency is not guaranteed by this approach. Direct manual updates in the database are not checked by the constraint check routines in the programs. It is also likely that a programmer may make a mistake and omit or incorrectly implement a constraint check in one of the programs.

The after-the-fact query is a better approach, where the application performs the update and then checks the constraint by issuing a query to the database.

This allows the expression of more complex constraints, but the constraints are still scattered in many programs and no guarantees are given for database consistency .

Periodic query is the third approach. In this approach, an independent program periodically checks the constraints by querying the database. Although this solves the problem of scattered constraints, this approach has other drawbacks. Since the checking of the constraints is done by an independent program there is no way to detect which program violated the constraint. Another issue is the choice of polling frequency . How often should the constraint checker query the database? There is no good answer to this question. Excessive polling will slow down the system; polling too infrequently risks missing constraint violations.

A single interface program handles all the updates and constraint checking in this fourth approach. This concentrates all the constraints in one piece of software, facilitating easy modification and checking. It is possible for this interface program to notify programs that try to update the database in a way that would violate a constraint. Unfortunately , manual updates are not check. Another problem is the potentially large volumes of data that need to be transfer between the interface program and the database. In other aspects, this approach is similar to an active rule approach where the monitoring facilities are incorporated in the database.

The active database approach has all the benefits of the previous approach, and handles the manual update problem. In this approach the active database monitors and reacts to changes in a database. The database guarantees the consistency constraints and the performance of the system is improved in comparison with the other approaches described above. The active rule approach promises guaranteed consistency and efficiency .

To guarantee consistency, all information subject to monitoring should be present in the database. This imposes a performance penalty for updating and retrieving information compared to systems where program variables store part of the information.

2.2.2 RULE MODELS

There are several existing rule models for database management systems. The most common model for active rules in databases is the ECA model [McC89] where 'E' is the event that triggers the rule, 'C' is the condition monitored, and 'A' is the action to be performed when the event 'E' occurs and the condition 'C' is satisfied. Active rules have the form:

```
ON event IF condition DO action
```

Many systems support this type of rule, *e.g.,* Starburst [Wid90], and Postgres [Sto90]. The events can be database operations such as retrieve or update. Some systems [Buc92][Cha93] support composite events and event composition algebras to handle rules related to several events or combinations of events.

The HiPAC project [Day88] defined a five-part rule model. Besides ECA, they introduced two couplings: E-C and C-A. The E-C coupling specifies when the condition 'C' will be evaluated relative to the occurrence of the event 'E'. The C-A coupling specifies when the action 'A' is executed relative to the evaluation of the condition 'C'. In addition to the couplings, three possible coupling modes were presented:

- Immediate
- Deferred
- Separate

The **immediate mode** preempts the ongoing transaction and evaluates the condition. If the condition is satisfied, the rule processor immediately executes the action within the current transaction.

The **deferred mode** also operates within the current transaction, but evaluation of conditions and execution of actions is deferred until the commit of the transaction.

The **separate mode** forms a separate transaction. The E-C and C-A couplings can be combined into coupling modes such as immediate-immediate. The immediate-immediate coupling mode means that the E-C coupling is immediate, and the C-A coupling is immediate.

Another paradigm for active databases is the conditional-action rule (CA rule) [Ris93][Han92]. As opposed to ECA rules, CA rules do not specify an event, meaning that the rule is triggered whenever the condition is true, regardless of which event caused the condition to be satisfied. The ECA rules can specify different actions depending on which event occurred. This is not possible with CA rules. On the other hand, CA rules are easier to write, since they avoid the specification of all possible events that could affect the condition.

A third class of rules used in database systems are the deductive rules [Ull88], which query the database state using predicate logic. Deductive rules are declarative and demand-driven and cannot actively react to changes in the database state. The difference between active rules and deductive rules is that the active rules modify the database state instead of deducing information from facts and rules at the time of the query .

2.2.3 COMMERCIAL SYSTEMS

Research in active database systems have moved from ideas, concepts, early prototypes, full-fledged active database management systems, and finally to new functionality in commercially available DBMSs.

Active database facilities have been incorporated into several commercial DBMSs. For example, both INGRES [ING90] and SYBASE [SYB90] have trigger systems integrated into their relational DBMS engines.

At present, the rule facilities in such commercial systems are restricted. Common restrictions are that events cannot span several relational tables and the number of triggers is limited. Some systems only allow one trigger per table.

2.3 Active Mediator Object System

The active mediator object system [Fah93], AMOS, is a research platform for experimenting with specialized database systems. It is a fast main-memory object-relational database with a data dictionary, a query language, transactions, database procedures [Flo96], multi-database queries [W er96], access to relational databases [Fah94][Fah96], and active rules [Skö96].

The query and modeling language of AMOS, AMOSQL, is a derivative of OSQL [Fis89] and is based on the DAPLEX functional data model [Shi81]. The system supports high-level object-oriented abstractions and declarative queries for extracting and manipulating data. A foreign function interface allows external programs and drivers to be linked to AMOS. A cost-based optimizer optimizes the queries [Lit92]. Mediators for heterogeneous databases (*e.g.* SYBASE) and external file systems are provided [Fah93].

2.3.1 ACTIVE RULES IN AMOS

The AMOSQL modeling language has active rules facilities that detect updates to the database or to a data source [Fah93] [Ris93]. A rule condition can be any AMOSQL query and specify when the rule should be triggered. The action part can be a procedure call to a foreign function or any database operation.

Active rules in AMOS have the format:

```
when query(parameters)
  do procedure(parameters)
```

The AMOS system supports rules that follow the set-oriented condition-action paradigm [Wid90]. Currently, two coupling modes are supported: deferred, and a user defined coupling mode where the application or the user decides the rule processing points [Skö95].

2.4 ARAMIS

The ARAMIS (A Robotic and Manufacturing Instruction System) layered hierarchical control system architecture [Hol92][Lob91] [Lob94] is based on a graphical language which is a hybrid between a production rule view and traditional imperative programming languages. The fundamental concepts of ARAMIS, workers and actions, are adopted from a workshop metaphor and the process of modeling and specifying the behaviour of a workcell is essentially similar to the normal design process of a production engineer. The goal of the hierarchical architecture was to introduce abstraction levels and to simplify programming of the different parts of the software-like high-level operation descriptions, control software, and device drivers.

World model paradigm

The ARAMIS philosophy involves a world model, WM, which is a state model of the objects in the environment. The world model allows programmers to describe high-level interactions by using well-defined and simplified world model objects. The highest layer in the architecture coordinates operations in the external environment using the world model as a blackboard [Hay85] to communicate with the control level layer.

The control layer is closer to the environment process and acts as a servo mechanism using the world model as a reference value. Finally, a physical layer sets up the I/O-interface to actual sensors and actuators. The programming at this level is usually done in C or assembly language. The ARAMIS system uses a data repository to represent the world model. The ARAMIS system is documented in [Lob94][Hol92].

Relation to the thesis

Many concepts and ideas in ARAMIS are used without modification in this thesis. For example the language used in CAMOS is functionally similar to the ARAMIS task language apart from the constructs for data management. The idea of storing an abstract model in the database is derived from the world model concept in ARAMIS [Lob91]. In the work presented here, the simple navigational data repository in the ARAMIS system has been replaced with a main-memory-based object-relational database, AMOS. The original ideas for the integration of an active database management system and a control system were developed at RTSLAB [Lob93].

2.5 Transaction models

Control applications require capabilities to manage long-running transactions [Gra92]. Long-lived transactions in a database are a complicated issue. Whereas simple transactions are executed in tenths of a second, the duration of a long-lived transaction can exceed days or months in length. This type of transaction needs different programming techniques and a different concept of duration. In the literature, there are several suggestions for new execution models to allow control of long duration transactions which may consist of an arbitrary number of subtransactions *e.g.*, nested, chained, conditional. [Gar87], [Gar91], [Reu90]. Another approach is to use relaxed transaction models [Chr93] [Geo95].

2.5.1 LINEAR SAGAS

Linear Sagas [Gar87] were originally proposed to solve the problem of long-lived transactions. The model was later extended to parallel, nested, and generalized Sagas [Gar90][Gar91]. The Saga model allows a transaction to release resources before commit time. A Saga transaction is a sequence of sub-transactions (Saga-steps) that can be interleaved with other transactions. The Saga-step is a separate transaction with ACID properties. Each step also has a defined compensating transaction.

Undo is accomplished by invoking the compensating transactions associated with each Saga-step. These compensating transactions undo the changes made by the Saga-step, but do not necessarily restore the database to the same state.

2.5.2 CONTRACTS

The ConT racts model [Reu90] of handling long-lived computations in databases is a development of the Saga approach. The operations are divided into steps where each step has ACID properties. There is a guaranteed forward recovery that ensures that once started, an operation never gets lost or stuck in an inconsistent state. Instead, it will either be executed to its normal end or be brought to an acceptable state by a compensating mechanism.

2.6 Relation to the thesis

How do the background sections relate to the present thesis? This work proposes a layered control system architecture (see Section 2.1). Active rules (Section 2.2) are used to integrate an object-relational model of the manufacturing environment with a real-time server . The use of ECA and CA rule models (Section 2.2.2) is further discussed in Chapter V. The database system used in this work, AMOS (Active Mediator Object system), is presented in Section 2.3.

The ARAMIS system (Section 2.4) has been the starting point for this thesis. Many concepts from the ARAMIS are used without modification in the architecture proposed in the thesis. Transaction models for long-lived operations (Section 2.5) are also discussed in the thesis.

Chapter III
Related Work

Applications of active database systems have extended beyond the early toy examples. This chapter gives an overview of systems that use active rules to control processes. The first section presents the Berkeley Process Flow language. The second and third section describe previous attempts to use active rules for the control of operations in manufacturing and workflow systems.

3.1 Berkeley Process Flow Language

In his dissertation, C. J. Hegarty proposes a CIM architecture with a shared, integrated, distributed database and a new language for semi-conductor operations called BPFL [Heg91]. BFPL is used for handling operations concerned with the fabrication of integrated circuits (IC). It controls movements of material, issues instructions, allocates resources, and maintains a log.

The system runs in a distributed computing environment, with a local-area network connecting factory computers, database servers, and cell controllers. A database stores information about the equipment, test data, inventory data, and orders.

The operation flow specification and operation state information are also stored in the database so a fabrication run can be restarted after a system crash. This also enables modifications to running operations. BPFL is an extension to the Common Lisp CLOS system [Ste90] with constructs to specify loops and rework in the IC fabrication process.

3.1.1 USE OF DATABASE.

The BPFL system uses an INGRES relational database. Three types of information are stored in the database. The first type is data structures representing operation flows, and material. The second type describes the fabrication facility . The third type describes the current state of the process flow .

Since CLOS objects cannot be stored directly in the INGRES system, special translator methods must be used to transfer objects to and from the database (the impedance mismatch problem). Relational database tables are defined for each object type in the system. Queries over the fabrication process are difficult to handle since all process information is stored in the translated database format.

Relation to thesis

Although appropriate for the IC fabrication application, the BPFL model assumes that all the processes are controlled by the system and the database is only used as a record of the execution state and the locations of the lots. Therefore, active rules are not needed in the BPFL system. This makes it difficult to set up cooperating operations other than pure sequences and rework loops. This limits the flexibility and applicability of this approach to area of control applications.

3.2 ADBMS in control

There have been many attempts to use active databases for the control of processes, tasks, workflows, activities, operations [Seg95], [Bar95], [Nor95], [Cha95], [Kap95], [Gep95]. This section studies these earlier attempts.

3.2.1 THE DATABASE RESEARCH GROUP (ETH ZENTRUM)

The database research group at ETH in Zürich take the view that a CIM system is a form of cooperative work among component systems. The emphasis is on coordination, system-wide consistency [Nor94], and methods to adapt and combine multi-databases in CIM applications [Nor95]. They have developed methods to incorporate components without database capabilities by means of local agents that provide the needed database functionality. The agents provide a coordination interface that specifies which objects in the local system are globally important, *i.e.,* affect the system-wide consistency. The agents are responsible for maintaining the consistency by local actions.

3.2.2 THE CHIMERA SYSTEM FOR MANUFACTURING CONTROL (POLITECNICO DI TORINO)

Baralis and colleagues present an active rule application for the control of a manufacturing plant (SNIA)[Bar95]. The control of machinery is modelled using extended automatas, known as controllers. Each controller has a set of states and transitions. The states correspond to phases in the process performed by the machinery and the transitions model the progress of the process. The transitions are associated with control variables. A transition is enabled if all of its control variables are active. Active rules are used to implement the state machine. This system does not handle large volumes of data or complex independent processes.

Even though the rule set for modeling the state machines consist of only six rules, it is difficult to predict the behaviour since the rules interact. A tool for the analysis of interacting rules is presented.

Relation to the thesis

This work is more focused on developing methods for rule analysis, rather than solving problems in manufacturing control. Their operation model roughly corresponds to the DFA mechanism in the architecture proposed in this thesis. There is no equivalent to the high-level manufacturing operations in CAMOS.

3.2.3 AOODBMS FOR DISCRETE CONTROL

The project 'Active Object-Oriented DBMS for Discrete Control' at the University of Michigan focuses on feed-forward and feedback control of semiconductor manufacturing facilities. In feed-forward control, deviations in the processing of a wafer in one cell are compensated for by adjusting the processing in the next cell. In feedback control, a cell that has processed a wafer receives advice from a later cell to adjust its process. The idea is to use active rules to achieve inter-cell feed-forward and feedback control of the manufacturing process. The research is focused on developing models for fuzzy data and fuzzy processing of active rules [Cha95]. The motivation is that the manufacturing of semiconductors is not well understood and measurements cannot be directly used to decide which action to take. As far as we know, the experience of the use of active rules in this application has not been reported.

3.2.4 LONG-RUNNING OPERATIONS

A flexible way to express control of long-running operations using active database systems was presented by Umeshwar Dayal and colleagues [Day90]. This was the first effort where an active rules was used to express the control-flow of workflow applications.

A combination of coupling modes and nested transactions control the time and transaction context of rule execution. Rule priorities, a pipelining property, and transaction cycles define the rule ordering.

Relation to thesis

Dayal and colleagues present a flexible way to express event-driven coordination of long-running operations. Their approach is similar to the approach in this thesis in the sense that only parts of an application is coded with active rules. The coordination of operations and check of integrity constraints is encoded using ECA rules. The need for rule ordering and context switching is pointed out. Two mechanisms for context switch are proposed: flag-functions (similar to the approach presented in Chapter V), and special events indicating a context switch.

A difference between their work and the work presented in this thesis is that here we propose the use of active rules only as basic building blocks for the implementation of process coordination, while their work uses active rules directly to express the control flow of operations. In our system, programs are written in a specialized manufacturing control application language which then is compiled into active rules and queries, and runtime code.

3.2.5 AORDBMS FOR CIM SYSTEMS (LAWRENCE BERKELEY NATIONAL LABORATORY)

A CIM system based on an active object-relational database was presented by a group of researchers from Lawrence Berkeley National Laboratory [Seg95]. They implemented a prototype CIM system using the commercial ADBMS Illustra [Ill96]. The ADBMS was connected to an automated machining shop through a real-time server. The operation of the CIM system was specified by an event-rule framework. Illustra does not accept external events; in order to get the rule system of the Illustra database to trigger on events in the environment the research group introduced event-tables for external events. An event manager periodically polls the factory server for interesting events and updates the event-table. This makes it possible to indirectly state rules of events such as 'operation completed'. The Illustra database imposes several restrictions on the application. For example, the Illustra rule manager does not support events that involve several tables and cannot combine events using event composition algebras.

A modified form of Object Flow Diagrams (OFD) is used to describe processes in the CIM system. These are translated into rules over the event-tables. Four primary functions are identified and implemented using active rules: lot release, lot arrival, start of operation, and completion of operation.

Relation to the thesis

The work at Lawrence Berkeley National Laboratory relies on events from the controlled processes, whereas the approach proposed in this thesis is based on an abstract world model.

Another difference between their work and the work presented in this thesis is that here we propose the use of active rules only as basic building blocks for the implementation of process coordination, while their work uses active rules directly to express manufacturing operations.

3.3 Workflow management systems

Workflow management systems (WFMS) have become a hot topic during the past three years. Workflow management systems (WFMS) [Rus94][Geo95] are used for the specification of data and control flow between different operations. A workflow is typically a collection of operations organized to accomplish some typical business process, such as the processing of loan applications or purchase orders. WFMSs define the order of operation invocation and the conditions under which operations must be invoked, *i.e.* operation synchronization and data flow. An overview of the field is provided in [Geo95]. The WFMS come from three different directions [Pos95]:

- WFMSs based on enhanced mail systems
- WFMSs based on extended transactions
- WFMSs built from scratch

The most notable system in the first group is 'Lotus Notes' [Spu94]. Several research projects examine the use of databases in WFMSs [Moh95], [Kap95], [Dav95].

3.3.1 ADMBS IN WORKFLOW

A few systems use ADBMSs to implement workflow enactment. The TriGS $_{flow}$ [Kap95] is a WFMS implemented on top of the object-oriented database GemStone [But91]. Another workflow system is implemented on top of the SAMOS database [Gep95].

3.3.2 TRIGS $_{flow}$

The business domain objects are modelled in the database. The database has been extended with an active component, TriGS [Kap94]. A frequently changing part of the workflow is modelled using ECA rules. The TriGS $_{flow}$ system contains a 'Roles' module. Traditional object-oriented systems require objects to belong to exactly one class. The concept of roles allows objects to learn and forget roles, and in this way belong to several classes or change class during the lifetime of a workflow.

Active rules in TriGS $_{flow}$

The execution order in this WFMS [Kap95] is defined by an operation net built using control structures such as sequencing, branching, and joining. These control structures can be combined into more complex composite structures. The ECA rules are derived from the control structures. The event part of the rule is mapped to the termination of the preceding operation.

The condition part checks whether the succeeding operation has to be performed or not. The action part places the operation into the work-list of an appropriate agent in an appropriate role which will then perform the operation. Active rules are also used within agents to start processing of new operations when the previous operation has terminated. This gives the system basic workflow capabilities. Clearly, the initiation of new intra-agent operations could have been implemented with message passing instead of active rules.

3.3.3 COOPERATIVE AGENTS

A workflow system implemented on top of the SAMOS active database system is described in [Gep95]. They propose brokers and services as constructs to build a service-oriented workflow system. The brokers can detect situations in which they have to react, so that control and coordination can be decentralized. The situations are more complex than simple service requests. In a conventional WFMS roles are used to define grouping of capabilities. In this system, roles are used to model the responsibilities of a broker at some point during its life time.

ECA-rules are used to model the brokers. Each role specification consists of a set of ECA-rules and state variables. The following event types can be used: service requests, service replies, time events, update events, and method events. To model more complex situations, composite events can be constructed from the primitive event types.

The rule conditions are expressed over the state of brokers. The actions part of the broker role rules can be service requests and replies, or calls to methods. A partial ordering of rule processing can be defined by using a precedence clause. Here rule processing means the process of checking conditions and executing actions.

Precedence clauses must be specified if the action part of one rule affects the condition part of another rule.

The broker rules can model many kinds of task dependencies: execution, value, and temporal dependencies. An execution dependency defines that upon the termination of one task, another task should be started. A value dependency can, for example, state that task A should be executed if task B terminates with an output value below a certain threshold. Temporal dependencies defines timing constraints on tasks, such as task B has to be started within 6 weeks after the termination of task A.

Relation to the thesis

Although both WFMSs and the CAMOS system proposed in this thesis coordinate operations, the systems differ in many ways. The purpose of WFMS systems is different and the number of processed operations per second in WFMSs is much lower than in the CAMOS system. To be useful in any real applications a control system must process several hundred thousand operations per hour. Workflow systems do not need such high performance. The high-level operations in the control application language can be viewed as an efficient workflow specification of operations in control applications. A WFMS contains a model of the processing but does not provide models of the environment the operations operate in. Another difference is the execution model. WFMS are constructed for control of business operations and do not provide means for dynamic scheduling.

3.4 Research efforts on rule groups

This section reviews rule grouping techniques related to the dynamic rule context developed by RTSLAB (Real-Time Systems Laboratory) and EDSLAB (Engineering Database Systems Laboratory) at Linköping University. Rule groups were first introduced in the context of expert systems. In particular Mycin [Buc84] used rule groups in its inference mechanism. There have been several research efforts on rule control in the database area.

The rule execution of active database management systems such as POSTGRES [Sto88], ARIEL [Han89], HIPAC [Day88], and AMOS [Fah93] incorporate a flat transaction model. The rule systems use priorities to define processing order among rules. More expressive models use extended nested transaction models to support nested triggerings of rules [Hsu88]. RDL1 [Sim92] supports a set of constructs for specifying control structures among rules. The rules are defined in 'modules' with static rule control structures such as 'sequence', 'disjunction', and 'saturate'.

A more flexible approach is to specify separate rule graphs for each event [Jaw94]. These graphs define the execution order of conditions and action for different events. This enables the system to use different control structure for different events.

Relation to the thesis

In this thesis, I point out the need for mechanisms that allow efficient dynamic activation and deactivation of active rules. As a result of the CAMOS development, dynamic rule contexts were introduced in the AMOS database management system. Dynamic rule contexts facilitate development of the CAMOS operation manager .

Chapter IV
Architecture & Language

This chapter provides an overview of the CAMOS (Control Applications Mediator Object System) architecture and describes how the components of the architecture interact. The CAMOS system relies on several database mechanisms. Section 4.4 describes the use of the object-relational modeling language, declarative queries, transactions, and active rules in the CAMOS system. CAMOS has been implemented using the AMOS database management system developed at EDSLAB, Linköping University.

4.1 The architecture

The four key elements of the CAMOS architecture are the embedded active database, the operation manager, the real-time server, and the control application language called CAMOS(L).

The embedded database management system, AMOS, used in the architecture was developed by EDSLAB at Linköping University. The database stores an abstract object model of the external environment, known as the world model (WM). The world model gives the programmer the opportunity to ignore aspects of low-level control algorithms that are not relevant for the definition of high-level operations. In the introductory scenario (see Figure 1.1) the world model is a simplified representation of the state of materials and of the equipment. Active rules and queries are used to support the execution of manufacturing operations.

The operation manager controls and coordinates high-level operations, active rules, and the control algorithms. It consists of a scheduler , a rule coordinator , and an interpreter for runtime code. The scheduler is a round robin scheduler that handles synchronization with external events and temporal dependencies between operations. The rule coordinator controls the monitoring of the world model by activating and deactivating active rules in the database. The interpreter is used to prepare the execution of new operations according to the runtime code. All parts of the operation manager operate directly on the objects in the database, with no need to translate from and to database records.

The real-time server handles all sensors and actuators. The real-time server [Hol92] is a separate platform running a traditional real-time operating system. The control algorithms in the real-time server are programmed using traditional real-time techniques [Bur90]. Coordination of control algorithms and communication with the real-time server is achieved through a command and status protocol which channels status messages when control algorithms are initiated, terminated, or aborted. If an algorithm changes the physical environment, then the real-time server sends an update statement to the database (world model).

The control application language CAMOS(L) is used as a high-level programming language for manufacturing operations. A compiler translates expressions and conditions in the operation descriptions into runtime code, database queries, active rules, and database update transactions.

The runtime code describes sequences of possibly concurrent operations and interoperation dependencies. The queries are used to retrieve and derive information about the environment as stored in the database. The execution of high-level operations is synchronized with the changes in the state of the world model using the active rule facility of the AMOS database.

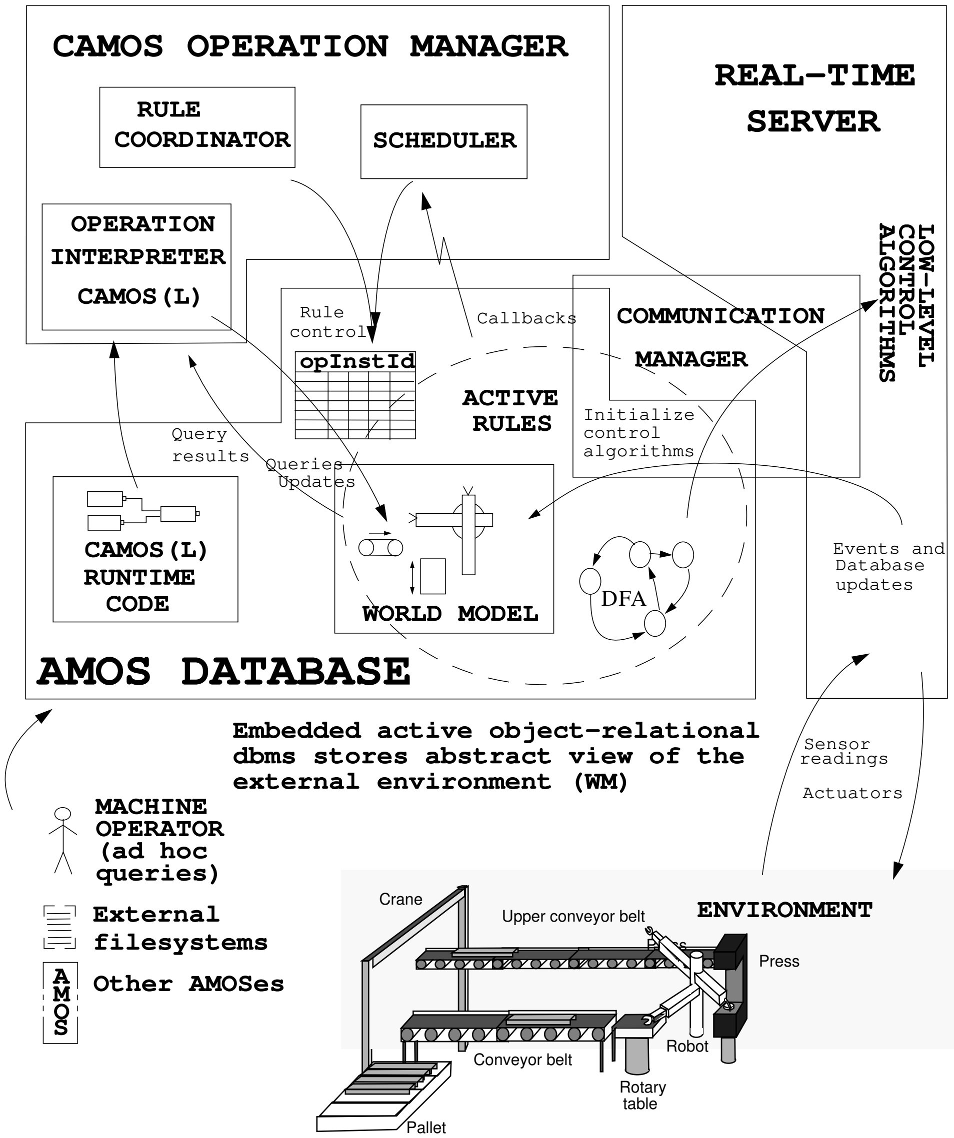

Figure 4.1: CAMOS database-centered architecture

4.2 Modelling active behaviour

The desired behaviour of objects in the environment is modelled at two different levels:

- CAMOS operations
- State models of active objects.

The CAMOS operations describe manufacturing operations at a high-level whereas the state models describe the behaviour of individual objects in the external environment (such as equipment and machinery).

CAMOS operations coordinate execution of high-level manufacturing operations. To achieve this, the control application language, CAMOS(L), has constructs to specify control flow and common abstractions in manufacturing control, *e.g.,* process synchronization and sequences of operations. Database update statements are primitive operations in the CAMOS(L) language. These primitive operations modify the world model stored in the embedded database. Basically, the language coordinates a series of database updates which modify the model of the external environment (WM) stored in the database.

The *state model* of an active object is represented as a deterministic finite state automaton (DFA) consisting of *nodes* and *transitions*. This automaton specifies allowed state changes for all the active objects in the environment. Each node in a DFA corresponds to a state of an external object. The node has a *node constraint* describing the situation when the active object is allow to be in that particular state. The nodes are connected by directed arcs (*transitions*) representing state changes. To carry out the state change, the transition is associated with a control algorithm in the real-time server.

If any database update satisfies a *node constraint* of a DFA, the active object changes the state and the communication manager sends a message to the real-time server which initiates the corresponding control algorithm.

The control algorithm will influence the environment and the resulting changes in the environment will be sensed, causing further updates to the database. This feedback works like a servo mechanism, keeping the environment consistent with the world model.

We have developed a general description model for operations and processes in control applications within the context of AMOS. The database stores operation descriptions and the state of the currently executing operations (processes). The operation descriptions and the processes are first-class objects in the database.

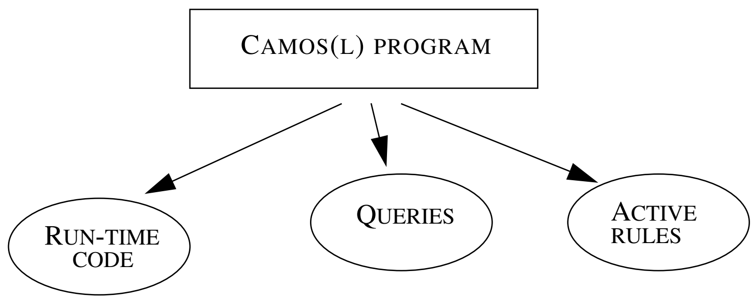

Figure 4.2: Compiling CAMOS(L) code

4.3 The language

A computer -based control system must sense the environment and directly influence it through actions. These actions are normally organized in high-level operations with a common goal. In large and complex control systems we typically find many related operations and complex dependencies between these operations. A paper presented at the Euromicro conference [Lob93], proposed active rules as a mechanism to coordinate operations in control systems. It is now clear that active rules alone cannot be used to build a control system. Other mechanisms to organize and coordinate the active rules are needed.

The goal of the CAMOS(L) language (Control Application Mediator Object System Language) is to provide mechanisms that can control and coordinate these operations and provide data management for the operations. CAMOS(L) is not intended to be a new general purpose programming language. It has been devised to simplify programming of a wide range of data-intensive manufacturing and control applications. The language aims at an efficient and general approach to modeling and data management in control applications. The data modeling language of the database is used to model and organize information about the environment.

4.3.1 CAMOS (L) OVERVIEW

This section contains a brief review of the control application language, CAMOS(L), being used as a programming language for high-level operations. It is **not** intended as a reference manual for the language.

CAMOS(L) is a successor to the programming language used in the ARAMIS system [Lob93] and addresses the need for data management in control applications. To overcome limitations in the access to the database the CAMOS language is extended with query constructs and transactions.

The basic concept of the language is the *operation*. There are several types of operations: *composite operations, iterative operations, primitive operations*. *Composite operations* and *iterative operations* are based on procedural abstraction, treating a group of possibly concurrently executing operations as a single unit. When initiated, the iterative operation repeatedly performs a set of operations as long as the *iteration condition* is satisfied. *Composite operations* are delayed until their *wait condition* is satisfied.

Composite and iterative operations can be decomposed into groups of guarded *sub-operations* that are dynamically scheduled (called selection of sub-operations). This provides alternative behavior of operations depending on which guards are satisfied.

The sub-operations can be partially ordered by *temporal dependencies* (dependency statements), stating that some operations must be executed in sequence. Operations that do not have temporal dependencies may execute in parallel. The temporal dependencies introduce constraints on possible schedules for a composite operation at runtime. Note that the dependencies are conditional. The scheduling depends on the guards for the sub-operations and dependency statements. The execution is synchronized with the external environment using wait conditions. Wait conditions, iteration conditions, and expressions can be parameterized and may contain aggregate expressions.

Figure 4.3 shows an example of an operation with query constructs, temporal dependencies, sub-operations and a wait condition. The purpose of the composite operation is to simultaneously fetch a new part from the elevating rotary table and to move the processed part from the press to the upper conveyor belt.

The operation "`getPressedFetchNew`" in Figure 4.3 is a composite CAMOS(L) operation with three parameters. The parameters are objects in the database world model. The "`as when`" clause defines the wait condition for the operation. The IF clauses defines the guards for the sub-operations. This getPressedFetchNew operation contains three sub-operations. Sub-operation 2b shown in Figure 4.3 is a primitive operation that will alter the world model. The `DEPENDENCIES` declaration introduces a temporal dependency between sub-operations 2a and 2b.

The syntax of conditions, guards, and expressions in CAMOS(L) adheres to the standard AMOSQL [Flo96].

```
CREATE OPERATION
  getPressedFetchNew
    (robot r, e_r_table t, press p)
AS WHEN
  partsat(p)
  AND position(p)="open"
  AND palletCapacity(currentPallet())>=
    COUNT((SELECT b
            FOR EACH part b
            WHERE
              position(b)=currentPallet()))
    IF NOTANY(partsat(upperConveyor())) THEN
      1:getpressedpart(r,p)
    IF SOME((SELECT b FOR EACH part b
          WHERE
            name(position(b))=
              "e_r_table")) THEN
      2a:getnewpart(r,t);
      2b:SET referenceElevation(t)=0;
      DEPENDENCIES:(2a,2b);
    IF SOME(partsat(t)) AND SOME(partsat(p))
THEN
      DEPENDENCIES:(1,2a);
```

Figure 4.3: An example of an operation

Why a new programming language?

The CAMOS architecture does not require a new language. Many programming languages could be used to code manufacturing operations and interact with active rules. What is the benefits of introducing the CAMOS(L) language?

First, the CAMOS(L) language provides convenient abstractions for manufacturing control and avoids the problems of a flat rule structure by introducing procedural abstraction.

Since the language is tightly integrated with the query processor, the system has uniform access to diverse categories of information including process data, external databases, information about the current execution, timing information, sensor data, actuators. All this information can be accessed directly by CAMOS(L) programs. This way, the database presents a heterogeneous world as a **homogeneous** programming environment and provides extended **power of expression** for high-level programming of manufacturing operations.

Note how the active rules are hidden from the programmer. The programmer only sees a language with data management capabilities and a rich set of synchronization mechanisms.

4.4 Database Interaction

The CAMOS system relies on several components in the AMOS architecture. This chapter describes how the different database facilities (data repository, DDL, DML, transactions, queries, extensibility and active rules) are used to implement the CAMOS control system architecture. The first section describes how an object-relational database can model the environment of manufacturing control systems.

4.4.1 THE OBJECT-ORIENTED WORLD MODEL

The main function of the database is to store information about the external environment, operation descriptions, and the current execution state. The information about the environment originates from the real-time server. The information is transferred to the world model (WM) in the database using a communication manager. The information can be sensed by sensor routines in executing control algorithms, or be implicitly known (believed) at the time control algorithms terminate in the real-time server.

The WM is defined using the object-relational data definition language of AMOS. An object in the WM represents a meaningful entity in the external environment — such as a robot or a metal part. The AMOSQL extensible type system is used to collect entities of similar properties into classes (see Figure 4.4). In AMOSQL, the term type corresponds to class in object-oriented programming languages.

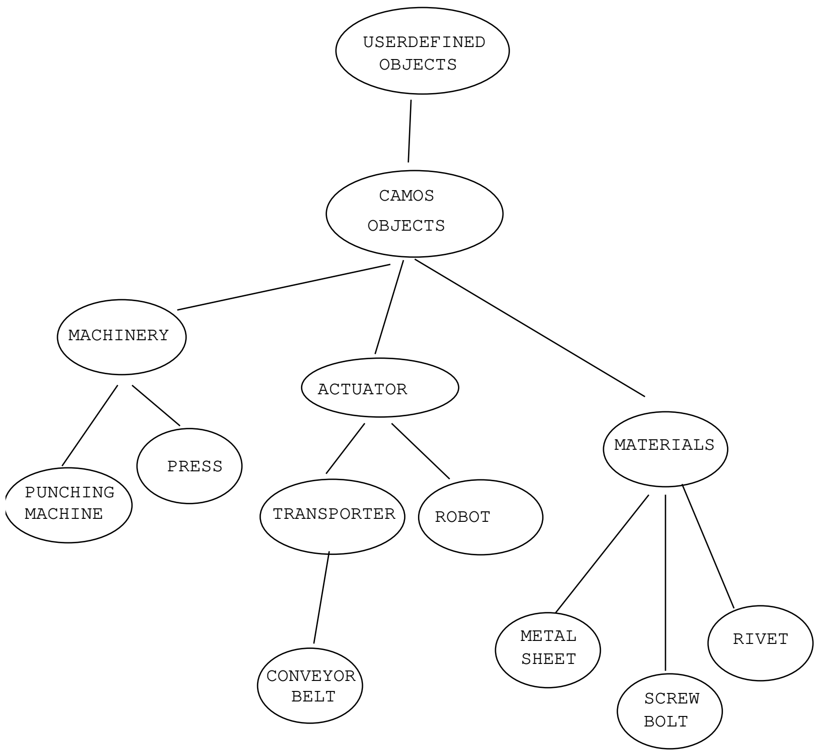

Figure 4.4: An example of a class hierarchy

Usually , the model of the environment includes objects with complex internal structures, which are composed of many sub-objects, typically of another type (class) [Ors96].

The example in Figure 4.5 illustrates how composite machinery can be modeled with *part-of* relations or using *multiple inheritance* . The production cell scenario contains a model of a two-armed robot as an object with four sub-objects attached (see Section 1.3). Note how multiple inheritance extends the object with additional functions (actualElevation and referenceElevation) belonging to the elevator class.

```
CREATE TYPE actuator (name CHARSTRING);

CREATE TYPE rotator ( actualRotation REAL,
                      referenceRotation REAL)
  SUBTYPE OF actuator;

CREATE TYPE elevator( actualElevation REAL,
                      referenceElevation REAL)
  SUBTYPE OF actuator;

CREATE TYPE gripper(actualGripperPosition
BOOLEAN,referenceGripperPosition)
  SUBTYPE OF actuator;

CREATE TYPE arm (actualExtension REAL,
referenceExtension REAL)
  SUBTYPE OF actuator;

CREATE TYPE robot (
           /* part-of relations */
           gripper gripper,
           upperarm arm,
           lowerarm arm,
           robotBase rotator) SUBTYPE OF actuator;

CREATE gripper INSTANCES :gripper1,:gripper2;
CREATE arm(gripper) INSTANCES
                 :upperarm(:gripper1),
                 :lowerarm(:gripper2);
CREATE rotator INSTANCES :robotBase;
```

```
CREATE robot(robotBase,upperArm,lowerArm)
  INSTANCES :robot1(:robotBase,
                    :upperArm,
                    :lowerArm);
ADD TYPE elevator to :robot1;
```

Figure 4.5: Example of an object class taxonomy

4.5 Use of Queries

This section contains information on how conditions and expressions in CAMOS(L) are translated into queries and active rules. Queries are the only way to retrieve information about the environment and executing processes.

There are six places where queries are used in the execution of CAMOS(L) programs (see Section 4.3.1).

- Wait conditions of composite operations
- Iteration conditions of iterative operations
- Selection of sub-operations
- Update operations for the world model
- Evaluation of parameters in the invocation of operations
- Definitions of constraints for nodes in DFA state machines

These queries are compiled and stored in the database.

Formal parameters

The processing of queries is complicated by the presence of formal parameters in CAMOS(L) operations. The example in Figure 4.7 below illustrates how a wait condition (see Figure 4.3) can be rewritten into a form that use a stored process representation (the instantiated operation) to make parameters available to the query processor. It consists of two parts: a database query (function cellReady) over the WM which yields a boolean result and an active rule (rule cellReadyCondition) controlling the operation execution.

```
CREATE FUNCTION cellReady(operationInstId id) AS
  SELECT TRUE WHERE
      partsat(arg(id,3)) AND
      position(arg(id,3))="open" AND
      palletCapacity()>=COUNT((SELECT b
          FOR EACH part b
            WHERE
              name(position(b))=currentPallet()));
```

Figure 4.6: Wait condition translated into a query

```
CREATE RULE cellReadyCondition(opInstId id) AS
  WHEN active(id) AND cellReady(id)
    DO BEGIN
        SET status(id)=:running;
        ADD currentProcesses()=id;
      END;
```

Figure 4.7: Active rule monitoring a wait condition

The `create function` command (Figure 4.6) instructs the AMOS DBMS to create and compile a new query. The query parameter (`opInstId`) refers to a database object that represents an executing operation. This operation instance object allows the transfer of formal parameters to the rule processor. The `arg` vector stores the formal parameters of each executing operation in the CAMOS system.

4.6 Use of transactions

This section describes why traditional transactions cannot be used for manufacturing operations. There is a clear difference between a conventional transaction that only modifies the database and a transaction that affects physical processes in the environment. This complicates the use of transactions since there is no **undo**, in the conventional transaction sense.

Suppose a database update transaction initiates a control algorithm which drills a hole in a metal blanket. The undoing of this transaction would not only involve undoing the changes in the database, but also doing compensating actions in the environment such as welding the hole [Siv92].

It is common that several CAMOS(L) operations cooperate in achieving a goal. The operations manipulate the world model and wait for physical processes in the environment. The classical transactional approach of establishing locks on objects involved in a transaction cannot be used for the world model, since there are long-running operations which potentially involve large parts of the world model [Reu95]. The use of a normal transaction mechanism provides little concurrency and make it impossible to use cooperating operations [Shu92]. It is also unacceptable to abort or simply delay execution if concurrency problems, in the traditional transaction sense, occur . Considering the characteristics of the world model, the following conclusions can be reached:

- The operations in the database are short lived, but they are associated with physical processes in the environment which are long lived.
- The isolation property of transactions is not desirable for CAMOS operations, since the operations must be able to interact through the world model.
- The problem of operations needing exclusive access to objects in the world model cannot be solved by a database locking mechanism in the database. There is no automatic mapping between database locks and locks on objects in the environment. Exclusive access can be guaranteed by predicate locks or wait conditions.
- Rollback of physical processes is not always possible to accomplish. Even acceptable compensating actions can be hard to find.

Consequently , the correct correctness criterion for CAMOS operations cannot be serializability . The concurrency problem has to be solved in more application-dependent ways where conflicts are handled explicitly instead of aborting when problems arise. There are several properties of CAMOS(L) programs that can be helpful in achieving this. For example, all update transactions are predefined as opposed to normal transaction systems where the transactions are not known in advance.

4.6.1 TRANSACTIONS IN THE EXECUTION MODEL

CAMOS operations describe activities resulting in several updates to a world model. The CAMOS operations themselves are not encapsulated in transactions. Instead, the database transaction mechanism is used to ensure atomic updates of the database world model. Each update of the world model in the database is run in a separate regular transaction. CAMOS uses a simplified model of sequences of atomic transactions. This means that CAMOS operations do not execute in isolation (in a database sense) but all modifications of the world model are executed atomically , consistently , and persistently .

The CAMOS(L) language provides a rich set of constructs for coordinating and synchronizing both high-level operations, control algorithms and physical processes in the environment. The constructs should be used to avoid conflicts in the processing of operations.

4.7 The Runtime Environment

This section presents how the execution of the control application language is carried out using the active rules. First we describe how the operation manager handles operation instances (processes).

The operation manager consists of an interpreter , a scheduler , and a rule coordinator .

The interpreter processes operation instances in the database, where each operation instance corresponds to an executing operation. The interpreter in the operation manager is based on the previous ARAMIS interpreter developed at RTSLAB at Linköping University [Hol92]. The scheduler organizes operation instances in different queues depending on their current status.

As mentioned earlier in this chapter , there are three types of operations: composite operations, iterative operations, primitive operations. Each operation type has its corresponding operation instance type. When a composite operation instance is processed, it will initiate the selection of sub-operations according to the operation definition and creation of operation instances for these. All sub-operations in the operation definition are tested and those sub-operations that have satisfied sub-operation conditions are selected for execution. Each time a sub-operation is selected, an operation instance is created to control the execution of this particular sub-operation. The operation instance is discarded when the operation has terminated, unless the operation is iterative.

It is possible and even likely that there will be several concurrently executing operation instances for each operation defined in the system. The operation manager maintains the set of operation instances which refer to operation definitions and queries associated the operation. When a primitive operation instance is selected for execution, the corresponding update transaction will be run, which will update the database world model.

4.7.1 DESCRIPTION OF STATE TRANSITIONS

Active objects are separated from passive material in the world model (database model of the environment). The active objects react to changes in the world model and are coupled to algorithms in the real-time server . A typical active object can be a piece of machinery or equipment, whereas passive objects are nuts and bolts.

There is no fundamental difference between passive objects (materials) and active objects (machinery) except that passive objects cannot react to changes in the world model.

A set of state variables are associated with the real-time server control algorithms for equipment and machinery . A subset of these variables (called exported variables) is maintained in the database, thus introducing abstraction and separation of shared data in the database and local data in the control algorithms. The "actualRotation" in Figure 4.5 is an exported variable.

The interface to the real-time server involves checking whether a transition belongs to a legal sequence of operations on an object, finding the appropriate control algorithm, transferring the data to the real-time server with different formats, and receiving results from terminating control algorithms.

The objects with active behavior in the environment are modeled as deterministic finite state automatas (DFA), which introduce restrictions on the usage of the object. The nodes of the DFA characterize different states of machinery or different phases of processes in the external environment. Each node in a DFA contains a node constraint and the transitions are associated with control algorithms in the real-time server that will take the modeled object from one state to another . When a sequence of database updates satisfies a node constraint, the algorithm associated with the transition to this node will be scheduled for execution.

If there is no transition to the node with a satisfied node constraint or there is no node with a satisfied constraint, then an error handler must be invoked. When the algorithms terminate, they report the new values of the exported state variables to the database.

In this way, the DFA describes restrictions on legal transitions for active objects, and therefore restricts the possible sequences of algorithms that can be executed.

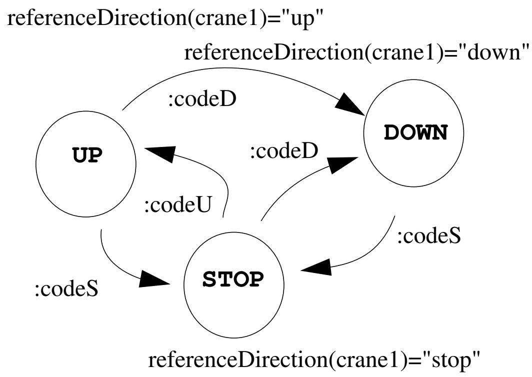

Figure 4.8: Crane modeled as a DFA

As an example, consider the crane depicted in Figure 4.8. The crane is represented by an active object with three nodes and five transitions. A set of database queries is associated with each node. The transition is enabled if the query evaluates to true. These queries can involve objects in the world model as well as other information in the database. A transition occurs if there is an enabled transition from the current node. The programmer must construct the transitions and queries so several transitions cannot be simultaneously enabled.

In the example in Figure 4.8, the directions of the crane motor (up, stop, down) can be directly reversed if it is going up. If it is going down the motor must be stopped before the direction can be reversed.

A DFA is modeled in the database as a set of nodes, a current node, and a set of active rule activations that are responsible for the state transitions in the DFA. The conditions of the active rules consist of several parts.

The conditions must test whether there exists a satisfied node constraint, if the object is already executing a control algorithm, and if the current node satisfies certain constraints. The action part of the active rule sends a message to the real-time server to initiate a previously downloaded algorithm. In this way, the objects in the world model are monitored and controlled by the active rules.

The transition rules are parameterized. The same generic rule will be used for all transitions. In AMOSQL, the activation of a parameterized rule corresponds to the creation of a rule instance with bound parameters [Ris93]. Figure 4.8 shows how the DFA is represented in the database.

In summary, the invocation of control level algorithms is not directly expressed in the CAMOS(L) control application language. Instead, CAMOS(L) interacts indirectly with the world model using the primitive operations (update transactions). This separates the control view of the object and the operations the object is participating in.

4.7.2 REPRESENTATION OF THE EXECUTION STATE

In the implementation of the CAMOS system, the execution state of manufacturing operations is modeled in the database. The state of each executing operation is represented by operation instance objects in the database. This opens new possibilities for monitoring the control system. The following types of queries can be made:

- Why has the execution stopped?
- How many welding operations are currently active?
- Is any instance of this operation running?

The state of the execution is saved in the database for three reasons. First, it provides some degree of software fault tolerance, so that the execution state is not completely lost if the system crashes or the power fails.

Second, it becomes possible to examine the state of the system in case of a failure due to a software error . The database query facility can be used to pose queries in terms of the defined operations. For example, if the execution is halted, then it is possible to detect which operations are suspended, and which environment processes or algorithms in the real-time server they are waiting for . It is even possible to query about the binding of the formal parameters in the invocation of executing operation instances.

Third, constructs with expressions and conditions involving details of the execution state will be possible as the state of each operation instance is accessible from CAMOS operations. This type of construct is generally considered to be bad programming practice, but it provides a flexible way to monitoring operations.

4.7.3 EXECUTION CYCLE

The interpreter creates an operation instance object for every composite and primitive operation, which is then scheduled by the scheduler . A terminating operation implies deletion of the corresponding operation instance. Two basic synchronization mechanisms depend on the rule check phase:

- Rules that monitor the transitions in a DFA must be able to detect and react to changes in the world model. Therefore, primitive operations must be followed by a subsequent rule check operation in the database.
- A rule check operation must also be performed after the control algorithms have terminated and reported the exported state variables. This will enable the operation wait conditions to trigger .

Since the rule facility supports deferred semantics, the execution model contains two commit points and therefore two rule processing points. The operation manager repeatedly carries out the following sequence:

1. Communicate step
2. Update world model
3. Rule processing (wait conditions)
4. Dynamic scheduling
5. Processing primitive operations
6. Rule processing (DFA rules)
7. Creating operation instances
8. Rule coordination

In the communication step, the real-time server notifies the CAMOS operation manager about terminated control algorithms and changes in the physical environment. In step two, the update world model step, the world model is updated according to the changes reported from the real-time server . This is followed by rule processing where wait conditions are checked (step 3). The operation manager schedules for execution all operation instances that have satisfied wait conditions (step 4). Any pending primitive operations (database updates) are processed in step 5.

This will result in changes in the world model. In the sixth step, the rule processing is initiated once more, this time to check if any DFA transitions have been enabled. Finally , the operation manager selects and initiates a new composite operation (step 7) and notifies the rule coordinator to activate appropriate rules in the database (step 8).

The operation manager repeats this process until there are no queued messages from the real-time server and there are no executable operation instances in the scheduler queues.

4.8 Current implementation

Basically , a control application system and a database system can be combined into a single unified architecture in the following two ways:

- An embedded approach: by embedding the database queries and the active rules into the control application. The database will be a tightly integrated system function from the control application point of view.
- A loosely coupled approach: by coupling database and control application components as parallel, communicating subsystems. In this approach the database is isolated from the control application but the systems can communicate and influence each other.

The advantage of the embedded approach is that the control system can interact efficiently with the components of the database system (the transaction manager, the query processor, the storage manager). To achieve tight integration, the architecture of the database system must be open. The advantage of the loosely coupled approach is flexibility. The control system could interact with several different database systems and there is no need for the database to run on the same processor. Unfortunately, there is a potential for poor performance in the loosely coupled approach due to the communication overhead. Typical high-level operations interact often with the world model stored in the database. How efficient this type of integration will be depends on the capacity of the communication channel and on the amount of information that has to be exchanged.

Another problem with the loosely coupled approach is that query results cannot be used directly without translation of the data formats and transmission of messages. It is also difficult to implement features that require direct access to database internals, such as application specific concurrency control, transaction models etc.

In the CAMOS project, it was decided to embed the object-relational database AMOS in the control system architecture (the first alternative). A control application language was built around the query processor and the active rule facility in AMOS.

The database operates in the same address space as the control application. In particular , the aim was to overcome the impedance mismatch problem between traditional programming languages and database management systems by providing a unified type system and tight integration of the query processor .

4.8.1 PARTITIONING THE SYSTEM

The current implementation is composed of two simultaneously executing processes: the operation manager process including the database; and the simulator for control algorithms and environment. These processes operate asynchronously and communicate though UNIX pipes.

Controller protocol

An asynchronous ordered queue of messages is used as a communication and synchronization medium between control algorithms in the real-time server and high-level operations in the operation manager .

Status messages are sent from the real-time server to the operation manager when terminating or aborting control algorithms. The control algorithms change the physical environment. Therefore, the message interface needs to include information on the changes in the environment. The following runtime events/messages are used:

- Start algorithm (from operation manager)
- Terminate algorithm (from operation manager)
- Aborted algorithm in real-time server (from real-time server)
- Update world model (from real-time server)
- Acknowledge algorithm completion (from real-time server)

The *start algorithm message* is sent from the operation manager to the real-time server to initiate a control algorithm. Algorithms are stopped using the *terminate algorithm message* . This message is used at system shutdown.

The message associated with aborting control algorithms is sent to the operation manager to allow error handling. An earlier thesis written at RTSLAB contains a discussion of error recovery techniques [Lob94]. The fourth message type represents changes in the environment that should be propagated to the world model in the database. This is a write-only update transaction [Ade95]. The last type of message informs the operation manager that a control algorithm has terminated in a normal way. This allows temporal dependencies to be resolved and new operations to be scheduled.

Chapter V
Rules in Control

This chapter discuss how the semantics of the active rules affects an integrated control system based on active rules. In particular, the coupling modes and the rule ordering mechanism are investigated.

The following terminology is used in this chapter. An **active rule** monitors the information in the database. Active rules can be **checked**. When the **condition** of the active rule becomes **satisfied**, the rule is **triggered**. Triggering results in processing of the **action** part of the rule. A rule is only triggered if it is **activated** (enabled for triggering).

5.1 Rule semantics

The general problem of monitoring a large set of rules is to know when to check the rules. There are many semantic choices when designing a rule processor. This chapter presents how these choices affect an integrated control system. The first section discusses how coupling modes and transaction contexts of active rules influence the semantics of an integrated control system.

In the following sections issues of integrity constraints in control systems are examined. The last section in this chapter describes the choices made in the current implementation of CAMOS.

5.1.1 ECA OR CA — DOES IT MATTER ?

This section presents two paradigms for active rules (ECA and CA) and discusses coupling modes and how they affect an integrated control system based on active rules.

The most common paradigm in active database management systems is the event-condition-action paradigm (ECA). An ECA rule has three parts:

Event: Which events will cause the rules to be triggered.

Condition: What condition should be satisfied to execute the action.

Action: What to do when the event arrives and condition is satisfied

Typical events are data modification events, time events, and external events, *e.g.,* from the control system. When an event occurs, all rules that are specified to trigger on that particular event will check their conditions. In most systems, the conditions are expressed as database queries. If the condition is satisfied, the ECA rule will process the action part of the rule.

Another rule paradigm is the condition-action (CA) which omits the event part of the rules. A CA-rule has two components.

Condition: A declarative expression defining in which situations the rule should trigger .

Action: What to do when the rule triggers.

This resembles the production rule system of expert system shells. In many situations, the CA paradigm leads to more compact rules, since the system derives the relevant events from the monitored condition. The programmer does not need to specify which events are relevant for the rules.

Consider the following rule: “Do not manufacture any parts on press “c” on a holiday, unless there is a supervisor present at the plant”. The rule expressed in CA-format is shown in Figure 5.1 below:

```
CREATE RULE dontPressOnHoliday() AS WHEN
SOME((SELECT s FOR EACH schedule s,
                        holiday h,
  WHERE name(type(machine(s)))="press"
    AND name(machine(s))="c"
    AND NOTANY((supervisor(plant(machine(s)))))
    AND date(s)=h))
DO ABORT;
```

Figure 5.1: Translation to CA rule

The ECA rule corresponding to the rule in Figure 5.1 contains 15 possible events that could trigger the rule (see Figure 5.2 on page 78). In the ECA model, it is easy to limit relevant causes. The programmer simply specifies exactly which events should trigger rules. It is impossible to limit events in the same manner for rules based on the CA-paradigm, since the conditions cannot access the events. This is why CA-rules are generally considered more declarative and powerful, but less expressive in their nature.

There are proposals to allow the conditions in CA rules to use information in the internal database event log [Dit95]. A rule could then access the event log to determine which event triggered itself. Note that both the declarative CA semantics and the event-based ECA semantics is suitable for detecting changes in the database world model used in this thesis. The CAMOS system is based on the CA model although an ECA based model would be equally efficient.

```
ON add(schedule), update(schedule), delete(super-
visor), update(supervisor), add(holiday),
update(holiday), add(type), update(machine)
add(machineName), update(machinename),
add(typeName), update(typeName), delete(plant),
update(plant), update(date) WHEN

SOME((SELECT s
       FOR EACH schedule s,
                holiday h
  WHERE name(type(machine(s)))="press"
  AND name(machine(s))="c"
  AND NOTANY((supervisor(plant(machine(s)))))
  AND date(s)=h))

DO ABORT;
```

Figure 5.2: Translation to ECA rule

In many manufacturing applications the number of rules would be larger using a system based on ECA rules. Since the rules are automatically generated from CAMOS operation descriptions, the use of ECA rules would not complicate the programming of the control system.

5.2 Coupling modes

Active database researchers use the term *coupling mode* [Day88] to describe when and how events, conditions, and actions relate to each other. The coupling modes affect the semantics of the rule processing and therefore also the semantics of an integrated control system.

5.2.1 RULE TRIGGERING

The first issue of coupling modes is the time of triggering for the rule. There are four possibilities:

- Immediate: the rule processing starts immediately upon an event occurrence.
- End of database operation: the rule is delayed until the end of the database operation. In this alternative, the rule is processed before the next database operation.
- End of transaction: triggering of the rule can be delayed until the end of the user transaction.
- After the transaction: the activation of a rule can be delayed until after the end of the transaction.

There are implementation and efficiency problems with the immediate mode. Consider a transaction consisting of a sequence of database operations. An immediate rule could trigger at any database operation. Therefore, the rule conditions must be monitored for each database operation [Skö96]. The performance of monitoring is better when the rules are triggered at the end of a transaction or after the transaction. The last alternative does not support rollback of the triggering transaction. We choose the *end of transaction* alternative, also called *deferred* rule execution. Deferred rule triggering provides efficient and flexible support for an integrated control system.

5.2.2 TRANSACTION CONTEXT

Another issue related to the coupling modes is the transaction context. The rule actions can be executed in the same transaction or in a separate transaction (detached coupling mode). If the actions are executed in the same transaction, they will commit or abort together with the triggering transaction. If the actions run in a separate transaction, then they will continue regardless of whether the triggering transaction aborts or commits. With separate transaction contexts, it is not possible to create integrity constraint rules that use the *abort* command to maintain integrity. For example, the rule below ensures that no crane will attempt to lift more than its maximum load.

```
CREATE RULE maxLoad() AS
FOR EACH crane c
WHEN weight(liftedObject(c))< maxLoad(c)
DO ABORT;
```

Figure 5.3: Integrity rules protect machinery

In some situations, the triggered transaction **must** execute in a separate context. For example, if attempted violations of constraints were to be recorded in a log and the recording takes place in the same transaction, then the record of the violation would be rolled back if the violating transaction was aborted. Thus, no record of the attempted violation would be left. The rule in Figure 5.3 will not work if the rule action part is independent.

In summary, the rule processor should support both transaction contexts.

5.2.3 RULE ORDERING

If two rules are triggered at the same time, the system has to determine the ordering of the rule processing of conditions and actions. The rule processor can use a variety of rule ordering principles:

- Only one rule is processed. The other rule is ignored.
- Both rules are processed in random order.
- Both rules are processed according to predefined priorities.
- A randomly selected rule is processed and then the other rule is checked. If still triggered, it will be processed.
- The situation is treated as an exception. No rules are processed.

Each possibility may result in a different database state. Recall how active rules are used in the CAMOS architecture:

- Monitoring the world model for the purpose of initiating algorithms in the real-time server
- Monitoring the world model for the purpose of synchronizing the operation manager with the processes in the external environment
- Integrity constraints

In the first case, a situation where two or more rules trigger is possible and even likely. Updates in the world model may trigger transitions in several DFAs. The only possibility here is to processes all the rules. Naturally, some transitions of active objects in the controlled environment may be more time-critical than others (*e.g.,* emergency shutdown). Here priorities could be used to determine in which order the DFA rules are processed so that the critical transition rules are processed first.

In the second case, an update in the world model notifies the operation manager that some operations should be scheduled for execution. These operations have been pending due to unsatisfied wait conditions. Here, the ordering of the rule processing is less important, since the scheduler in the operation manager will determine in which order the pending operations will be executed. In the CAMOS system, there is no situation where the processing of one rule could inhibit the triggering of another rule, since the action part of the rules only contains callbacks to the operation manager, and no updates that could trigger yet another rule.

The only situation in the rule processing that could reverse the triggering of a rule is an update in the database that undoes the satisfied condition of the other rule. This will never occur in a system where all updates are scheduled by an external operation manager.

From a control system perspective, it is never desirable to ignore any triggerings in the system. Priorities may be used to impose a partial ordering on the rule processing when needed.

5.3 Integrity of world model

It is important to avoid machine failures and other disastrous or hazardous situations. This section describe how active databases can enforce safety and integrity constraints in control applications. W ithin CAMOS, active rules can be used to specify integrity constraints in the world model. The rules can express generic safety conditions to avoid, for example, overload situations and collisions.

Monitoring can be performed at the operation level and at the world model level. The operation level constraints are already supported by the DFAs defining the behaviour of individual equipment or machinery .

There is also a need to specify global integrity constraints in the world model that are not restricted to operations on isolated objects. This type of constraint is more difficult to monitor and maintain since they cannot be checked in any single operation. The constraints may also involve several objects. It has been long recognized that active rules can be used to support database integrity constraints [W id96]. The same type of active rule can now be used to provide integrity constraints over the world model.

An example of a dynamic integrity constraint is that a robot arm should not be moved into a closed press. The constraint involves the press, both robot arms, and all operations that control these objects. Figure 5.4 shows how the integrity constraint is coded as an active rule.

```
CREATE RULE protectRobot(r)
AS WHEN ((SELECT TRUE FOR EACH press p, robot r
    WHERE actualPosition(p)="closed" AND
            actualRotation(arm(r))>= 70.0 AND
    actualRotation(arm(r)) <= 140.0)) DO ABORT;
```

Figure 5.4: Active rule for integrity constraint

The rule triggers in the situation when the press is closed and rotation angle of an arm of a robot is between 70.0 and 140.0 (the press zone) [1]. If the situation occurs, the active rule will issue an `ABORT` command, which rolls back the transaction that caused the situation.

5.3.1 PROCESSING OF INTEGRITY RULES

The presence of integrity rules complicates the rule processing, since other active rules coexist in the database. The behaviour of the system depends on which order the rules are checked in. Conflicts with active rules associated with wait conditions and DFAs must be avoided. It is critical that integrity rules are checked and processed **before** any other rules are allowed to execute. Consider the DFA rules responsible for start of algorithms in the real-time server . If an update violates an integrity constraint, the integrity rule can rollback or repair the world model before any transitions occurs. This is only possible if the integrity rule is processed before the DFA rule. If the integrity rule is not checked **before** all the DFA rules, then a DFA rule could have initiated processes in the real-time server . This would lead to problems with rollback, since the real-time server affects the physical environment. A rollback at this stage, leads to an inconsistent world model.

As shown earlier , coupling modes [W id96] can be used to control when rules are triggered. If the integrity rules are checked using an immediate coupling mode, and the DFA rules operate using a deferred coupling mode, everything will work satisfactorily since the immediate constraints are checked after the execution of every database operation and the DFA rules are processed at the end of the transaction. Unfortunately , immediate rule processing means low performance. A better alternative is to use the dynamic rule contexts proposed in Section 5.6.

1. In real applications, values with finite domains are preferred, since they simplify programming and analysis.

5.3.2 COUPLING MODES OF INTEGRITY RULES

This section examines how coupling modes affect the semantics of the integrity rules. There are three different types of integrity constraints:

1. Aborting constraints: Restore the state of the world model if an operation violates a consistency constraint.
2. Compensating constraints: Adjust the world model to an appropriate state without canceling the violating operation.
3. Reacting constraint: Do something completely different, such as recording the violation on a log.

Modeling the action part of the integrity rule as an independent transaction implies that the logging and recovery of both the triggering and the triggered transaction are independent. This leads to problems with *aborting constraints*, since the rollback of the triggering transaction cannot be initiated by the integrity rule. The independence work both ways: if the triggering transaction decides to abort, the triggered transaction (started by the integrity rule) will not abort. The same applies to the compensating constraints.

Reacting constraints are preferably run in a separate transaction. Consider a set of reacting constraints that notify an operator of violations. The attempted violation would be reported regardless of the commit or abort of the violating transaction.

5.4 Rules are not a programming language

Let us focus on the maintainability of rule bases and the controlling software. Consider a system with active rules in a database and external programs that update this database. The active rules notify the external programs of changes in the database. The maintainability of such a system involves many intricate details:

- How is the total system affected if some external programs are changed?
- How is the total system affected if some of the active rules are changed?
- How easy is it to program a system consisting of both traditional software and active rules?

These question are not easy to answer. Clearly, rule interaction cause problems in system development. It is difficult to realize how the change of a few rules affects the total system. This problem becomes even greater when the rules interact with other software. Generally, the ECA and CA rules are too far from the application to be useful as a programming language for control applications. The programmer needs a programming language that is easier to program in than active rules.

Applications with interacting rules are hard to develop in a system with a flat active rule structure. Even a very small number of dependencies between rules will make it difficult to maintain the system [W id94]. There is also the issue of interactions between the controlling software and the active rules. All these interactions makes the development and maintenance of an integrated control system difficult.

The CAMOS system solves these problems by introducing a new programming language, CAMOS(L). The idea behind CAMOS(L) is to use one unified language to express operations in the control system. The language hides the active rule facility from the programmer, and uses the active rules as a database assembly language. Thereby, the abstraction level is raised to concepts such as synchronization of operations, temporal dependencies, and updates of the world model.

The actual condition-action rules are derived from the operation definitions written in CAMOS(L) and therefore hidden from the programmer. This approach may result in easier maintenance, better understanding of the control flow and absence of undesired and unanticipated interference between the application programming language and a rule system in the database.

5.5 Rule efficiency

Control applications are highly dynamic. This section discusses how a rule coordinator improves the efficiency of active rules in dynamic control applications.

Many control applications continuously change focus resulting in frequent activation and deactivation of rule sets. Considering the number of rules needed in a manufacturing control system, it would be inefficient to use active rules alone to control the processes.

Four measures were taken to improve performance of the CAMOS system:

1. Declarative queries are used where possible
2. Stored procedures are used where (1) is impossible
3. Active rules are used only when (1) and (2) are impossible
4. Rules are active only when needed.

These design principles reduce the workload for the rule engine in the database. Rules are only responsible for detecting and reacting to complex conditions in the database that otherwise requires extensive polling of the database. Several synchronization points in the execution model can be managed with wrappers, rendezvous techniques or messages instead of active rules. In the current implementation, active rules only monitor wait conditions and the transitions in the DFAs.

5.5.1 RULE COORDINATION

The approach has been to use the rule system, but make necessary improvements of the functionality and the performance. The improvements include a new rule coordinator ,which ensures that rule instances are active only when they are needed.

In this way, the control system benefits from the dynamic nature of the applications and the procedural description of operations. In CAMOS, active rules associated with wait conditions are activated when an operation is initiated and immediately deactivated when the rule triggers (trigger-once rules). Only rules associated with currently waiting processes will be active. The limited number of simultaneously activated rules reduce the workload for the rule engine. The rule coordinator maintains a table of all rule instances, activating and deactivating the rules as required. The table contains the parameters for the rule instance and the rule status (activated/deactivated).

5.5.2 QUICK ACTIVATE AND DEACTIVATE

Each time a rule is activated, the AMOS rule facility creates data structures for book-keeping and incremental evaluation of conditions. This is a costly procedure. There is a way to avoid repeated activations of rules. Note that the number of possible rules is fixed for a given CAMOS(L) program. By providing a pool of activated reusable rule instances, the performance of the system can be improved.

To provide a mechanism to turn on and off the reusable rule instances, the rule conditions of the rule instances are augmented with boolean-valued flag functions. Using the flag functions the rule coordinator can temporarily turn on and off rule instances without using the costly activate/deactivate statements in AMOSQL. To ensure efficient checks of the flag functions, the query processor is instructed to check the flag before processing the actual rule condition. The approach will imply a small performance degradation during the rule check phase while the flag functions of deactivated rules are checked.

5.5.3 EXECUTION OF OPERATIONS

At the time an operation is scheduled for execution, the parameters are transferred to the rule coordinator and an appropriate reusable rule instance is activated.

The individual wait condition rule instances are derived from the same generic rule.

The instances have individual parameter bindings and an associated operation to which the wait condition belongs. The rule instances can be reused if an intermediary table is used to store parameters and operation identity . A predefined number of inactive rule instances can in this way be created at compile time, which will reduce the number of costly activate statements at runtime.

Typically only a small number of the wait condition rules is active simultaneously . In CAMOS this is related to operation invocations; when an operation is invoked, the scheduler activates a rule instance to monitor the wait condition and the rule is deactivated directly after is triggered. Built-in system support for rule contexts would make it possible to check only the rules in currently active rule contexts. This mechanism is further discussed in a joint paper from RTSLAB/EDSLAB at Linköping University [Skö95].

5.5.4 AVOID CASCADING RULE TRIGGERING

Systems involving active rules must also deal with the cascading rule triggering problem. There are situations where one rule may introduce a change in the database that activates another rule, which may introduce yet another change and so on, resulting in unpredictable response times. In the current implementation of CAMOS the action part of active rules are foreign function calls that will reschedule operations in the scheduler . The problem of cascading rules is avoided because none of the generated rules contain database updates. Therefore the rule processing cannot trigger other rules in the system.

5.6 Rule groups

The dynamic nature of control applications and the experience of developing CAMOS inspired us to develop *dynamic rule contexts* in cooperation with EDSLAB at the Department of Computer and Information Science at Linköping University . This section briefly describes the idea behind the rule contexts. For a more detailed description, see the proceedings of RIDS'95 [Skö95].

Active database systems for control applications must continuously check the rules, but the rules are used by several users and processes. The application requires a mechanism to control which rules are checked and when. In our application area the rules are highly dynamic, *i.e.* different items should be monitored at different steps in the manufacturing process. This results in a high frequency of activation and deactivation of rules.

In the present system, boolean-valued flag-functions are used to implement efficient rule groups. Dynamic rule contexts offer a general solution to these problems. The idea is to place rule instances into contexts that can be rapidly activated and deactivated. Activation of a context enables triggering of all rules placed in that context. The contexts are also associated with a dynamic coupling mode. This mechanism replaces the unintuitive flag functions mentioned in Section 5.5.2.

The dynamic rule contexts group operations into modes where the domain knowledge is separated from the control knowledge. Rule contexts can be used to organize rules into context hierarchies. This mechanism will provide valuable support for the rule coordinator in CAMOS.

5.7 Conclusions

This section summarizes the experience of using active rules in a control application setting. The main problem of active rules are: efficiency , rule interactions, and flexibility in a dynamic situation.

Our solution is based on an operation manager . The operation manager contains a rule coordinator that simplifies the management of active rules in control applications. First, the rule coordinator dynamically controls the status of the reusable rule instances (activated/deactivated). Only a small subset of the rule instances is kept activated. The reduced number of rules improves the efficiency of the monitoring.

Second, the rules only *monitor* the world model. This eliminate problems associated with rule cycles, rule interaction, and cascading rule triggering, since the rules are not allowed to update the world model. Naturally , this approach only moves the problem of cycles and interactions from a flat rule setting to the CAMOS(L) setting. Poorly designed CAMOS(L) programs can result in similar problems, but the programs are easier to analyze. Petrinet-based techniques has been proposed for the analysis of manufacturing operations expressed in a manner similar to CAMOS(L) [Lob95].

Third, the deferred coupling mode provides an efficient and flexible mechanism to check DFA-rules as well as wait-condition rules.

The introduction of an external operation manager allow several rule paradigms, coupling modes, and rule orderings principles to be used. Special attention must be given to the processing of integrity rule. These must be check before any other rules are processed.

Chapter VI
Requirements

The work presented in this thesis requires functionality and performance not found in regular databases. This chapter presents the requirements on the database management system. It is divided into three parts: general database requirements, requirements on the active rule facility , and performance requirements. In summary , the database must support object-oriented modeling, dynamic active rules, integration facilities, and high performance.

6.1 General database requirements

- Complex modeling of the controlled environment requires an object-oriented data definition language (DDL). To avoid unnecessary join processing, the DDL must be able to express composite object without joins.
- The integration of a database system and a control system require efficient facilities to tightly interface the query processor with external applications. The results from queries should be directly accessible without translation of data formats and message passing. Data in the database must be

accessible directly from the language used to implement the operation manager without using queries. The same information should also be accessible from the query language. The requirement of multi-language access to the information in the database has been pointed out in [Sch93].
- A foreign function interface is needed. In many situations the database need to access data and programs outside the database (communication manager , operation manager)
- The DBMS must support fast and short transactions. The standard transaction model can be used to implement short transactions.

6.2 Requirements on the active rule facility

- Efficient change monitoring techniques. The main motivation for using active rules in the execution model is that rules can handle operation dependencies involving large volumes of data efficiently . The incremental evaluation technique [Skö94][Skö96] used by AMOS rules is more efficient than repeated polling or simple indexing of rule conditions.
- Fast rule context switch. The rules in a control system are not static. Since activation and deactivation are very frequent operations in the control system architecture, they must be efficiently implemented.
- Facilities to tightly integrate the rule processor with applications. The active rules should be able to signal back to applications.
- Clear semantics of the rule processing.

It is also important to be able to extend the database system with specialized data structures to model more closely the domain of control applications (processes, operation descriptions and so forth). New data structures will enhance the performance of an integrated control system. The application-specific storage structures should be implemented at the lowest possible level to ensure best performance. The performance is essential if the database is used to store information about processes. Our implementation of CAMOS uses specialized ordered collections to represent schedules for executing processes.

The database must support extensions to query analysis, optimization, execution strategies, data access methods, and storage methods. AMOS is extensible with new data types using a foreign function interface. To provide the additional functionality and data structures efficiently, the new dedicated access methods have to be incorporated into the query optimizer. This is achieved using the open cost model in AMOS [Lit92].

6.3 Performance requirements

A computer-based control system must sense the environment and directly influence it through actions. Such systems are subject to time constraints related to the environment in which they operate. For instance, failures, such as dropped objects, may occur unless a conveyor belt is stopped quickly enough.

The thesis takes the view that the timeliness of the high-level operations involving large volumes of data is normally not a critical issue. Real-time system research has focused on developing mechanisms to support predictable execution of periodic control tasks with minor data dependencies [Bur90]. These traditional ways of handling time-critical operations are incorporated into the real-time server in the CAMOS architecture, whereas the high-level operation manager does not provide any guarantees for timeliness. This puts the responsibility on the programmer to decide what operations should execute in real-time.

Nevertheless, conventional disk-based database systems are too slow for extensive use in control applications. The main problem arise in the slow and unpredictable disk-accesses. The database must be very fast to handle the stream of updates from the real-time kernel.

A promising alternative is object-relational main-memory databases where all data are kept in main-memory.

6.3.1 DISK-BASED VS. MAIN-MEMORY DATABASES

Even if a disk-based system had enough RAM to hold all the data in main-memory, it would still be slower than a main-memory database [Hey95]. The reason for this is the underlying assumption that data is stored on disk for the greater part of time. All storage structures, indexing methods, and execution plans are optimized on the basis of this assumption. By contrast, main-memory databases assume that data is in main-memory for the greater part of the time. This way, we can eliminate many of the checks and copies from buffers that must be made in disk-based systems. The query optimization can be optimized for main-memory access.

In the past, main-memory databases have been limited to research prototypes, but several factors suggest that they will be a better choice for control application than the conventional disk-based database systems. The dropping prices and the increased density of RAM chips make it possible to build computers with several gigabytes of main-memory. Improved RAM designs and new battery back-up memories have increased the mean time to failure (MTTF) to the same region as disk-based storage. Some major vendors of control systems for industrial applications consider RAM more reliable than disk-based storage. Furthermore, 64-bit addressing means that main-memory databases are no longer limited to small address spaces.

All these trends, in combination with the superior speed of access, suggest that main-memory databases are a good alternative for control applications.

Chapter VII
Thesis summary

An increasing number of control applications require database-like functionality. The management of data becomes a problem when the control applications grow, become complex, and operate in environments with large volumes of data. CAMOS (Control Application Mediator Object System), presented in this thesis, aims at an efficient and general approach to modeling and data management in control applications, where the data modeling language of the object-relational database AMOS is used to model and organize information about the environment. The benefits of the CAMOS architecture are:

- Organization of process data in an object-oriented world model
- Access to a query language and data definition language for the world model
- Uniform access to information (process data, conventional data)
- Support for monitoring of the world model using active rules
- Providing manual intervention and queries with a high-level query language

This thesis documents the experience of using active rules in control applications. One conclusion is that active rules are low-level mechanisms too far from the control applications to be used as a programming language. The rule processing in its present form faces difficulties which make it unsuitable for modeling complex control applications. These problems stem from the inability to cope with highly dynamical rule sets. To overcome these problems, the thesis contains a proposal for a specialized control application language and a combination of active rules and a supervising operation manager to implement the event based part of the control system architecture. The control application language, CAMOS(L), hides the active rule facility from the programmer and uses the active rules as a database assembly language.

Since programs are written in a single language, the maintainability should be better than if the programs were written using the active rules and a traditional programming language. This is yet to be investigated. At least this approach eliminates the issue of interaction between control flows of the rule system and the traditional programming language.

The CAMOS(L) approach also raises the abstraction level to concepts such as synchronization of operations, temporal dependencies, updates of the world model, instead of rules. Basically, programs written in CAMOS(L), are compiled into queries and active rules which are stored in the database.

A complex set of active rules tend to interact. The actions of one rule may trigger another rule causing potential cascading rule triggerings or rule cycles. In the thesis, a simple solution is presented where these problems are avoided. The CAMOS(L) compiler only generate a restricted form of active rules where the action part does not contain any database updates that could trigger other rules.

There will always be a need for fast control systems with fast control algorithms. The performance of today's database technology cannot provide data access that is sufficiently fast for all control applications. The CAMOS architecture can, however, combine the speed of control algorithms written with traditional techniques and the coordination and data management of CAMOS operations.

Chapter VIII
Future work

The future of manufacturing control systems will be based on distributed environments. The thesis presents an approach based on a distributed architecture but with a central active database management system. The database makes it possible to integrate information from data sources in the environment. The world model can therefore serve as a basis for coordinating operations in several distributed real-time servers. An interesting extension would be to use to the multi-database facilities to organize hierarchies of cooperating distributed world models. Since multi-database queries are already integrated into the query processor of the AMOS database, this would be feasible. This approach would require extensions to active mediator technology for the multi-database access.

The present lack of real-time processing of CAMOS(L) operations in the operation manager must be dealt with. Fortunately, there is nothing inherently non-real-time in the processing of high-level operations. Given a real-time database, the same architecture would be able to guarantee predictable execution times. Another shortcoming is the lack of data management at the real-time kernel level.

In most situations the control algorithms do not need any data management for the real-time server since information is of limited size and kept locally. An attempt to integrate high-performance real-time databases in the real-time servers would be a step closer to better datamanagement at all levels in the layered control system architecture.

The thesis does not contain a precise definition of the semantics of the CAMOS(L) language. The intention was keep the focus on how active rules can be used and not on the language issue. The semantics should be formally defined. Clearly, this semantic description depends the semantics of the underlying active rule system and query processor.

We are also developing new tools for specifying and verifying important characteristics of the modeled system using Modified Timed Petri Nets [Pen93]. The long-term goal is a design environment for the modeling of large real-time systems that allows the designer to explore different alternatives and to verify timing properties of the design. We will conduct a case study of a typical industrial production cell to test this approach.

References

[Ade95] Adelberg, B., Garcia-Molina, H., Kao, B., Applying Update Streams in a Soft Real-Time Database System, ACM SIGMOD, 1995.

[Årz88] [1] Årzen, K-E., Knowledge-Based Real-Time Control Systems: IT4 Feasibility Study, Studentlitteratur, Lund, Sweden, 1988.

[Årz91] [1] Årzen, K-E., Knowledge-based Real-Time Control Systems: IT4 Project: PhaseII, Studentlitteratur, Lund, Sweden, 1991.

[Årz94] [1] Årzen, K-E., Parameterized High-Level Grafcet for Structuring Real-Time KBS Applications, Proc. 2nd IFAC Workshop on Computer Software Structures Integrating AI/KBS Systems in Process Control, Lund, Sweden, August 1994.

[Åst84] [1] Åström, K.J., Wittenmark, B., Computer Controlled Systems, Prentice-Hall, New York, 1984.

1. References containing Swedish national characters are duplicated here for the convenience of English speaking readers.

[Bar95] Baralis, E., Ceri, S., Paraboschi, S., ARACHNE: A Tool for the Analysis of Active Rules, Proc, 2nd International Conference on Application of Databases, San Jose, Ca, USA, Dec. 1995.

[Bar89] Barkmeyer, E. J., Some Interactions of Information and Control in Integrated Automation Systems, Nato ASI Series, Vol. F53, Advanced Technologies for Industrial Material Flow Systems, ed. Nof, S.Y., Moodie, C.L., Springer-Verlag, Heidelberg, 1989.

[Bor89] Borda, J., How to Use the Manufacturing Information System as a Competitive Weapon, IFAC Information Control Problems in Manufacturing Technology, Madrid 1989.

[Buc84] Buchann, B. G., Shortliffe, E. H., Rule-based Expert Systems, The Mycin Experiments of the Stanford Heuristic Programming Project, Addison-Wesley, 1984.

[Buc92] Buchmann, A., Branding, H., Kudrasss, T., Zimmerman, J., REACH: A REactive, ACtive, and Heterogeneous mediator system, IEEE Data Eng Bull, Special issue on Active Databases, Dec. 1992.

[Bur90] Burns, A., Welling, A, Real-Time Systems and Their Programming Language, Addison-Wesley Publishing Company, 1990.

[But91] Butterworth, P., Otis, A., Stein, J., The GemStone Object Database Management System, C ACM, 34(10), Oct. 1991.

[Cat91] Cattell, R. G. G., Object Data Management, — Object-Oriented and Extended Relational Database Systems, Addison-Wesley Publishing Company, 1991.

[Cer86] Ceri, S., Interfacing Relational Databases and Prolog Efficiently , Proc. 1st International Conference on Expert Database Systems, Charleston, S.C., April 1986.

[Cha93] Chakravarthy , S., Krishnaprasad, V ., Anwar , E., Kim, S.-K., Anatomy of a Composite Event Detector , Tech Report UF-CIS-TR-93-039, University of Florida, Department Computer and Information Science, Florida, USA.

[Cha95] Chaudhry , N., Moyne, J., Designing Databases with Fuzzy Data and Rules for Applications of Discrete Control, T echnical Report, Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor , USA.

[Chr93] Chrysanthis, P .K., Ramamritham, K., A Formalism for Extended T ransaction Models, Proc. of the 17th VLDB Conference 1991

[Dav95] Davis, J., Du, W ., Shan, M-C., OpenPM: an Enterprise Process Management System, Data Engineering Bulletin, 18(1), March 1995.

[Day88] Dayal, U., Blaustein, B., Buchmann, A., Chakravarthy , S., et al., The HiPac Project: Combining active databases and timing constraints, SIGMOD Record, 17(1), March 1988.

[Day90] Dayal, U., Hsu, M., Ladin, R., Organizing Long-Running Activities with T riggers and T ransactions, Proc. SIGMOD, Atlantic City , May 1990.

[Day95] Dayal, U., T en Years of activity in active database systems: What have we accomplished?, Proc. Active and Real-T ime Database Systems, Skövde, Sweden, June 1995.

[Dit95] Dittrich, K.R., Gatziu, S., Geppert, A., The Active Database Management Manifesto, Proc. 2nd International Workshop on Rules in Database Systems, Athens, Greece, Sept. 1995.

[Efs92] Efstathiou, J., Architectures and Techniques of Artificial Intelligence in Process Control, IFAC AI in Real-Time Control, Delft, Netherlands, 1992.

[Elm89] Elmasri, R., Navathem S. B., Fundamentals of Database Systems, Addison-Westley, 1989.

[Eng88] Englmore, R., Morgan, A., Blackboard Systems, Addison-Westley, London, 1988.

[Fah93] Fahl, G., Risch, T., Sköld, M., AMOS - An Architecture for Active Mediators, NGITS'93, Haifa, Israel, 1993, pp. 47-53

[Fah96] Fahl, G, Risch, T., Query Processing over Object Views of Relational Data, Technical report LiTH-IDA-R-95-11, Linköping University, Sweden, (Revised version accepted at *VLDB J.*)

[Fah94] Fahl, G., Object Views of Relational Data in Multidatabase systems, Licenciate thesis No. 446, Linköping University, Sweden, 1994.

[Fal95] Falkenroth, E.T., Törne, A., Risch, T., Using an Embedded Active Database in a Control System Architecture, Proc. 2nd International Conference on Applications of Databases, San Jose, Ca, USA, Dec. 1995.

[Fis89] Fishman D., et al, Overview of the Iris DBMS, Object-Oriented Concepts, Databases, and Applications, ACM press, Addison-Wesley Publ. Comp., 1989.

[Flo96] Flodin, S., Karlsson, J., Risch, T., Sköld, M., Werner, M., AMOS User's Guide, Available at URL http://www.ida.liu.se/labs/edslab/amos/amosdoc.html

[For81] Forgy, C., The OPS5 User's Manual, Carnegie Mellon University, Technical Report, 1981.

[Gar87] Garcia-Molina,H., K.Salem, SAGAS, Proc. SIGMOD, San Fransisco, May 1987,pp. 249-259.

[Gar90] Garcia-Molina, Gawlick, D., Klein, J., Kleissner, K., Salem, K., Coordinating Multi-Transaction Activities, in Proc. IEEE Spring Compcon, 1991.

[Gar91] Garcia-Molina, Gawlick, D., Klein, J., Kleissner, K., Salem, K., Modeling Long-Running Activities as Nested Sagas, Bulletin of the Technical Committe on Data Engineering, IEEE, 14(1), March 1991. Compcon, 1991.

[Geo95] Georgakopoulos, D., Hornick, M., Sheth, A., An Overview of Workflow Management: For Process Modeling to Workflow Automation Infrastructure, Distributed and Parallel Databases, 3(2), April 1995.

[Gep95] Geppert, A., Kradolfer, M., Tombros, D., Realization of Cooperative Agents Using an Active Object-Oriented Database Management System, Proc. 2nd International Workshop on Rules in Databases, Athens, Greece, Sept 1995.

[Gra92] Graham, M.H., Issues in Real-Time Data Management, Journal of Real-Time Systems, 4, 185-202, 1992.

[Ham88] Ham, I., Lu, S., Computer-aided process planning: The present and the future. Annals of the CIRP, 37(2):591, 1988.

[Han89] Hanson, E., An Initial Design Report on the Design of Ariel, SIGMOD Record, 18(3), Sept 1989.

[Han92] Hanson, E., Rule Condition Testing and Action Execution in Ariel, Proc. SIGMOD 1992.

[Har94] Harhalakis, G., Lin, C-P., Mark, L., Muro-Medrano, P., Implementation of Rule-Based Information Systems for Integrated Manufacturing, IEEE Trans. Knowledge and Data Engineering 6(6), Dec. 1994.

[Has86] Haspel, D.W., Taunton, C.J., Applying linguistic control to process optimization, Proc. KBS'86, UK, 1986.

[Hay85] Hayes-Roth, B., A blackboard architecture for control, Artificial Intelligence, 26, pp 251-321, 1985.

[Heg91] Hegarty, C.J., Process-Flow Specification and Dynamic Run Modification for Semi-Conductor Manufacturing, PhD thesis, Memorandum No. UCB/ERL M91/40, College of Engineering, University of California, Berkeley, Ca, USA, April 1991.

[Hey95] Heytens, M., Listgarten, S., Neimat, M-A., Wilkinson, K., Smallbase: A Main-Memory DBMS for High-Performance Applications Release 4.2, Technical paper, HP Labs, Palo Alto, CA, USA, Sept. 1995.

[Hol92] P.Holmbom,P., Loborg, P., Sköld, M., Törne A., A Model for the Execution of Task Level Specifications for Intelligent and Flexible Manufacturing Systems, Proc. of the International Symposium on Artificial Intelligence, ISAI92, Cancun, Dec 1992.

[Hsu88] Hsu,M., Ladin, R., McCarthy, D., An execution model for active database management systems, Proc. 3rd International Conference on Data and Knowledge Bases, Washington, June 1988.

[Iec95] IEC International Standard CEI IEC 1131-3, Programmable Controllers, Part 3, Programming Languages, First ed., Bureau Central de la Commission Electrotechnique Internationale, Geneve, Suisse, 1993.

[Ill96] Illustra, home page, Available at URL http://www.illustra.com/

[ING90] INGRES Product Division, INGRES Version 6.3 Reference Manual, Alameda, Ca., USA, Sept 1990.

[Jaw94] Jawadi, R., Su, S.Y.W., Incorporating Flexible and Expressive Rule Control in a Graph-Based Transaction Framework, Technical Report TR94-030, Department of Computer and Information Sciences, Univ of Florida, Gainesville, USA, 1994

[Kap94] Kappel, G., Rausch-Schott, S., Retschitzegger, W., Beyond Coupling Modes: Implementing Active Concepts on Top of a Commercial OODBMS, International Symp. on Object-Oriented Methodologies and Systems, ed. Bertino, E., Urban, S., Springer Verlag LNCS 858, 1994.

[Kap95] Kappel, G., Rausch-Schott, S., Retschitzegger, W., Workflow Management Based on Objects, Rules, and Roles, Data Engineering Bulletin, 18(1), March 1995.

[Kee085] Intellicorp, Inc., KEE Software Development System User's Manual, Intellicorp, Mountain View, CA, 1985.

[Kee088] Intellicorp, Inc., Core reference manual, KEE Version 3.1, Intellicorps, Inc.,Publication # K3.1-CRM-2

[Kra93] Kramer, R.,K.,Senehi,M.,K., Feasibility study: Reference architecture for machine control system integration Nat. Inst. of Standards and technology, 1993.

[Kri91] Krijgsman, A.J., Jager, H.B., *et. al.*, DICE: A framework for intelligent real-time control, Proc. 3rd IFAC Workshop on AI in real-time control, 1991.

[Lew94] Lewerentz, C., Lindner, T., Case Study "Production Cell" - A Comparative Study in Formal Software Development, FZI-publication 1/94, Forschungszentrum Informatik Haid-und-Neu-Strasse 10-14, Karlsruhe, Germany, 1994.

[Lit92] Litwin, W., Risch, T., Main Memory Oriented Optimization of OO Queries using Typed Datalog with Foreign Predicates, IEEE Transactions on Knowledge and Data Engineering, 4, Dec. 1992.

[Lob91] Loborg,P., Törne A., A Hybrid Language for the Control of Multimachine Environments, Proc. EIA/AIE-91, Hawaii, June 1991.

[Lob93] Loborg, P., Risch, T., Sköld, M., Törne, A., Active OO Databases in Control Applications, in Microprocessing and Microprogramming, Vol. 38, No 1-5, p.255-264. Proc. 19th Euromicro Conf., Barcelona, Sept 1993.

[Lob94] Loborg, P., Error Recovey Support in Manufacturing Systems, Linköping Studies in Science and Technology, Lic Thesis no 440, Department of Computer Science and Information Science, Linköping University, Sweden, ISBN 91-7871-376-5, 1994.

[Lob95] Loborg, P., Törne, A., A Layered Architecture for Real Time Applications, in 7th Euromicro Workshop on Real Time Systems, (Odense, Denmark), June 1995.

[McC89] McCarthy, D., Dayal, U., The Architecture of an Active Data Base Management System, Proc. ACM SIGMOD, Portland, Oregon, 1989.

[Moh95] Mohan, C., Alonso, G., Guntör , R., Kamath, M., Exotica: A Research Perspective on W orkflow Management Systems, Data Engineering Bulletin, 18(1), March 1995.

[Nor94] Norrie, M., Blott, S. M., Schek, H-J., W eikum, G., The database research group at ETH Zurich, SIGMOD Record, 23(1) March 1994.

[Nor95] Norrie, M.C., Schaad, W ., Schek, H-J., W underli, M., Exploiting Multidatabase T echnology for CIM, Technical Report, Department Computer Science, ETH-Zentrum, Zurich, Switzerland.

[Ors96] Orsborn, K., Management of Product Data Using an Extensible Object-Oriented Query Language, Sixth International Conference on Data and Knowledge Systems for Manufacturing and Engineering, T empe Arizona, October 1996.

[Pen93] Peng, Z., Törne, A., A Petri Net Based Modeling and Synthesis T echnique for Real-T ime Systems, 5th Euromicro W orkshop on Real-T ime Systems, Oulu, Finland, 1993.

[Pos95] Position, Statement, Databases and W orkflow management: What is it all about, Position Statement, Proc. 21st International Conference on VLDB, Zurich, Switzerland 1995.

[Pri89] Prieto, F ., Sastron, F ., *et. al.,* Database support in workcell design, IF AC symposium series, Information control problems in manufacturing technology , Madrid 1989.

[Reu90] Reuter ,A., Wächter ,H., Grundkonzepte und Realisierungsstrategien des Contract-Modells Informatik, Forschung und Entwicklung No. 4, 1990.

[Reu95] Reuter, A., Schwenkreis, F., ConTracts: A Low-Level mechanism for Building general-purpose workflow management systems, Data Engineering Bulletin, 18(1), March 1995.

[Ris93] Risch,T., Sköld, M., Active Rules based on Object-Oriented Queries, IEEE Data Engineering (Quarterly), January 1993.

[Rus94] Rusinkiewicz, M., Sheth, A., Specification and execution of transactional workflows. In W.Kim (ed.): Modern Database Systems. Addison-Wesley, 1994.

[Sch93] Schreier, U., Database Requirements of Knowledge-Based Production Scheduling and Control: A CIM Perspective, Proc. 19th International Conference VLDB, Dublin, Ireland, 1993.

[Seg95] Segev, A., Mendel, M., Patankar, A.K., An Implementation of a CIM System Using an Active, Object-Relational Database, Proc. 2nd International Conference on Applications of Databases, San Jose, Ca, USA, Dec. 1995.

[Shi81] Shipman, D.W., The Functional Datamodel and the Data Language DAPLEX, ACM TODS, 6(1), March 1981

[Shu92] Shu, L., Young, M., Correctness Criteria and Concurrency Control for Real-Time Systems: A Survey, Technical Report SERC-TR-131-P, Software Engineering Research Center, Purdue University, Nov. 1992.

[Sim92] Simon, E., Kiernan, J., de Maindreville, C., Implementing high-level active rules on top of a relational DBMS, Proc. 18th International Conference VLDB, Vancouver, 1992.

[Siv92] Sivasankaran, R. M., Ramamritham, K., Stankovic, J., Towsley, D., Data Placement, Logging and Recovery in Real-Time Active Databases, Department Computer Science, University of Massachusetts, Amherst.

[Sköd94] Sköld, M., Active rules based on Object-Relational Queries: Efficient Change Monitoring Techniques, Licenciate Thesis No. 452, Department of Computer Science and Information Science, Linköping University, 1994

[Skö95] Sköld, M., Falkenroth, E., Risch, T., Rule Contexts in Active Databases: A Mechanism for Dynamic Rule Grouping, Proc. RIDS'95, Athens, Greece,1995.

[Skö96] Sköld, M., Risch, T., Using Partial Differencing for Efficient Monitoring of Deferred Complex Rule Conditions, 12th International Conference on Data Engineering (ICDE'96), New Orleans, Louisiana, Feb 1996.

[Spu94] Spurr, K., Layzell, P., Jennison, L., Richards, N., Computer Supported Co-operative Work, Wiley & sons, New York 1994.

[Ste90] Stele, G.L., Common Lisp: the Language, 2nd ed., Digital Press, 1990.

[Sto88] Stonebraker, M., Hanson, E., Potamianos, S., The POSTGRES rule manager, IEEE Trans. Software Engineering, 14(7), July 1988.

[Sto90] Stonebraker, M., *et. al.*, On Rules, Procedures, Caching and Views in Database Systems, Proc. SIGMOD, Atlantic City, USA, May 1990.

[SYB90] Sybase Inc., Sybase Version 4.0 Reference Manual, Sybase Corp., Emeryville, Ca., USA, June 1990.

[Tör96] Törne, A., Falkenroth, E., Loborg, P., Databases in Control and Simulation, Proc. 1st International Workshop on Active Real-Time Databases, March 1996.

[Ull88] Ullman, J. D., Principles of database and knowledge-base systems, Vol. 1., Computer Science Press, Maryland, USA, 1988.

[Vin92] Vina, A., Dominguez, J.M., Reactive Process Control Using a Black Board Architecture, Proc. IFAC on AI in Real-Time Control, Delft, Netherlands, 1992.

[Wal88] Walters, J.R., Nielsen, N.R., Crafting Knowledge-based Systems — Expert Systems Made Easy/Realistic, Wiley & Sons, 1988

[Wer96] Werner, M., Multidatabase Integration using Polymorhpic Queries and Views, Lic Thesis 546, Department Computer and Information Science, Linköping University, Sweden, 1996.

[Wid90] Widom, J., Set-Oriented Production Rules in Relational Database Systems, Proc. ACM SIGMOD conf., Atlantic City, New Jersey, 1990, pp. 259-270

[Wid94] Widom, J., Research Issues in Active Database Systems, ACM SIGMOD Record, Vol.23, No.3, September 1994.

[Wid96] Widom, J., Ceri, S., eds., Active Database Systems - Triggers and Rules for Advanced Database Processing, Morgan Kaufmann Publishers, San Fransisco, USA, 1996.

[Årz88] Årzen, K-E., Knowledge-Based Real-Time Control Systems: IT4 Feasibility Study, Studentlitteratur, Lund, Sweden, 1988.

[Årz91] Årzen, K-E., Knowledge-based Real-Time Control Systems: IT4 Project: PhaseII, Studentlitteratur, Lund, Sweden, 1991.

[Årz94] Årzen, K-E., Parameterized High-Level Grafcet for Structuring Real-Time KBS Applications, Proc. 2nd IFAC Workshop on Computer Software Structures Integrating AI/KBS Systems in Process Control, Lund, Sweden, August 1994.

[Åst84] Åström, K.J., Wittenmark, B., Computer Controlled Systems, Prentice-Hall, New York, 1984.