BUILDING DISTRIBUTED CONTROL SYSTEMS USING DISTRIBUTED ACTIVE REAL-TIME DATABASES

MATS LUNDIN

Submitted by Mats Lundin to the University of Skövde as a dissertation towards the degree of M.Sc. by examination and dissertation in the Department of Computer Science.

September 1998

I hereby certify that all material in this dissertation which is not my own work has been identified and that no work is included for which a degree has already been conferred on me.

_____

Mats Lundin

ABSTRACT

From the field of control theory we can see that varying communication delays in a control system may be hard or even impossible to handle. From this point of view it is preferable to have these delays bounded and as small and as possible in order to adapt the control process to them. On the other hand, in some cases delays are inevitable and must be handled by the control system:

A control system may for different reasons be distributed, e.g., because of a distributed environment or severe environment demands such as heat or dust at some locations. Information in such a system will suffer from delays due to transportation from one place to another. These delays often show up in a random fashion, especially if a general network is used for transportation. Another source of delays is the system environment itself. If big amounts of information are to be processed within a distributed system, a platform where information naturally is distributed is appropriate to use, e.g., a distributed database. For predictability reasons a distributed real-time database is preferable if the delays are to be controlled.

A straightforward way of handling delays in a control system is to build the system such that the delays are constant, i.e., building a time invariant system. The time from sensor reading to actuation is made constant by either adding a delay to have a total constant delay or by using time-triggered reading and actuation. Even though these are simple ways of controlling the delays, they may be very inefficient because worst time execution time must always be used. Other ways of handling varying delays are by using more tolerant control algorithms. There are two suitable control models proposed by Nilsson (1998) for this purpose. The latter approach is assumed in this work.

This thesis looks into the possibility of using a distributed active real-time database system as a basis for building control systems. One of the main objectives is to determine how active functionality can be used to express the control system, i.e., how rules in the database can be used to express the control algorithm and for handling propagation of information. Another objective is to look at how the choice of consistency level in the database affect the result of the control system, i.e. how different consistency levels affect the delays. Of interest is also to categorize and what type of applications each level are suited for.

Table of Contents

1 INTRODUCTION 1

1.1 ORGANIZATION OF THE DISSERTATION 2

2 BACKGROUND 5

2.1 CONTROL SYSTEMS 5

2.1.1 Real-Time Systems 6

2.1.2 Feedback Control Systems 8

2.2 TIMING PROBLEMS IN CONTROL SYSTEMS 12

2.2.1 Modeling and Control of Delays in Control Systems 14

2.3 DISTRIBUTED REAL-TIME DATABASE SYSTEMS FOR CONTROL SYSTEMS 17

2.3.1 Active Behavior: Rules 20

2.3.2 DeeDS Architecture 22

3 PROBLEM DEFINITION 25

3.1 MOTIVATION 25

3.2 PURPOSE OF DISSERTATION 26

3.2.1 Objective: Use of Rules for Control 27

3.2.2 Objective: Study Delay Dependency of Rule Distribution Schemas 28

3.2.3 Objective: Investigate How can Rules Support Distribution 29

3.3 ASSUMPTIONS 29

4 EXPRESSING CONTROL SYSTEMS USING ACTIVE RULES 30

4.1 IMPLEMENTATION USING ACTIVE RULES 30

4.1.1 Information Flow of Control Process 32

4.1.2 Basic Control Functions 34

4.1.3 Control Management Functions ... *36*

4.2 Mode Changes for Altering Control System ... 40

4.3 Importance of Rule Ordering ... 41

5 DISTRIBUTION OF CONTROL FUNCTIONS ... 44

5.1 Issues in Distributing a Control Application ... 45

5.2 Handling Random Time-Delays in System ... 51

5.2.1 Using Tolerant Control Algorithms ... *52*

5.2.2 Calculating Delays ... *52*

5.3 Where to Distribute Functions ... 53

5.3.1 Delays in Different Allocation Schemas ... *55*

6 RESULTS ... 57

6.1 Implementing Control Application Using Active Rules ... 57

6.2 Distribution of Control System ... 60

6.2.1 Issues in Distributing the Rules ... *60*

6.2.2 Compensating Delays ... *61*

6.2.3 Allocation of Rules ... *62*

6.2.4 Using Databases for Control Systems ... *63*

6.2.5 Level of Consistency ... *64*

6.2.6 Discussion of ASAP Replication versus Bounded-Delay Replication ... *67*

7 RELATED WORK ... 68

7.1 Active Rules for Data Management in Control Applications ... 68

7.2 Active Databases is Semiconductor Manufacturing ... 69

7.3 Database for CIM Applications ... 70

7.4 A Distributed Database Used in a Cold Tandem Mill ... 70

7.5 Modeling and Design of Distributed Control Applications ... 70

7.6 Feedback Control Systems with Random Delays ... 71

7.7 REAL-TIME CONTROL SYSTEMS WITH RANDOM DELAYS 72

7.8 HRT-HOOD: A STRUCTURED DESIGN METHOD FOR HARD REAL-TIME SYSTEMS 72

8 CONCLUSIONS 74

8.1 SUMMARY 74

8.1.1 Expressing Control System Using Active Rules 75

8.1.2 Distribution of Control System Rules 76

8.1.3 Other Conclusions 76

8.2 CONTRIBUTIONS 78

8.3 FUTURE WORK 79

9 ACKNOWLEDGMENTS 81

BIBLIOGRAPHY 82

List of Figures

1.1	Overview of the problem area	3
2.1	A closed loop control system with delays	9
2.2	Markov chain and delay distributions	16
2.3	The DeeDS architecture	23
3.1	The four allocation schemas	28
4.1	A simple control system	32
4.2	A control system with an estimator	33
4.3	Rule for compensating transients	40
5.1	Control system with several sensors	46
5.2	Control system with delays	47
8.1	Compensation for real delay	79

Chapter 1

Introduction

Future feedback control applications will operate in distributed environments that involve large volumes of data. The development of such applications can be tedious unless tools are available for building these applications on a suitable platform. As a result of the distributed nature of such applications, delays will be present, and may cause instability unless the delays are compensated for. These delays are often random, as the communication between the nodes in the system often takes place via a general network. There are proposals for how to compensate for unpredictable delays in those systems [Luc90], and for using more delay-tolerant control algorithms [Cha96, Nil98, Ray94]. Recently, there has been some work done on building distributed feedback control applications using such tolerant control algorithms, e.g., see [Nil98, Tör95].

A distributed database offers a way of handling large volumes of data and distribution it in a natural way, and thus, can be a suitable platform for building

distributed control systems. The time for replication of data between the nodes will be random due to unpredictable network delays.

This dissertation address the area of building distributed control systems using an active distributed real-time database as the base. We examine how a control application can be modeled using active rules in order to implement it in the database. Then we examine the issues that are raised as the control application is distributed, i.e., by partitioning of the rules to the different nodes in the database. We look at how tolerant control algorithms can be used to compensate for delays in database replication.

It is shown that a control application can be implemented using active database rules as well as other management control functions. There will not be any conflicts in rule ordering as long as some special case of management control functions are used. An ordinary control system will have the same information flow independent of where the rules are placed. The delays in the control system can be calculated (and compensated for) if the time when the information written in the database are known. An overview of the problem area of this dissertation is provided in figure 1.1.

1.1 Organization of the Dissertation

Chapter 1 has presented an introduction to the area of building distributed feedback control systems.

Chapter 2 gives an overview of real-time and feedback control systems, delays in control systems, distributed database systems, active rules and the DeeDS architecture.

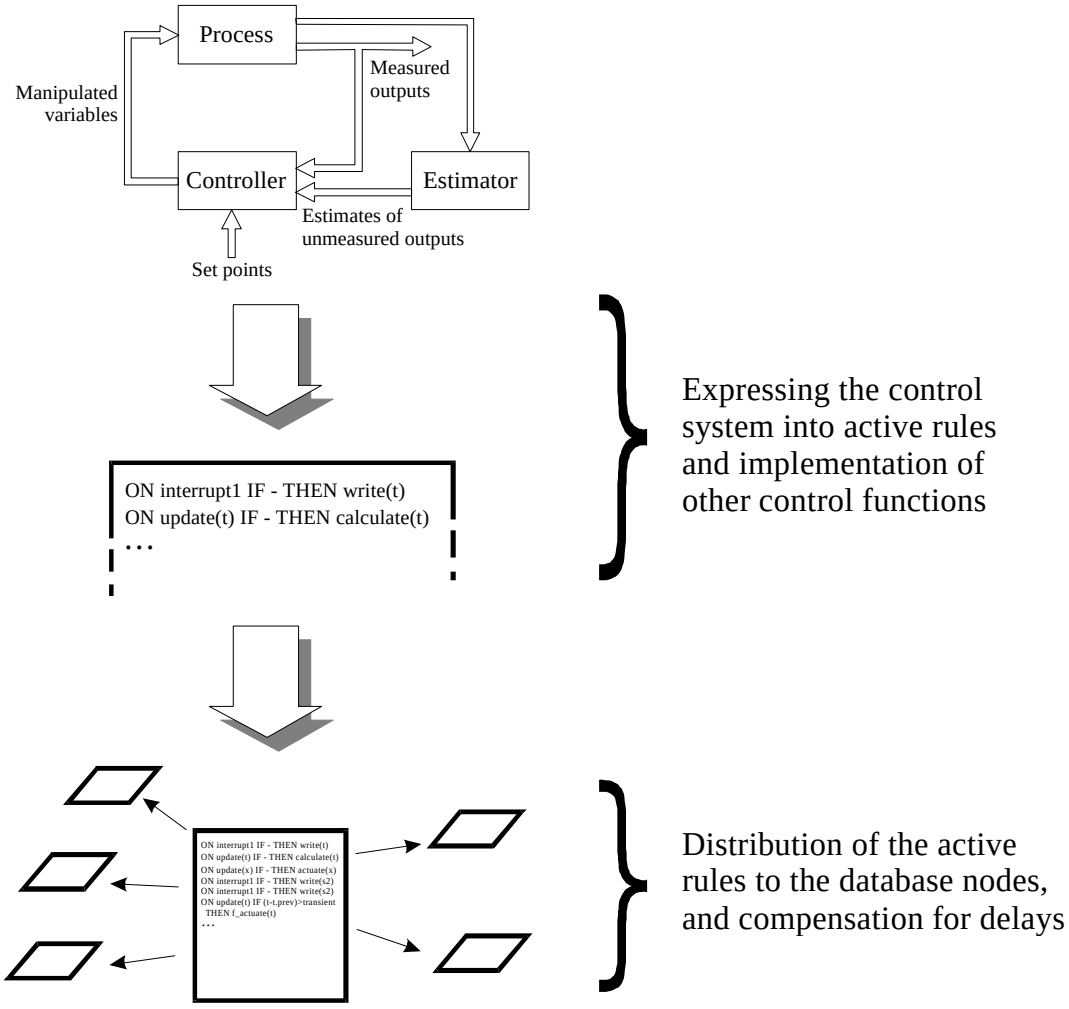

Figure 1.1 - Overview of the problem area

Chapter 3 presents the problem definition of this dissertation. The motivation, aims and objectives are presented.

Chapter 4 describes how a control application is expressed using active rules. Both basic control functions in control theory and additional functions for building control systems are presented.

Chapter 5 describes how the control system is affected by the delays introduced by database replication and what can be done to compensate for them.

Chapter 6 gives the results and discusses what is important when building control system using active rules.

Chapter 7 presents work done in close relation to this dissertation, and compares it to the work done here.

Chapter 8 summarizes the dissertation and contributions are highlighted. Some suggestions for future work are given.

Chapter 2

Background

This chapter introduces areas important to the dissertation. The following is an introduction to control systems (section 2.1), timing problems in control systems (section 2.2) and distributed real-time databases for control systems (section 2.3).

2.1 Control Systems

The use of computers for control purpose is of great importance in the automation of real-world processes, such as chemical plants and CIM (Computer Integrated Manufacturing) environments. The most general way of using a computer for control is to have sensors connected to a computer, a control program and actuators that affect the real world processes according to decisions made by the control program. This way of building a control system may be sufficient for simple control problems where the demands are not be too rigorous. For other problems the control system must be built with greater care, e.g., because the controlling must be exact and reliable in all situations or if the control problem is distributed by its nature.

Definition 1 - A general *control system* is a system that continuously uses updated information about an environment for making decisions or calculations and delivers an output.

In order to achieve total control over a controlled environment, data is fundamental [Rod89]. In this dissertation we consider how a distributed real-time databases (RTDB) can be used as basis for building process control systems (see further section 3.2), i.e., systems that controls a continuous processes.

There are two fundamental types of control systems used in the computer science and engineering field respectively. In computer science a real-time system is a control system that reacts to the controlled environment in a timely manner. Such systems mostly work in environments where events or timer interrupts are used to start computations that results in discrete events. In section 2.1.1 real-time systems are described. On the engineering side control theory are used to control continuous real-world processes. Today many control systems uses sampled data and a computer to perform the computation. In section 2.1.2 such control systems are described.

2.1.1 Real-Time Systems

A real-time system is built to deal with the *dependability aspects* [1]of a control system and to ensure its correct behavior in a timely manner, i.e., within a bounded time. The terms “time-critical” and “real-time” does not necessarily imply very fast execution

[1] This is further discussed in [Lap94]

but fast enough for the chosen application. In [Bur97] a real-time system can be defined in the following way:

Definition 2 - A *Real-Time System* is a system that is required to react to stimuli from the environment (including the passage of physical time) within time intervals dictated by the environment.

Some of the dependability aspects that individually must be considered and handled by real-time systems are [Lap94] and [Mul94]:

- *Availability* – the delivery of correct service when needed, i.e. readiness for usage.
- *Reliability* – continuous delivery of correct service over time.
- *Safety* – non-occurrence of catastrophic consequences as a result of one or more failures.
- *Maintainability* – ability to quickly undergo repairs and reintegration of system.

The time at which the output of the system must be produced is called a deadline. Deadlines are categorized according to their criticality depending of how important it is to finish in time. If a missed deadline is of catastrophic nature the system is said to be *hard critical*. A *soft* real-time system is a system where the missed deadline is of the same magnitude as the utility of the operational system [Mul93]. The systems can further be classified into the following criticality classes according to the consequences of missed deadlines [Bur91]:

- *Hard critical* – a missed deadline leads to catastrophic consequences as great economic loss, great environmental pollution, injury, or even death.
- *Hard essential* – costly penalty, e.g., damage of equipment or material.
- *Firm* – loss of service, e.g., a missed departure or plane reservation.
- *Soft* – degraded service, e.g., loss of certain functions as less optimal fuel consumption, less smooth ride etc.

Another important design aspect of real-time systems is the way they respond to external events, either the system is event triggered or time triggered. An event-triggered system is a system where the execution is dependent on externally generated events, e.g., sensor readings. A time-triggered system on the other hand uses the system clock to trigger the execution. The former system executes within a bounded time from the generation of the event and the latter uses the clock to execute at pre-defined points in time. This means that delays in an event-triggered system can be less than in the time-triggered system, but the execution is easier to predict in the latter.

2.1.2 Feedback Control Systems

A feedback control system is a special case of a control system that controls a continuous process by feedback. This is done by sampling the readings from the sensors at precise points in time, i.e., it is a time-triggered system with a given sampling interval.

Definition 3 – "A *feedback loop* is a control system in which a desired effect is achieved by operating on various inputs to a controlled system until the output, which is a measure of the desired effect, falls within an acceptable range of values." [Tör98]

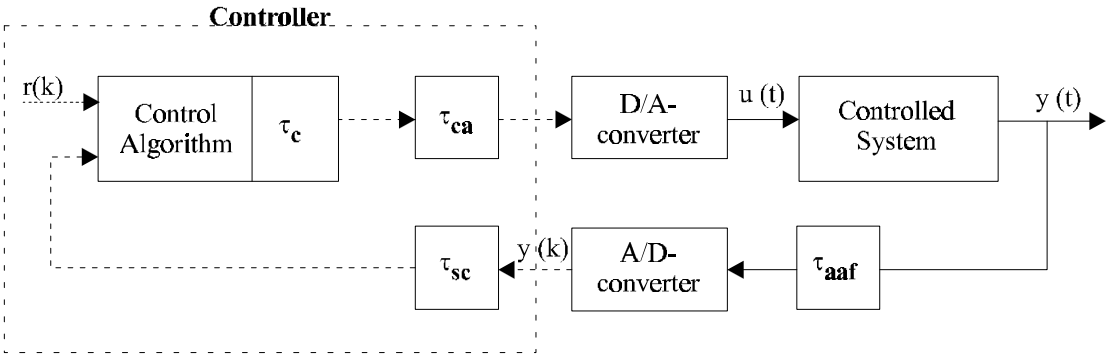

Figure 2.1 - *A closed loop control system with delays*[2].

In this work we will from now on refer to a *feedback loop* when we use the term *control system*.

The principles of a sampled data system are illustrated in figure 2.1, see further in Törngren [Tör95]. The information in such a system is a mixture of both continuous and discrete time signals, here illustrated with continuous and dashed lines respectively. The connection between the controller and the real world goes through the analog to digital (A/D) and digital to analog (D/A) converters. Time delays in the system are modeled by t, where t_c is the controller delay (computation time), t_{aaf} is the

[2] Modified figure from [Tör95]

delay if a *antialiasing*[3] filter is used, t_{sc} is the sensor to controller delay and t_{ca} is the delay from controller to actuator. The signal *r(k)* is the desired output reference, *u(t)* is the control signal to the actuator, *y(t)* is the actual output and *y(k)* is the sampled output.

In control theory PID-controllers are usually used since it is simple and handle many situations. The P stands for Proportional, it compensates for the error, i.e., the difference between the actual and the desired output. The I stands for Integration, and integrates the error over time to reduce it as close as possible to zero. Finally, the D is for Derivation. The error value is derived to be able to compensate for fast changes. In a control system there are constants, called the *system variables,* that are used in the computation of the output to the actuators.

Sampled data systems are in general time invariant, i.e., the delays in both the control algorithm and in conversions are constant due to the electronic circuit properties. This time invariant view is mostly sufficient for feedback control purposes. In a distributed environment this might not be sufficient as delays are introduced. There is a straightforward way of handling too long response delays, it is to raise the sample rate. A rule of thumb is to choose a sample rate of 10-30 times the desired natural frequency of the closed loop system [Åst90], i.e., the frequency that the system will oscillate with if the control system is too slow.

[3] An antialiasing filter is used to filter ambiguous information after sampling.

This type of control is, nevertheless, sensitive if delays are introduced [Cha95], because the delays is not compensated for in the model of the environment that the control system is built for. This means that the control performance is affected as delays are introduced to a system that is not designed to handle them. For example, a consequence might be that the system starts to oscillate because the system is late to compensate for an error in the output [Nil96]. In this dissertation we will only look at the influence of t_{sc} and t_{ca}. The effect of t_c can be embedded in t_{ca} and t_{aaf} embedded in t_{sa} [Nil96].

The change from sample to sample are normally small, a control system is built fast enough to be able to handle these changes. However, there may be fast changes in the environment that the system is not fast enough to handle, these are called *transients*.

In this dissertation we assume that a control system can be decomposed into the following basic parts, each called a control function:

- *Sensor reading* – The sampling of the sensor value.
- *Actuation* – The actuation of the output from the control algorithm.
- *Control algorithm* – The computations to compensate for the error between the actual and desired value.
- *Digital filter* – A filter in the computer that smoothes the sensor readings.
- *Estimator* – A function for deriving values that are not directly measured, e.g., other values must be used to indirectly calculate the extreme heat of a gas.

Definition 4 – The term *control performance* are used as a measure of the system performance, a better accordance to a wanted behavior is named *better control performance.*

The flow of information in a control system we define as:

Definition 5 – A *control process* is the flow of information in a control system. The flow of information is defined by the information being passed from one control function to another. The control process starts with the information written by one or several sensors and end when the control is being delivered to the actuator(s). A control process may have several *paths* of control as there may be several sensors and actuators.

2.2 Timing Problems in Control Systems

In most systems there are delays introduced in the system by different sources. In a centralized control system the most significant delay is usually caused by the control algorithm. If the system, on the other hand, must be built in a distributed manner, the delays in communication (t_{sc} and t_{ca}) become significant. The reason for building distributed control system can be several [Mul93, p. 424, Cha95]:

- The sensor(s) and actuator(s) are at different physical locations.
- A computing element is used for several control process algorithms.

- The environment at the sensor and actuator representatives is too severe for a computer control site to work in, e.g., by high heat, dusty environment or the signal noise of the computer might affect the process.

Time delays in a control system can be divided into two main types of delays, namely *jitter* and *control delays* [Tör95].

Definition 6 - ***Jitter*** *is the non-intentional variation in sampling period.*

Definition 7 - *The* ***Control Delay*** *is the actual delay from sampling to actuation. In terms of figure 2.1, it is the total delay* $t = t_c + t_{sc} + t_{ca} + t_{aaf}$.

Another source of delays is the clock drift between the nodes in a distributed system. In this dissertation we will see it as jitter and will therefore not consider it further.

There may be situations where one sample is not measured or where it disappear before it reaches the actuator, this is called *vacant sampling*. Törngren in [Tör98] and Ray in [Ray88] propose that *oversampling* can be used to solve this situation, i.e., a sample period shorter than actually needed. It will however reduce the time-variations at the cost of increased resource utilization.

When more information is appearing more rapidly than is possible to send on the network a queue must be used for handling the transmission. In [Cha95] a way of

handling delays in these situations. This will not further be discussed in this dissertation.

The delays introduced by communication in a distributed system are not always easy or even possible to predict, especially if a general network is used. In the case of a general network the delays are among others defined by the properties of the communication media, e.g., the bandwidth and the latency of sending messages. Even if these properties are known there are other sources of delays as well such as the load and traffic patterns of the network causes unpredictable delays. Time-varying delays on control system performance and the modeling of time-varying discrete-time control systems were investigated in [Luc88].

It is not only the communication from one node to another that affects the delays in a distributed system, the choice of algorithm for distribution and synchronization is also affecting the total delays [Lun97].

2.2.1 Modeling and Control of Delays in Control Systems

Depending on the properties of the system, e.g., by consistency level, communication media, and algorithms for communication, the delays are of different kinds and can be categorized in three different ways:

- *Constant delays* – the delays are known in advance and do not change over time.
- *Randomly and independent delays* – the delays are not predictable and are totally random over time, i.e., no variations in distribution caused by the load of the network, etc.

- *Stochastic delays* – the delays have a known distribution, e.g., by affection of variations in network load.

The constant delays can be dealt with by the standard control theory and are only problems if they get too large, see e.g. [Sch88]. The method for controlling constant delays are called LQG-control, Linear Quadratic Gaussian control. The varying delays are, on the other hand, a larger problem because of their unpredictability. There has recently been done work in this area and theories have been proposed for dealing with randomly independent delays [Ray94] [Nil98] and stochastic delays [Nil98].

A straightforward way of handling delays in a control system is to build the system such that the delays are constant, i.e., building a time invariant system. Firstly the time variations are reduced to a minimum by suitable choice or modification of the execution strategy. Secondly the time from sensor reading to actuation is made constant by either adding a delay to have a total constant delay or by using both time-triggered reading and actuation. Luck and Ray [Luc90] proposes a solution where buffers are used for compensation at the controller node and the actuator node. The problem with this method is that it is pessimistic and highly dependent on modeling accuracy, i.e., worst time delays must be considered.

In order to make a control model that manages delays it can be assumed that the transfer delay is independent of previous delays. A comparison between a model with constant delays and one with independent varying delays are done in [Nil96, Ray94].

The case in a distributed control system is not totally random delays but delays that are dependent of earlier delays, e.g., by network load at the moment. A way to model random but dependent delays is by using an underlying Markov chain [Krt94, Nil96, Nil98]. Having the system do a transition in the Markov chain every time a transfer is made can be used to capture effects of varying delays.

Example 5 - *Markov chain (from [Nil96])*

In a simple model of a network we can let the network have three states: One with low, medium, and high network load respectively. In figure 2.2, the Markov chain gives the transitions between the three network states. Together with each state in the Markov chain there is a corresponding delay distribution modeling the delay for that network state, e.g., by the one in figure 2.2.

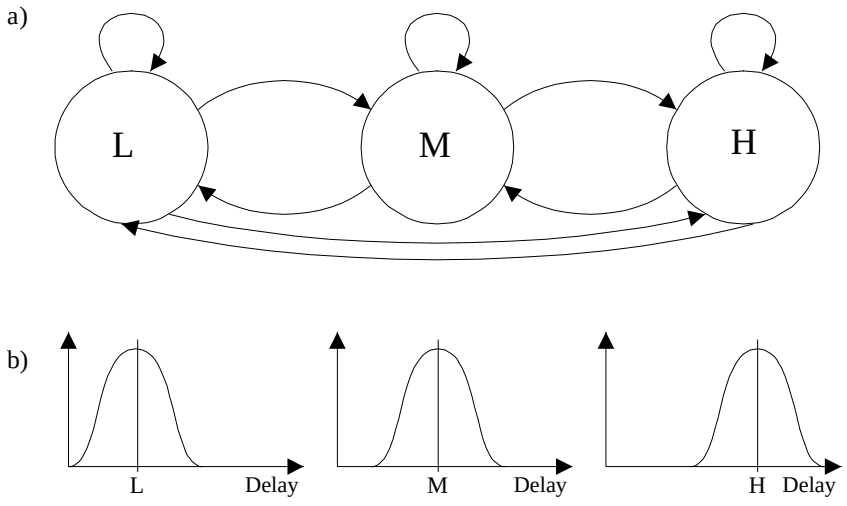

Figure 2.2 *- Markov chain and delay distributions*

In 2.2 a) an example of a Markov chain modeling the states in a communication network. L is low, M is medium and H is high network load. The arrows show the possible transitions in the system. In b) the delays distribution is showed for each case respectively.

The models described above are suboptimal solutions to the control problem, but can be good tools for handling randomly varying delays in a system.

In this dissertation we will only consider control delays but not jitter in the system. This is because the control models described in [Nil96, Nil97, Nil98] does not take jitter into account.

2.3 Distributed Real-Time Database Systems for Control Systems

A distributed database is a database that has some common data distributed over several sites and allows coordinated sharing and updating of that data. One of the reasons for distribution is that the data must be reachable at several sites at the same time. An advantage compared to a conventional centralized database is fault tolerance and modular growth at each site [Mel97]. On the other hand, this also raises the issue of replication of changes from one node to another; there must be some synchronization that ensures consistent data at all nodes. The price for this synchronization is the use of a protocol that consumes processor and communication resources and also takes elapsed time to perform. This elapsed time is often hard to predict because of the unpredictability of the communication medium.

There are different protocols proposed for handling data consistency and for ensuring atomic execution of transactions in a database, i.e., to ensure that an operation is executed at all sites or none at all [Lun97]. A usual way is to use an *atomicity protocol* which replicate the read and write transactions to all other nodes, one such protocol is the *Distributed Two-Phase Commit Protocol*. This protocol and other further developments are all using considerable resources for execution.

In databases, so called ACID properties are used to specify a correct behavior of the database, e.g., to ensure its correct behavior. In [Gre93, Elm94] these properties are defined as follows:

- *Atomicity* – A transaction's changes to the state are atomic: either all happen or none happen. These changes include database changes and messages.
- *Consistency* – A transaction is a correct transformation of the state. The actions taken as a group do not violate any of the integrity constraints associated with the state. This requires that the transaction is a correct program.
- *Isolation* – Although transactions execute concurrently, it appears to each transaction, T, that others executed either before T or after T but not both.
- *Durability* – Once a transaction completes successfully (commits), its changes to the state survive failures.

When we say that a database is distributed in this dissertation we will assume that all data is available at all nodes, i.e., the database is *fully replicated*.

In close relation to the level of consistency is the level of isolation. In [Gre93] four levels (degrees) of isolation are identified, namely: 1) Chaos, 2) Browse, 3) Cursor Stability, and 4) Isolated. A closer comparison is done in table 2.1.

Issue	**Degree 0**	**Degree 1**	**Degree 2**	**Degree 3**
Common name	Chaos	Browse	Cursor stability	Isolated, serializable, repeatable reads
Protection provided	Lets others run at higher resolution	0^o and no lost updates	No lost updates No dirty reads	No lost updates No dirty reads Repeatable reads
Lock protocol	Set short exclusive locks on data you write	Set long exclusive locks on data you write	1^o and set short share locks on data you read	1^o and set long share locks on data you read
Rollback	Undo cascades, can't rollback	Undo incomplete transactions	Same as 1^o	Same as 1^o
Dependencies	None	WRITE → WRITE	WRITE→WRITE WRITE→READ	WRITE→WRITE WRITE→READ READ→WRITE

Table 2.1 *- Description of the different levels of isolation*[4]

In some cases it is not important to have immediate consistency, e.g., when an application is tolerant to temporal inconsistencies, but instead a more efficient usage of the resources is to prefer [Lun97]. In such cases an *eventual consistency protocol* can be used, i.e., a protocol that does not guarantee immediate consistency but consistency at some future point in time. The result of such a protocol is that an unknown time delay to achieve consistency of the database is introduced.

[4] This is a part of the table presented in [Gre93]

A way of controlling replication delays for critical transactions is to use different latency classes in the communication. This will make it possible to shorten some delays at the cost of lengthening others.

Traditional databases do not focus on real-time issues but on integrity and consistency of data. They often store the database on a secondary permanent storage medium such as a disk and do not have to take timely behavior into account. A database for a real-time systems must take the access delays into account to ensure timely behavior. A way to do this is by using a main-memory database, which will also improve the performance [Lis97]. Also the time of creation are of greatest interest, since the data are only valid for a given period of time [Rod89]. A designer of a simple PID controller assumes that the data are measured at specific points in time.

2.3.1 Active Behavior: Rules

Traditional database systems only support operations such as insertions, deletions, and retrievals. In order to build systems with active behavior, e.g., for supporting business rules, the application programmer must explicitly do this. In a distributed environment, this is an even harder task. There are two fundamental ways of implementing rules in a distributed environment using an ordinary database [Fal96]:

- *Embedded rules* – An embedded rule is not a part of the database itself but a part of the program code at each node. This means that the rule must be hard coded at the construction of the system and compiled to each node specifically. When a change in the database is propagated to the nodes in the system and the system should

react to this change code must be written to handle this situation. As the information comes to a node, it must be evaluated whether it is relevant or not, and if so, trigger an action.

- *Polling* – the rules are enforced by a process that constantly scans the database for changes and executes the corresponding rules if a relevant change is found. The polling procedure must be performed at each node in the database. An important design consideration in an application using polling is at which frequency the polling should be performed.

Both of these approaches have drawbacks. The first approach means that the same rules must be programmed and executed at each node that uses or updates some relevant data. Maintenance and updating of applications using embedded rules is thus costly and changes in implementation may cause inconsistencies in data. The second approach uses a large amount of resources: processing power, communication bandwidth, and database usage. It also requires a trade-off between resource usage and time to discover changes. Further elaboration in the area can be found in [Fal96].

A better approach than using the proposed ways of implementing rules, active functionality can be integrated into the database itself. An active database system is such a database system that monitors situations of interest and, when they occur, triggers an appropriate response. In real-time database systems this must happen in a timely manner.

Active rules in a database can be executed at different points in time. There are three coupling modes describing these three [Cha89]:

- *Immediate mode* – This mode immediately evaluates the condition. If the condition is satisfied the action is immediately executed within the current transaction, the ongoing transaction is preempted till after the execution of the action.
- *Deferred mode* – This mode will also execute within the transaction, but evaluation of condition and execution is deferred to the transaction commit.
- *Decoupled mode* – In this mode the execution forms a separate transaction after the commit of the current transaction. Evaluation of condition can either be immediate or performed in the separate transaction.

2.3.2 DeeDS Architecture

The DeeDS (Distributed active real-time Database System) architecture and prototype at University of Skövde is an attempt to keep up with the demands of timely behavior, complex environments and large storage capacity. It is an event triggered database system that uses dynamic scheduling and is built for flexibility [And95, And96, Mel97]. Transactions can be of different criticality and the reactive behavior is modeled using Event-Condition-Action rules. The architecture of DeeDS is shown in figure 2.3.

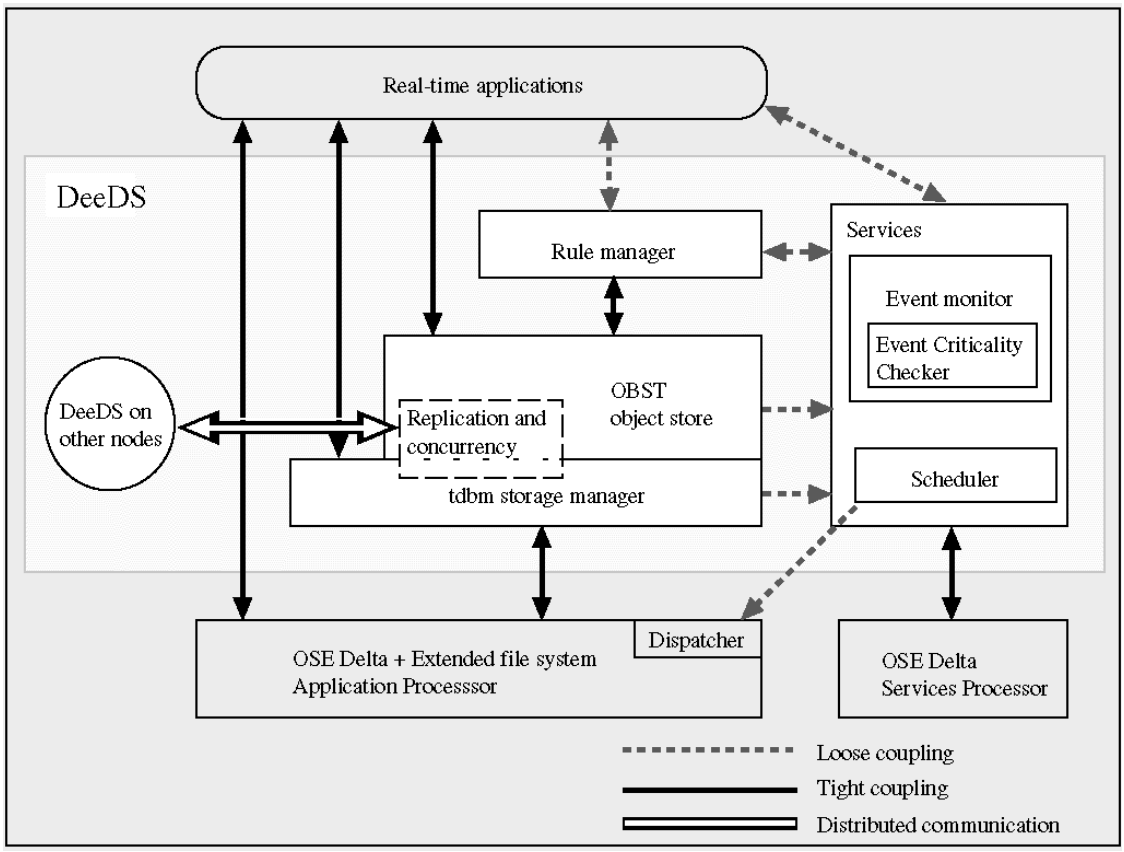

Figure 2.3 - The DeeDS architecture.

The application-related functions in DeeDS are separated from the critical services by running them on different processors. The reason for this is that it simplifies the timing model, i.e., is advantageous for predicting the temporal behavior. Each processor executes a distributed real-time kernel (OSE Delta).

The database used is a public domain object store, OBST (Object Management System of STONE) [Cas92], but the storage manager has been replaced with tdbm [Bra92]. The main reason for this is that *tdbm* supports nested transactions, which is a desirable functionality in active databases and for exception handling and efficient

main-memory access techniques. The reactive behavior in DeeDS are modeled after the *event-condition-action* (ECA) paradigm described in [Cha89].

DeeDS have two replication policies, namely ASAP [GUS95] and bounded delay [Lun97]. All transactions commit locally before the updates are replicated to the other nodes, requiring an eventual consistency protocol to be used. This gives a short latency within a physical node as the execution of next transaction can start immediately after the previous. In this dissertation we assume this local execution before replication.

Chapter 3

Problem Definition

This chapter defines the purpose and motivation of this work and gives an overview of the problem that we wish to address in this dissertation. The main purpose is divided into objectives that can be evaluated.

3.1 Motivation

Building a control system today can be a very difficult task, there are often considerable information to take into account and the control process can be anything but trivial. In order to deal with the complexity in manageable way, a control problem can be divided into subproblems that individually can be solved in a satisfactory way. If a situation also raises the demand of distribution, e.g., by inherent geographic distribution or demands for fault tolerance, there are other types of problems raised:

- Transportation of information from one place to another
- Coordination of remote control system.
- Timing problems such as deadlines and delays
- Reliability despite distribution

In a control system where a lot of information is to be used in a distributed environment, a platform that handles this automatically is to prefer. This is the reason why a distributed real-time database system is to be used as the basis. One of advantageous features of such a platform is the active functionality that can be used to express the control system at different places in the system. One of the big problems, on the other hand, is the delays that comes from the distribution of the platform.

There are mainly two ways of handling the problems of communication delays in a distributed control system, either from the computer science or the control engineering perspective. There are some work done on how to model different delays in control theory, e.g., for use in distributed real-time systems. There are, however, no specific study done in using a distributed real-time database for building a control system.

For simplicity and to ensure consistency the DB are the only medium to send information in the system. This also implies that rules must be used although they might be inefficient if used in to a large extent. On the other hand, few rules are activated at the same time, which improves the performance.

3.2 Purpose of Dissertation

The main purpose of this dissertation is to look at advantages and disadvantages of using active functionality in a distributed real-time database system when it is used for building distributed control systems. The first aspect of interest is whether a control system can be expressed using active rules and if some other control functions can be defined. Secondly, the delays introduced by the distribution of the rules are considered.

This also raises the question of whether precaution must be taken to maintain consistency in the database.

3.2.1 Objective: Use of Rules for Control

The active functionality of a database can be used with advantage in many situations. In this dissertation we will look at how rules can be used to express a distributed closed-loop control system.

Hypodissertation 1 – *Rules can be used to facilitate distribution and allocation of control.*

In normal situations the environment that a control system works in is easy to handle and does not have any transients, i.e., fast or unforeseen changes in the process. To have a better control performance without utilizing the systems resources more than necessary a rule can be defined to handle this.

Hypodissertation 2 – *By using active rules of a distributed real-time database transients can be handled.*

We will also have a look at how the level of consistency that a database delivers affects the delays. One interesting question that is investigated is whether an eventual consistency protocol with ASAP replication is sufficient for a control performance, or if bounded delay replication must be used.

3.2.2 Objective: Study Delay Dependency of Rule Distribution Schemas

One of the most obvious delays in the system depends of the network, which can be the most troublesome to predict. These delays depend not only on the normal latency and bandwidth of the communication medium, but also of the network load, traffic patterns, topology, and protocol.

In a distributed control system there are four static allocation schemas possible, except for the schema where the sensor, controller, and actuator are at the same physical node. These four allocation schemas are shown in figure 3.1. Depending on which of these that are used, the delays are present in different parts of the control process. These four allocation schemas are all evaluated with respect to the case where the sensor, control, and actuator are at the same node. A physical nod is a separate processor with memory and a replica of the database. Each physical node can hold several control functions.

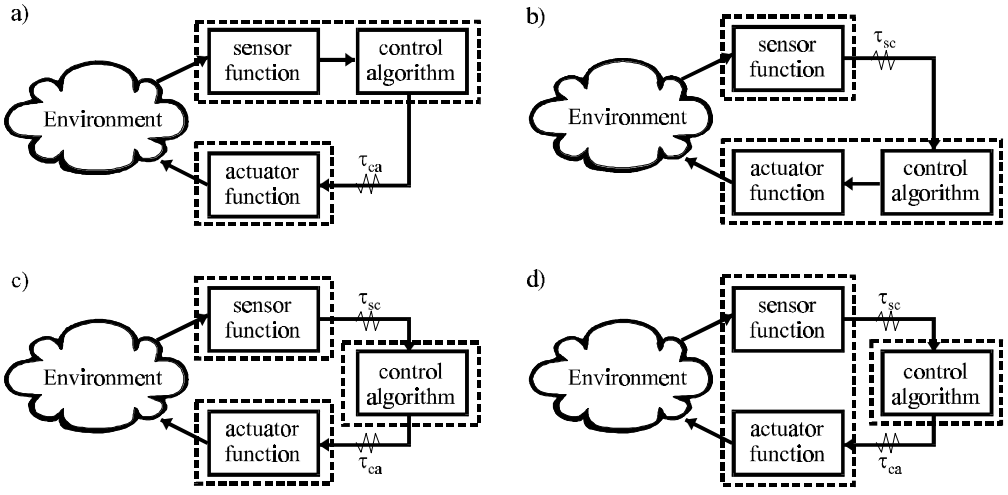

***Figure 3.1** - The four allocation schemas.*

In figure 3.1 the physical node is defined by a dotted line. In 3.1 a) the sensor function and the control algorithm are at the same physical node, in 3.1 b) the actuator function and control algorithm are at the same physical node, in 3.1 c) all control functions are at separate physical nodes and in 3.1 d) the sensor function and actuator function are at the same physical node.

3.2.3 Objective: Investigate How can Rules Support Distribution

We will also have a look at how rules can be used to distribute the control system to the different nodes in the system. When the rules are distributed there will be delays. We will examine how these affect the performance of the control system.

3.3 Assumptions

We make the following assumption in this dissertation:

- All communication between the control functions goes through the database. The delay from two functions at the same physical node are insignificant compared to the delay in replication between two physical nodes.
- A time-driven sensor representative. The readings by the sensor are done periodically, i.e., the sensor function will always have enough resources at the sample instant. The updating of one or several database variables triggers the other control functions, including the actuator function.
- The control algorithm keeps a history of previous delays.
- Bounded clock drift and sufficient clock granularity.

Chapter 4

Expressing Control Systems Using Active Rules

In order to build a distributed control system with a distributed database the control system must in some way be expressed, either in separate code or in the database itself. The control functions in a control system are defined in section 2.1.2. In this chapter we consider how the control functions can be defined using active rules in an active RTDB and how these rules are combined into a control application. Some important aspects while building the control application are timely behavior and fast execution due to the control applications sensitiveness to delays.

4.1 Implementation Using Active Rules

In the area of control theory a control system consists of at least a control algorithm that performs the controlling of a physical process. Often there are other parts as well, depending on what process that is to be controlled. In this section the functions that a

control system can consist of are defined and expressed as rules. The reasons for expressing the control system into rules are:

- The possibility to use an active database as basis for the control system
- To allow distribution of the control system over several nodes

Further, the reason for distributing the control system over several nodes can be one or several of the following reasons:

- *Hostile environment* – There may be a severe environment where normal high performance processors cannot work, e.g. in the reactor of a nuclear plant or in a dusty or hot environment.
- *Distribution of system* – The system representatives are distributed over a large spatial area.
- *Distribution of information* – Information from one sensor can be used at several locations in the whole system

In order to build a control system out of an active database the control system must be modeled into the database, i.e., into active rules. In the following section we examine the information flow in the database when it is used for control systems. Next is a section where the control functions are expressed into active rules and then a section where some management control functions are presented.

4.1.1 Information Flow of Control Process

There is a flow of information through a control system: The sensors produce new information, the control functions use it to calculate an appropriate behavior of the system and finally the actuators perform what is decided by the control functions. This flow of information must in some way be implemented into the database. One way of doing so is by using the rules to define an implicit information flow.

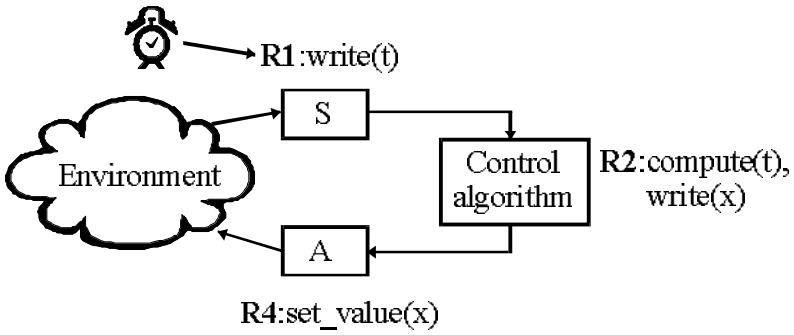

Figure 4.1 *- A simple control system with a sensor, control algorithm and actuator implemented in using active rules.*

Example 6 – A simple control system consisting of a sensor, control algorithm, and an actuator can be modeled by three active rules, see figure 4.1. The first rule (R1) is time-triggered and read a value from the sensor and writes it to the database. The second rule (R2) is trigged by the update and calculates the control value and puts it in the database. Finally the last rule (R4) is triggered by the second update and sets the new value to the actuator.

In some control applications the control algorithm uses information from multiple sources, e.g., from several sensors or from both a sensor and an estimator. One way of modeling this and to ensure a correct information flow of information is by using composite events to trigger a rule.

Example 7 – A control system with an estimator have multiple ways of information in the control process, the control algorithm uses information both directly from the sensor and information that has been derived by the estimator (see figure 4.2). A composite event at the control algorithm ensures that new information from both the sensor *and* the estimator is available before the computation takes place.

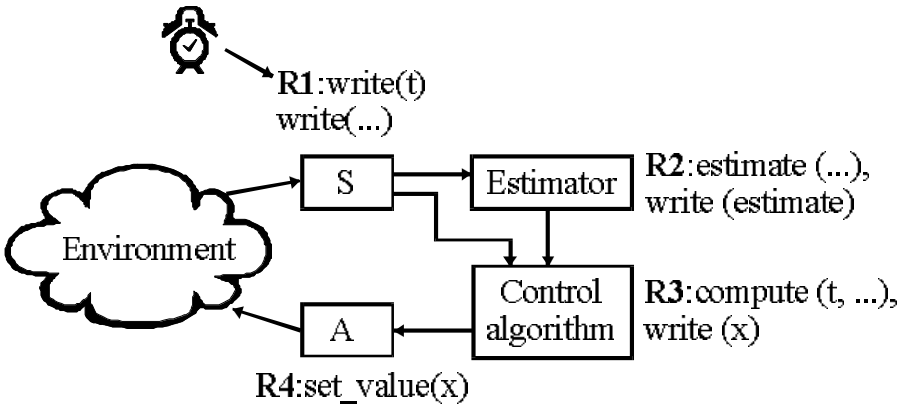

***Figure 4.2** - A Control system with an estimator implemented into active rules.*

Each function (or rule) will write its result to the database. The value written by each function is unique and is not written by any other function. Thus, the control process is implicitly defined and only has one way trough the control system.

4.1.2 Basic Control Functions

The basic control functions (sensor reading, actuation, control algorithm, predictor, estimator, and digital filter) defined in section 2.1.2 all have one thing in common, namely they all process information. When information is received they calculate new values and deliver it as output. This commonality makes them all suitable to be expressed in the same way using active rules.

Example 8 – An example of expressing a control application into active ruls.

A control application is illustrated in table 4.1 that consist of an estimator and a control algorithm. Some of the measurements cannot be done in the process, but have to be estimated using an estimator.

There are four basic and separate control functions that can be identified to perform the actual control process, namely:

1. *Sensor-reading* – Information about the current status of the controlled system are measured by sensors.
2. *Estimator* – Estimation of values that are hard or even impossible to measure directly in the environment.
3. *Control algorithm* – The actual control process that calculates the new control values to correct for a possible error compared to the desired behavior, e.g., speed, flow, position etc.
4. *Actuation* – Perform the corrections in the environment via an actuator.

In this example the basic control functions can be implemented by active rules that uses the updating of the database as events, except for R1 that are trigged by a timer.

Rule	**ON (Event)**	**IF (Condition)**	**DO (Action)**
R1: Write_sensor_data	timer_interrupt(I1)	-	Read_senors (), Write (t), Write (…)
R2: Estimate	update (…)	-	Estimation (…), Write (estimate)
R3: Calculate	update (t) AND update (estimate)	-	Calculate (t, estimate), Write (x)
R4: Actuate	update (x)	-	Actuate (x)

Table 4.1 - *The basic control functions implemented as ECA-rules.*

In table 4.1 the basic control functions are implemented as ECA-rules. The first rule (R1) is trigged by a timer interrupt (I1). R1 reads the information from several sensors and write the values into the database. R2 is then trigged by the updating of the database variables (here represented by (…)) written by R1. The estimation is calculated and written to the database. R3 is trigged by the composite event of *update (t)* and *update (estimate)* which ensures that the control algorithm only starts when a new and updated variable from the estimator is available. Finally, the fourth rule (R4) is trigged by the update of *x* in the database and *x* is sent to the actuator.

The other basic control functions (predictor and digital filter) are also possible to implement as rules in the same way as the control algorithm and the estimator. The event(s) that trigger a function is the update made by another basic control function(s) earlier in the control process.

4.1.3 Control Management Functions

A control application may be a control system that handles a simple process. On the other hand, the problem at hand can be a more troublesome process that must be treated with greater care. The usage of active rules gives the possibility to implement more functionality to the control application. We consider some functions that are of interest:

- *Supervision* – Some control applications work autonomous without any supervision, while others are of greatest interest to a supervisor. There may be special situations when an alarm must be set to inform a supervisor.
- *Adaptation to changes in environment* – The environment that a control system work within may change over time and to maintain a given control performance, the control system must adapt under run.
- *Handling of other changes in environment* – There may be other changes in the environment such as disturbances that causes the control process to go out of hand. To prevent the system from a hazardous behavior actions must be taken.
- *Handling of transients* – An ordinary control system is built sufficiently fast for handling changes in the environment. If faster changes appear the system may have an unwanted behavior or even become unstable. It is therefore desired to handle this.
- *Handling vacant sampling* – If, by some reason, new information from the latest sensor reading does not show up at the controller the system must continue to work.

Supervision: The supervision of a process is either done periodically or at special occasions, e.g., when a given threshold is passed. The supervision can consist in a message sent to a GUI or data written to a logfile for later inspection. In either case the same mechanism can be used.

Consider an active rule that triggers on the update of a database variable will capture all new data in the database. As the database is the only communication medium for information in the system the rule will therefor not win anything in checking the data more frequently. To specify this rule it is sufficient checking the *update* event, i.e., an EA-rule can be used.

To check the environment for special occasions, an ECA-rule are used to specify if the occasion has occurred, i.e., if a given threshold has been passed. It is also possible to specify a rule that triggers if the actuator has not yet received an actuator value at the time of next sample instant. This can be done by an interrupt-driven rule at the same physical node as where the actuator function is.

Adaptation to changes in environment: The environment in which the control system works within may change, e.g., by drastic changes in measured values or because the different units in a CIM-application must be treated differently.

An active rule triggering on a specific change in the environment can change the control system variables or even the information flow to adjust its operation.

Example 9 – Consider a CIM application where different parts can be manufactured. The control system can adjust its parameters to have an optimal treatment of the manufactured part. As an example the pressure in a lathe might be adjusted to the varying degree of hardness on the material. The pressure is adjusted by changing the control system variables that in turn is the basis for the calculations in the control algorithm.

Example 10 – Consider a water purifying plant where one step of the control application is to add chemicals to prepare the water for a forthcoming step of biological cleaning. Normally, it is sufficient for this control application to regulate the acidity in order to provide a convenient environment for the bacteria. Some bacteria are sensitive to too low quantities of the organic matter that it is supposed to eat and break down. To prevent the bacteria from dying, you might have to *add* more of the polluting organic matter. An active rule can check if the level of organic matter fall under a given value and then start to add that organic matter.

Handling of other changes in environment – Active rules can watch for dangerous situations that happen in the environment and react accurately. Either the situation can be handled with an exception or even by shutting down the entire system, i.e., a fail-safe system. In the case of using an exception the dangerous situation either is treated by a correcting action or the whole system must enter a new state. The fail-safe case is easily expressed using an active rule, while the last is more complex. The more

complex situation can be handled using *mode change* in the database, this will be treated in section 4.2.

Handling of transients – A control system is built sufficiently fast for a given environment. If a transient appears in that environment, an unwanted behavior of the control system is avoided if the transient can be handled. *One* of the reasons that the built control system does not handle the situation is that it reacts to slow, i.e., the environment changes a lot before the system reacts. The solution to this is to build a system that reacts faster when a transient is present.

The delays in a control system as described in figure 4.1 are those that arise at the detection and execution of R1, R2 and R3, we name them τ_s, τ_c and τ_a respectively. The delays τ_s and τ_a are necessary and we cannot do anything at all to them. The only possibility is to do something to reduce τ_c. The problem is that we still want to keep a good control performance and unless we cannot do anything to optimize the control algorithm an alternative solution is necessary. One way of solving this is by defining an additional rule that is trigged by a transient and executes a faster (and less accurate) algorithm than the ordinary one. In figure 4.3, R4 is trigged by an update as in R2, but with a condition that detects a transient:

***R4: ON** update (t) **IF** ABS(t.previous - t) >= transient **THEN** call fast_algorithm (t)*

As the delay of R4 (τ_{c2}) is shorter than τ_c the total delay from sensor reading to actuation has decreased by τ_c- τ_{c2} and thus we have a faster response to a transient.

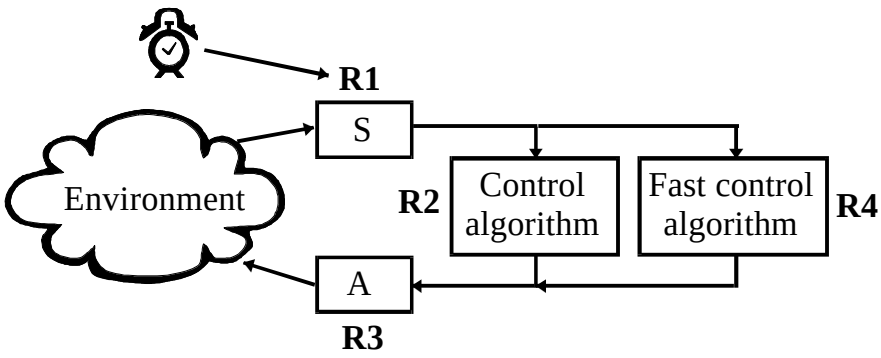

Figure 4.3 - *An additional rule (R4), trigged by a transient and executes a faster control algorithm than R2.*

This solution implies that we can decide the rule ordering. R4 must trigger before R2 in order to be useful. Even R3 must trigger before R2, otherwise the total delay will increase by τ_{c2}. In the case of a central active database the introduction of priorities to handle this, the situation is troublesome as R2 is trigged before R2 in the case of a transient. Even so, the idea of handling a transient in this way is theoretical as the execution time of the control algorithm is to be compared to the database interaction.

4.2 Mode Changes for Altering Control System

When control systems are built using a distributed active RTDB there are other database functions that can be used. One of these is the possibility of using mode changes that are available in many databases. This functionality can be used as a way

of adapting the control system to changes in the environment as described in the preceding section.

The use of a mode change will give a very fast response to a given event, such as when a dangerous situation is detected. In one step the entire system can exchange all control functions to have a proper behavior in the detected situation, i.e., the mode change is propagated and executed on all the other physical nodes in the system before other rules on other nodes are executed.

Using mode changes can also be a simple way of compensating for a broken node in the system. If a rule can detect that a node in the system is not working in a proper way, a mode change can easily alter the system so that the rules on the broken node are distributed on a working node.

4.3 Importance of Rule Ordering

There are several rules used to create the control process, several of them may be on the same physical node. There will however only be one way of information flow through the control process and, thus, there will not be any need for a specific rule ordering.

In contrast, the management control functions requires some consideration. Supervision of the control system may be important but can probably be executed when the system has spare time. It will at least be executed before the next sample interval unless the system suffers from an overload situation. If the supervision

function writes a log, the delay to execution of the function can be compensated if the delay is known.

To have the system prepared to the changes within the present sample period in the environment these management control functions must be executed before the basic control functions start to process new information. This may however be a problem as the sensor control function will write its information to the database and, thus, the information is propagated to the other nodes in the system. This situation can be solved in three ways: (1) the rule for adjusting the control system variables has a higher priority than the propagation of information. (2) we include a check of the condition in the read-sensor rule. (3) we add an extra rule that make a copy of the sensor value written to the database and let the control functions use the timestamp of this new value in the calculation.

The first solution handles the situation of running the rule for detecting a specific condition in the environment before the basic control functions start to handle the new information. There is however a problem if something in the control system environment is to be changed in this sample period, e.g., the control system variables. The values this rule writes to the database must be propagated *before* the information written by the sensor function. Otherwise, all or some of the control functions in the system use the old control system variables in this sample interval.

In the second solution all the rules that check for specific conditions in the environment must be written into one rule. This also implies that all the extra control functions must be executed at the sensor node.

The third solution implies that an extra database variable must be used, a value that must be propagated via the network to the other nodes. It also implies that priorities of the rules must be used so that the function for detection of the specific condition is executed before the copying of the sensor value. Otherwise the information written by the detection-function is not used until the next sample interval.

There is also the function for compensation for transients in the system. If the two control algorithms, the fast and the ordinary (see section 4.1.3), are at the same node in the control system, priorities must be used. In order to have any effect, the function for detecting a transient and calculating a response must be executed before the ordinary control algorithm.

Chapter 5

Distribution of Control Functions

In the preceding chapter the modeling of a control application into active rules were described. The final step of building a distributed control system is the distribution of these active rules to the different nodes in the system. As the control system relies on the database replication functionality there are delays introduced into the system as the data is replicated. The introduced delays are of greatest importance as control systems are sensitive to delays. This chapter describes the consequences of distributing the rules to different nodes. In the section 5.1, considerations of importance when deciding on distribution of active rules will be discussed. Further, in section 5.2, we look at how the random time-delays can be handled by using existing tolerant control algorithms and active rules. Lastly, in chapter 5.3, a discussion is hold on where to distribute the rules when building a control system.

5.1 Issues in Distributing a Control Application

Building a distributed control system with a distributed active RTDB as basis introduces several aspects compared to the development of an ordinary control system. The replication of data between the nodes in the database is handled via a network of some kind. To ensure a given delay for all data replicated in the database, a real-time network must be used. A real-time network is a network with timely and reliable behavior. Even so, it is most unlikely that we get a constant communication delay. The distributed environment of the database implies that several nodes causes network load as all updates in all the nodes are replicated, i.e., there will not be only one node sending information. If the communication goes via a non real-time network such as CSMA-CD, the delays will be even more unpredictable. We assume that the replication delays will have a random distribution. A closer study of how delays in CAN and Ethernet networks are distributed can be found in [Nil98].

Time sensitive functions – The basic control functions are all sensitive to delays introduced in the system, especially in the case of random delays. This means that the delays must be considered and compensated for in order to deliver a given control performance. The management control functions are not sensitive to the randomness of the delays. It is instead the actual length, i.e., the latency, which is important.

Synchronization of multiple sources of information – A control application where there are several distributed sensors and actuators implies several different delays (see figure 5.1). We assume that sampling in the sensor nodes are done synchronized and that the information are written directly to the database. The measurements of interest for a given control function arrives at different points in time and, thus, have individual random delays. Normally the computation in the control function cannot start until the information from the sensor with the longest delay has arrived. It might, however, be possible to define a rule that use a certain number of the needed values and infer the others. This rule can either be time-triggered and start the computation at a given point in time or trigger when enough values are present.

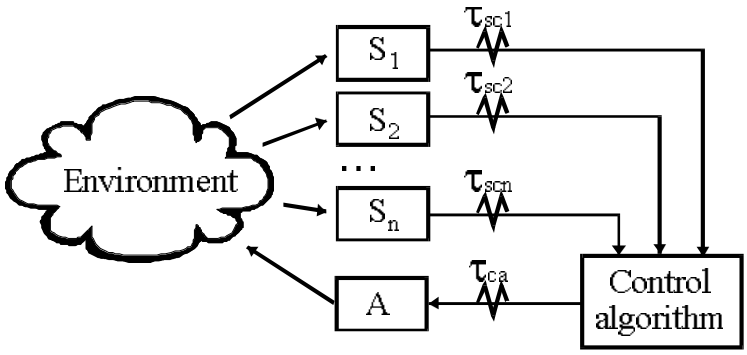

Figure 5.1 *- Control system with several sensors*

Example 11 – Consider the setup in figure 5.1 where we have a system with several sensors connected to a control algorithm performing calculations. We will have n delays, τ_{sci}, $i \in \{1,\ldots, n\}$. The information from each sensor arrives at different points in time to the control algorithm. Calculation cannot start until the sensor reading with

the longest has arrived. The calculation of the delays must only be done once as it is assumed that the sensor readings are synchronized.

Total delays – Using information in several steps means that the delay increases in each one of them. When the delay in the system is to be compensated for, the total delay from the sensor reading to actuator must be used, not only the delays from the previous and next steps in the control process.

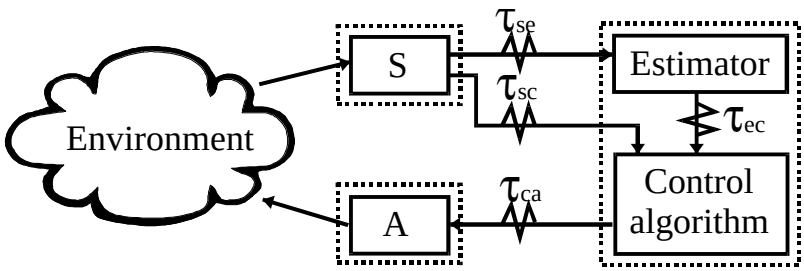

Figure 5.2 - *Control system with delays*

Example 12 – Consider figure 5.2 where the delays are to be compensated for in the control algorithm. The control system is built with three separate physical nodes where the estimator and control algorithm are at same physical node. The delays are: τ_{se}, delay from sensor to estimator; τ_{sc}, delay form sensor to controller; τ_{ec}, delay from estimator; and $\tau_{ca,}$ the delay from algorithm to actuator. We define $\bar{t}$ as the average time for replication of information between the nodes. We assume the time for calculation in the estimator and control algorithm are insignificant. We also make the assumption that $\tau_{ec} << \tau_{se}$, $\tau_{ec} << \tau_{sc}$ and $\tau_{ec} << \tau_{ca}$ since the database update and event detection is done within the node. Consider the case where the delays in the system

should be compensated for. Using the delay from previous function in the control process and to the next function in the control process in compensation gives an average delay of $\bar{t}$. The total average delay that affects the system, and that should be compensated for, is instead $2\bar{t}$ ($\tau_{ce}+\tau_{ec}+\tau_{ca} \approx 2\bar{t}$ or $\tau_{ce}+\tau_{ec}+\tau_{ca} \approx 2\bar{t}$).

Level of consistency – In this dissertation we consider the consistency that the database can assure. We will assume three separate levels of consistency, namely:

1. **Immediate consistency** where data is replicated to all nodes and committed at the same time in all nodes. To ensure this level of consistency a distributed atomic commitment protocol must be used [Mul94].
2. **Eventual consistency** where data is replicated at a later point in time, this can either be done with as soon as possible (ASAP) [Gus95] or within a bounded time [Lun97].
3. **No guarantee** of consistency is given.

The three levels of consistency involve different time delays, due to control systems sensitiveness to delays the lowest appropriate level of consistency should be used. To ensure the immediate consistency in a database a distributed atomic commitment protocol (DACP) must be used. This protocol has overhead in communication between the nodes in the system and, thus, takes valuable time to execute. In [Mul94, chapter 6] a DACP is described. Under the assumption that the underlying network have a broadcast facility, the DACP involves two sendings of

information from the coordinator, the node where the information is updated, to the participants, the other nodes in the database. The participants each sends one response on the network before the transaction are committed and one more each when committed. If the time for one transmission on the network is *d* and *n* is the number nodes, there is *d (2+(n-1))* time needed for a commit and another *d (n-1)* before the protocol has finished. There are also some overhead involved as the protocol is executed in the coordinator and participants.

The second and third levels of consistency both involve a propagation of data using the broadcast facility of the underlying network, i.e., *d* of time for delivery. The second level also involves some overhead in node where the data is propagated, the time for the overhead depends on the algorithm. In [Lun97] a version vector algorithm is described. This algorithm will use *n*m*l* number of comparisons, where *n* is the number of nodes, *m* the number of objects, and *l* the number of transactions in the log filter, to detect an inconsistency. Only in the case of an inconsistency, a conflict resolution mechanism is executed.

In choosing an appropriate level of consistency in a control system based on a distributed database we must examine if inconsistencies can occur, and if they do, in what situations. Inconsistencies occur when we have a read-write-conflict or write-write-conflict in the database, i.e., if one or several control functions use and update the same data in the database. First we look at the basic control functions.

All the basic control functions in the control process read and write data in the database. In normal operation of the control system we assume that all control functions in a control process caused by a sample instant, *ht*, has been reached before the next sample instant, *h(t+1)*. Each function in a control process receives data from

the previous function as the update event of a database variable is detected. The function makes a calculation based on the database variable, and updates another database variable when finished. There will not be any two functions updating the same database variable, and thus, there are no write-write conflicts. There are not any two basic control functions having any dependence between each other that can cause a read-write conflict.

If there is a high load in the system and, thus, the control process has not yet reached the actuator at the time of next sample instant, *h(t+1)*, there might be a problem. This may cause a control function to update the same value as already been updated recently and a queue is therefore needed. If a FIFO queue is used there will still not be any inconsistencies. If such a high load in the system is allowed, the performance may be enhanced by the technique described in [Cha95].

Among the management control functions have supervision function. This function delivers information either to a log, or to a Graphical User Interface (GUI) for presentation of interesting information. However, this function will not write the same information as any other function and, thus, will not cause any conflict.

The control function for adoption of the control process to the present environment can update the control system variables that are used by management control functions. No management control function will make any change to the control system variables, but there may be several rules of this kind that can change the values. In the case that two or more of these rules are executed concurrently, either on one node or on separate, there may be both read-write conflicts and write-write conflicts.

Functions for handling of exceptions in the environment are not the source of conflicts unless they are executed concurrently and result in contradictory decisions. This must not be allowed.

When transients are to be handled an extra rule is used to shorten the delay from the sensor readings to actuation. Consider the case where only one rule is used for actuation. In this case, both the ordinary control algorithm and the fast control algorithm must write data to the same database variable to have it actuated. This may be a source of a write-write conflict. However, the system will result in an expected behavior. Although the system output is correct if the lowest level of consistency is used, the inconsistent state may not be wanted. To avoid this situation we may instead use two actuation rules, one for the ordinary control algorithm and another for the fast control algorithm. Now there will not be any write-write conflicts.

5.2 Handling Random Time-Delays in System

Using the database replication mechanism for sending information between the different functions in a control system introduces delays. We cannot know the exact delays in advance and, therefore, cannot build a control system using standard control theory[5]. Standard control theory implies that we have a constant time delay. What we can do is to timestamp all information with a global clock. By doing so we will have

[5] What we can do if we have a bounded delay, is that we wail until a given deadline is reached (see [Luc90]). This will however affect the control performance.

the age of the information by comparing the time-stamp with actual time when we want to use it.

5.2.1 Using Tolerant Control Algorithms

The two control algorithms proposed in [Nil98] uses knowledge of both the present delay(s) from sensor(s) to controller and a history of earlier time delays from controller to actuator. It is assumed that a control system only consists of three different control functions: sensors, controllers, and actuators. In this work it is assumed that the control system can be divided further by defining separate control functions. In order to use the control algorithms from [Nil98] with management control functions between the controller and sensor/actuator, we must have the total delay from sensor to control algorithm and from control algorithm to actuator.

5.2.2 Calculating Delays

In section 5.2.1 we use knowledge about the delays in the control process. In order to use the control algorithms in [Nil98] to compensate for the delays we need:

1. A timestamp of each data written to the database. This timestamp will be used to calculate the delay from the previous step in the control process by comparing it to the actual time.
2. Knowledge of delays in previous steps of the control process.
3. Knowledge of historical delays in forthcoming steps of the control process.

The first requirement is fulfilled by the assumptions that the database provides timestamps of information written and an available global clock.

To fulfill the second requirement to be fulfilled there are at least two possible solutions. Either (1) we use a time counter to add the delays through all paths or (2) we know all delays and what paths the information went. The first solution requires that we add extra information in the database about the delay from the sensor, i.e., a counter that is increased with the latest delay. In the second solution the total delay can be calculated with the knowledge about the control process path and by the fact that the database is fully replicated. All data with timestamps earlier in the control process is available at the time for calculation, but it is actually only the timestamp of the data written by the sensor function that is needed.

The third requirement can also be solved partly with the same solutions as the second requirement. However, both solutions must be used with an extension. The control algorithm must know the final delay in the control process in next sample period. In both solutions the actuator rule must explicitly write the timestamp at actuation in the database, and let the database replication mechanism deliver the timestamp to the control algorithm.

5.3 Where to Distribute Functions

When building a control system with the basic and management control functions there are many ways of distributing these functions. Generally the system would have the same behavior wherever the control functions are distributed, but the delays are

introduced in database replication and affect the performance. The most important consideration in distributing the control system is that the total delays in the system are as small as possible. The possible performance of a control system decreases as the delay gets longer.

When the basic control functions are distributed to the different nodes in the system, using tolerant control algorithms and knowledge about the delays can compensate for the delays. To the management control functions, on the other hand, a short latency is desirable. This implies that these functions, if possible, are placed on the sensor node.

In section 4.1.3, a function is proposed to handle transients that come into the system. Used in single node the function is of little use as the delays introduced are small. Moreover there is also the problem with rule ordering (see section 4.1.3). In a distributed environment the delays introduced by database replication are larger. Thus, the profit of using this function can be greater. In case that the ordinary and the fast control algorithm are at separate nodes, the problem of rule ordering will not either occur.

In section 3.2.2 we introduced four separate allocation schemas. All four are simplified and only displays the control system as three separate logical nodes (sensor, controller, and actuator node) that can be distributed to separate physical nodes. Following is a section studying of the consequences of using each schema. We will assume that delays in a physical node are insignificant compared to the delays in replication between two separate physical nodes.

5.3.1 Delays in Different Allocation Schemas

In figure 3.1 a) the sensor- and controller functions are at the same physical node. There is a delay between the sensor/controller node to the actuator node. In this case the function for handling transients are of less use, as the only timesaving is that of the faster algorithm compared the normal one. There is no problem of rule ordering in this case. The tolerant control algorithm will have to use historical time-delays for *prediction* of delays.

In figure 3.1 b) the controller and actuator functions are at the same physical node. The delay will be between the sensor node and the controller/actuator node. As in the previous schema there is no significant timesaving in using the function for handling transients. There will also be need for rule priorities to have it to work properly. In this case the tolerant control algorithm can use *actual* delays instead of predicted, thus it can achieve a better performance.

In figure 3.1 c) there are delays between all three functions. This makes it more suitable to use a transient handler function. Under the assumption that the delays between sensor to controller and controller to actuator are equal, the timesaving will be τ_{ca} plus the difference in execution time between the two control algorithms. The tolerant control function will need to use both the present delays and a prediction based on historical delays. The performance will be less good than in 3.1 a) and 3.1 b).

Finally, in figure 3.1 d) the sensor and actuator functions are at the same physical node. In this case the timesaving for the transient handling function is $\tau_{sc}+\tau_{ca}$, and

allows also an almost immediate response. The performance of the control system in other respects is the same as for 3.1 c).

Chapter 6

Results

This chapter summarizes the work done and elucidates the results. A methodology is discussed for dividing a control application into rules. There is also a discussion of what should be considered when distributing the rules over several nodes. Further, there are some implementation issues discussed concerning how transients are handled and how the level of consistency affects the performance of the system.

6.1 Implementing Control Application Using Active Rules

In order to use an active database as the only base for building a control system, it must be shown that all functions of a control system can be expressed using active rules. A control system using ordinary control theory is sensitive to delays. Implementation in a distributed environment can thus be a problem as random delays are introduced in communication. A distributed database has delays as information is replicated between the different nodes.

In this work we have shown a way of expressing all basic control functions into active rules. Each control function is defined by a rule that is triggered by the update of one or mode database variables. When the rule is triggered it uses the database variable(s) and performs its calculation, thereafter writing the new value to the database variable. This new variable will be replicated to all the other nodes in the system and does trigger the next function in the control process. In this way the control process is implicitly defined by the way the rules are triggered. All the basic control functions normally can be expressed using EA-rules, an exception is when a transient handling function are used. In this case an ECA-rule are necessary if the normal control algorithm must not be executed when the fast (and less accurate) control function are executed.

To start the control process we uses a time-driven process at one or several sensor nodes. This time-triggered process reads the sensor and put the value into the database. The end of a control process is when the actuator function is triggered by the output value update, and performs the actuation.

We have also shown how some additional control functions can be implemented using active rules. The rules can be used to detect specific events in the environment and react in a proper way. These control functions are not sensitive to random time delays but instead of the latency. We therefore propose to put these control functions on the sensor nodes when possible. The reason to this is: (1) To have a small latency, as no replication is necessary before detection. (2) The function executes before the rest of the control system and can therefore make fast changes if necessary, e.g., a

mode change can alter the control process to compensate for a change in the environment. (3) If a RTDB is used we can also allow a bounded latency of replication. Compared to the basic control functions, these functions also uses the condition part in ECA-rules. The condition is used to separate the ordinary events from those of special interest.

A control system operating in an environment is adjusted to have an optimal behavior. If an unwanted but foreseeable transient appears, the control system might be to slow to react for a proper behavior. We have shown how an active rule at the actuator node can be used to allow a fast response to the transient. If this function is used in a one-node control system there will be problems because of difficult rule ordering. The gain would also be modest, as there are insignificant delays. In contrast, used in a distributed environment, the delays are significant due to replication of data over the network. The problem with rule ordering is solved if the control algorithm and actuator functions are at separate physical nodes. The transient-handler function trigger the updating in the database, at worst, at the same time as the control algorithm.

All communication between the functions in the control system uses the database replication, and thus, the same information will be available at all nodes in the system. The reason for using the database replication are: (1) The database replication mechanism is automatically distributing all information to all the nodes in the database. This means that a given rule will have access to the same information and therefore can be distributed to any node. (2) We does not need to explicitly define what

way the information should go through the network, and thus no reconfiguration of paths are necessary if the control process is changed.

It is shown that the basic control functions will not require any priorities to work properly. The function for detecting important changes in the environment may need priorities of the rules. We propose a solution where an extra rule is used to make a copy of the sensor data, i.e., this rule will triggered by an update of the sensor variable *S* and write a copy to *S'*. *S'* are used by the other control functions instead of *S* and, thus, there will be no conflict.

6.2 Distribution of Control System

When the control system is modeled using active rules, they can easily be distributed to the different nodes in the system. There are some considerations in distributing the rules.

6.2.1 Issues in Distributing the Rules

The control system that is distributed to different physical nodes in a system suffers from delays in communication as information is distributed. Using a distributed database as basis the delays in communication is database replication latency. All the basic control functions uses control theory and, thus, are sensitive to delays. To have a system that can work under these conditions the delays must be compensated for.

As we have seen in chapter 5, a control algorithm may use information from several sources. Several sources of information also imply that all information does not arrive at the same time. An easy way of waiting for all information to be present is to

define the control function using composite events, the calculation will not start until all needed information is updated and available. There is one consideration that must be done; the shorter the time is from sensor reading to actuation, the better the performance. In this case the control process waits in an idle state till all the information is available. It is therefore important that the delays from the sensors to a junction point are about equally long. We make the assumption that the execution time of each function is short compared to the replication between the nodes. This gives us that each path from the sensors to the junction point should have equally many replications between separate nodes.

To compensate for delays in the control system we must know the total delay in the control system. By using the timestamp from the previous function in the control process we only have a part of the total delay. We must know the total delay from the sensor(s) to the actuator(s).

6.2.2 Compensating Delays

To calculate the delays in the system we use timestamps and compare them with the actual time. We proposed two separate ways of calculating the delays: (1) A counter keeps track of the delay in a specific path in the control process or (2) the control process is known and, thus, the timestamps can be used to calculate the delays. We propose that the second alternative is chosen because, in the first alternative, information about what sensor the information should be used in must anyway be known, i.e., the path to the sensor must be known.

In order to use knowledge about previous delay from control algorithm to the actuator, a timestamp must be written in the database by the actuator function. There will be a delay in replication, unless the control algorithm that uses the timestamp for delay calculation and the actuator are at the same physical node. The calculation, and thus the control algorithm, cannot start until information is received from all previous functions in the control process. In the case of the allocation schema in 3.1 a), the control algorithm must wait for the timestamp from the actuator.

6.2.3 Allocation of Rules

When a control system using the basic control functions is expressed in active rules and the delays are not considered, it will have the same behavior wherever the rules are distributed. The delays will however affect the control system and, thus, should be as small as possible. We have examined how control systems are affected as the control functions are distributed over the different nodes. It can be summarized in the following:

- The functions that observe things of interest in the environment must be executed before all basic control functions. These functions should be placed at the sensor node or at the node where the first control function that is dependent of it is.
- The function for handling transients should be placed on the actuator node for best result. This function shows the best result when the sensor and actuator are at the same physical node and the control algorithm is at a separate physical node. The reason is that the delay for detecting the transient is almost zero and the delay for the management control function is at least $2d$.

- The function for supervision is not very sensitive to delays unless an alarm should be presented fast on the GUI. The maximum delay is however maximum one sample period. Otherwise, if the function is used for writing a log the delay are not important as the timestamp are known.
- If the controller and actuator are at the same physical node there is a rule ordering problem. Either this situation is avoided or priorities of rules are needed.
- If the controller is placed at the actuator node the actual delay can be compensated for, it will not be necessary to use predictions based at old delays. In this case we have a better performance of the control system.

6.2.4 Using Databases for Control Systems

Using a distributed active RTDB as basis for building a control system both have its advantages and disadvantages. The largest disadvantage is the delays introduced in the system when the data is replicated between the nodes in the system. There are however delays due to communication in a distributed environment even if not the database replication mechanism would be used. The main difference is the difference in delays. We have examined how the different levels of consistency affect the delays and have shown that when no consistency must be ensured we only have the communication delay of sending one broadcast message on the network. If we assume that the overhead involved in replication is negligible, we have the same delay as when the message is sent directly over the network. The advantages are several:

- A “standard” platform with defined characteristics. We may use the database functionality and do not need to explicitly write all code. If a real-time database is used we can rely on the timely behavior.
- Automatic distribution of information between the nodes. We do not have to write the code for handling the message sending between the control functions in a distributed environment. All information is available at all nodes only at the cost of storage place and delay in replication.
- A defined level of consistency can be ensured. By using the database functionality to ensure a specific level of consistency we can avoid situations where this might harm the control system performance.
- Easy distribution of rules. When the database is expressed in active rules it can easily be distributed to the separate nodes in the system. The delays can be calculated in the same way at all nodes.
- Possibility of defining extra database functionality. The database *mode change* can be used to quickly alter the control system.

6.2.5 Level of Consistency

An application using a database may be sensitive to inconsistencies. To avoid an inconsistent state different algorithms for data replication can be used. In this work we have identified three levels of consistency that a database may provide: (1) Immediate consistency where data is replicated to all nodes and committed at the same time in all nodes. (2) Eventual consistency where data is replicated at a later point in time, this

can either be done ASAP or within a bounded time. (3) No guarantee of consistency is given. We have examined and compared the delay, i.e., the time for execution, for each of the algorithms. We use d as the time it takes to send one message on the network. To provide the highest level of consistency, $d\ (2+n)$ of time, except for overhead in the nodes, are needed to complete one replication in the database. The second algorithm needs d plus execution time of $n*m*l$ comparisons, where n is the number of nodes, m the number of objects and l the number of transactions in the log filter, to detect an inconsistency. When no consistency is guaranteed the delay will only be d as only one broadcast message is needed. The highest level will require much more time to execute its algorithm, as several messages are needed for each propagation. Consider a system with three separate physical nodes. In highest level it requires $5d$ plus overhead to complete the replication. In the middle level it requires d plus the time of $5*m*l$ comparisons at the receiving nodes. When no consistency must be guaranteed only d is needed for replication. Under the assumption that the communication delay is greater than the execution time for comparisons, we can clearly see that the highest level of consistency is most expensive. Next is the eventual consistency level and, finally, the level of no guarantee of consistency. The two last levels are not actually compared to each another but a fair guess is that they are about equally costly.

Further, we have examined whether an algorithm is needed to ensure a higher level of consistency. For most of the control functions there are no read-write or write-write conflicts that can occur, but there are special exceptions: The function for adopting the control system to changes in the environment may cause both read-write and write-write conflicts in the system. In this case we can (1) use an algorithm to ensure consistency or (2) not allowing these functions to run concurrently. In the second case

this implies that these rules must all be executed at the same physical node and that rules cannot run in parallel.

The function for handling transients may also cause an inconsistency as the two control functions may cause a write-write conflict if they are using the same database variable to write their result. In this case there is not much help in using an algorithm to ensure a higher level of consistency as a costly (causes longer delay) conflict resolution must be used. Instead, using separate actuator functions for the two control algorithms can solve the situation.

In choosing an appropriate level we must consider what level is necessary. Most of the control functions will not need any assurance of consistency, but there are situations when is might be needed. The highest level implies a lot of communication and, thus, long delays. We must avoid long delays as much as possible so this level seems inappropriate. The second level with eventual consistency is less costly when no inconsistency occurs, almost the same delays as when no consistency is guaranteed at all. This algorithm may be inappropriate if the situation is complex and a large conflict resolution algorithm must be used. This is however not likely as the system uses simple functions and the worst thing that can happen is that the computations must be done again. We therefor propose that the eventual consistency algorithm is used when consistency is necessary. Even if so is not the case, the cost in *time* for using it in replication all the time is not much higher than for the level of no consistency.

6.2.6 Discussion of ASAP Replication versus Bounded-Delay Replication

In this work we have used a distributed active RTDB. Depending on the real-time properties of the database it can be used to ensure the performance of the control system. In the current version of DeeDS the architecture has a real-time behavior at each node but no guarantee in replication. [Gus95] proposes an ASAP algorithm for replication and, thus, do not ensure a timely behavior. In the case that an ASAP algorithm is used, a control system using this database can either wait for the replication or use a timeout to compensate for a late replication. In the case of a bounded-time replication as proposed in [Lun97], the deadline is not necessary, as we know that the replication is on time.

Chapter 7

Related Work

There are some work done in using databases for control operations in manufacturing and workflow systems [Fal96, Cha98, Lee92, Wan89] and in modeling and design of distributed control systems [Tör96]. In [Krt94, Ray94, Nil98], delays introduced in a distributed environment are handled. There is however no explicit work done in the area using an active distributed database for building *feedback* control systems. This chapter gives an overview of these related areas.

7.1 Active Rules for Data Management in Control Applications

Falkenroth [Fal96] describes in his licentiate dissertation how active rules can be used to data management problems in control applications. He proposes a way of combining traditional control algorithms and high-level operations using active rules. The database stores information about the controlled environment and machinery and the control algorithms are executed on a separate real-time server. The rules are used to interface the model of the environment, as stored in the database,

and the control applications. The information access is improved by a tight connection to the database query processor.

In [Fal96] the controlled environment is controlled in two separate ways: (1) A real-time server is used for feedback control of the continuous process and (2) active rules are used for data management. The key difference to this dissertation is that the active rules are not used for feedback control.

7.2 Active Databases is Semiconductor Manufacturing

In [Cha98] by Chaudry, Moyne and Rundensteiner at the University of Michigan, an approach for multi-step control in manufacturing facilities is described. They have developed the Active Controller, a generic, adaptable and reusable software enabler for multi-step control. The Active Controller uses active rules are used to check for conditions in the environment when compensation for errors is needed. They also investigate issues of integrating the Active Controller with existing CIM (Computer Integrated Manufacturing) environments.

In [Cha98] the active rules are used for correcting errors in a controlled environment, the rules are used to send correction messages forward and backward in the controlled CIM-environment. The rules are not however used for feedback control in a continuous process.

7.3 Database for CIM Applications

Lee and Cho [Lee92] uses a main memory database for building control applications to CIM environments. The WatchMan database that is used, is a real-time database used to store information about the controlled environment. In the database the data can be classified into three categories: (1) *temporary data* maintained with only the latest value, (2) *real-time trend data* maintained with only the recent short-term history of values, and (3) *historical trend data* with all history of data.

In this work the database are not used explicitly for process control but for storing information about the controlled environment. In this dissertation we will use the database rules directly for feedback control purpose.

7.4 A Distributed Database Used in a Cold Tandem Mill

In [Wan89] the requirements for distributed databases controlling a high-speed cold tandem mill are described. A main memory database is used for predictability and moderate latency. They use the database for storing information about the controlled system. The database does not have an active behavior.

7.5 Modeling and Design of Distributed Control Applications

In the doctoral dissertation by Törngren [Tör95] considerations in building distributed control systems are handled. Timing requirements of distributed control applications are considered and ways of compensating these are discussed. Complex control

systems where different speeds for control are used, so called multi-rate systems, are also covered. Methodologies for building distributing control systems are described.
In this work, active rules are not used explicitly for building the control applications but related areas are covered.

7.6 Feedback Control Systems with Random Delays

The paper [Krt94] considers time-delays due to information exchange over communication networks. It is assumed that the traffic conditions in a network will cause random delays that affect a feedback control systems performance and stability. They introduce a control algorithm that is shown to hold the necessary and sufficient stability conditions for control systems. Numerical tests are performed and finally the results are demonstrated in distributed multiprocessor environment with random communication delays. This work handles communication delays in feedback control systems but does cover the platform to build the distributed environment on.

Ray at the Pennsylvania State University also handles communication delays in [Ray94]. His work builds on the conventional linear quadratic Gaussian (LQG) with are extended to handle delays, delay compensated LQG by using minimum variance filtering and dynamic programming. It is argued that the control algorithm proposed are applicable to future generation aircraft where the distributed environment is needed. There are some results presented from simulation using the proposed control algorithm.

7.7 Real-Time Control Systems with Random Delays

In the doctoral dissertation by Nilsson [Nil98], random time delays in distributed environments are handled. There are two control algorithms proposed that handle delays with different delay distribution, either totally random distribution or another for delays with stochastic distribution. There are an experimental study of delays in the CAN (Controller Area Network) and Ethernet where the distribution are shown to be more or less randomly varying. In a separate chapter problem such as sample jitter, setups with several sensors and the use of timeouts are considered.

In this work the delays in different distributed environments are handled, but a platform on which to build the system is not proposed.

7.8 HRT-HOOD: A Structured Design Method for Hard Real-Time Systems

Burns and Wellings proposes a structured design method in [Bur94 and Bur97] for designing systems that can be analyzed for their timing constraints. The method is called HRT-HOOD (Hard Real-Time Hierarchical Object Oriented Design) and is an extension of HOOD that enable common hard real-time abstractions to be presented. There are two activities of in the design of a system using this method: (1) *logical architecture design* and (2) *physical architecture design*. The logical architecture embodies commitments, which can be made independently of the constraints imposed by the execution environment, and is primary aimed at satisfying the functional requirements. The physical architecture takes these functional requirements and other

constraints into account, and embraces the nonfunctional requirements. The physical architecture forms the basis for asserting that the application's nonfunctional requirements will be met when the detailed design and implementation have taken place.

In HRT-HOOD the system is implemented using a programming language. Languages as assembler, sequential languages and high level concurrent programming languages are discussed. In this dissertation we use an active distributed real-time database and therefore have a natural distribution of information in the distributed environment, i.e., we do not have to explicitly write code for the distributing information in our system.

Chapter 8

Conclusions

8.1 Summary

In this dissertation, the main goal has been to use a distributed active RTDB as a base to build control systems. To achieve this, we have developed methods for: (1) Using active rules of the database to model the control system on the database. (2) Distributing these rules to the nodes in the system. When the control system rules are distributed in the database, there are delays introduced by replication of data between the nodes. Depending on where the rules are placed there are different situations to handle. To compensate for these delays, we estimate them and use a tolerant control algorithm for compensation. Some situations require that we use an algorithm for ensuring the consistency level in the database, i.e., immediate or eventual consistency.

8.1.1 Expressing Control System Using Active Rules

The basic functionality in a control system can be expressed using active rules. We have defined some basic control functions as: sensor reading, actuation, control algorithm, predictor, estimator, and digital filter. These basic functions can all be expressed using Event-Action-rules.

The control process can be defined by using database variables as output from each control function. By this solution, the order in which the rules are triggered are implicitly defined and will not change even if the rules are at separate nodes in the system.

To improve the applicability of the control system, other management control functions can be defined as well:

- Supervision of important changes in the environment can be expressed using Event-Condition-Action-rules. Special situations can be recognized and reported to a supervisor or written to a log.
- Adaptation to changes in environment. The system may sense changes in the environment and adjust its system variables for better control performance.
- Handling of special events in the environment. Due to special occasions in the environment the system can alter its control algorithm to handle the new situation. Mode changes supported by the database can be used in this purpose.
- Handling of transients. Rules can be expressed to handle occasional transients in the system.

8.1.2 Distribution of Control System Rules

When a control system is expressed using active rules, it can easily be distributed over the nodes of the system. If the delays are not considered the control performance of the distributed system will be the same regardless of placement.

We have seen that it is important to have optimistic replication in the control system, as a faster algorithm for replication can be used when consistency must be ensured. If a distributed atomic commitment protocol is used we have slow replications between the nodes and, thus, a worse control performance of the system.

We have also seen that the ordinary control functions do not need a higher level of consistency than the eventual consistency, only the function for handling changes in the environment are in some situations dependent of immediate consistency. These situations are when several of these functions are used at separate nodes and, thus, can be executed in parallel.

The rise in control performance gained by the use of transient handler functions is insignificant if it executes at the same node as the ordinary control algorithm. The largest advantage of this function is if it can be located at the actuator node and the ordinary control algorithm is at a separate node. In this case the time saving can be between one to two replications between two physical nodes in the system.

8.1.3 Other Conclusions

In order to build a control system with a distributed active database we require:

- A real-time clock

- Interaction with the outer world
- Time-trigged rules for sensor readings and perhaps for compensation of long delays
- Composite events to define functions with several sources of information
- A protocol for ensuring a given level of consistency (immediate, eventual, or none)

The real time properties of the database can be used to ensure that the system delivers an actuation value within a specified time. In case when the database permits local real-time properties, using a deadline and prediction of the missing value can compensate for the unpredictability of latency in data replication. This may however decrease the control performance of the system to some degree.

The performance of a system built in this way has a graceful degradation as random delays in the system are compensated for. Thus, the system can handle situations with temporary high load with just a somewhat worse result compared to normal utilization of the system. In situations with low load the control performance is improved. The active functionality allows an immediate response, i.e., the sensor(s) to actuator(s) delays will be as small as possible at a specific point in time.

Conventional disk-based systems are slow and too unreliable to use in control applications. A main memory database is much faster and can have real-time properties. There is much speaking for using main memory database instead of conventional disk-based systems in control applications in the future. The dropping

prices and increased density of memory circuits makes it possible to build systems with large amount of memory.

8.2 Contributions

We have made the following contributions:

- We have used active rules to express control system in a distributed active real-time database, dividing a control system into separate functions to be separately defined as rules. In the literature most work has been done in expressing production rules into database rules.
- We have expressed functions in database rules, which allows simple distribution to the nodes in the system.
- We have studied how the rules placed at different nodes in the system affect the system performance.
- We have found a solution where the delays can be calculated at the time they arrive to the control algorithm. The delays from the control algorithm can be calculated in the next sample instant if the actuator rule write a timestamp to the database
- We have defined a separate rule for handling transients in the environment. This rule will work best when placed at the actuator node and separate from the ordinary control algorithm.
- We have shown that consistency used not be ensured unless several functions at separate nodes are used to react for changes in the environment.

8.3 Future work

In this work we have used the tolerant control algorithms defined in [Nil98] as the basis for compensating delays. Using these algorithms in an environment such as in figure 3.1 a), c) and d) on page 29 will involve prediction of the delays to the actuator by using information from previous sample periods. This will, however, affect the performance of the system as the real delays cannot be used, i.e., an estimation of the delay will differ from the actual delay. An idea would be to use the tolerant control algorithm to compensate for delays from sensors to the control algorithm. To achieve better performance, rules could be placed at the actuator(s) to compensate for the *actual* delays instead of the predicted. See figure 8.1.

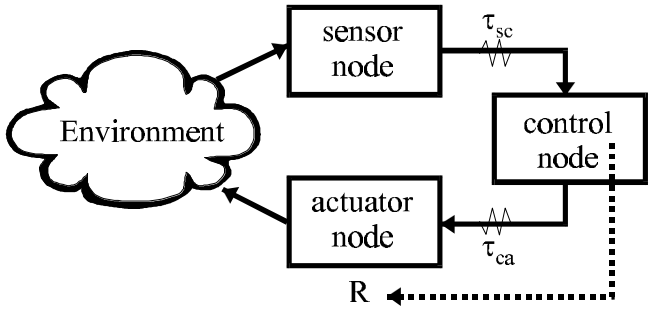

Figure 8.1 *- Compensation for real delay*

In this dissertation we have discussed how the distribution of rules affects the performance of the system. There may be a way to automatically distribute rules to the nodes in the system by taking delays into account. Other consideration to take into account are the processing power, network load and environmental factors. This may

also be extended to a toolset for support of implementing control applications in a distributed database.

Chapter 9

Acknowledgments

First, I would like to take the opportunity to thank my supervisor Prof. Sten F. Andler for all encouragement and help during the work process. I would also like to thank Jonas Mellin for getting me interested in this work and for his valuable comments. The other members of the DRTS Research Group at the University of Skövde have also been of great help with their comments and discussions around the work. Another person who has helped me a lot, and to whom I would also send my thanks, is Daniel Eriksson. He has been exposed to a lot of questions and a number of discussions.

Finally, I would like to express my gratitude to my closest friends for their understanding and support during the periods of hard work. – Thank you!

Bibliography

[And95] S. Andler, M. Berndtsson, B. Eftring, J. Eriksson, J. Hansson, and J. Mellin, DeeDS: A Distributed Active Real-Time Database System, Technical report, Department of Computer Science, University of Skövde, Sweden, 1995

[And96] S. Andler, J. Hansson, J. Eriksson, J. Mellin, M. Berndtsson and B. Eftring, DeeDS Towards a Distributed and Active Real-Time Database System, *SIGMOD Record*, March 1996

[Bes97] A. Bestavros, K. Lin, S. H. Son, Real-Time Database Systems – Issues and Applications, Kluwer Academic publishers, 1997.

[Ben88] S. Bennet, Real-Time Computer Control, The University Press, Cambridge, Great Britain, 1988, ISBN 0-13-762485-9

[Bra92] B. Brachman, G. Neufeld, TDBM: A DBM Library with Atomic Transactions, *In Proc. USENIX*, San Antonio, 1992

[Bri94] O. Bridal, LÅ. Johansson, R. Johansson, H. Lönn, DICOSMOS – Project Definition and Results, Chalmers University of Technology, Lund Institute of Technology, Royal Institute of Technology, 1994

[Bur91] A. Burns, Scheduling Hard Real-Time Systems: A Review, *Software Engineering Journal*, pp. 116-128, May 1991

[Bur94] A. Burns, A.J. Wellings, HRT-HOOD: A Structured Design Method for Hard Real-Time Systems, In *Real-Time Systems*, no. 6, pp. 73-114, 1994

[Bur97] A. Burns, A.J. Wellings, Real-Time Systems and Programming Languages, Addison Wesley Longman, 1997, ISBN 0-201-40365-X

[Cas92] E. Casais, M. Ranft, B. Schiefer, D. Theobald, W. Zimmer, STONE – An Overview, Forschungszentrum Informatik (FZI), Germany, 1992

[Cha89] S. Chakravarty, B. Blaustein, A. P. Buchmann, M. Carey, U. Dayal, D. Goldhirsch, M. Hsu, R. Jauhari, M. Livny, D. McCarthy, R. McKee, A. Rosenthal, HiPAC: A Research Project in Active Time-Constrained Database Management, technical report XAIT-89-02, reference number 187, Xerox Advanced Information Technology, July 1989

[Cha95] H. Chan, Ü. Özgünger, Closed-loop Control of Systems over a Communication Network with Queues, *International Journal of Control*, vol. 62, no. 3, pp. 493-510, 1995

[Cha98] N. Chaudhry, J. Moyne, E. A. Rundensteiner, Active Controller: Utilizing Active Databases for Implementing Multi-Step Control of Semiconductor Manufacturing, *IEEE Transactions on Components*, Packaging and Manufacturing Technology, 1998

[Dat77] C. J. Date, An Introduction to Database Systems, 2nd ed., Addison-Wesley, pp. 407-408, 1977, ISBN 0-201-14456-5

[Elm94] R. Elmasri, S. B. Navathe, Fundamentals of Database Systems, 2nd ed., Benjamin/Cummings Publishing Company, Redwood City, 1994

[Fal96] E. Falkenroth, Data Management in Control Applications – A Proposal Based on Active Database Systems, Department of Computer and Information Science, University of Linköping, S-581 83 Linköping, 1996

[Gre93] J. Grey, A. Reuter, Transaction Processing: Concepts and Techniques, Morgan Kaufmann Publishers, San Mateo, California, 1993, ISBN 1-55860-190-2

[Gus95] P.M. Gustavsson, How to get Predictable Updates Using Lazy Replication in a Distributed Real-Time Database System, Master's dissertation, University of Skövde, 1995

[Had93] V. Hadzilacos, S. Toueg, Fault-Tolerant Broadcasts and Related Problems, in S. Mullender, editor, Distributed Systems, 2nd ed., chapter 5, pp. 97-145, Addison-Wesly, 1993

[Ham96] H. Ham, S. Jeong, Y. Kim, Real-Time Shop Floor Control System or PCB Auto-Insertion Line Based on Object-Oriented Approach, *Computers & Industrial Engineeding*, vol. 30 , no 3, pp. 543-555, 1996

[Jon93] A. Jones, M.G. Rodd, Problems with Expert Systems in Real-Time Control, *Engineering Applications of Artificial Intelligence*, vol. 6, no 6, pp. 499-506, 1993

[Krt94] R. Krtolica, Ü. Özgünger, H. Chan, J. Göktas, J. Winkelman, M. Liubakka, Stability of Linear Feedback Systems with Random Communication Delays, *International Journal of Control*, vol. 59, no. 4, pp. 925-953, 1994

[Lap94] J.C. Laprie (ed.), Dependability: Basic Concepts and Terminology, International Federation For Information Processing (IFIP), 1994

[Lee92] H. Lee, J. Cho, J. Kim, Development of a real-time database-management system for production process-control applications, Microprocessing and Microprogramming, vol. 35, no 1-5, pp. 445-452, 1992

[LeL93] G. Le Lann, N. Rivierre, Real-Time Communications over Broadcast Networks: the CSMA-DCR and the DOD-CSMA-CD protocols, Technical Report 1863, Institut National de Recherche en Informatique et en Automatique, Unite de Recherche INRA Rocquecourt, Domaine de Voluceau, Rocquencourt, B.P 105, 78153 Le Chesnay Cedex, France, March 1993

[Lis97] S. Listgarten, M. A. Neimat, Cost Model Development for a Main Memory Database System, In *Real-Time Database Systems: Issues and Applications*, A. Bestavros, K. Lin, S. H. Son (eds.), Kluwer Academic Publishers, 1997

[Luc90] R. Luck, A. Ray, An observer-based compensator for distributed delays, Automatica, vol. 26, no. 5, pp. 903-908, 1990

[Luc94] R. Luck, A. Ray, Experimental verification of delay compensation algorithm for integrated communication and control system, International Journal of Control, vol. 59, no. 6, pp. 1357-1372, 1994

[Lun97] J. Lundström, A Conflict detection and Resolution Mechanism for Bounded-Delay Replication, Technical Report HS-IDA-TR-97-10, Department of Computer Science, University of Skövde, Sweden, 1997

[Mel97] J. Mellin, J. Hansson, S.F. Andler, Refining Timing Constraint of Applications in DeeDS, In *Real-Time Database Systems: Issues and Applications*, A. Bestavros, K. Lin, S. H. Son (eds.), Kluwer Academic Publishers, 1997

[Moo89] J. Moody, Digital Design with a Distributed Environment in Mind, Distributed Databases in Real-Time Control, Proceedings of the *IFAC/IFIP Workshop*, pp. 9-15, Budapest, Hungary, 16-18 October, 1989

[Mul93] S. Mullender, editor, Distributed Systems, 2nd ed., Addison-Wisley, ACM Press New York, New York, 1994, ISBN 0-201-62427-3

[Nil96] J. Nilsson, Analysis and Design of Real-Time Systems with Random Delays, Licentiate dissertation, Department of Automatic Control, Lund Institute of Thechnology, Lund, 1996

[Nil97] J. Nilsson, B. Bernhardsson, B.Wittenmark, Stochastic Analysis and Control of Real-Time Systems with Random Time Delays, Department of Automatic Control, Lund Institute of Thechnology, Lund, 1997

[Nil97b] J. Nilsson, B. Bernhardsson, B.Wittenmark, Timingproblem i realtidssystem (in swedish), Instutionen för Reglerteknik, Lunds Tekniska Högskola, Lund 1997

[Nil98] J. Nilsson, Real-Time Control Systems with Delays, Doctoral Thesis, Department of Automatic Control, Lund Institute of Thechnology, Lund, 1998

[Ray88] A. Ray, Y. Halevi, Integrated Communication and Control System, *ASME Journal of Dynamic Systems, Measurements and Control*, vol. 110, pp. 367-381, 1988

[Ray94] A. Ray, Output Feedback Control under Randomly Varying Distributed Delays, *Journal of Guidance, Control, and Dynamics*, vol. 17, no. 4, pp. 701-711, 1994

[Rod89] M. Rodd, Real-Time Issues in Distributed Data Bases for Real-Time Control, *Proceedings of the IFAC/IFIP Workshop*, pp. 1-8, Budapest, Hungary, 16-18 October, 1989

[Sch88] B. Schmitdtbauer, Analog och Digital Reglerteknik, ISBN 91-44-26601-4 Studentlitteratur, Lund, 1988

[Séd89] S. Sédillot, Process Control: How to Process Data in Less Then a Second, *Proceedings of the IFAC/IFIP Workshop*, pp. 17-22, Budapest, Hungary, 16-18 October, 1989

[Tör93] M. Törngren, B. Gerbergs, H. Berggren, A Distributed Computer Testbed for Real-Time Control of Machinery, *5th Euromicro Workshop on Real-Time Systems*, Uolu, Finland, 1993

[Tör95] M. Törngren, Modeling and Design of Distributed Real-Time Control Applications: An Automatic Control Perspective, Doctoral Thesis, Royal Institute of Technology, KTH, Sweden, 1995

[Tör96] M. Törngren, J. Wiklander, A Decentralized Methodology for Real-Time Control Applications, *Journal of Control Engineers,* Practice, February, 1996

[Tör98] M. Törngren, Fundamentals of Implementing Real-Time Control Applications in Distributed Computer Systems, *Real-Time Systems*, vol. 14, pp. 219-250, 1998

[Ver93] P. Veríssimo, Real-Time Communication, in Distributed Systems, pp. 447-489, S. Mullender (ed.), Addison Wesley, 1993

[Wan89] J. Wang, H. Zheng, J. Zhu, A Real-Time Distributed Database System in Cold Tandem Mill, Proceedings of the *IFAC/IFIP Workshop*, pp. 83-86, Budapest, Hungary, 16-18 October, 1989

[Åst90] K.J. Åström, B. Wittenmark, Computer Controlled Systems, Theory and Design, 2nd ed., Prentice Hall, 1990, ISBN 0-13-172784-2