Knowledge-Based Real-Time Control Systems : IT4 Project: Phase I

Rytoft, Claes; Hoggard, Nicholas; Åberg, Anders; Uneram, Martin; Gerding, Christer; Rosenberg, Börje; Årzén, Karl-Erik; Larsson, Jan Eric; Petti, Thomas

1990

Knowledge-Based Real-Time Control Systems

IT4 Project: Phase I

ASEA BROWN BOVERI

ALFA-LAVAL

SATTCONTROL

Department of Automatic Control
Lund Institute of Technology

KNOWLEDGE-BASED

REAL-TIME CONTROL SYSTEMS

IT4 Project: Phase I

Asea Brown Boveri AB

SattControl AB

Department of Automatic Control
Lund Institute of Technology

1990

Persons to contact:

Claes Rytoft
ABB Corporate Research
IDEON Research Park
223 70 Lund
Tel: +46 46 168522

Börje Rosenberg
SattControl AB
Box 62
221 00 Lund
Tel: +46 40 226878

Karl-Erik Årzén
Department of Automatic Control
Lund Institute of Technology
Box 118
221 00 Lund
Tel: +46 46 108782

Published 1990
Printed in Sweden
Studentlitteratur

Table of Contents

Preface

"Knowledge-based Real-time Control Systems" is a project within the Swedish Information Technology Research Programme (IT4) that aims to specify and verify a system concept that allows knowledge-based techniques to be integrated with conventional programming techniques in future *Knowledge-Based Control Systems (KBCS)*. The project partners are Asea Brown Boveri AB, SattControl AB, and The Department of Automatic Control, Lund Institute of Technology.

This report is the documentation of the first phase of the main project. Previously, a Feasibility Study has been performed (IT4, 1988). The first phase of the project has concentrated on the specification of the concept and the development of two prototypes that show some important aspects of the concept. The food engineering process Steritherm, a process for UHT sterilization of liquid food products, is used as a demonstrator in project.

The material which this report is based on has been written by the following members of the project group: Claes Rytoft, Nicholas Hoggard, Anders Åberg, and Martin Uneram from ABB; Christer Gerding and Börje Rosenberg from SattControl; and Karl-Erik Årzén, Jan Eric Larsson, Mats Andersson, and Thomas Petti from the Department of Automatic Control, Lund Institute of Technology. Thomas Petti's contributions were made while on study visit from the University of Delaware. The material has been compiled and edited by Karl-Erik Årzén.

1

Introduction

1.1 BACKGROUND

The most important and valuable asset of an industrial company, the knowledge capital represented by the expertise of different key-people, has not been a possible target for computer representation and processing until recently. Knowledge is mostly qualitative, abstract information which is hard to represent and process in traditional systems.

Current industrial use of computers is essentially restricted to applications which can be described formally. The type of information that can be handled efficiently is restricted to quantitative entities expressed in numeric or alpha-numeric forms. In a typical computer program for, e.g., accounting, material administration, technical calculation or process control the computer only handles repetitive and algorithmic functions such as arithmetics, logic, and sequential operations such as sorting or merging.

As a consequence, we have been forced to concentrate the use of computers on well specifyable and repetitive processing of mainly numerical information. This has resulted in a tremendous improvement in the handling of "quantifyable" activities and assets in the industry.

New programming technologies, e.g., object oriented programming, knowledge-based system techniques, symbolic programming, declarative programming languages, applied artificial intelligence, etc., will in the future make it possible to represent and process knowledge.

The visionary and extremely important potential consequences of these new technologies are that the knowledge resources of a company will be subject to the same far-reaching use of computers and increased efficiency as we have experienced in areas suited for conventional computer techniques.

1.2 INCENTIVES FOR DEVELOPMENT

Knowledge handling is of specific importance in process control systems. Control systems are carriers of a collected but extracted knowledge about the whole controlled process. A program for how to supervise, control, and activate the process is the final, concentrated result of a complex processing of knowledge about

- the controlled process itself,
- the components involved and their ways of interaction,
- the produced products and their properties,
- control theory,
- demands and conditions for service and maintenance, etc.

Significant knowledge is accumulated in the process computer and among the operating personnel when a process has been operated for a few years. This knowledge is scattered and neither well represented nor well organized in conventional systems. The reason for this is that today's control systems are not at all suited for processing this type of knowledge. The main weaknesses are the following.

- Incapability to express and implement knowledge-based control functions.

  Control functions based on experience, heuristics, fragmentary knowledge, or qualitative knowledge must today be handled "manually".

- Incapability to represent and communicate the underlying knowledge of different types of control functions.

Although there is a lot of knowledge involved in the analysis and design phases which precede the final programming of the process control system, only the final algorithmic representation and possibly some verbal explanation of it is processed by the computer. The control system is a black box whose content possibly can be explained by the programmers of the system but often by nobody else. All expertise and underlying reasoning is documented separately, or in the worst case communicated verbally to the users who need it in order to improve or extend the process.

- Incapability to visualize a complex process in a user and knowledge oriented way

  Processes, automation, and control systems become more and more complex. This puts increased demands on operators and other users of the control systems without giving them any new supporting tools. Today's systems can, in an excellent way, visualize the process operation based on process signals, but cannot transfer this to higher, more knowledge-oriented abstraction levels suited for the various users of the system.

1.3 PROJECT GOALS

The goal of the project is to define and partly verify a uniform concept for a knowledge-based real-time control system that later on can be used as a base for development of commercial products.

A guidance for the work is a visionary goal of a real-time control system

- that supports all the different user categories associated with the process that is controlled, and
- which uses both knowledge-based and conventional techniques to implement the control system functions needed.

1.3.1 Areas of specific importance

As a result of the first phase of this project, we have reached the conclusion that the solution to the following problems are of fundamental importance for the development of knowledge-based control systems.

- Integration of conventional and new technologies in one uniform system.

- Integration of design knowledge into the operational control system.
- The representation of different types of knowledge in one uniform knowledge-base.
- A uniform interface towards the knowledge in the control system.
- A practical solution of the real-time problem.

1.3.2 Participators

The participators in the project are:

- Asea Brown Boveri AB and SattControl AB, the two leading suppliers of automation systems in Sweden. ABB is one of the worlds largest electro-technical companies and belongs to the forefront in many areas. Some examples are robotics, power system monitoring, and automation. SattControl is among the world leaders in process control and automation. The company has a broad competence within information technology in general, and man-machine communication systems in particular.
- The Department of Automatic Control at LTH, engaged as a consultant, has an international reputation in research on knowledge-based control systems and good contacts with most of the important research institutions around the world working with knowledge-based control systems.

1.4 PROJECT ACTIVITIES

In order to work with realistic problems we have chosen an industrial process as a demonstrator. The process, Steritherm, is an example of an industrial process in the dairy industry – a typical branch for which the concept of a knowledge-based control system could be of interest. Even if the Steritherm process is not in need of a more advanced control system, it is complex enough for being useful as a demonstrator for this project. We have collected knowledge about this process from documentation as well as from designers and we have access to both operators of a real plant and a lab scale version of the process. A simulation model of the process has been implemented in order to have an environment to run prototypes in.

Two prototypes have been built to verify different aspects of our concept. The primary goals for these prototypes are to test ideas concerning how the operator's interface could look and how the knowledge could be structured. The first

prototype is implemented with the use of the multimedia tool Plus and an Apple Macintosh II computer. The other prototype, which is the larger of the two, has been implemented using the real-time knowledge base system tool G2.

In addition to the work done on the prototypes we have looked at what requirements the different users may have of a future real-time control system, what the total concept of a knowledge-based real-time control system could look like and how the central component in such a system, the knowledge, should be structured and managed.

The background and motivation for the project have been presented at the American Control Conference 1989 (Årzén, 1989) in Pittsburgh. The current status of the project will be presented at the American Control Conference 1990 (Årzén, 1990) in San Diego. The project has also been presented at the G2 User Group Meeting in Tokyo, December 12, 1989, at the IT4 conference in Stockholm January 31 – February 1, 1990, and at the DUP conference in Stockholm, March 14, 1990. At these three occasions, the G2 prototype was demonstrated. During this phase of the project, we have exchanged ideas with, and received feedback from, an industrial group from the DUP programme with interest in, and experiences from, expert system applications in the process industry (mainly the pulp and paper industry).

1.5 PROCESS DEFINITION

Process control is the sum of the different tasks that interact with a specified process and the different users of the process. This definition of process control includes control, monitoring, diagnosis, maintenance, planning, simulation, etc. The process control system is the system that realizes these tasks. In the context of this paper a process is defined as a set of operations which perform a physical or chemical transformation or a series of such transformations, and includes transportation of matter or energy and transmission of information (IEC, 1975).

This definition of process includes the process industry, the manufacturing industry, and telecommunication systems. As pointed out by Dhaliwal (1985), processes in these domains have a number of common features:

- The complexity of the systems is such that no single individual or small group of individuals can fully understand them.

- The operations and maintenance manuals may cover several tens of volumes. Maintenance of the documentation is a particular problem. It is difficult to access relevant and correct information speedily.

- Systems are continually changing and evolving since: the process and the operating environment changes; shortcomings in the original specifications come to light; bugs are discovered; and, technology advances.

- The rate of change means that old methods of training and retraining staff are no longer adequate.

- Different users of the systems need markedly different styles of interaction with the system.

- Speedy and accurate correction of faults is required. The hazards of slow or incorrect treatment are:

  - Faults not treated early enough may propagate catastrophically.

  - Dormant faults undetected or left untreated may greatly affect the overall reliability and maintainability of the system.

  - The existing control system may itself be prone to failure and thus may mask the true cause of misoperation.

  - Wrongly identified faults and consequential repair actions may make matters worse.

  - The high reliability of the systems gives problems. Some failures are so rare that it is difficult to ensure that maintenance staff are appropriately predisposed or equipped to handle them

1.6 KNOWLEDGE-BASED CONTROL SYSTEMS

Process industries contains many application areas where knowledge-based systems (KBSs) have been successfully applied. These includes process and control system design, on-line monitoring and diagnosis, closed loop control, off-line troubleshooting, planning and scheduling, administration support systems, etc.

The combination of KBSs and control systems has, from the point of implementation, gone through three different stages:

- separated systems,

- interfaced systems, and

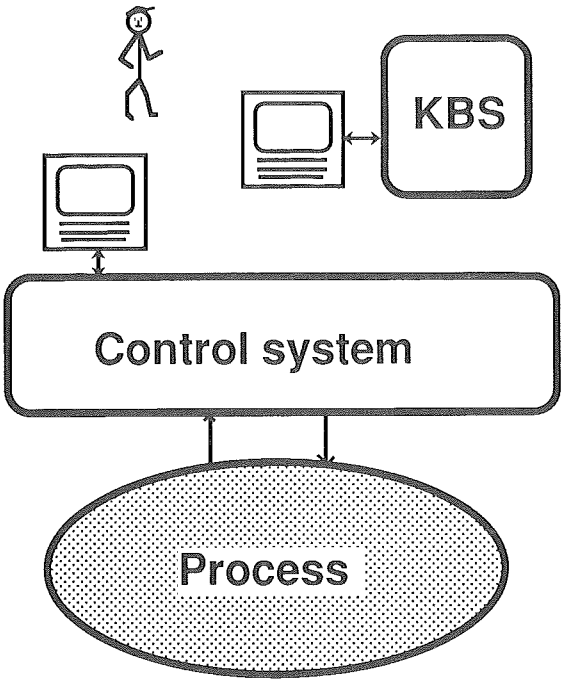


Figure 1.1 Separate systems

- integrated systems.

1.6.1 Separated systems

In the first systems that arrived in the end of the seventies and the beginning of the eighties the KBS and the control system were totally separated. The expert system tools that were used were simple rule-based off-line tools of the EMYCIN type aimed for applications where the user typically interacted with the system in a question and answer consultation dialogue. The user, typically the operator, must manually, upon request, type in all the data the expert systems needed for its reasoning. The situation is shown in Fig. 1.1.

Clearly this solution has many drawbacks. All applications that require a real-time response from the expert system are excluded. Also applications with large amounts of data that must be manually supplied are excluded. However, for certain applications the solution is feasible. Examples are process and control system design, off-line troubleshooting, analysis, and some planning applications.

The initiative to systems of this type came typically from the user industries, i.e., the chemical industry, the power industry, the steel industry, etc. The large

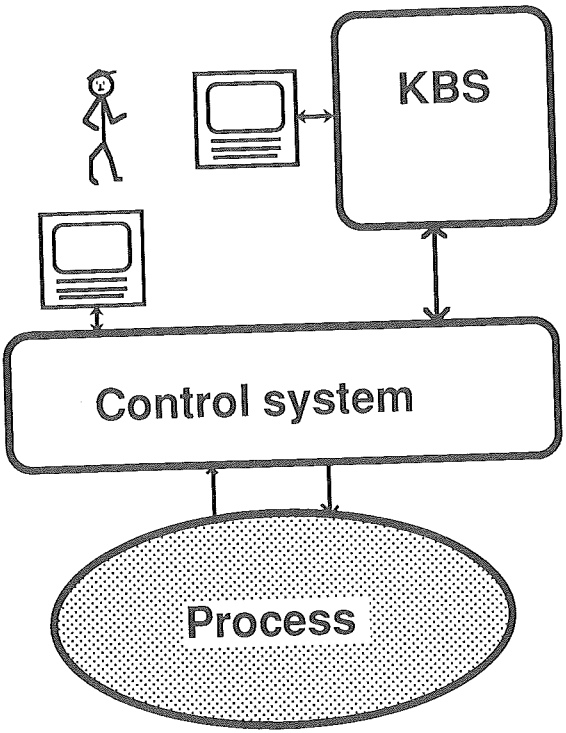


Figure 1.2 Interfaced systems

interest in expert systems in general in the beginning of the eighties made the process industry aware of the new technique and spawned several applications.

1.6.2 Interfaced systems

The interest among the process industry for KBSs created a market for expert system tools specially aimed at the process industry. These tools execute as separate systems that have an on-line interface to the control system as shown in Fig. 1.2. Through the interface process measurements and events are transferred to the expert system for analysis. The output from the expert system is advice to the user, also now typically the operator, and parameter changes to the control system.

The expert system tools used range from systems that essentially are off-line tools that automatically take their input information from the control system to advanced real-time tools with methods for reasoning about dynamic, changing environments; reasoning about time; reasoning under time constraints, etc. Interfaced systems are, at least theoretically, sufficient for most applications, real-time as well as off-line.

There are however important problems with the approach. The major problems stem from the fact that the systems are different. The systems come from different suppliers, require different expertise, use different hardware and software and above all, in many cases have different end-user interfaces. The interface between the expert system and the control system is separate from the internal communication network of the control system and may cause communication bottlenecks. Large amounts of data and information such as process parameters and schematics must be represented redundantly in both systems causing problems with consistency.

1.6.3 Integrated systems

Integration has been a major trend in control systems during the last 30 years. PLC systems and digital control systems have merged. Instrumentation systems and separate supervisory control systems are merging together. Therefore it is natural that the control system suppliers take the initiative to merge together conventional control systems and real-time KBSs to form *Knowledge-Based Control Systems (KBCS).* Here the conventional algorithmical methods for control and supervision are integrated with knowledge-based techniques into one uniform system intended not only for operator support but also for other user groups such as process engineers, designers, maintenance personnel, electricians, etc., as shown in Fig. 1.3.

Integrated systems aim to combine the strong features of real-time expert system tools, i.e., explicit knowledge representation, support for representing heuristic knowledge, object-orientation, rule-based reasoning, support for reasoning about dynamic environments and for temporal reasoning, and user-friendly interfaces with the strong features of modern distributed control systems, i.e., support for algorithmic representation of control logic, speed, distribution and communication, hierarchical control system decomposition, good graphics, and hardware and software reliability.

KBCSs use a single development and end-user interface. The systems are based on a common, distributed or centralized, knowledge and database that makes redundant information unnecessary. In a KBCS, knowledge-based techniques can be integrated at all levels in the control system hierarchy from the local control loops to the supervisory level.

1.7 THE CONCEPT

A key issue in the system concept is the representation of knowledge. In the project the term *knowledge* is used in a wide sense including what is normally termed knowledge in the expert system community, control logic, written documentation, process models, drawings and photographs, dynamic data, etc.

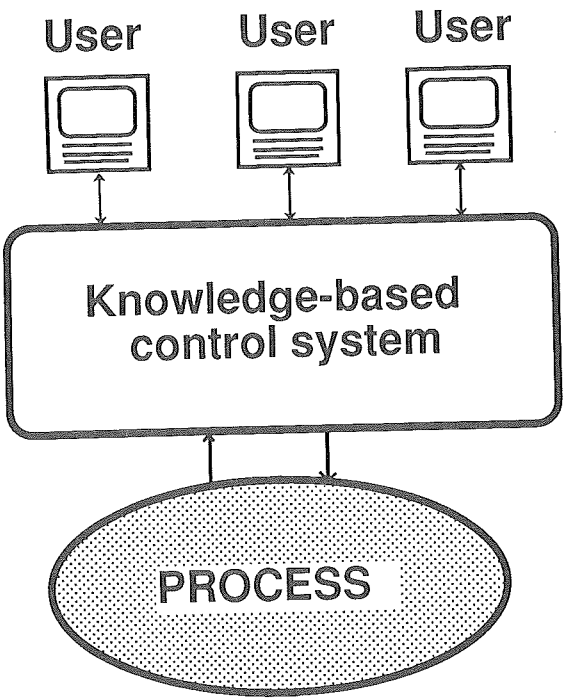


Figure 1.3 Integrated systems

Conventional control systems contain only the final algorithmic representation of all the knowledge involved in designing the control system. A KBCS must also have means for representing other types of knowledge. One example is the different types of written documentation that are today delivered together with the process and the control system. The documentation includes user manuals; installation, maintenance, and operation descriptions; various checklists and instructions; component data sheets; flow schematics, mechanical assembly drawings, etc. New hypermedia techniques for, e.g., representing text and pictures will make it possible to include this in a KBCS.

Another type of knowledge that should be represented in the KBCS is the heuristic, experiential knowledge that different users have about the process and which today is not made explicit and transferred to other users. The process designer bases his design of years of experience, rules of thumb, and heuristic considerations. The skilled process operator knows from experience which are the important process variables to monitor or what may have caused a certain set of fault symptoms.

Models in terms of, e.g., quantitative or qualitative equations that describe the behaviour of the process and its components under various conditions are another important type of knowledge. Used properly they can be the basis for training simulators, decision support simulators, and model-based diagnosis schemes.

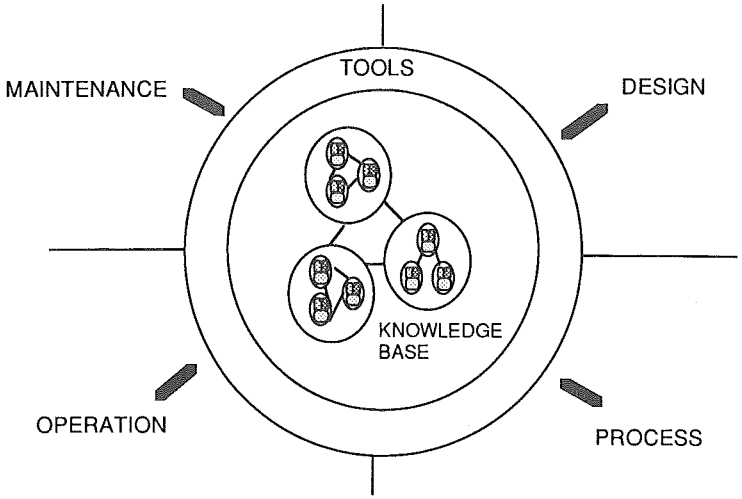


Figure 1.4 The KBCS concept

Specifying a complete KBCS concept is a major task far beyond the range of this project. Some important issues are knowledge representation mechanisms, real-time knowledge-based system issues, implementation and distribution, man-machine interfaces, and communication. In this project we focus on the knowledge representation issues and on the real-time knowledge-based system issues.

The concept is based on a common knowledge base. This knowledge base can be centralized or distributed. The knowledge base contains the knowledge of importance for the application including the issues previously described. The knowledge base is operated upon by a set of tools. The tools implement the different functions in the system. The tools can be divided in two groups: design tools and realization tools. Design tools assist the designers in building up the knowledge-base. Realization tools provide the different system functions used during operation. Examples of realization tools are control tools, diagnosis tools, simulation tools, and planning tools. The task of the realization tools is to extract the appropriate knowledge from the knowledge-base, possibly convert it to a form better suited for execution, distribute it to the different processing units in the system, and set up the communication links and the links to the user interfaces. The concept is shown in Fig. 1.4.

1.8 DEVELOPMENT SITUATION

Development and research in industrial knowledge-based applications is currently very intense, with many large international projects. The driving forces are three different groups: user industries, AI companies, and control system suppliers.

Among the user industries, it is primarily the manufacturing industry, the chemical industry including oil refineries, the paper and pulp industry, the power systems industry, the steel industry, and the telecommunications industry that have started activities. Many systems have been developed and a few also fielded. In the nuclear industry alone, 298 expert systems were reported by June 1989 (Bernard and Washio, 1989). In Sweden, the scene is dominated by the paper and pulp industry and the power industry.

Small AI consulting companies aimed at the process control market are currently emerging. Some of these companies also have expert system tools aimed at the process industries. The best example of this kind is Gensym Corporation, Cambridge MA with their tool G2. G2 is today the technically most advanced and also most widely spread tool of the "interfaced system" type. Over 300 licenses have been sold for a variety of applications in different industry branches. Several of these are used on-line. Other examples of companies of this kind are Cambridge Consultants with their tool Muse, Sagem with their tool Chronos, and Talarian Corporation with TALOS – R-TIME. Also conventional consulting companies such as Framentec, PA Consultants, and Stone & Webster Engineering Corp. are active in the AI – process control area.

The interest is also high among the manufacturers of conventional control systems. Japanese companies like Hitachi, Toshiba, and Yokogawa are developing their own expert system tools tightly interfaced with their control systems. Manufacturers like Honeywell, Fisher, Siemens, ABB, Yokogawa, Allen Bradley, etc. have interfaces between their systems and G2. Honeywell, Bailey Controls, Foxboro, Siemens, and Combustion Engineering all have active AI research groups and products on the way. Combustion Engineering has developed the GDS (Generic Diagnostic Shell) for knowledge-based diagnostics. They have also developed a system for the pulp and paper industry based on G2. Honeywell are developing a shell for monitoring of batch processes. Bailey Controls have a rule-based module that can be embedded in their Super 90 system.

Several large international, as well as national, research programmes concerning knowledge-based control systems are going on at the moment. The most ambitious effort is found in the European Community's ESPRIT I & II programmes with more than 15 large projects (> 40 MSEK) within the AI – process control area.

With few exceptions, all the activity taking place today concerns interfaced systems. Integrated systems based on a common knowledge base are still systems for the future. However, ideas similar to ours are also found in other places. *In order for expert systems to be a surviving technology in the process industry in the long term it is essential that the development of integrated systems take place.*

1.9 OUTLINE OF THE REPORT

This report has the following organization:

Chapter 2: User Requirements. This chapter presents a specification of requirements of the different users of a knowledge-based control system. Examples taken from the demonstrator, the Steritherm process, are included. The requirements may or may not be realistic, but they are suggestions for what the requirements on a future control system could look like. These requirements could be used as a requirements specification for the concept of a future knowledge-based control system.

Chapter 3: The KBCS Concept. Here, a system concept for the integration of knowledge-based techniques with conventional techniques is proposed.

Chapter 4: Knowledge Representation. This chapter presents different techniques for knowledge representation. Besides conventional techniques, objects, rules, quantitative equations, qualitative equations, causal digraphs, multilevel flow models, hypermedia, etc., are discussed. Every knowledge representation technique has its advantages and disadvantages, and which to choose depends on what it will be used for. Many of the different knowledge representation techniques will probably come to use in a knowledge-based control system.

Chapter 5: The Main Knowledge Base. This chapter presents how the knowledge in the main knowledge-base could be structured in order to handle the different uses of knowledge and the different kinds of knowledge representations that these involve. One fundamental idea is that knowledge should be non redundant, which entails hard demands on the structuring of the knowledge and the checking of its consistency. The idea of using a language for knowledge structuring and representation is put forward and discussed.

Chapter 6: Requirements and Limitations. The requirements and limitations that will arise when our proposed concept is to be implemented are discussed. Concerning hardware, e.g. CPU power, memory capacity and interface support, the resources needed are available or will be in a near future. When it comes to software technology it is harder to predict what will come. It is possible to implement the concept using current software technology, but it is most likely that future advances will make it much easier.

Chapter 7: Prototypes. The two prototypes that we have developed are presented. This has been a major part of the project so far. The prototypes have been used to visualize some of the aspects in the system concept.

Chapter 8: Technical Survey Update. An updated version of the technical survey of research and development in the field of KBS for control systems that was presented in the Feasibility Study (IT4, 1988). The G2 system that represents state-of-the-art today is described in detail.

Chapter 9: Further Project Activities. Possible future activities within the project are presented. Three primary alternatives emerge. Firstly, put together a requirements specification of a language to describe the common knowledge-base. Secondly, produce a more detailed specification of the total concept. And thirdly, do further work on the G2 prototype.

Chapter 10: Summary.

Appendix A: Steritherm. The Steritherm process is described in detail.

Appendix B: Travel Notes. Several visits to companies and universities with related projects have been made during this phase of the project. To get new ideas for our system concept and to get an overview of research and development in the field of KBS for process control we have travelled to the USA and Japan and participated in the ESPRIT '89 conference.

Appendix C: Glossary. A short glossary of terms used in this report.

2

User Requirements

2.1 INTRODUCTION

In order to write specifications for a new system it is essential to be aware of the future users' demands and wishes.

The main purpose of this chapter is to try and formulate the users' requirements for a future knowledge-based real-time control system. In the Feasibility Study (IT4, 1988) we tried to analyze and identify the type of knowledge used for operation, control, and maintenance. This time we go somewhat deeper and try to analyze the work of different user groups in order to determine their needs.

2.1.1 User categories

A KBCS is aimed at all different user categories that interact with the process. In this chapter special emphasis is placed on what we believe are the four main user categories:

- Operators
- Maintenance personnel
- Designers
- Production Engineers

The operator is the person who is mainly responsible for the day to day operation of the plant. Maintenance personnel should assist the operators when something goes wrong. What designers do is obvious, but maybe it is not so obvious that there are several different designers with quite different demands. Production engineers is the category responsible for what, how and when things are produced.

There are of course other user categories in a plant. However we have found them to be either mainly covered by the main categories, or to be unusual user categories from a traditional process control viewpoint. The categories we have in mind are:

- Process engineers
- Management
- Sales
- Quality

Process engineers work with redesign of the process and try to find new and more efficient ways to run the process, in order to optimize the process function. The demands from this group can, to a large extent, be found in other groups.

Management is interested in general knowledge about how the production is running, such as goal fulfillment, actual production figures, etc. Most of this information has been covered by other groups.

The sales department is interested in information such as storage content, delivery information, production stops, etc.

The quality department is interested in following the product through the process from raw materials to the end product. These aspects are to some extent covered by the operator. (A practical example of how a quality support function could be implemented is shown in Section 7.3.7.)

The requirements are grouped according to the different user categories. The different user categories are of course not homogeneous groups – for example, designers includes process designers as well as electrical designers and designers of screen layouts for the operator stations, This suggests that there could be quite different demands for functionality within a user category. Some of the requirements are the same for different users, and these are only mentioned once although they belong to two or more of the groups.

For most of the requirements a short practical example from the Steritherm process is given.

The requirements specified in this chapter basically originate from the common experience of the project group. To some limited extent interviews have been conducted with process designers and users. The requirements have also been

discussed with engineers from the Swedish pulp and paper industry. Ideally, the requirements should be verified together with real users. This has not been done due to limited resources and due to the problems of presenting and illustrating the ideas in a proper manner.

2.2 OPERATORS

The operators are responsible for the day to day operation of a plant. In large plants there are of course several operators responsible for different sections of the plant. The operator is the user that works closest to the plant. It is therefore natural that most of the requirements are listed here although many of them are also valid for other categories. The main tasks of the operators are:

- Monitoring,
- Diagnosis, and
- Control.

The functions that the control system must provide the operators with are primarily functions related to these tasks. However, there are other activities, e.g., training, that must also be supported.

All of these activities require a lot of knowledge about the process and the control system. Conventional control systems of today do not provide the operators with this.

2.2.1 Monitoring

Geographical process overview

There should be physically and if possible isometrically correct pictures of the whole process, e.g., photographs and 3-D drawings. The main function of these pictures is to find a link to the operator's reality. The pictures should probably be organized hierarchically.

> Steritherm: A geographical overview of the process could be given in a picture. Every pertinent pump, tank, valve, sensor, indicator, etc., that is of interest for the operator is included and positioned as in the actual plant. A geographical process overview of Steritherm is shown in Fig. 2.1.

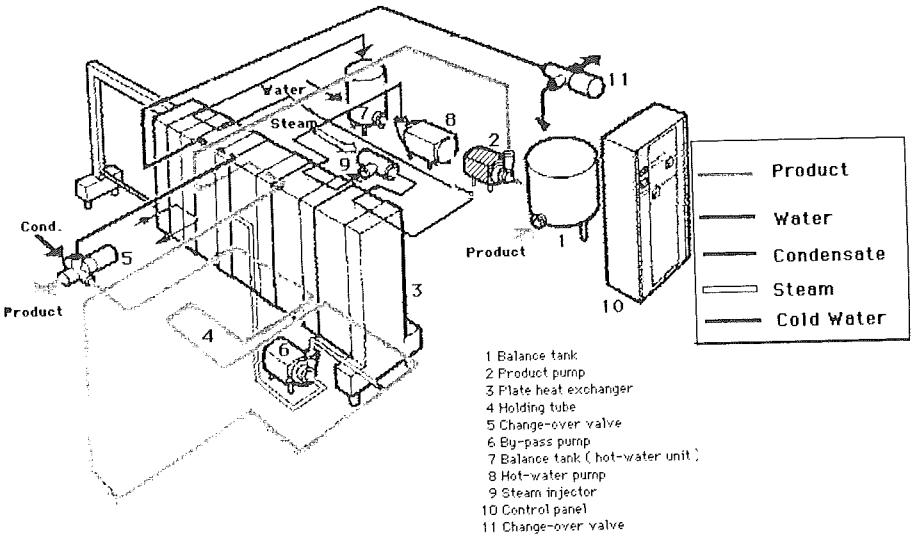


Figure 2.1 Geographical overview of the Steritherm process.

Topological process overview

There should be a topological description of the total system. Process variables, such as pressures, temperatures, flow rates, levels, flow directions, and active/nonactive units, should be presented or indicated in, for example, a picture similar to the one used in the geographical overview, or the form of a process schematic. This overview should be hierarchically structured. It should be possible to move between different descriptions in the system.

> Steritherm: Sensor values and active components could be presented in, e.g., a process schematic of the process as shown in Fig. 2.2.

Focusing on a part of the process

There should be a picture for every topological subsystem that is relevant to the operator. The reason for these pictures is to prune among the available information in order to make it easier for the operator to concentrate on one subsystem. In many cases, the subsystems will have to be divided into further subsystems. In some cases, the lowest level will consist of a single object and a description of how it functions.

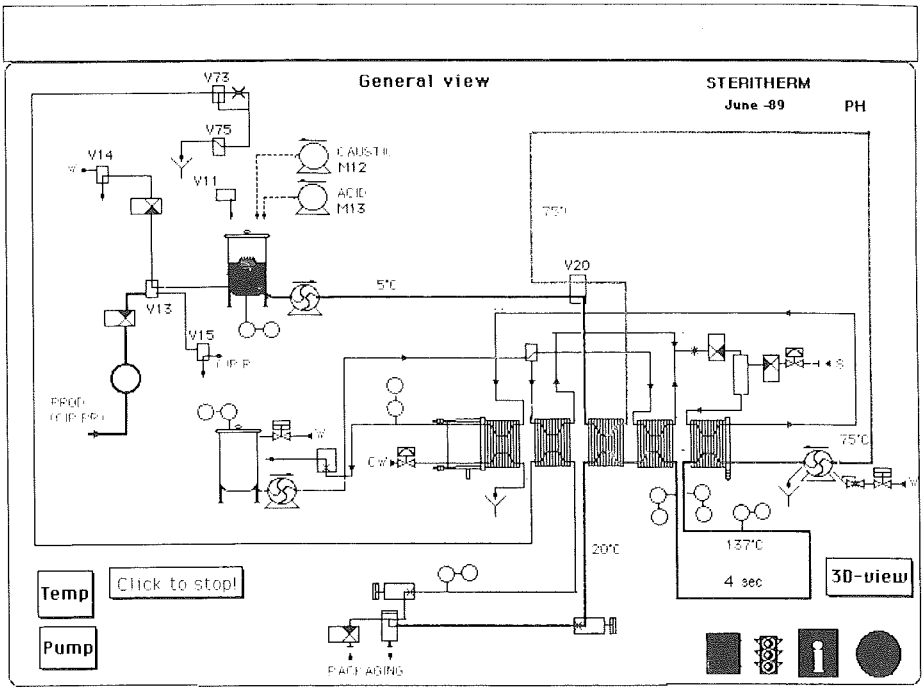


Figure 2.2 Topological overview of the Steritherm process

Steritherm: Examples of topological subsystems are the product flow or the heat exchangers with relevant information about temperature, pressure, flow rate, etc.

Support for automatic picture selection

It should be possible to choose to have automatic selection of which pictures to show, i.e., at every moment the system chooses the pictures which are most suitable with regard to the present state of the process. The user should have an opportunity to change or add among the conditions that the choices are based upon. The operator should always be free to not follow the advice of system concerning the choice of pictures.

Steritherm: An example of automatic picture selection is when the process enters a phase where the temperature of the product in the holding tube is very critical. In that case, a detailed picture of the holding tube could be automatically presented to the operator.

Highlighting

As a complement to focusing on subsystems, there should be the possibility to highlight some predefined topological or geographical subparts of the process. With this solution one would have access to general information at the same time as extra information about a specific function is presented in the same picture. While focusing involves a new, specific picture for every subsystem, highlighting is done in pictures normally used for other purposes.

> Steritherm: A subsystem that could be highlighted is the product flow, which then should be coloured in a more perceivable way, and sensor values of relevance for the product flow, some of which normally would be hidden, should be presented in a conspicuous format. Another way to use the highlighting is to indicate every component and value that, e.g., the pressure of the heat water flow depends on.

Control logic presentation

There should be a presentation of the different sequences of the process, e.g., as a sequence chart. This will be a description of the behaviour of the process, e.g., the state of the different components in each phase.

> Steritherm: A sequence chart could show the status of the controlled components in the different phases of the process. It should also be possible to show the status of the conditions, time, limits, etc., that have to be fulfilled in order to enter the next phase.

Presentation of process data

There should be a continuous presentation of sensor values or calculated process values and there should be possibilities to access other values, e.g., alarm limits and control parameters, such as numbers, meters, etc. Dynamic values should be updated in real time. All these values ought to be presented in connection with pictures of the process, e.g., the overview pictures or focusing pictures mentioned above. Historical data should be available, as graphs. Where this data comes from, e.g., which sensor, and how it is compiled, should be presented.

> Steritherm: An example of process data to be presented is an indication of the degree of burn-on, e.g., the pressure difference in heat exchanger I, or more 'fuzzy' announcements, e.g., 'small', 'rapidly increasing', 'large', etc. The temperature of the product in the holding tube is another critical value that ought to be clearly presented.

Presentation of the overall state of the process

Especially to help inexperienced operators to interpret the information from the control system, there should be a presentation of how things are going with regard to quality. This could consist of, for example, a Chernoff's face that smiles when

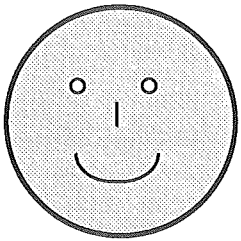


Figure 2.3 A Chernoff's face showing the process state.

the process is going well and looks sad when things are going bad as shown in Fig. 2.3. This information should be available from every picture of the system.

> Steritherm: To present the state of the process one could use an icon in the shape of a Chernoff's face, that is green and smiling when the process is ok, but sad and red when the burn-on is getting dangerously high.

Focusing on an event

It should be possible to focus on events, and not only on topological components, as mentioned above. Depending on the operators choice, a new picture might be built automatically.

> Steritherm: Presentation, in a specific picture, of everything that concerns the temperature in the holding tube, e.g., the pressure just before valve V78 and the product temperature, could be a relevant overview of an important event.

Process alarms

The alarms should be analyzed before being presented, i.e., irrelevant or misleading parts of the information should be discarded, and conclusions based on what is left should be presented as alarms of a higher level.

Alarm messages could be complemented with audible and visual signals. However, one has to be very careful not to overdo it. The messages should require an acknowledgement, after which only the text remains. The text should remain until the cause of the alarm is taken care of or the alarm has become irrelevant. Alarms could be presented in the following forms:

- Every picture in the control system has a specific box for alarm messages.
- In overview and focusing pictures the components relevant to the alarm should be marked.

- A special alarm list, where all alarms are shown, including activation time, value, limits, unit, time when acknowledged, priority, etc.

Tendency alarms should also be shown, e.g., as graphs.

> Steritherm: If valve V44 malfunctions, the temperature in the holding tube may increase and cause burn-on. Alarms should be activated for valve V44 and the temperature increase. However, the system should primarily inform the operator about the effect, i.e., about the burn-on.

Process warnings

Warnings differ from alarms as they are activated before something has gone wrong and it still is possible to avoid malfunctions. The activation of a warning is normally based on the value or the rate of change of process data. When one of these passes a preset limit, a warning is given. Warnings are to be handled as alarms, see above.

> Steritherm: A warning should be given when something is about to go wrong or might go wrong. For example when there is a very rapid increase in the burn-on or the temperature of the product in the holding cell is just above 137 C, and therefore easily could drop below this limit. Fast changes and values near alarm limits are normally the sources for warnings.

Quality monitoring

The system should provide a possibility to monitor the process with regard to the quality of the product. A list of important product parameters for each unit or every x minutes could be mapped to a list of important process parameters. When the product quality deteriorates the system could inspect the corresponding process parameters and in this way get the reason for the problem. Any deficiencies and the causes for these should be presented automatically to the operator as alarms or warnings.

> Steritherm: When the fouling of the product gets too high the system should inform the operator and if possible also present the cause, e.g., something is wrong with valve V44, causing an excess temperature in the holding tube.

Condition monitoring

Performance of process components should be monitored, in order to find signs of wear among the components. This is a complement to pre-scheduled maintenance.

> Steritherm: A situation when condition monitoring could be used is when irregularities are observed concerning the function of the regulator that controls V44. At this situation the system should propose that the regulator should be replaced.

2.2.2 Diagnosis

On-line diagnostics

A more or less automatic interpretation of an alarm or warning or a group of alarms and warnings, should be given. Communication with the operator should be a part of the analysis.

> Steritherm: For example, when the alarm "The temperature of the product in the holding tube is too low" is activated the system should ask something like "Check the pressure in the holding cell (pressure indicator, PI-33). Is it below 3.5 bar?" Depending on the answer the dialogue would continue until the system had pin-pointed the cause. The system would then present a remedy.

2.2.3 Control

Operation in normal state.

For every normal state of the process it should be possible to ask the system what the operator should do next, what should happen when this is done, and what should remain unchanged. There could be things that are not or should not be possible to change, depending on the phase of the process.

The answers could be in the form of text and pictures and it should be possible to have them presented in many different ways. For example, as a complete instruction manual, by referring to the actual (or a hypothetical) state of the process, or as answers to specific questions in natural language. The possibility to get all operational information about a special part of the process should also exist. It should also be possible to handle explanations to these answers and hypothetical discussions.

> Steritherm: It should be possible to ask the control system what actions are to be taken in connection with changes from one sequence to another. For example, what conditions should be fulfilled before changing from production to aseptic intermediate cleaning and how these can be achieved.

Operation in abnormal state

The control system should be able to start a dialogue by informing the operator that the process is in an abnormal state.

> Steritherm: An example of an abnormal state in the Steritherm process is when the temperature in the holding cell is under 137 degrees C during production. If that is the case the operator should be informed about what is happening and what he should do, e.g., "The temperature in the holding cell is decreasing. Do this and this".

2.2.4 Other functions

Flexible data interpretation, processing and presentation

The exchange of information between the operator and the control system should be as close to natural language as possible and try to emulate a conversation with an expert. The control system should be able to handle and analyze fuzzy or incomplete information. For example, handle approximate input from the operator or make heuristic analysis of graphs and diagrams.

> Steritherm: An example of flexible data interpretation could be if the operator states the temperature of the product in the holding cell as "just above 143 degrees C" or "something between 140 and 145 degrees C" and the system answers "That sounds rather high. You should check the sensor TT 42 once more and...".

Product information

Information about how a specific product influences the process, (e.g., with respect to choice of parameters) and vice versa, should be available.

Steritherm: A typical Steritherm product is milk.

Product : Milk.

Parameters:

TSL 44: 137 degrees C.

TSL 64: 85 degrees C.

Operation:

Time between start of production and aseptic intermediate cleaning: 240 minutes.

Time between first and second aseptic intermediate cleaning: 200 minutes.

Time between second aseptic intermediate cleaning and final cleaning: 160 minutes.

Duration of aseptic intermediate cleaning: 30 minutes.

Duration of cleaning: 70 minutes.

Duration of sterilization: 40 minutes.

Cleaning program: Amount of acid and lye (e.g., caustic soda) and in what order to use them.

Expected values:

TT 45: 137 degrees C.

TT 71: 20 degrees C.

TT 64: 85 degrees C.

Typical errors:

(Burn-on etc)

Knowledge browsing

It should be possible to access in an easy way more knowledge than is automatically presented. There should be explanations available for alarms, warnings, diagnosis, pieces of advice, trends, etc. It should also be possible to go deeper into the accumulated knowledge of the system and, e.g., inspect how the sequence logic of the system looks like or what the production constraints are.

> Steritherm: A question that the system should be able to answer is "Why does the system believe that the decrease in temperature in the holding cell depends on a heat water leakage?". Another example of knowledge about the Steritherm process that should be available is descriptions of the components, pumps, valves, tanks, etc.

Presentation of historical process data

In addition to implicit use of historical data, as in the case of tendency alarms, historical process data should be explicitly available, e.g., as text, graphs, diagrams, and mean values.

> Steritherm: Historical data that it should be possible to access is, e.g., the variation of the burn-on during the last hour or the logging of the state of TSL 44 and V71.

Prediction

Prediction should mainly be done automatically, as in the case of some alarms, warnings and pieces of advice. However, it should also be possible to ask more explicitly about the future, e.g., "What will the pressure in this pipe be twenty minutes from now?".

> Steritherm: The system should be able to answer questions like: "When will the burn-on reach 10%?", or "What will the temperature in the holding tube be an hour from now?"

Training

A simulator that reacts identically to the process should be available for training. It should be possible to start in the middle of a batch in difficult situations.

Comments and notes

It should be possible to make notes about faults, oddities, questions, etc., in an easy and convenient way.

2.3 MAINTENANCE PERSONNEL

Maintenance personnel are obviously very important in order to run a plant efficiently. The requirements from the maintenance personnel largely overlap the operators' requirements. However, there are several demands that are more closely connected to maintenance. In this section we only deal with requirements that not have been discussed previously. The main tasks for the maintenance personnel are:

- Off-line diagnosis
- Maintenance reconfiguration
- Preventive maintenance
- Spare part handling
- Maintenance simulation

As is the case for the operator functions, all these activities require a lot more knowledge about the process and the control system than is available in the conventional control systems of today.

2.3.1 Off-line diagnosis

Cause and remedy

When a fault occurs the system should present a cause and a remedy. There may be more than one cause and if so they should be ordered in accordance with their probability. The maintenance personnel should be able to make their own proposals, which the system then evaluates. This off-line trouble-shooting is more advanced and thorough than the on-line diagnosis that the operators will have access to.

> Steritherm: For example, when the temperature of the holding tube is too low the system should make a diagnosis and present a cause and a remedy for this, e.g., "Replace valve V 78 with a new one. This will increase the pressure, and the temperature, in the holding tube".

Overview of the process regarding faults

The location of the fault should be shown in three different ways (preferably as pictures):

- Geographical: Which component(s).
- Topological: Which component(s).
- Operational: Which state(s).

  Steritherm:

  - Geographical: Valve V78.
  - Topological: The holding tube.
  - Operational: Production, phase P/2.

2.3.2 Maintenance reconfiguration

It is often the case that it takes some time to replace or repair a faulty component. In the mean time, the production may continue using an alternative way of production, i.e., the process is reconfigured.

> Steritherm: For example, if one of two packing machines breaks down, a solution is to disconnect the faulty one and halve the production by changing some process parameters.

2.3.3 Preventive maintenance

Maintenance planning

The system should be able to make a maintenance plan that decides when different parts should be tested, replaced, or optimized with regard to cost and reliability.

> Steritherm: An example of a part of a maintenance plan for a Steritherm plant is "Test the pumps M2 and M3 after x hours in operation. Replace the following sensors TT 44, TT 71 and TT 64 after y hours in operation ...".

Preventive actions

Warnings about when to make tests, replace components, etc., according to the maintenance plan or based on new data, should be given.

> Steritherm: A warning that informs the maintenance personnel that it is time for intervention could look something like this: "The holding tube has been in operation for x hours and it is now time to inspect it".

2.3.4 Spare part handling

Component information

Information about a component, and its different parts, such as behavioural, topological, geographical, and functional information, specific for the component should be available.

> Steritherm: A component that the system should have information about is valve V78. The information that could be of interest is: "It is a pneumatic valve of type ARC-SMS-38-30-10; it consists of the following parts ... (perhaps a picture); pressure information ... (perhaps a graph); installation instructions, etc".

Spare part catalogue

There should be information about the spare parts available. Information about whether a component of a certain type could be replaced by one of an other type, and if this will have any negative effects on the performance, should be available.

> Steritherm: For example, the system should be able to give information on whether there are any Pt 100 temperature sensors in the store and, if not, is there any substitute, and what does one have to consider if one uses this instead.

Spare part planning

To optimize the stock of spare parts needed, an estimation should be done based on, e.g., the number of used parts in the plant, probability of a fault in a part, price if the part is not available, etc.

> Steritherm: The system should propose how many holding tubes would be needed in storage, based upon fault probability, price, etc.

2.3.5 Maintenance simulation

There should be an opportunity to simulate different maintenance actions.

> Steritherm: Something to test in the simulator could be what will happen if one waits another 100 hours before the rpm of pump M2 is corrected.

2.4 DESIGNERS

The design of a industrial plant is a complicated task. It involves several designers, e.g. process designers, mechanical designers, control system designers, etc. The design process includes the problem definition phase as well as the actual

design of the process and its subprocesses. In practice the problem definition is often done by a different organization compared to the rest of the design. For example, the problem definition is done by the customer or a consultant to the customer, while the design is done by a supply company or a group of supply companies. Although, the problem definition could be also be supported by knowledge-based tools, these are not included in our concept. However, we believe that it is important to have the possibility to interface the output from the problem definition phase (design parameters, etc.) in some way.

In common for all types of design is that the work, after the problem definition has been done, can be split up in the following phases:

- Structuring
- Detailed construction
- Verification
- Documentation

The phases above seem to be a reasonable way to structure the functional demands that designers will have on a future supervision and control system.

Redesign of a plant follows the same structure as above but a separate support function is probably needed to handle modifications in an existing database.

A special case is the situation during the start-up phase of a plant. This situation is a combination of verification of the original design and redesign. A special support function is probably needed that partly has the same functionality as during redesign. The start-up phase is normally not carried out by the designer, but by a special start-up crew or by production engineers. Demands from the start-up phase should, of course, influence the design functions.

2.4.1 Structuring

Structuring a plant is not a homogeneous task. It means different things for a process designer and a control system designer. However, all designers have to structure their design. A normal way to work is that the process designer lays out the overall structure of the plant. In the case of Steritherm, this design will be in terms of a preheater, heater, cooler, etc. This design structure is then used as an input for the other designers, such as mechanical designers.

The following function is needed by the designers during structuring:

Structuring support system

The structuring support system should help and guide the designers with knowledge about plant design, control system design, etc. An example of an input to this function could be that a dairy should produce 80,000 litres UHT milk/day. The designer should, with the help of the system be able to structure the plant and decide whether there should be one Steritherm with a capacity of 10,000 l/h or two Steritherms of 5,000 l/h each, etc.

The function should also help the designer with deciding the level of automation, whether it should be a distributed or centralized control system, etc.

2.4.2 Detailed construction

When the basic structure of the process has been decided, it is time for the designers to fill in this structure with design data. This does not necessarily have to be done by the same designer that did the structuring. In fact, it is more common that it is done by designers specialized in mechanics, control systems, etc. During this phase the designers need functions like:

Access to a design library

There should be a library of earlier design solutions, e.g., control algorithms, available in the system and the designer should have access to this knowledge. Help should be available to evaluate which solution fits best. In the case of a Steritherm the task could, for instance, be to design a pre-heater for a temperature of 75 C and with a flow of 10.000 l/h. The designer should then be able to find earlier solutions to this problem. If there are no such specific solutions, he should be able to pick a similar solution and modify it.

Automatic generation of design solutions from specifications

When no acceptable ready-made solution is available there should be a function that can generate a basic design from requirements in a specification.

Support for making original designs

If everything else fails, it should be possible to design certain parts of the plant from scratch. This support will probably be very simple and help the designer only with functions like calculation of capacity, etc.

Access to a knowledge pool

It is important for a designer to get design feedback. Therefore it is essential to have access to a knowledge pool with experience from earlier plants. This knowledge pool then follows the plant and is hopefully updated by start-up crews,

operators, process engineers, etc. In the case of Steritherm it could, as an example, be information about a certain component, e.g., a temperature switch, which frequently has caused problems.

2.4.3 Verification

When the design is partly ready it should be possible to verify it. An example of such a function is:

Simulation

This function must be very flexible and allow simulation of designs in different stages of completeness. From the first level of block drawing to the complete, detailed design.

2.4.4 Documentation

When the design is ready it contains of a lot of documentation. This documentation is used for the assembly of the plant. It should be possible to generate the documentation from the plant database. Examples of functions in this area are:

Documentation during construction

During the design the system should automatically transfer documentation from design libraries, etc. In the case of a new design, the system should encourage the designer to produce documentation. When putting in a new component in a Steritherm, the system should ask for complementary information such as spare part lists, maintenance instructions, etc.

Document generation

When the design is ready the system should support the user with all wanted documents. Note that it should also be possible to generate new types of documents that have not been foreseen earlier, e.g., a list of all components within a certain geographical area, etc.

2.4.5 Redesign

During redesign the designer needs access to the complete knowledge base. Functions that can guide and help the redesigner are necessary. It may obviously be a difficult task to try to alter a complex database. Some of the redesign tasks may take place during operation, e.g., on-line modifications of the sequence logic.

Support for design changes

This function is intended to give the redesigner help, e.g., point out consequences for supply functions when a process part is changed, etc.

In the case of Steritherm, the support could, e.g., be to calculate the consequences for the steam supply when the flow through the plant is boosted.

2.5 PRODUCTION ENGINEERS

Production engineers are, in our definition, the persons responsible for the production in the plant including what is produced at any moment in time, how it is produced, and when the next production change should take place. Many functional requirements of a production engineer, such as monitoring and alarm handling, are basically the same as for the operator. As these functions are treated in the description of the user requirements for the operator, this description will mainly concentrate on the requirements that are specific for the production engineers.

One way to structure the work of a production engineer is to talk about work with:

- Future production data
- Current production data
- Historical production data

All the production engineers work can be categorized into one of these three groups. In the work with current production data, the demands for functions are nearly identical to the operator's. In the work with future and historical production data there are some demands for functions that are unique for production engineers. As mentioned in the design section, it is possible that the production engineer should have access to some specific functions related to start-up procedures.

2.5.1 Work with future production data

When a production engineer is working with questions concerning the future it normally concerns planning or optimization. Examples of functions are:

Production planning

Production planning includes, planning of how much product of a certain quality should be produced, when it should be produced, etc. In an advanced system the result of the production planning will automatically be transferred as a production order to the operator. The production planning system must have dynamic

connections to the control system in order to make recalculations in case of the failure of some equipment.

Resource planning

The resource planning function should be possible to calculate the amount of raw material needed during a certain production period. The result of the resource planning can then be used as an input to the production planning.

Production optimization

The production optimization function makes possible to optimize the production capacity, minimize environmental disturbances, minimize the size of storage capacity, etc. It may involve, e.g., the choice between machines with different capacity with the aim to minimize the throughput time or the use of raw material. Possibilities to simulate different solutions should be included in the planning and optimization functions.

2.5.2 Work with current process data

Although most of the demands here are the same as for the operator, there are some functions specific for the production engineer, such as:

Goal fulfillment monitoring

The goal fulfillment monitoring function will help the production engineer to initiate control actions. The control system must have information about the goals of the process. In the case of a Steritherm, the goal could be to produce as much UHT-milk as possible without any concern for taste, production economy, etc. On the other hand the goal could also be to produce UHT-milk with the best possible taste.

Monitoring of the mass and energy balances

Balance monitoring is used for the same purpose as the previous function. Depending on the level of process knowledge this function could also be used by the operator.

Browser for the knowledge base

The browser should make it possible to navigate between different views of information in the database. It could be used, e.g., to find the functional description for a certain part of the process, find component descriptions, etc. This function is general and should be accessible by all the different user groups.

Product treatment optimization

The product treatment optimization function should make it possible to optimize the process parameters, such as flow temperature, pressure, etc., with the aim of getting the best possible quality of the product. This function should also include advice on normal parameter settings, etc.

Product knowledge

General information about the product should be available in the system. This information could be very valuable during development of new process steps or when the process should be tuned for new products.

Steritherm:

Product : Milk.

Product characteristics:

Colour: White to yellow

Density: 1.028 - 1.034

Freezing point: -0.54 - -0.59 C

pH: 6.6 - 6.7

Recipe knowledge

If the plant is running with different products, it is essential to be able to get information about the recipes. In the case of Steritherm, a plant could produce plain milk one day and chocolate milk the next day.

Process knowledge

As described earlier, there is a lot of knowledge about the plant in the control system. This knowledge should be accessible for the production engineer. Process knowledge includes theoretical information about the process. In the case of Steritherm, it should be possible to get some basic knowledge about how sterilization of milk is done, the function of the main process, and of the support systems such as steam, water, electricity, etc.

Start-up function

This function is used during start-up of the plant and it gives possibilities to note in the knowledge base when a process function is checked and verified. It should also be possible to include heuristic knowledge from the start up period in the knowledge base.

2.5.3 Work with historical process data

In this category, the main interest is to support analysis of the production history, in order to get information for redesign, etc. This could be done with a function like:

Data logging

A possibility to specify data to be logged. There should be possibilities to analyze the data in different ways, e.g., statistically, using graphical displays, etc. The input to start a log sequence could be a process event, manually controlled, or time controlled.

2.6 SUMMARY

Above, we have briefly tried to describe rather detailed user functions for some user groups. As we have seen, many of these functions are shared by several user categories. Below, we have tried to list what could be called main user functions. The table also shows which user categories are the main users.

The table indicates that many user functions are of common interest for several users. Maybe there is a slight difference in how the function is used, but the basic needs are the same.

2.6.1 Table of main user activities versus users

Users:	Operators	Maintenance personnel	Designers	Production engineers
Monitoring	X	X		X
Diagnostics	X	X		X
Control	X			X
User assistance	X			X
Interpretation		X		X
Planning	X	X		X
Simulation	X	X	X	X
Browsing	X	X	X	X
Support for structuring			X	
Design assistance			X	
Redesign support			X	
Documentation	X	X	X	X
Optimization	X	X		X
Logging	X		X	
Start-up support	X			
Plant knowledge	X	X	X	X

3

The KBCS Concept

3.1 INTRODUCTION

In the Feasibility Study (IT4, 1988), a KBCS concept was outlined. A hierarchical object model was presented in which the basic entities were physical components, e.g., pumps, valves, and tanks, and control functions, e.g., sensors, controllers, and PLC code. Intelligent tools should operate on this object model, as shown in Fig. 3.1. The tools should provide different users with different information about the process and perform the various functions of the system. This concept has been the base for further work which is presented in this chapter.

The chapter describes a concept for a KBCS based on a common knowledge base. The chapter includes a presentation of the goals that we aim at and how we think these goals could be reached, a short description of the knowledge base that will be the core of our system, an overview of the different kinds of tools that are used for creating and modifying the knowledge in the knowledge base and for generating executable code from this knowledge, a short description of the interface between the knowledge base and the tools, and finally some ideas about how different implementation configurations may look.

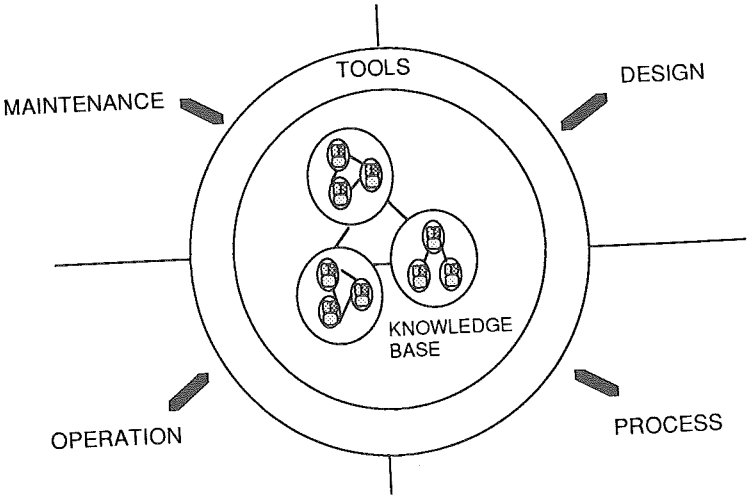


Figure 3.1 The basic concept.

3.2 OVERVIEW

A total concept for KBCSs is no minor technical issue. The concept has to include what hardware modules the system should consist of, the network architecture, communication protocols, software architecture, knowledge base language, user interfaces, implementation language, etc. As this is outside the scope of the project, we have concentrated on the software architecture, including the structure of the knowledge base, and the nature of the tools that operate upon the knowledge base.

Today, when knowledge-based systems are applied in the process industry, the pre-dominant solution is the "interfaced systems" architecture described in Chapter 1. In these systems, the programming techniques or, in expert system terminology, the knowledge representation formalisms in the add-on knowledge-based systems are different from those in the conventional control system. Typically, rules and objects are used in the KBS whereas equations and procedures dominate in the conventional control system. There are many reasons for this separation. Conventional control systems are primarily designed for the actual control of the process. This is normally implemented with boolean, sequential or combinatorial logic, and with procedures for, e.g., PID control. High computation speed and inexpensive hardware modules are important design criteria. On the other hand, knowledge-based techniques such as, e.g., rules are, typically, time-consuming and require more powerful hardware.

The interfaced solution also matches the way knowledge-based techniques are usually applied in the process industry. Operator support applications such as supervisory monitoring, diagnosis, and control are by far the dominating appli-

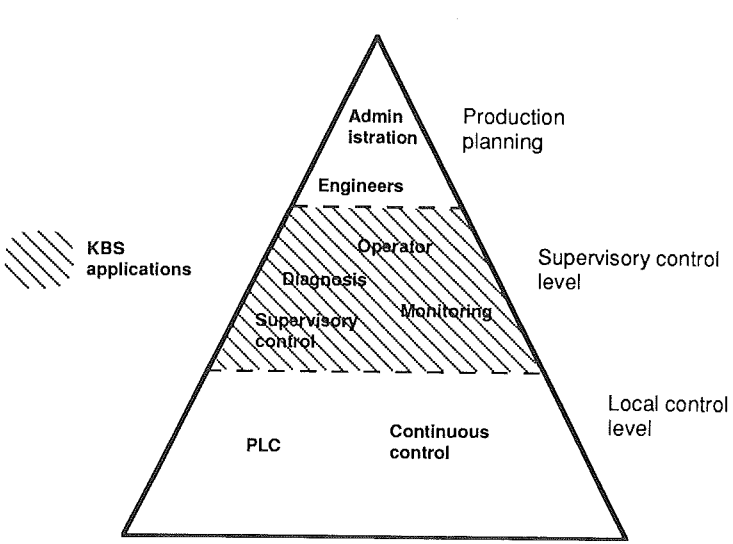


Figure 3.2 KBS applications today

cation types. The role of knowledge-based systems in the overall system today is indicated in Fig. 3.2.

This way of using knowledge-based systems will probably dominate also in the future. However, it should be possible to use use knowledge-based techniques also on the other levels. KBSs for planning, design, and administration support are becoming more and more common. In fuzzy and expert control, knowledge-based techniques are used also on the local control level. It is necessary that a KBCS can support knowledge-based applications on all levels in the system, as shown in Fig. 3.3.

In a KBCS, different representation formalisms such as equations, procedures, rules, etc., can be mixed together. The knowledge base of the KBCS includes, among other things, the executable units of the system, i.e, rules, equations, procedures, etc. The knowledge base has information about the distribution of the units, i.e., in which processing modules the units should execute. In the ultimate realization of the concept all types of units can be executed on all hardware modules in the system. This implies that also the processing modules on the local control level have the capacity to execute rules, with all the hardware requirements that follows. In a perhaps more realistic implementation, different hardware modules can execute different types of units. For example, the local control modules may only be able to execute the conventional units, i.e., equations and procedures, whereas the supervisory control modules have full processing capacity.

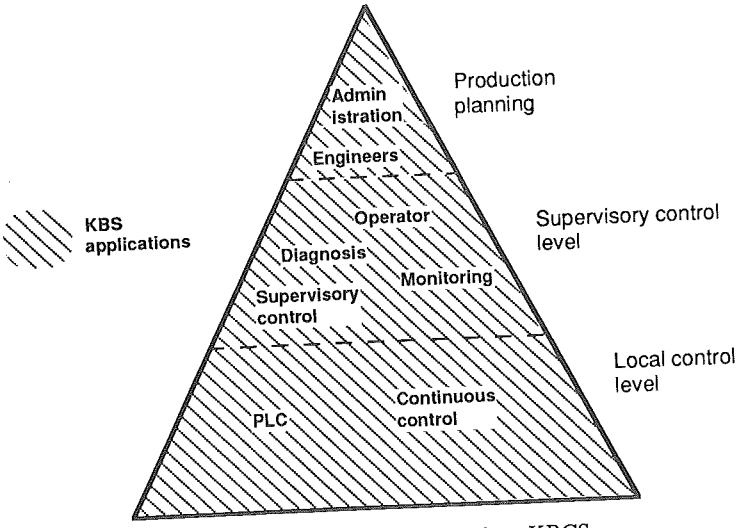


Figure 3.3 KBS applications in a KBCS

Compatibility

As described, the total concept more or less implies a closed system. In the same time it is extremely important that the KBCS is compatible, and can coexist, with other control systems. As previously described, the hardware modules of a KBCS include different processing modules that can execute all, or parts of, the executable units in the knowledge base. These processing modules constitute the internal hardware of the KBCS.

However, it must also be possible to use external hardware, e.g., processing modules from other suppliers, and which are not compatible with the internal hardware. In that case it must be possible to extract the parts that can be executed in the external hardware from the knowledge base and transform them into a format that the module understands. PLC code is one such example. It should be possible to convert the PLC representation in the knowledge base to a format that can be down-loaded to an external PLC system.

Another possibility is that some parts of the control system in a plant are implemented with conventional systems and some parts with a KBCS. In that case the KBCS must be able to communicate with the other systems. In the knowledge base the structure of the entire control system is represented, with the non-KBCS parts indicated as modules whose internal structure the KBCS cannot access.

3.2.1 Concept goals

The goals of the concept of a KBCS primarily are:

- One uniform knowledge base for all knowledge.

- The same knowledge may not be described more than once in the knowledge base.
- The knowledge base should have a uniform interface.
- The KBCS should be as modular as possible.
- The same knowledge should be available to many users.
- It should be possible to distribute knowledge to other processors in order to achieve necessary run-time performance.
- On-line modifications should be possible.

Our proposal of a concept which tries to attain these goals is based on a KBCS, which consists of one knowledge base and a set of tools. The tools operate on the knowledge in the knowledge base, as shown Fig. 3.4. The knowledge consists of the process knowledge needed in the KBCS and the tools are used for creating and modifying it, generating executable code from it, and distributing it to the processing module where it should execute. The management system of the knowledge base should, with the help of the tools, keep the knowledge updated and manage redundant information.

3.2.2 The concept

Knowledge about the process and how to control it, i.e., the knowledge needed by the KBCS, is located in the common knowledge base. This knowledge base contains a variety of different knowledge types as indicated in Fig. 3.5.

The common knowledge base is divided into the main knowledge base and local databases. The main knowledge base is a uniform, but possibly distributed, knowledge base for different kinds of knowledge needed in the control system. It contains all of the knowledge types defined in Fig. 3.5 , except for the dynamic process data. The main knowledge base should be accessed in a uniform way by all the tools. The structure and contents of the main knowledge base are presented in Chapter 5.

It is not a practical or possible solution to use only the main knowledge base for all information during run-time. Dynamic process data should reside in local databases with links and references from the main knowledge base. This is to avoid an overwhelming amount of communication and transferring of data between the main knowledge base and the distributed control programs. However, it is the main knowledge base that should be responsible for delegating the handling of process data to the local databases and for management of the links between the local databases and the main knowledge base. The local databases are primarily intended for handling data. However, more complex functionalities

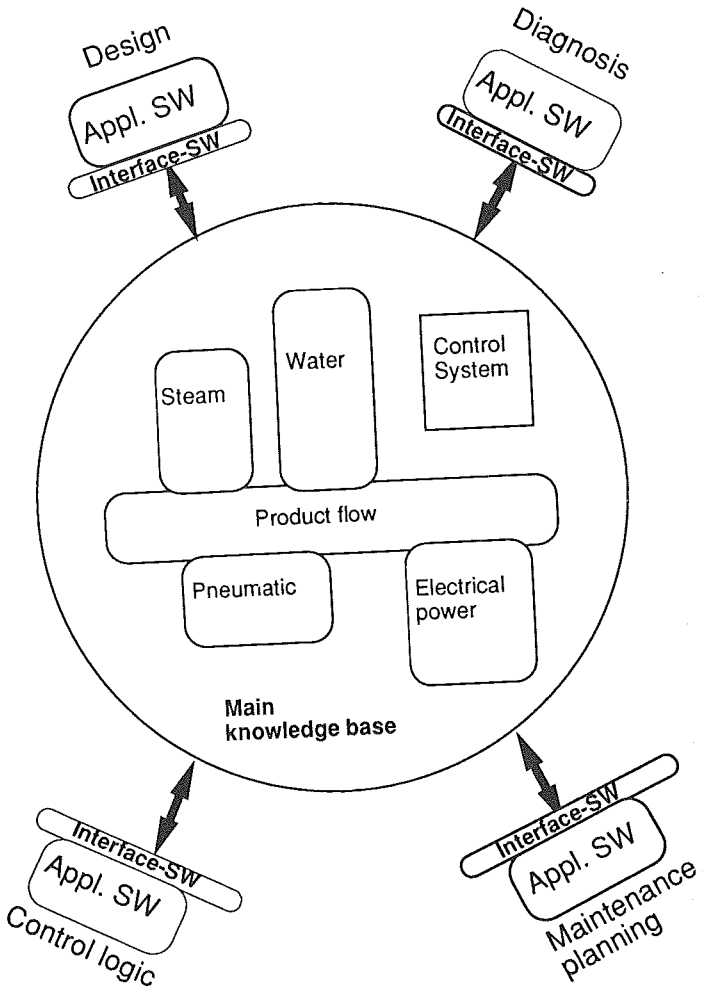


Figure 3.4 The concept of a KBCS

could be desirable. In that case it would be more relevant to talk about local knowledge bases.

The tools can be divided into two kinds: design tools and realization tools. The design tools are used for construction of the main knowledge base, modification of its knowledge, and for insertion of new knowledge. These tools are of different complexity. Some are for rather primitive operations like the insertion of a text or an object, while other are more advanced and used for more complex knowledge that is intended for specific use. An example of the latter is a tool that assists the designer in defining causal models that are to be used for model based diagnosis.

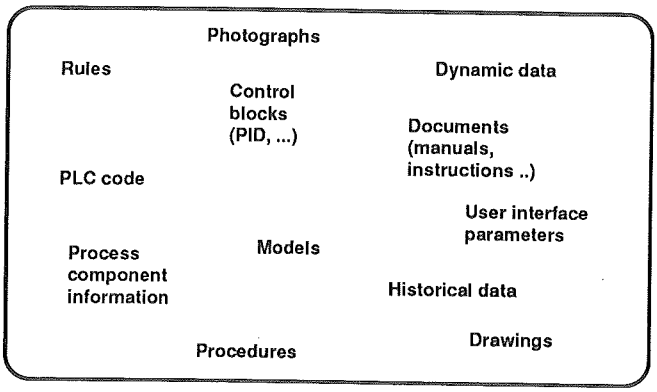


Figure 3.5 Knowledge base content

Realization tools realize the different functions of the control system. They extract relevant knowledge from the main knowledge base, put it together into software modules, and adapt these to the proper operational environment. In other words, the realization tools generate an executable control program from the knowledge in the main knowledge base. If necessary, the realization tools distribute the software modules and set up links for communication with the main knowledge base. If allowed, it is possible to modify the main knowledge base via these links. The realization tools could delegate the responsibility for dynamic process data to local databases and set up links between the main knowledge base and the local databases. An example of a realization tool could be a tool that extracts, converts, and distributes knowledge concerning on-line diagnosis.

Although we talk about one main knowledge base, it does not have to be physically implemented in one place. This means, of course, that neither does the total KBCS, which includes the main knowledge base, have to be implemented in one place. On the contrary, the need to distribute the computational burden to many processors is even higher in KBCSs than in conventional control systems. A distributed KBCS configuration is shown in Fig. 3.6. The fundamental idea, however, is that the main knowledge base, from a functional point of view, should look and function like one centralized entity with a uniform interface towards its surroundings, i.e., towards the different tools, the control program modules, and the local databases.

3.2.3 Fulfilment of the concept goals

The concept of a main knowledge base for all knowledge satisfies the goal that there should be one uniform knowledge base for all knowledge. It also supports the goal that the same knowledge should be available to many users. However, some knowledge has to reside in local databases. Due to the links between the local databases and the main knowledge base, the knowledge in these

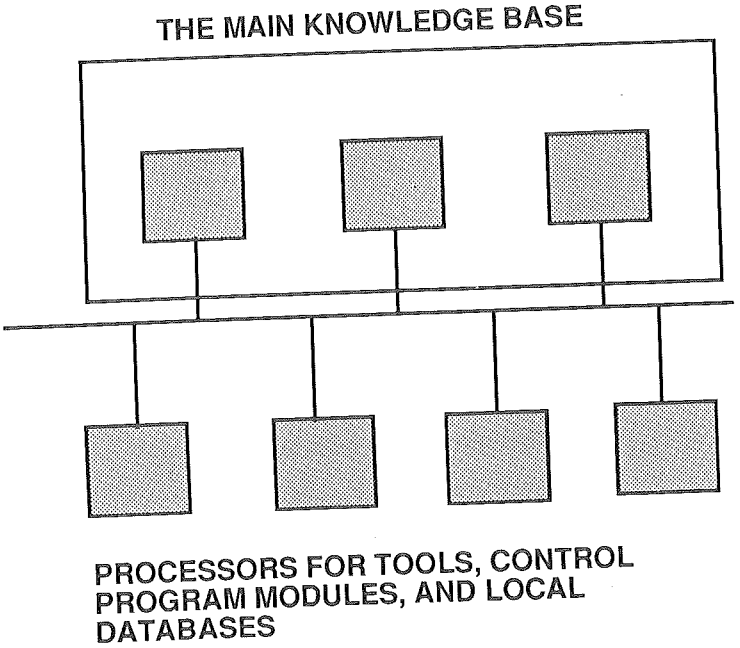


Figure 3.6 A possible KBCS configuration.

databases is always available. Thus, the main knowledge base together with the local databases constitute one common knowledge base for all knowledge in the KBCS.

How the main knowledge base complies with the desire to avoid redundant information is described in Chapter 5. However, redundant information has to be allowed in the KBCS. The reason for this is the need for local databases, the contents of which will sometimes have to include redundant information. However, this should be taken care of by an advanced management system that manages redundant information and handles modifications of, and accesses to, this information.

The KBCS should be as flexible with regard to changes, e.g., addition of new functionalities and tools, changed configurations. This is supported by partitioning the KBCS into a main knowledge base, local data bases, and tools. The strict structure of the main knowledge base and the modularity of the tools are further support for high flexibility. This is described in Chapter 5 and Section 3.4.3 respectively.

Different users need different interfaces to the KBCS. However, all users should also have the possibility to inspect the contents of the main knowledge base in an uniform way. This is solved by including a general browser function, which is presented in Section 3.5.4.

To attain distribution in order to achieve necessary run-time performance dif-

ferent measures are available. First, realization tools enable distribution of executable code to different processors. Second, local databases are used for distribution of dynamic process data. Third, it is possible to distribute the main knowledge base itself to several processors.

On-line modification is a functionality that is critical for control systems in many process industries. It has to be included in any control system that is intended for a wide application area. This is also true for a KBCS.

3.3 KNOWLEDGE IN THE KBCS

A short description of different kinds of knowledge needed in the control system will be given in this section. Where the different kinds of knowledge could reside and how to handle distributed knowledge will also be discussed.

3.3.1 Different kinds of knowledge

The knowledge stored in the common knowledge base includes not only what is usually termed knowledge in the expert system sense, but also the control logic, dynamic process data, different types of documents, drawings and schematics, etc. An overview of different types of knowledge representation formalism that can be part of the knowledge base is given in Chapter 4.

3.3.2 The main knowledge base

The main knowledge base is the kernel of the common knowledge base and the KBCS. It supports the idea that all knowledge should be available for all users of the control system. The main knowledge base is one uniform knowledge base for the different kinds of knowledge needed in the control system. All the tools operate upon the main knowledge base. The design tools handle the modifications of knowledge and the realization tools obtain and convert knowledge into executable code. The internal structure of the main knowledge base will be discussed in detail in Chapter 5.

3.3.3 The local databases

As mentioned above, dynamic process data has to reside in local databases, e.g., one local database for every control program module that receives time-critical or storage-critical process data. The main knowledge base has references to each of these local databases and information about how to handle their data transfers and when to update knowledge, if necessary. For example, a process parameter, that is needed for control purposes in a local program module and for simulation purposes in the main knowledge base, is measured every other second and stored

in a local data base. With a 20 seconds interval a mean value is transferred to the main knowledge base to be used in the simulation.

It is also quite possible that a module needs process data from the local database of another processing module. It would be very circumstantial and inefficient to have to access this data via the main knowledge base. Besides, in order to provide a reliable system, the processing modules must be able to, at least temporarily, function without access to the main knowledge-base. This implies that the process data needed for each module must be directly accessible instead of being accessed through the main knowledge base. What this could look like is indicated in Fig. 3.7.

The management of the knowledge distribution could be implemented by having a cross reference table in the main knowledge base. When a new software module is generated, the reference or address to its local database is inserted into this table together with information about what is to be stored there. When a module needs knowledge that resides in local knowledge base of another processing module, the realization tool extracts the reference to the local database that is needed. It then enters into the table that this local database is used by the module. The latter action is carried out in order to make it possible to inform the module whether the local database of another module is deleted, blocked, or moved. When the module executes and has to access process data that reside in another local database it can be done directly. The cross reference table and the administration of it is managed by the knowledge base management system.

Databases for dynamic process data are already a part of modern control systems. These databases, with some extensions, are probably suitable as local databases in our concept.

3.4 TOOLS

A tool can be many different things, e.g., something that is used to design a functionality of a system or something that realizes a functionality. Examples of this are rule editors and a simulation tools respectively. The tools are the means by which the operations on the common knowledge base are executed, as is shown in Fig. 3.8.

3.4.1 Design tools

The design tools are used to modify knowledge and insert new knowledge. These tools should be available for the designers only. There are many different kinds of designers, e.g., process designers, mechanical designers, and control system designers. The design tools have to offer the different designers different capabilities, as shown in Fig. 3.9.

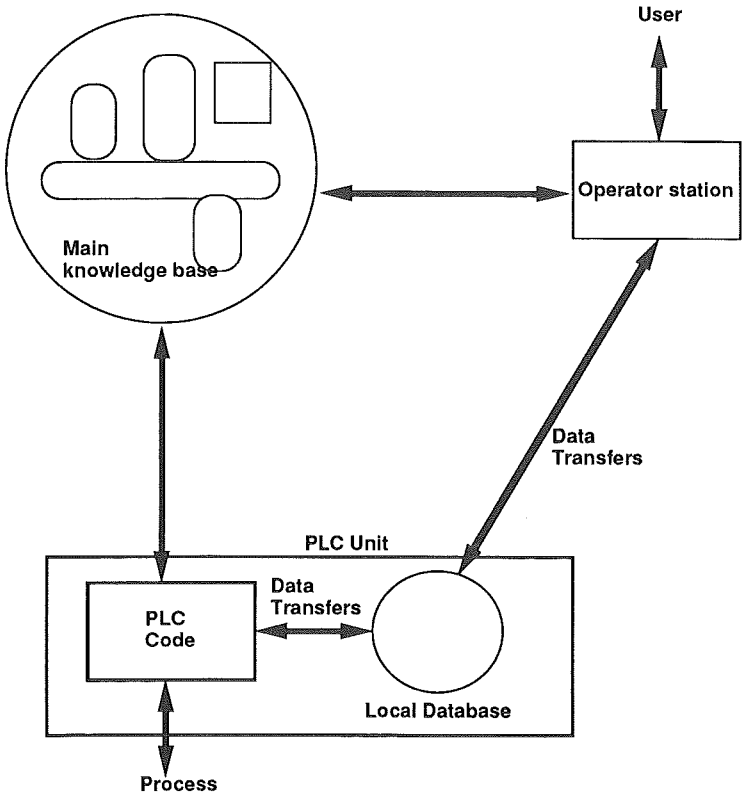


Figure 3.7 Local databases and their connections to the main knowledge base.

If the knowledge is to be used outside the main knowledge base, for example, if it is to execute in a local processor, the designers have to tell the system where the knowledge should be used.

Access of knowledge

In theory, the design tools have full access to any knowledge and full privileges to modify, insert, and delete. However, it should be possible to protect some or all knowledge from some or all designers or designer categories. The knowledge base management system and the structure of the main knowledge base should support the avoidance of redundant information. The different design tools could also support this by checking the work done by the designers. The designers should, with the help of design tools, be able to decide which parts of the main knowledge base that the other users should be able to modify.

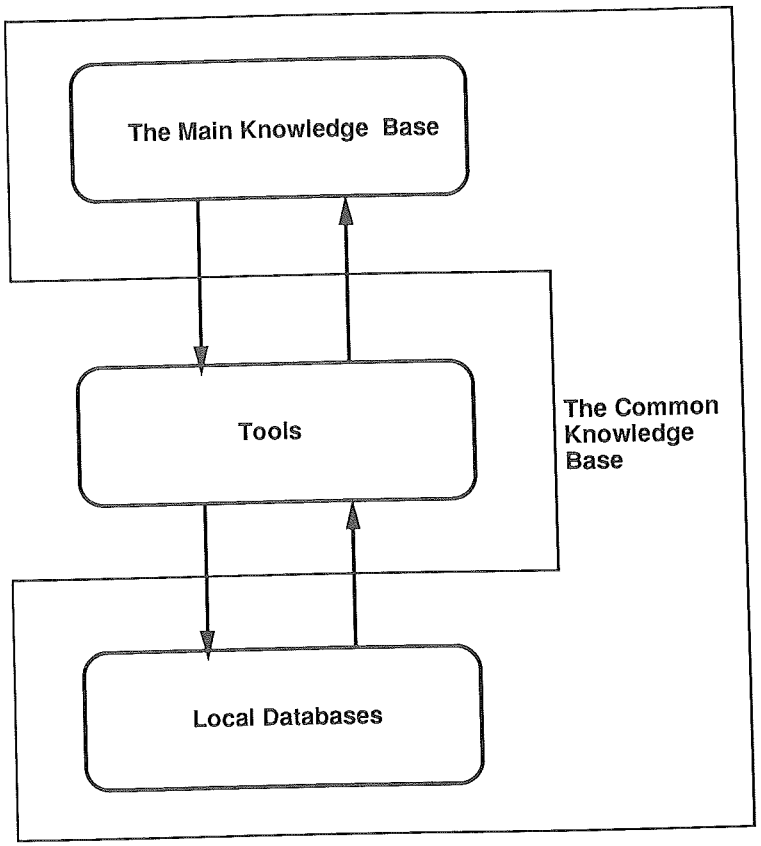


Figure 3.8 Tools and the common knowledge base.

Design tools for different kinds of knowledge

There are different kinds of design tools for different kinds of knowledge, or different knowledge representations. E.g., text editors for text, rule editors for rules, and object editors for objects. Other kinds of knowledge and knowledge representations that must be handled are for example logic, control strategies, causal models, information about process components, control algorithms, physical equations, and photographs.

Design tools can have different degrees of knowledge about their field of application. For example, a design tool for text editing does not need to know anything about the content of the text, only about format, spelling, etc. A design tool for editing rules and constraints that are to be used for production planning needs to have knowledge about rule and constraint syntax, but should also have knowledge about the process, the products, and planning. For example, a design tool should be able to automatically extract the basic constraints that it can conclude

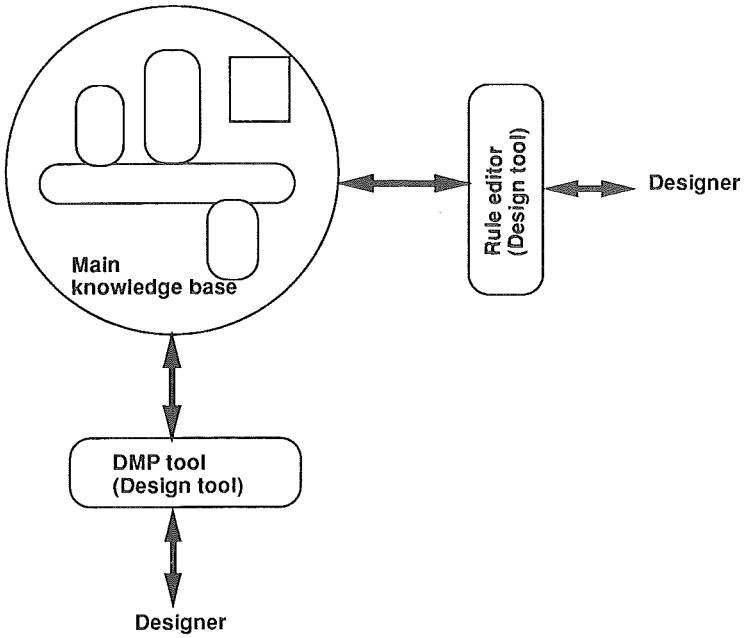


Figure 3.9 Design tools.

from the process knowledge in the main knowledge base, e.g., that the desired capacity of a process prohibits the use of a certain pump in a certain place.

The knowledge that the design tools insert into the main knowledge base is sometimes used for many different applications, e.g., quantitative equations describing the process can be used both for diagnosis, e.g., according to the Diagnostic Model Processor method described in Section 4.5.1, and in a simulation model of the process. In other cases, the knowledge is specific for one kind of application, e.g., descriptions of relations between symptoms and causes would probably have to be specific for the diagnosis method for which they were intended.

Interface to the designers

A basic and uniform interface towards the designers should be a part of the main knowledge base. However, it would be desirable to customize this interface for different designer categories and different development environments. To do this, the design tools have to adapt the knowledge they use so that the tools can handle it and present it in the way the designers prefer. To make these conversions, the design tools should use their internal capabilities as well as knowledge from the main knowledge base. Knowledge that may be needed from the main knowledge base is information about what kind of development environment the designer is working in, what privileges this designer has, an icon table that is to be used when making graphical presentations to a certain category of designers, e.g.,

mechanical designers, etc.

3.4.2 Realization tools

The realization tools are used for supplying the control system with the control program, i.e., the executable code that does the actual control. This program is realized by extracting knowledge from the main knowledge base and converting it so that it can execute in the appropriate processing module.

Generation of executable code

To be able to realize the functions of the control system, the realization tools extract knowledge from the main knowledge base. This could consist of e.g., logic, control procedures, diagnosis rules, fuzzy control rules, planning constraints, and simulation equations.

The knowledge is converted into executable code that, in some cases, will be downloaded to an operator station or a local control unit for efficient execution, while in other cases the environment in which the realization tool is located is sufficient. It is possible that a software module has to be executed together with a realization tool. For example, a numerical equation solver, that is needed for a simulator, could be a part of a realization tool. Examples of data conversions required are graphical data formats for the operator stations and program formats for the control processors. The steps are shown in Fig. 3.10.

The knowledge that the realization tools have to access for this conversion is of two kinds. First, the realization tools need information about the configuration of the control system, i.e., the environment of the processing module(s) in which a specific unit is to be executed. In this information, demands on formats are included. For example, logic that is to run on a certain control processing module has to be converted into the internal code format of this module.

Secondly, the realization tools need knowledge that enables them to do the conversion. One example of this is a tool that should adapt a product planning system to be run on a station for production engineers. Among other things, the tool needs to know what symbols and colours are to be used for presentation in this environment and for the concerned user category. Another example is when the graphical presentation for a maintenance planning expert system for the electrical subsystem of the process is to be generated by a realization tool. One thing the tool could be in need of are the symbols for electrical components according to, e.g., an American standard.

The specific knowledge needed for a particular unit, and where the unit is to be executed, is inserted into the main knowledge base by the designers, using design tools. This means that the generation and distribution of executable code could be done automatically by a realization tool, with no intervention by any personnel.

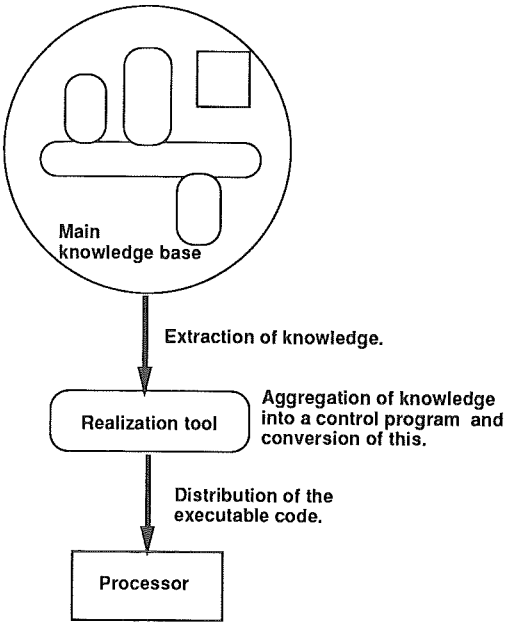


Figure 3.10 Generation of executable code

Integration of the control program

The executable units are connected to the main knowledge base via links that the realization tools automatically set up. In this way, the main knowledge base will be available on-line, as shown in Fig. 3.11.

It will be possible to inform the distributed units when their domain knowledge changes or becomes obsolete, as well as bring new knowledge, e.g., a manually adjusted parameter value, back to the main knowledge base. It should also be possible to update the distributed knowledge, e.g., when a redesign is made. Links to other parts of the system, e.g., to different local databases, are also automatically set up during the generation of the units. Information about these links are kept in the main knowledge base, and when changes occur, e.g., a local database is moved to a new location, the units concerned are informed by the main knowledge base.

To show the different kinds of realization tools that could exist, an example will be given. For generating a simulator, for example for training operators on critical but unusual situations, a realization tool extracts a quantitative process model, rules that are constructed especially for this simulator, general knowledge about the process needed for presenting pictures, etc. The extracted knowledge is brought together into a training simulator for operators. If this simulator is to be executed on the same processor as the tool it has to be adapted to this

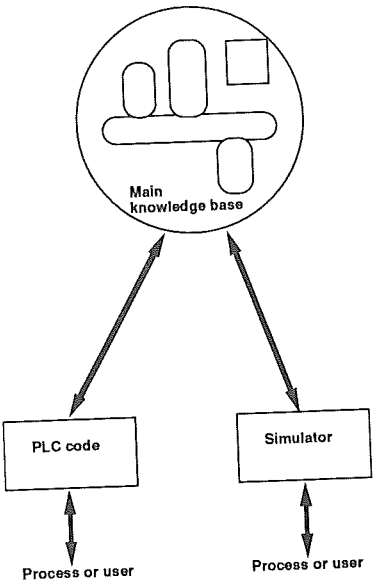


Figure 3.11 Links to the main knowledge base

environment. However, if the simulator is to be executed separately from the tool, it may have to be adapted to another environment. The knowledge about the environment, i.e., which format the code should be converted to, which graphical data format should be used, etc., is found in the main knowledge base. If the simulator is to be executed in another location it is then distributed. Finally, links between the main knowledge base and the simulator are established.

General or individual realization tools

There could be different realization tools for different application areas, e.g., tools for diagnosis, control, and simulation, respectively. However, it is possible that the realization tools should be classified according to the kind of conversions that they can handle and not according to what kind of control system functionalities that they are used for. A realization tool could be general when it comes to the kinds of knowledge it can handle but specialized on what kinds of conversions it can perform. I.e., a tool would only be able to convert knowledge to executable code for one or a few different operational environments, e.g., an operator station or a certain kind of processing module. What kinds of conversions the tool can perform depends on what is included in its toolkits.

3.4.3 The internal structure of the tools

A tool could be divided into several parts. One part is software unique to the tool, and other parts are toolkits and libraries that are shared by several tools.

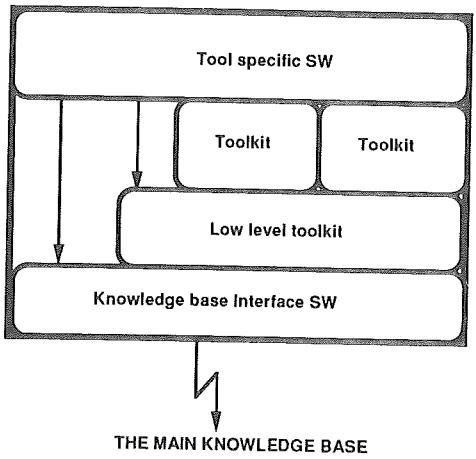


Figure 3.12 Internal tool structure.

The tools consist of a general interface to the main knowledge base, i.e., an interface towards the knowledge base management systems. On top of this is a general toolkit that is included in all, or most, tools. This toolkit is used for supporting capabilities that are general for the system, e.g., support to give all process knowledge a certain structure. Finally, unique software, which includes the tool specific knowledge, is added to give each tool a specific functionality. The interfaces to the users of the tools could either be included in the toolkit or in the unique software, i.e., if individual interfaces for some tools are preferred. This structure is shown in Fig. 3.12.

The important thing is a uniform interface to the main knowledge base. With this concept we pave the way for an open system that is independent of the internal implementation of the different tools and make it possible to easily add new capabilities to the system by adding new tools or modifying old ones.

Examples of the internal structure of different tools

As mentioned earlier, process knowledge should be located in the main knowledge base. However, the design tools definitely need to have information about how the knowledge in the main knowledge base should be handled. E.g., how rules ought to be structured when used for a particular purpose or what kind of planning strategy is to be preferred under certain circumstances. The realization tools might need to have some knowledge about the different implementation methods to enable them to collect the right knowledge and put it together into effective units.

To give a feeling of what the internal structure of the tools could look like, three somewhat simplified examples are described here:

- A design tool that supports the Diagnostic Model Processor method of Section 4.5.1 could consist of the following software: general software for interfacing with the main knowledge base, a general rule editor, tool specific software for generating model equations and dependencies, etc.

- A realization tool for adapting knowledge to a certain processing module should have general software for interfacing with the main knowledge base, general software that supports knowledge extraction, a translator for conversion into the internal code format of the processing module, and general software for establishing and maintaining links between the main knowledge base and the control program unit.

- The last example is a more advanced realization tool that would be able to generate and execute simulators. The tool could consist of general software for interfacing with the main knowledge base, general software that supports knowledge extraction, a compiler that converts the knowledge into executable code, and software for executing the simulator, e.g., a numerical equation solver.

3.5 KNOWLEDGE BASE MANAGEMENT

3.5.1 Interface

It may well be that knowledge used in one control program module is gathered from different parts of the main knowledge base with several different representation forms. However, the main knowledge base will have one common uniform interface that can handle all the different representation forms. The tools, the use of which will be the only way to interact with the main knowledge base, would not have to concern themselves with how the knowledge is represented in the main knowledge base. Neither do the tools have to be aware of how the conversion to and from the internal knowledge format of the main knowledge base is done.

3.5.2 Redundant information

Redundant information should be avoided in the main knowledge base and the system should support this. This is necessary as many different tools, control program modules, and users use the same knowledge in many different places, and for many different purposes, and often at the same time. This entails that one and the same piece of knowledge theoretically could be used and modified in many places at the same time. A critical case when it comes to concurrent handling of knowledge is on-line modifications of the control program, a thing that has to be possible in a future control system.

Redundant, duplicated knowledge may occur for two reasons. First, knowledge that is duplicated because the system requires it. Second, knowledge that is duplicated because the users accidentally or through laziness input the same knowledge in more than one place. The basic idea is that each piece of knowledge should only exist in one place. The idea is supported by the fact that knowledge as far as possible is stored in one place, the main knowledge base. However, in a few cases, the system makes it impossible to avoid redundancy, e.g., in the local databases. The second reason for redundancy is avoided by providing the users support with unique naming of objects, having a hierarchical structure of the main knowledge base, and including a powerful browser.

3.5.3 The knowledge base management system

To complete the concept, the main knowledge base should be equipped with a uniform management system, a knowledge base management system, as shown in Fig. 3.13. The management system should compile or translate the knowledge into the internal representation format of the main knowledge base, possibly performing syntax checking, and checking the knowledge to avoid redundant and contradictory information. The knowledge should then be available for the different tools via this management system.

Where the line should be drawn, i.e., where the different functions of the knowledge base management system should be placed, is not a question with a simple answer. To decrease the processing load on the processor on which the main knowledge base is implemented, as much as possible of the functionality of the knowledge base management system should reside in the different tools. However, to decrease the amount of communication between the tools and the main knowledge base, as much as possible of the functionality should be placed in a central position, i.e., tied to the main knowledge base.

3.5.4 Browser

The possibility to browse through the main knowledge base should be available to all user categories. In this way, the users should be able to inspect all knowledge in the main knowledge base. All users should have the knowledge presented in the same way and access to the same browsing functionalities. Some examples of browsing functionalities are graphical presentations of the different hierarchical structures of the knowledge, to be able to get all references to a chosen object, and to have all knowledge used in a particular control program module presented. The browser could be implemented as a part of the knowledge base management system, as it offers a function that should be available for all users, and it could be desirable to access it from many or all design and realization tools, or as a separate tool.

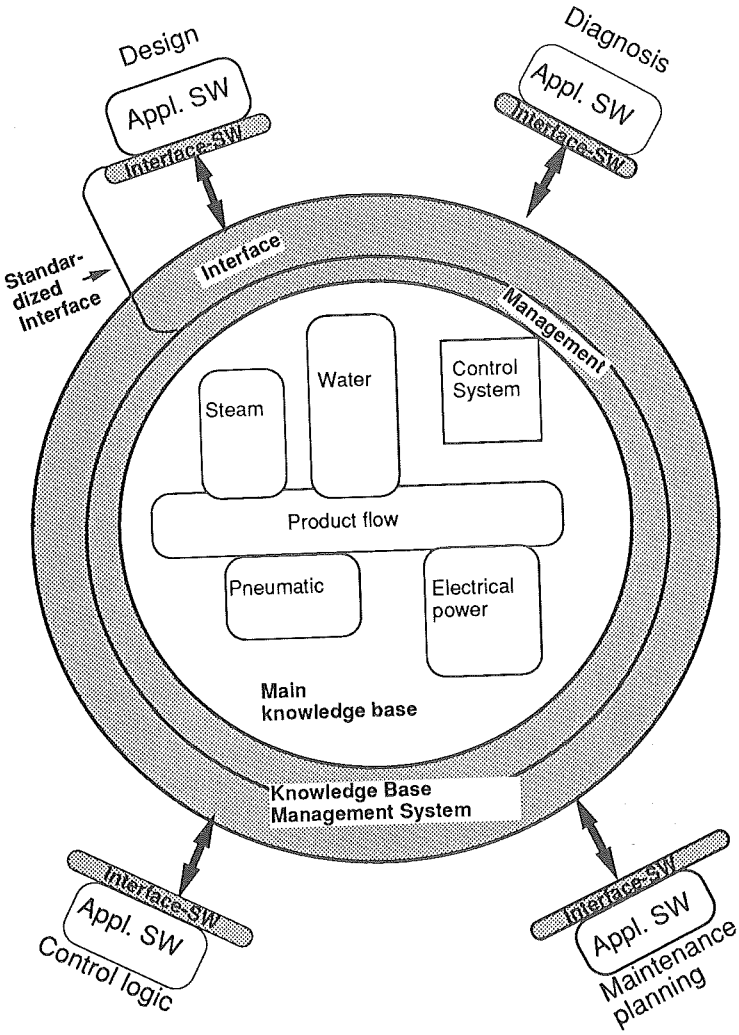


Figure 3.13 The knowledge base management system.

3.6 THE KNOWLEDGE BASE LANGUAGE

The contents of the main knowledge base is described in terms of a language. The language components, i.e., objects, procedures, rules, equations, etc., are described in Chapter 5. The knowledge base language has three representations: an internal representation, an external representation, and a graphical representation. The internal representation consists of the low-level data structures that implement the language, i.e., pointers, references, etc. The external representation consists of a textual, ASCII representation of the language. In this

representation the language can be manipulated with ordinary text editors. Finally, the graphical representation is what is seen when the knowledge base is viewed through the knowledge base browser.

The external representation makes it possible to transfer or copy knowledge from one main knowledge base to another. When the knowledge is collected from the main knowledge base of a KBCS it is translated from the internal representation to the external representation. After some minor conversions, that may be necessary, it should be possible to compile and load this into the main knowledge base of another KBCS. The translation from internal to external representation and vice versa should be done by a translator that could either be a part of the knowledge base management system or a separate tool.

3.7 POSSIBLE CONFIGURATIONS

We have presented a concept for an integrated knowledge-based real-time control system, and how it could be structured. However, we have not mentioned how it should be implemented or in which environment. Implementation details for such a system is too large a subject to discuss in this report. But some possible configurations for the system are discussed below.

We have been talking about an integrated control system and a uniform common knowledge base. However, this is only from a functional point of view. When it comes to implementation and the environment in which the system is to be executed in, the actual configuration is determined by many different parameters. Some of these are: complexity of the control system; the amount of knowledge that is to be placed in the main knowledge base; the kind of process that the system is to be used on and its physical structure; and the hardware and software that are to be used for the control of the process. In other words, the configuration of the system should be independent of the concept of the control system. The KBCS, and even the main knowledge base, could be distributed to many processors or executed on one single processor.

A likely configuration for a small or medium system would be one processor for the main knowledge base, and a handful of processors, in the form of engineer and operator stations, for different tools. The placing of the different tools depends on the environment they have to run in and the different users' need for access to them. Some parts of the control program are to be run on the same operator stations as the tools, while others are to be run on local control units. The hardware and software needed for the different control programs to execute, should be available in the respective nodes. All these processors, engineer stations, operational stations as well as local control units, should be able to communicate directly or indirectly with each other. The reason for this is, of course, the necessity to keep the system integrated and its knowledge available and updated.

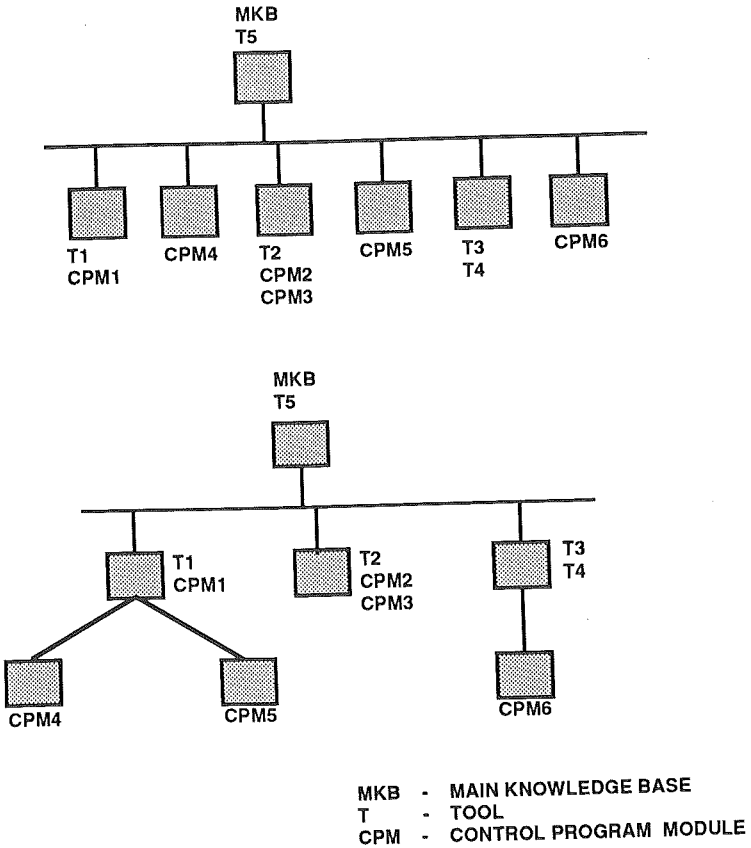


Figure 3.14 Examples of simple configurations.

E.g., a control unit would need to be able to send a new parameter value back to the main knowledge base if the parameter would influence the rest of the system. The most elementary configuration would be to have all the different processors connected to a bus. Another solution is to connect the processors of the main knowledge base and the engineer stations to the bus, and then connect each less sophisticated processor, i.e., those in which only control program modules reside, to the processors on which the relevant realization tools are. These two possible configurations are shown in Fig. 3.14.

For a major KBCS it could be necessary to increase the numbers of processors. An example is shown in Fig. 3.15. The reason for this increase is the need for more numerous and more capacity demanding tools, more numerous and more advanced control program modules, and a larger, and thereby also a more complex, main knowledge base. The major difference, regarding configuration, between a minor and a major KBCS is that the latter will probably have a

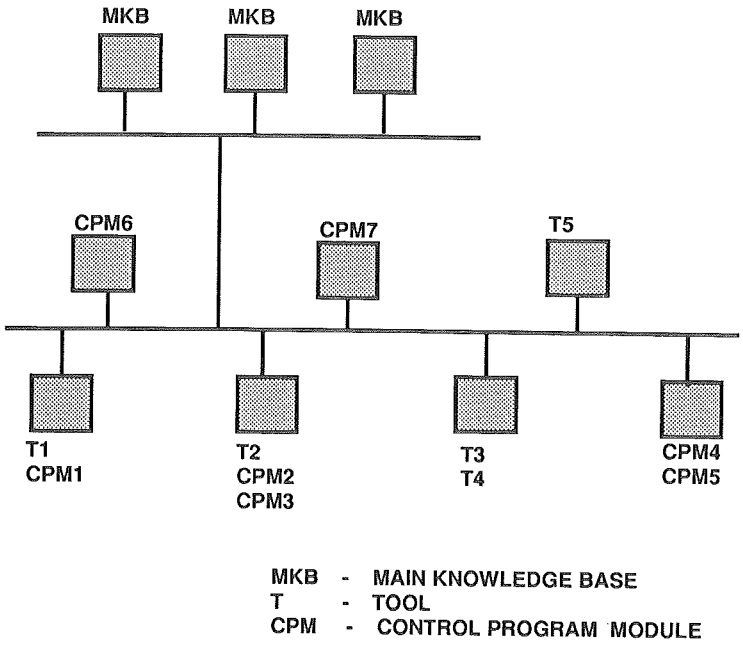


Figure 3.15 An example of a more advanced configuration.

physically distributed main knowledge base. However, the functionality of the main knowledge base is the same and from the outside it looks and acts like one single knowledge base. This will be possible due to the knowledge base management system, which will supply the main knowledge base with a uniform interface.

3.8 REAL-TIME ASPECTS

As the control system is meant for process control, the real time aspects of the system are very important. The possibility to distribute the system more or less extensively is a necessary support for real-time performance. Another support is given by the real-time constructs in the knowledge base language. These are discussed in section 5.4.

The tools do not have to reside on the same machine as the main knowledge base, but can run on different computers and in this way increase the performance of both the particular tools as well as the total control system.

The excutable knowledge base units can be distributed to many processing modules. The units do not have to be executed on the same processors, or in the same kind of operating environment, as the tools that were used to generate them.

There are two types of distribution to improve performance: partitioning of the

main knowledge base and distribution of knowledge (typically real-time data) from the main knowledge base to local databases. By partioning the main knowledge base and placing it on different processors, the computational burden is decreased. The partitioning of the main knowledge base can be governed either by geographical and logical constraints, i.e., the partitions mirror the geographical and logical structure of the process, or by the type of knowledge. In the latter case, several techniques such as object-oriented databases, multimedia databases, and relational databases are used to implement different parts of the main knowledge base.

3.9 THE KBCS AND ITS USERS

As stated in Chapter 1, an important goal is to develop a control system that fulfils the requirements of the different user categories. In Chapter 2 we presented our ideas about future user requirements. The concept that has been presented in this chapter aims at a flexible, powerful, and user friendly control system.

Some fundamental ideas when it comes to user friendliness that this concept supports are:

- It should be easy to add new capabilities to the control system.
- All knowledge should be available for all users.
- The users should get powerful support when it comes to advice and explanations.

As an example, consider a situation where a fault has occurred in the Steritherm process, e.g., an early and rapid burn-on. An alarm analysis is performed and the operator is informed that burn-on is occurring. This could be done by an audible alarm, an indication on the screen marking heat exchanger I and finally a text string that pops up on the screen. The text informs the operator that he should stop the production and start intermediate cleaning. The operator acknowledges the alarm, stops the production and starts intermediate cleaning. Next a diagnosis system starts to run. After a short time, which could include some questions to the operator, the cause is found and presented. Valve V44 has stuck open. A remedy is proposed: clean the valve and get it replaced at next plant stop.

The operator could now ask how the malfunction of valve V44 could cause rapid burn-on. This should be explained by the built-in explainer of the diagnosis system. (Valve V44 controls the amount of steam that is injected into the water line, and thereby the temperature in the water line and in heat exchanger I.) He could also mark valve V44 and demand information about this process component. This activates the browser, which enters the main knowledge base for

collecting the information asked for. With the help of the browser the operator could now ask for general information on the type of valve that V44 belongs to, about what components V44 is connected to, how the state of V44 influences the performance of the process, information about the regulator PI44 that controls V44, etc. In other words, the operator could browse through all, or a selected part, of the main knowledge base.

3.10 CONCLUSIONS

We have presented a concept for a knowledge-based real-time control system. The control system is an integrated system with a common knowledge base for all users. This makes it possible for knowledge to be available for all users and removes the need for redundant information. Distribution of knowledge to other processing modules, in order to achieve necessary run-time performance, is possible while still adhering to the concept of a common knowledge base.

The control system is also integrated in the sense that knowledge-based techniques are used combined with conventional techniques. Designers use the design tools to modify the main knowledge base. Realization tools are used to automatically extract knowledge from the main knowledge base and convert it into executable code. To facilitate a modular and flexible system where new tools, and thereby new capabilities, should be easy to add, the main knowledge base has a uniform interface.

4

Knowledge Representation

4.1 INTRODUCTION

A key issue in a KBCS is the representation of knowledge of various kinds. In our project the term *knowledge* is used in a wide sense covering both what is termed knowledge in the expert system community; what is found in conventional control systems today, e.g., control logic; and information in the form of text, drawings, and photographs. The motivation for this chapter is to give examples of the types of knowledge representation formalisms that may be used in a KBCS and, hence, the main knowledge base should support.

A good knowledge representation technique has the following characteristics. The representation should

- make important things explicit,
- suppress unimportant details,
- allow for completeness,
- be concise and efficient,
- support modifications,

- be transparent,
- facilitate storage and retrieval mechanisms, and
- have associated computation or deduction procedures that can 'execute' the knowledge representation and use it to conclude new knowledge.

Selection of knowledge representation is often a tradeoff between expressability and derivability. Consider the two extreme cases of predicate calculus and natural language text. Predicate calculus has a limited expressability. The derivability is, however, very high. Resolution is a computational procedure operating on predicate calculus through which theorems can be proved. Text has a very high expressability. What can be expressed in natural language text is unlimited. Written text has, however, a built in incompleteness and ambiguity that makes formal derivation procedures impossible. Although some research is performed on automatic story understanding, the computational procedures that exist for text are limited to search methods for matching, alpha-numerical patterns.

An important issue to have in mind when discussing knowledge representation is that *"A single uniform all-purpose knowledge representation technique does not exist"*. Each technique has its strengths and weaknesses and is more or less suited for a certain purpose. Knowledge representation and use cannot be separated.

In this chapter, commonly used knowledge representation techniques are described. The techniques described range from basic knowledge representation formalism such as rules, objects, etc., to more specialized methods such as the Grafcet method for representing sequential actions and the MFM (Multilevel Flow Model) methodology for representing functional process models. The amount of text describing each representation form does not reflect its relative importance. Representation techniques normally associated with expert systems such as rules and objects are described in more detail than conventional, well-known representation techniques such as procedures and equations even though the latter techniques are equally important. Also, techniques used explicitly in the prototypes such as the Diagnostic Model Processor method for model-based diagnostics and the MFM formalism are described in more detail than other techniques.

Emphasis is given to how the different techniques can be structured and implemented in an object-oriented system and to the hierarchical nature of the different formalisms. The graphical presentation of each formalism is also shown.

4.2 OBJECTS

Objects are the basic and most general knowledge representation formalism used in the project. Objects may represent general, physical as well as abstract concepts. A major strength of objects is that they represent a concept with its

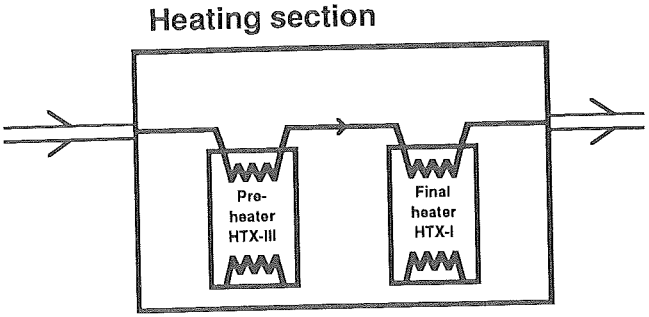


Figure 4.1 Composite model

associated characteristics as a single entity or "knowledge chunk". The characteristic properties of an object is represented by its attributes. Attributes may contain constant values of different types, variables, lists of items, other objects, procedures, icons, pieces of text, references to pictures, etc.

Through the distinction between class and instance objects, generic knowledge common to all objects of a class can be separated from specific knowledge pertaining to a single individual object. A class can have subclasses that represent different specializations of the class. The subclasses inherit the properties of the superclass as well as include their own specific properties.

4.2.1 Inheritance, views and composite objects

The superclass – subclass relation in combination with inheritance mechanisms make object-oriented representation compact and natural. Inheritance schemes where each class has only one superclass are called single. This leads to a hierarchical inheritance tree. In multiple inheritance a class can inherit attributes from more than one superclass. Multiple inheritance is natural when an object, at the same time, can be seen as an instance of more than one class, i.e., it belongs to more than one inheritance hierarchy. Through multiple inheritance, the properties are described as the combined properties of all the object's superclasses. The second reason for multiple inheritance is for structuring purposes. Properties that are common to many different classes but which cannot be naturally structured into a single class that may have instances can instead form a *mix-in* class. Mix-in classes are used to add a specific behavior or set of attributes to other classes and are never instantiated on their own.

Multiple views or multiple perspectives are used for the case when a single object, at every time, can be seen from more than one perspective, having different properties in the different contexts. In the different views the object may inherit from different superclasses. Multiple views can be seen as a version of multiple inheritance where the behavior and attributes from the inherited classes are kept separated in the object instead of being combined.

The name composite object is used for an object whose attribute values are other objects. Composite objects is one way to achieve hierarchical object structures where a sub-object of an object represents a more detailed description of a part of the object as shown in Fig. 4.1.

4.2.2 Graphical representations

The natural graphical representation for an object is an icon. If the object is used to model a physical entity, such as a pump or a heat exchanger section, the icon usually has some resemblance to the actual entity. In some cases, e.g., for electrical circuit objects, standardized symbol libraries exist.

Objects have relationships to other objects. Some types of relations are natural to show as graphical connections between objects. This is obvious in the case of physical connections among physical objects such as flow pipes connecting together process components in a flow schematic or electrical wires in an electrical schematic, but it can also be very useful to show certain relations between abstract objects. Examples of this are connections representing the caused-by relation in a network of event objects or connections representing influences in an influence diagram.

Connections may be typed, i.e., defined in terms of connection types, or classes, with associated attributes. Connections that have attributes may be used to represent structured connections, e.g., an electrical cable consisting of a number of electrical wires. Connections may also have directions.

For multiple view objects, one can think of a situation where an object has different icons and different connections in the different views as shown in Fig. 4.2.

The inheritance tree or network also has a graphical definition. Here, class definitions have associated icons, the interconnections of which represent the superclass-subclass relation and the instance-of relation as shown in Fig. 4.3.

4.2.3 The different roles of objects

Objects can serve three different purposes in a knowledge-based control system.

- Representing knowledge about some concept of interest.
- As a programming paradigm in the KBCS.
- As a programming paradigm in the implementation of the KBCS.

The first case has already been discussed. In the second case, object-orientation is used as a programming paradigm within the knowledge-based control system. In an object-oriented programming language, an object is a self-contained entity

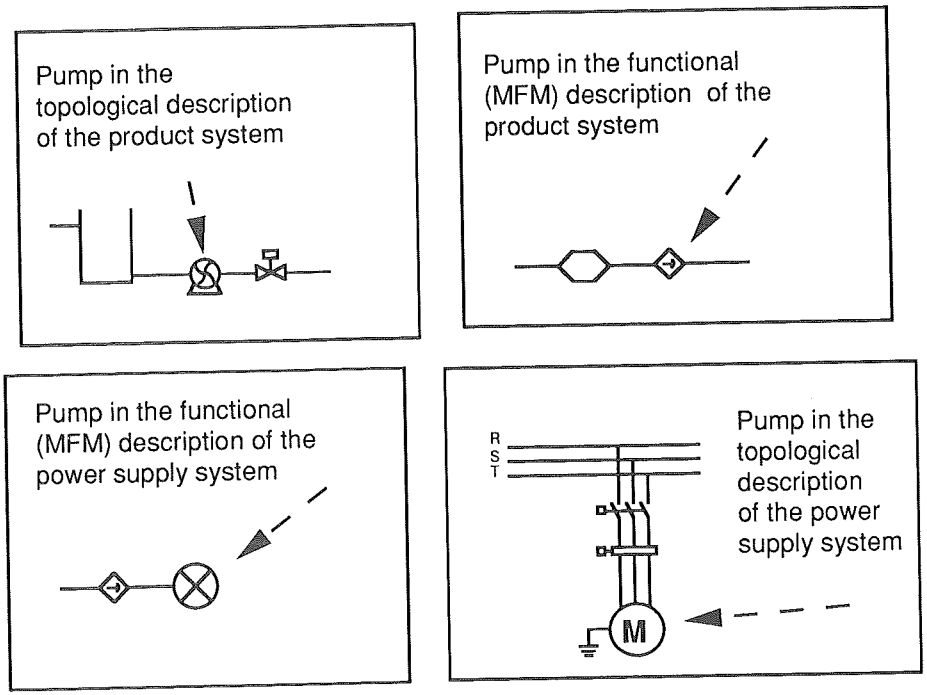


Figure 4.2 Multiple view, or perspective, object with different icons and connections

which has its own private data and a set of operations to manipulate that data. The set of operations defined for an object constitutes a uniform external interface to the rest of the system. Interaction with an object occurs through requests for the object to execute one of the operations in its interface. New objects can be defined as extensions of existing ones with the use of inheritance. When object-orientation is used for programming purposes, objects will also comprise several other knowledge representation techniques. For example, rules, procedures, and equations can all naturally be seen as objects.

The last case contains the situation when a conventional object-oriented programming language such as C++ or Eiffel, is used as the implementation language for an object-oriented knowledge-based control system. In this case, you have two levels of classes and objects, the classes and objects in the implementation language which are only visible to the system developer and the classes and objects in the knowledge-base visible to the designer and users of the control system. There is not necessarily a direct mapping between the two types of objects. On the contrary, it is likely that they have very little connection.

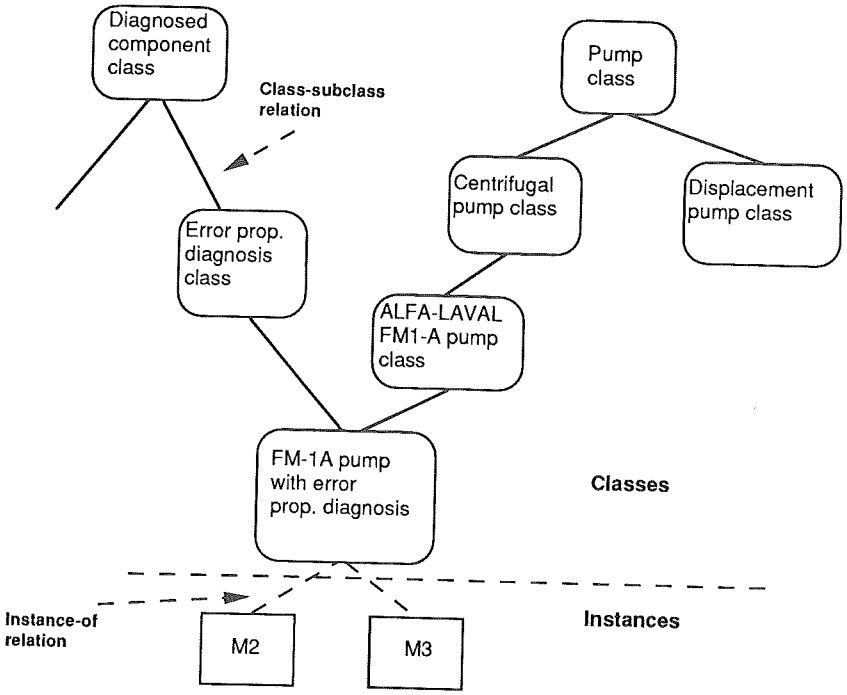


Figure 4.3 Inheritance network

4.2.4 Steritherm Examples

Here, some examples of what might be represented in terms of objects in the knowledge-based control system for the Steritherm process are given.

Process components: Process components such as pumps, valves, tanks, heat exchanger sections, pipes, etc., are naturally represented as objects. Associated attributes may contain operating status, physical parameters, component information, maintenance status, time in operation, inferred fault diagnosis information, equations describing the components quantitative or qualitative simulation models, etc. Larger parts of the process composed out of several individual components can be described as composite objects. Some examples are the heat exchanger consisting of the different sections, the balance tank including associated valves and pumps, the homogenizer, etc. In the same way the entire Steritherm process can be described as an object.

Sensors and control signals: Sensors can be described as objects with attributes such as measured value, reliability, noise range, internal models, or other information about the validity of sensor readings. Control signals may contain attributes such as operating range, saturation limits, etc.

Control system components: Control system hardware and software components such as CPUs, I/O boards, A/D converters, internal variables, and various types of function blocks such as combinatorial logic blocks, timers and counters, selectors, arithmetical blocks, comparators, alarm blocks, and different types of predefined controller blocks are also natural to view as objects. They can have associated creation date, validity intervals, expiration time, time history, information about normal and abnormal signal values, etc. Associated with the control blocks are attributes containing information about execution period, parameters, the procedure which implements the control block, etc.

Rules: Rules for, e.g., monitoring and diagnosis can be compared with control function blocks and be seen as objects in a similar way. They may have information about inference chaining, information to what rule groups the rule belongs, etc.

Steps: Sequential control is naturally expressed as sequences of states, or steps, and transitions in a Grafcet style. These elements can be seen as objects. A step may have associated information about initial and final conditions, control variable settings pertaining to that step, abnormal conditions that may occur in that step, possible substeps that the step is made up of, etc.

Events: Events that occur in the process are naturally represented as abstract objects. Alarms and operator warnings can be subclasses of events that have information of what caused the alarm, time of occurrence, alarm explanations, advice on countermeasures, etc.

Faults and symptoms: In a fault diagnosis system it is natural to view faults, fault hypotheses, and symptoms as objects. Faults may be physical faults or functional faults. Symptoms may be expressed as constraints on sensor values, the timing of events, and their order of occurrence.

Product information: Raw products can be seen as objects having attributes about chemical characteristics, processing characteristics, and what recipes it is used in (a recipe can also be seen as an object), etc. Final products may contain information about production data, production date, order and customer information, etc.

Various: In a wider perspective, the examples of what might constitute an object are infinite. Some further examples are production orders and production plans, maintenance plans, the parts of the electrical and hydraulical subsystems, etc.

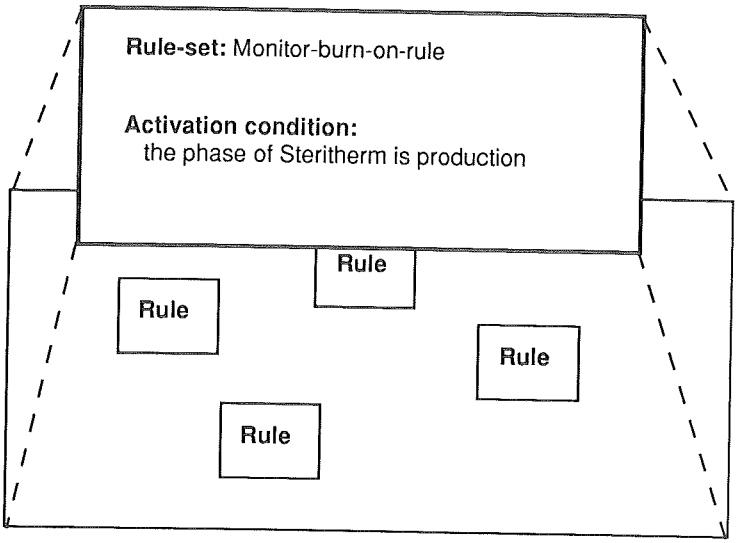


Figure 4.4 Hierarchical rules

4.3 RULES

Rules is the knowledge representation formalism usually associated with expert systems. The basic structure of a rule is

If conditions *then* conclusions.

The condition part represents a set of conditions on, e.g., process variables, that must evaluate to true in order for the conclusions or actions to take place. The rule conditions could be general logical expressions or patterns that must match the database that the rules operate upon. The way in which the rules are invoked, i.e., selected for evaluation, is determined by the inference strategy. The standard methods in conventional expert systems are forward chaining and backward chaining. For on-line systems it is useful to be able to associate a scanning interval with rules that determine how often they should be invoked. Explicit rule invokation through, e.g., meta-rules that invoke other rules is another possibility.

In the case of rules that operate on an object oriented database, two types may exist: rules that operate on instance objects and rules that operate on class objects. The latter, generic, rules correspond to a set of rules where each rule is specific to one instance of the class.

In an object-oriented system, rules can be described as objects that have attributes for the rule conditions, actions, scanning interval, chaining details, etc. Rules can either be individual objects or objects associated with the instance or class that they describe. The former alternative is necessary when the rule refers to several objects, as in the following case:

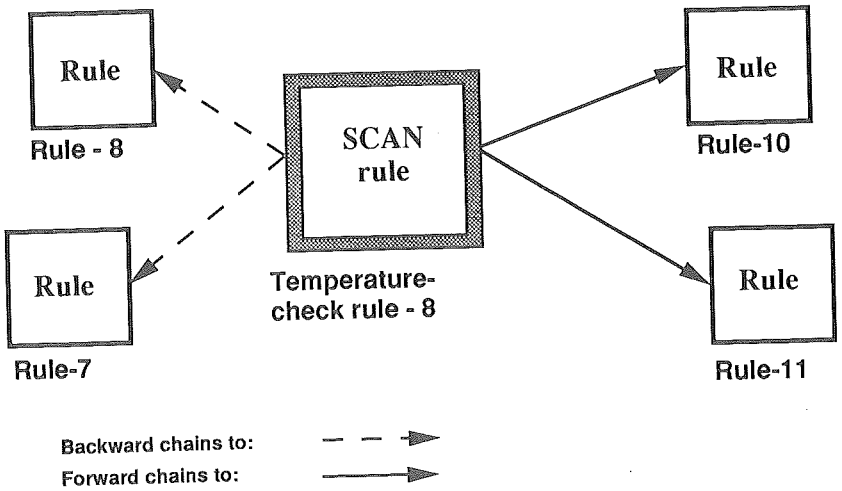


Figure 4.5 Graphical rule representations

```
for any tank
if the status of the tank is not normal
and
   the status of the valve connected to
   the tank is not normal
then
 conclude that the possible-fault-cause of the
 tank is valve-error
and
 conclude that the status of the valve is suspect
```

This rule refers both to valves and tanks. In fact, what it refers to is a pattern of interconnected objects. Therefore it is not natural to associate this rule with either the tank class or the valve class.

Hierarchical structures are not common for rules. However, a meta-rule that determines the applicability of its associated rules, or sub-rules, depending on some condition, can be seen as a composite rule that has an internal structure of rules. This is indicated in Fig. 4.4.

4.3.1 Graphical representations

Rules can have several different graphical representations. In G2 a rule is represented by a rule icon where the icon is the textual representation of the rule. It would be useful if the user had freedom to define the rule icon. Special icons could be used for generic rules, and for different rule groups. Graphical connec-

tions might be used to indicate the chaining between the rules as shown in Fig. 4.5.

The chaining and relations between rules are normally described graphically using tree and network structures. In Nexpert Object a graphical rule network is automatically generated and can be used to browse through the rule base as shown in Fig. 4.6.

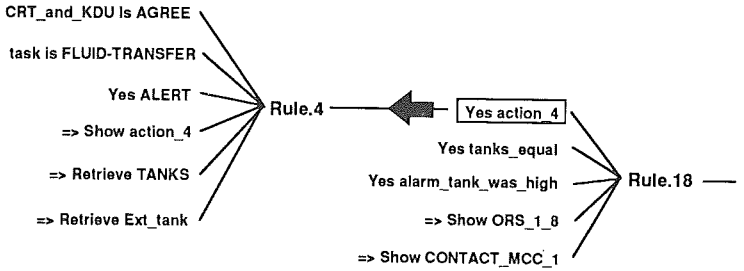


Figure 4.6 Nexpert Object rule network

Rules of the backward chaining type can be seen as an alternative way of realizing a decision tree with the obvious graphical representation shown in Fig. 4.7.

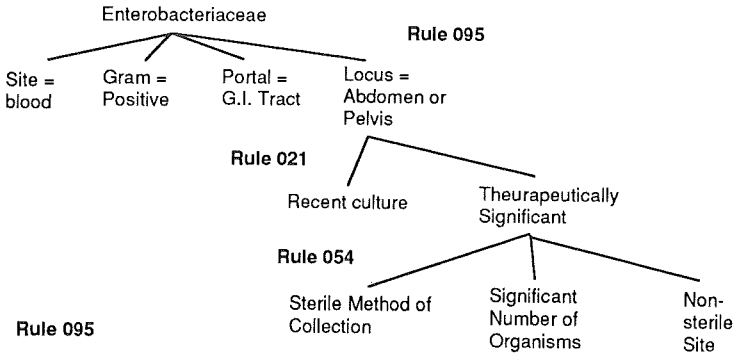


Figure 4.7 Rule and corresponding decision tree example taken from EMYCIN

Rules that operate directly on process variables can be compared with ordinary control function blocks. In that case it is natural to have the same graphical representation as function blocks have. This is shown in Fig. 4.8.

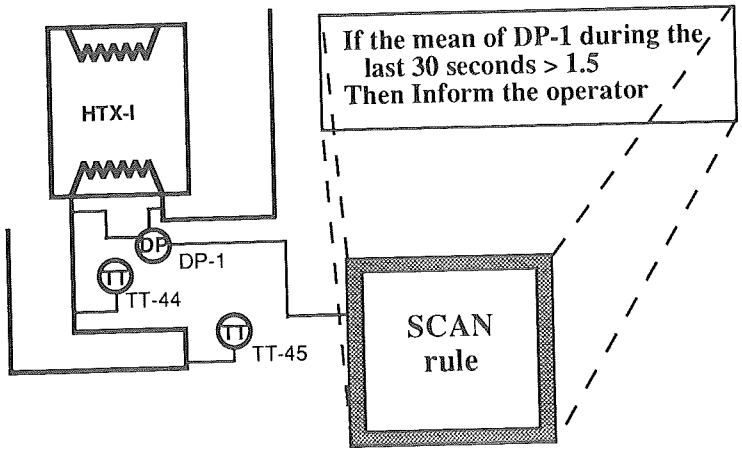


Figure 4.8 "Function block" rule

4.3.2 The double role of rules

As in the case of objects, rules may serve several purposes. The most natural purpose for using rules is to represent associations between, e.g., causes and effects. Rules of this kind often express heuristic, experiential knowledge. The knowledge is explicitly represented in well-defined rule modules. In some systems explanations can be automatically generated from the rule chaining. Rules can, however, also be used to represent other types of knowledge. Deep model-based knowledge can, e.g., be implemented in terms of rules that describe the generic behaviour of classes of components.

Rules, specially of the forward chaining, pattern-matching type, can also be used for general programming purposes implementing arbitrary algorithms. The action part of a rule can be seen as a procedure whose applicability is determined by the rule conditions. Using rules as a substitute for procedural programming can, in some cases, lead to code that is very difficult to understand and that would be better expressed in a conventional procedural high-level language. However, in other cases the modularity and flexibility of the rule and the non-sequential activation is an advantage.

4.3.3 Steritherm examples

In the Steritherm process rules have several applications. Some examples are supervision of burn-on, alarm analysis, and selection of cleaning procedure.

The cleaning to remove burn-on is usually done on a pre-scheduled basis where the time intervals depend on the actual product being processed. For processes equipped with differential pressure transmitters on the heat exchanger sections excessive burn-on may be detected by the control system. Rules that monitor

the time history of the differential pressure may notify the operator when the rate of change is too large, or when the magnitude exceeds a certain threshold value. Another way to detect burn-on is by monitoring the control signal of the PID regulator controlling the valve of the steam injector. Burn-on causes the heat transfer coefficient in the heat exchangers to decrease. Hence, the controller needs to increase the valve opening to maintain the temperature at the set-point. Monitoring of the control signal can be a secondary means for detecting burn-on if the pressure transmitter should fail. Examples of this kind of rules are given in Section 7.3.4.

Fault trees for helping the operators and service personnel to find the original fault that has caused an alarm are today normally not used in the plants. They can, however, be obtained from process designers and start-up engineers. These fault trees can relatively easy be transferred into a set of alarm analysis rules that give advice on fault localization. This is further described in Christiansson and Ericsson (1989).

The selection of cleaning procedure is an operation that requires a large amount of heuristics. The sequence, amounts, and cleaning time of the cleaning liquids depend on the type of product and how the process has been cleaned previously. It is possible to express this as a set of rules for how to select a proper cleaning sequence.

4.4 LOGIC

Standard logic systems such as propositional logic and predicate calculus are an often used formalism within the AI community. The advantage of using logic for knowledge representation is that it has a firm theoretical basis and well defined reasoning methods, e.g., resolution. The disadvantage is that the logic systems have restrictions with respect to what they can express and therefore can be quite rigid. To solve this problem several non-standard logic systems have been developed.

4.4.1 Propositional logic

The language of propositional logic consists of proposition which are either true or false. Propositions are combined into clauses using the ordinary set of logical connectives, i.e., `AND (&)`, `OR` ($\vee$), `NOT` ($-$), `IMPLIES` ($\Rightarrow$), and `EQUIVALENCE` ($\Leftrightarrow$).

The basic rule of inference is Modus Ponens. This simply says that whenever a fact `A` is known to be true and there is a clause `A` $\Rightarrow$ `B`, it is permitted to conclude that `B` is true. Propositional logic, as well as predicate calculus, is monotonic. If the logical statement `A` can be proved from a set of initial axioms, additional

axioms or information must not cause the negation of A to be provable. If this was the case, the logical system would be inconsistent. Due to the monotonicity property, the beliefs of the system are considered to be always true and the system monotonically draws new conclusions from the existing ones. Unfortunately, monotonic systems cannot handle three kinds of situations that often arise in real problem domains: incomplete information, changing situations, and generation of assumptions during the problem solving process.

4.4.2 Predicate Calculus

Predicate calculus or logic is based on predicates that are either true or false. Predicates have arguments that could be either constants, variables, or functions. Predicates are combined with the standard logical connectives together with the universal quantifier ("for all", $\forall$), and the existential quantifier ("there exists", $\exists$). Example of clauses in predicate calculus are:

$\forall$(x) man(x) $\Rightarrow$ human(x)

$\exists$(x) father(john,x) & female(x)

The inference method of predicate calculus is resolution. The inference problem is stated as a set of axioms A and a theorem T which we want to check if it follows from the axioms. This done be showing that the set $-$T $\cup$ A is unsatisfiable, i.e., that it leads to a contradiction. Resolution in its simplest form is a procedure that involves translating all logical formulas into clausal form, selecting two clauses, and unifying those and creating the resolvent clause. This procedure is continued until a contradiction is found in which case it has been shown that T follows from A.

Resolution can be mechanized and is the basis of the so called theorem provers. A special form of resolution is also the basis for the inference mechanism in Prolog (Clocksin and Mellish, 1981). Prolog is based on the Horn subset of predicate calculus.

4.4.3 Non-standard Logic Systems

Several non-standard logic systems have been developed. These include modal logics, temporal logics, fuzzy logics, etc. Several of these are surveyed in the Feasibility Study. For a good overview see (Turner, 1984).

4.5 EQUATIONS

Equations of different forms naturally represent relations and constraints among process variables. Equations can be quantitative or qualitative. Quantitative equations operate on numerical or boolean variables. Qualitative equations usually operate on the signs of numerical variables. Quantitative equations on assignment form are what is normally used in conventional control systems.

4.5.1 Quantitative equations

Quantitative equations include, e.g., algebraic equations, differential equations, difference equations, and differential-algebraic equations. Equations have many usages both in conventional control systems and in KBCSs.

Ordinary, boolean or real-valued, expressions for representing control logic can be seen as quantitative difference equations on assignment form.

Quantitative equations are the normal way of representing simulation equations used in, e.g., off-line training simulators or on-line predictive, decision support simulators. The simulation equations can have different resolution. Equations in simple simulation models may only cover the static behavior of the process. More detailed simulation models may contain important dynamic modes, and discrete events describing, e.g., mode changes. If the simulation is performed on-line and in real-time, simulation equations can be used as an alternative to procedures for implementing arbitrary dynamic filters. These could be used in state observers, as low-pass or band-pass filters, for model-based sensor validation, etc.

Quantitative equations are also used to represent constraint equations that during normal operating conditions should hold between different process variables. Examples of constraint equations of this kind are mass and energy balance equations. These could be used for diagnostic purposes in which case the diagnostic engine checks for violated constraints and uses that to infer possible faults.

Equations can be seen as objects with icons as their graphical representation in a similar way to rules. They can be internal to, e.g., objects, giving the actual value of some object attribute, or global. In some cases also other graphical representations are possible. Consider a normal boolean equation. It can be represented on equation form, as interconnected function blocks, or as a ladder diagram according to Fig. 4.9.

Simulation equation examples

As a part of the G2 prototype a real-time, numerical simulation model of the Steritherm process has been derived (Christiansson and Ericsson, 1989). The model contains algebraic and differential equations simulating pressures, flows, temperatures, levels, etc., in the various process components. In Chapter 7, the equations for the plate heat exchanger are shown.

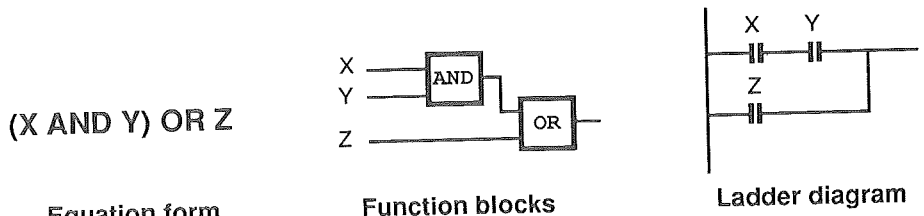


Figure 4.9 Different representation for boolean equations

Although the simulation equations are not a part of the control system knowledge-base in the G2 prototype, it is clear that in a real KBCS they should be a part of the objects describing the process components. The Steritherm simulation model is interesting from many aspects. The equations for calculating the product and water flows are not local to any process component. Instead global system equations are used to calculate the flows. In a hierarchical, object-oriented representation these equations would belong to a composite process object which, in this case would be the object representing the entire Steritherm process.

The differential equations modeling the heat transfer are described by generic differential equations describing the behavior of all instances of the heat exchanger class. In an object-oriented decomposition these equation would belong to the heat exchanger class.

The diagnostic model processor

An example of a model-based diagnosis method based on quantitative constraint equations is the Diagnostic Model Processor method (DMP) (Petti *et al*, 1990). The method has been implemented in the G2 prototype and the implementation is described in Chapter 7.4.6.

The diagnostic model processor is a method of using deep or model-based evidence to arrive at the most likely fault conditions of a process. The method was invented to rectify some of the problems associated with rule-based knowledge-based systems. These problems include lack of generality, poor handling of novel situations, and the tendency to fail suddenly. The main problem addressed by the method is that of generality. Traditional knowledge bases become invalid in the event that the target process undergoes changes. It is also difficult to apply the knowledge bases to other processes. It is therefore desirable to structure the fault analyzer such that the process specific knowledge (model) is maintained separate from the task specific knowledge (methodology). The architecture of the diagnostic model processor achieves this separation under the requirement that the process is represented in the format discussed in the following section.

The diagnostic model processor works under the premise that during fault free operation, the actual process and process model equations should produce similar outputs when driven by the same inputs. By examining the direction and extent

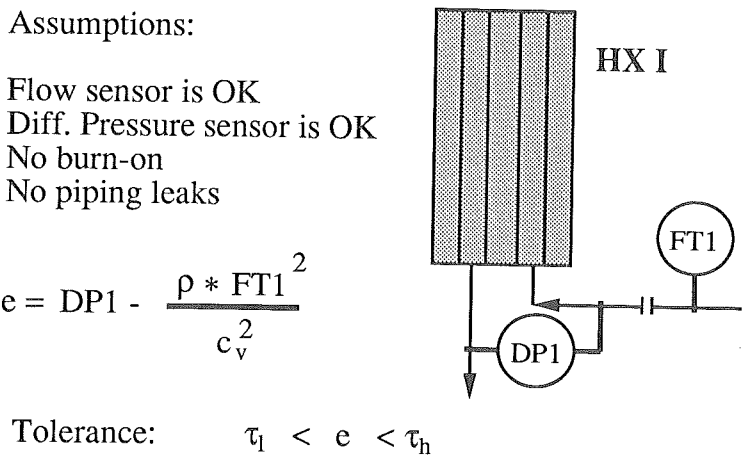


Figure 4.10 Example of the formulation of a model equation with its associated assumptions.

to which each model equation is violated and by considering the assumptions on which they depend, the most likely failed assumptions (faults) can be deduced. Redundancy which is available in the system leads to better performance because an assumption which is common to many violated equations is strongly suspect; whereas, satisfaction of equations provides evidence that the associated assumptions are valid. The formulation of the process model is very important to ensure the competence of the analyzer. Care must be taken to include all the applicable assumptions with the model equations so that the associated faults can be diagnosed.

Model equations: The process model is listed as a series of simple governing equations which describe the process. The model equations which can be used are dependent on the instrumentation of the target process. Associated with each model equation are tolerance limits which give an indication of when the equation is no longer representative of the process. Each model equation is written in a form such that it ideally equals zero. Process noise, modeling error, and faults prevent them from equaling zero; the discrepancy is called the residual. The tolerance limits are the expected (fault free) upper and lower values of the residual for which the equation is considered satisfied. Also associated with each equation is a set of assumptions which if satisfied, guarantee the satisfaction of the model equation. A simple example of an equation formulation and the associated assumptions is shown in Fig. 4.10. Notice that some of the assumptions are explicit in the model equations, such as correct sensor readings, and some are implicit such as the fact that there are no piping leaks.

The diagnostic methodology begins with the calculation of a vector of model equation residuals, e, from the process data P. The residual of the jth model equation,

$$e_j = c_j(P; a) \tag{4.1}$$

where a is shown to indicate that each equation is dependent on the satisfaction of a vector of modeling assumptions.

Since the residuals of the model equations are not uniform in magnitude, they are transformed into a metric between -1 and 1 which indicates the degree to which the model equations are satisfied: 0 for perfectly satisfied, 1 for severely violated high, and -1 for severely violated low. These values constitute the satisfaction vector, sf, which is calculated using the model equation tolerances, τ. For the jth model equation,

$$sf_j = \frac{(e_j/\tau_j)^n}{1 + (e_j/\tau_j)^n} \tag{4.2}.$$

The value of sf_j is given a positive value for a positive residual, e_j, and a negative value for a negative residual. The curve is a general sigmoidal function with the steepness determined by the constant n. The curve is shown in Fig. 4.11. If the tolerances are not symmetric about the origin, the upper tolerance is used for a positive residual and the lower tolerance is used if the residual is negative.

Relationship between equations and assumptions: A matrix of sensitivity values, S, which describes the relationship between each model equation and assumption is computed to weight the sf values as evidence. The ijth element of S, which represents the sensitivity of the jth model equation to the ith assumption is calculated as:

$$S_{ij} = \frac{\frac{\partial c_j}{\partial a_i}}{|\tau_j|} \tag{4.3}.$$

The larger the partial derivative of an equation with respect to an assumption, the more sensitive that equation is to deviations of the assumption. Similarly, equations with large tolerances τ, are less sensitive as they are more difficult to violate. Many model equations are non-linear in some assumptions; these partials are estimated by linear approximations. Assumptions which are implicit with respect to an equation (i.e., pump operation) are arbitrarily given a partial derivative equal to 1, unless experience suggests otherwise. Also, equations independent of an assumption have an associated sensitivity of zero.

Calculation of failure likelihoods: Conclusions about the satisfaction of each assumption (fault state) can be made by combining the evidence from the model equations, sf, with consideration to the sensitivity matrix S. We desire a method of combination which results in a normalized measure of the satisfaction of each assumption, a_i, and allows for direct contradiction of evidence which

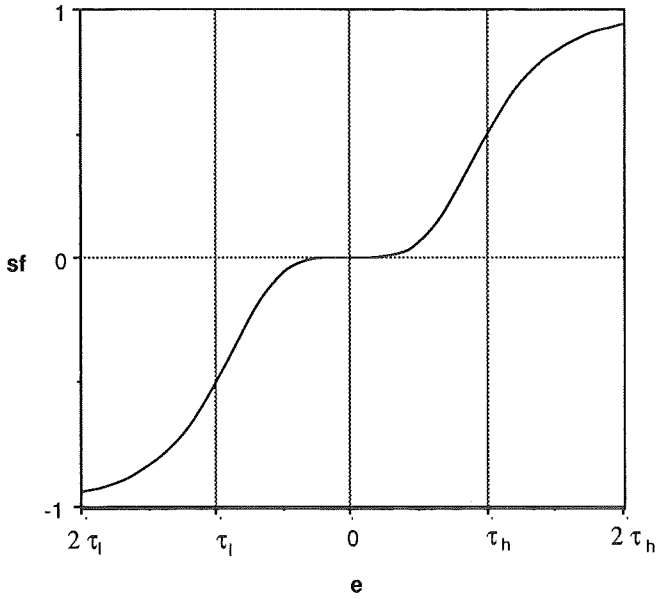


Figure 4.11 Satisfaction value as a function of equation residual. High and low tolerance is indicated. $n = 4$.

suggests failures in opposite directions (high and low). A combination which satisfies these requirements is the calculation of a vector of failure likelihoods, F, such that

$$F_i = \frac{\sum_{j=1}^{N}(S_{ij}\, sf_j)}{\sum_{j=1}^{N}|S_{ij}|} \tag{4.4}$$

where N is the number of model equations. It is evident that this method of combination allows the sf values of those equations which are most sensitive to deviations of assumption a_i to be weighted the most heavily in the calculation of F_i. The failure likelihood is interpreted as indicating a likely condition of assumption a_i failing high as the value of F_i approaches 1, while an F_i tending toward -1 indicates a likely failure low.

Example: A simple but realistic example is used to demonstrate the operation of the methodology. A series of temperature sensors are shown in Figure 4.12. The configuration is similar to that in the holding tube of the Steritherm process.

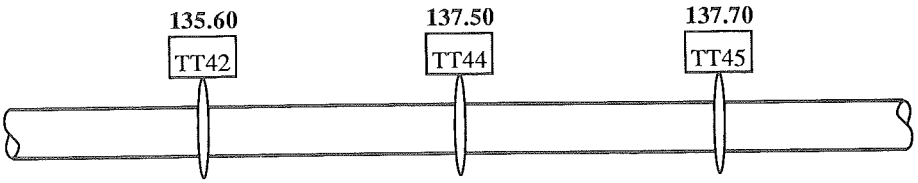


Figure 4.12 Example schematic showing three temperature sensors measuring a similar quantity

The three model equations can be written as follows:

$$c_1\text{: } e_1 = \text{TT42} - \text{TT44}, \text{ where } \tau_h = -\tau_l = 1$$

$$c_2\text{: } e_2 = \text{TT44} - \text{TT45}, \text{ where } \tau_h = -\tau_l = 1$$

$$c_3\text{: } e_3 = \text{TT45} - \text{TT42}, \text{ where } \tau_h = -\tau_l = 1$$

and the assumptions to be considered are

$$a = \begin{pmatrix} \text{TT42 is OK} \\ \text{TT44 is OK} \\ \text{TT45 is OK} \end{pmatrix}.$$

Using the sensor values shown in figure 4.12 and Equation 4.2 (with $n = 4$), the vectors e and sf are calculated.

$$e = \begin{pmatrix} -1.9 \\ -0.2 \\ 2.1 \end{pmatrix}, \text{ and } sf = \begin{pmatrix} -0.929 \\ -0.002 \\ 0.951 \end{pmatrix}.$$

The partial derivative of the equations with respect to the assumptions are all constant so the sensitivity matrix S is calculated using Equation 4.3 as

$$S = \begin{pmatrix} 1 & 0 & -1 \\ -1 & 1 & 0 \\ 0 & -1 & 1 \end{pmatrix}.$$

It is obvious that the equations are not linearly independent; however, this redundancy helps improve the diagnosis.

Using Equation 4.4 the failure likelihoods can be calculated. The vector

$$F = \begin{pmatrix} -0.940 \\ 0.464 \\ 0.477 \end{pmatrix}$$

clearly indicates the failure of the TT42 sensor and in the proper direction (low). This is the correct diagnosis as is obvious by examining the sensor readings in this simple example (two of the three sensors agree).

Advantages: The major advantage offered by the diagnostic model processor is a true separation between the process model and the diagnostic procedure. This allows for easy changes to the process knowledge to include altered plants or improved equations. Additionally, new processes can be handled by using the design equations associated with them. The separation discussed is shown in Section 7.4.6 to have a positive impact on an object-oriented implementation of the diagnosis method.

Another advantage of the methodology is the use of a non-Boolean measure for the classification of the model equations. This approach is superior to Boolean reasoning in that stability problems are avoided which may result if the equation residuals are bordering near the tolerances. Choice of tolerance levels are also less significant and a failure can be monitored as a degradation rather than a Boolean event.

The method of combining evidence in the procedure allows for direct contradiction of high and low failures; and, the evidence is weighted according to a measure of the sensitivity. Additionally however, because the method assigns zero sensitivity to relationships between independent equations and assumptions, there exists the ability to detect multiple fault situations if they are not competing in their influence. The methodology does not demand a single explanation of the evidence. This may present some difficulty, however, in that often more than one assumption has a failure likelihood which is indicating a possible failure.

4.5.2 Qualitative equations

Qualitative equations are in many aspects very similar to quantitative equations. The major difference is their domain. Quantitative equations operate on real-valued variables whereas qualitative equations operate on signs of variables, i.e., $\{-, 0, +\}$.

The work in the area of qualitative equations and models was initiated by Pat Hayes in the 'Naive Physics Manifesto' (1979). The motivation was to provide a physics that is closer to our everyday experience rather than the precise, mathematical approach of conventional physics. Three theories have emerged representing different views of qualitatively describing complex systems. Using the notation of R. Leitch (Leitch and Horne, 1989) these are called Qualitative Physics, Qualitative Engineering, and Qualitative Mathematics.

In Qualitative Mathematics as derived by Kuipers (1986), a qualitative model is abstracted from the structure of an underlying quantitative model. The abstraction is obtained using qualitative primitives such as monotonic increasing or

decreasing, qualitative derivatives, etc. A propagation algorithm (QSIM) is used to obtain a qualitative simulation of the system. The system produces a set of possible ordered sequences of events (envisionments) rather than a true, unique temporal simulation.

In Qualitative Engineering, structure is imposed by identifying physically distinguishable subsystems or components. Each component is described by a qualitative differential equation, or confluence. The components are connected together and a constraint propagation algorithm is used to determine an overall consistent set of qualitative values. A differential equation describing an endothermic chemical reaction and its corresponding confluence equation is shown below.

$$Ah\frac{dT}{dt} = F_i(T_i - T) - Q/\rho c_p$$
$$\delta T = [F_i][T_i - T] - [Q]$$

where

A = area
h = level
T = reactor temperatur
F_i = input volumetric flow
T_i = input temperature
Q = effect removed through cooling
ρ = density
c_p = heat capacitivity

Here, δT represents the sign of the derivative of T, i.e., it has the values increasing, decreasing or stationary, and $[Q]$ represents the sign of Q.

In Qualitative Physics the basic representational primitive is a process, e.g., a flowing liquid, a dropping solid, an expanding gas, etc. The processes are defined and combined to represent an overall system under different operating conditions.

As in the case of quantitative equations, qualitative equations have many uses. The two most usually mentioned are simulation and diagnosis. In simulation, qualitative techniques are used to give a set of possible futures for the process. The simulation could be used as a predictive operator support simulator. The technique can be of value when an exact numerical simulation model is not available. The price to be paid is the ambiguity of the method.

In diagnosis, a qualitative model is compared with process measurements. In the qualitative model, fault assumptions are introduced in order to find a fault which is consistent with process measurements and thus can be the true fault in the process. Unmeasurable or ambiguous variables also receive assumed values. An

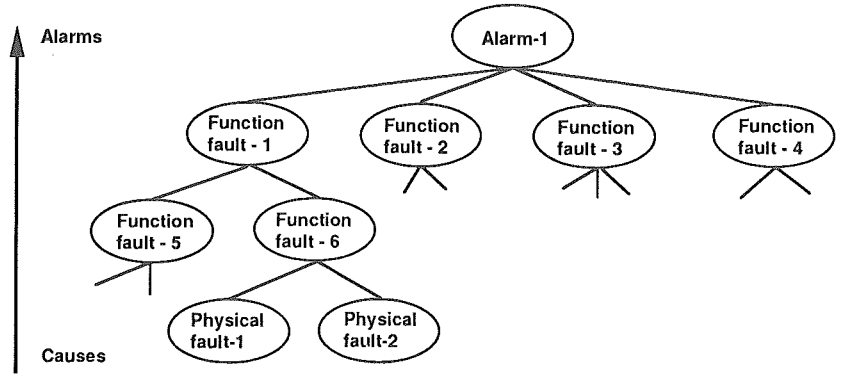


Figure 4.13 Alarm tree

assumption-based truth maintenance system combined with a constraint propagation algorithm are used to record, and withdraw assumptions and to propagate qualitative values through the process (Arlabosse *et al*, 1988).

4.6 TREES AND GRAPHS

Trees and graphs are general representation structures with several uses in a KBCS. Inheritance trees or networks describe the relations between classes, subclasses, and object instances as shown in Fig. 4.3. Relations among rules are naturally described as a rule network or a decision tree as shown in Fig. 4.6 and 4.7. The Diagnostic Model Processor method previously described is based on a graph consisting of model equations and fault assumptions connected together by dependencies.

The natural graphical representation for trees and graphs is a set of graphically interconnected nodes. In an object-oriented system a tree or graph could be seen as an object that has an internal structure of node objects that have different relations to other nodes. In some cases it is natural for a node in a tree or a graph to be hierarchical, i.e., have an internal structure of other nodes.

4.6.1 Alarm trees

Trees are a convenient representation form for tasks that involve searching for solutions to some problem or causes to some failure. An example is alarm trees where the root node may represent an alarm, the non-leaf nodes represent faults on some function of the process, and the leaf nodes represent physical faults in the process components according to Fig. 4.13. A tree of this type may assist the operators or the service personnel in off-line troubleshooting, i.e., help them

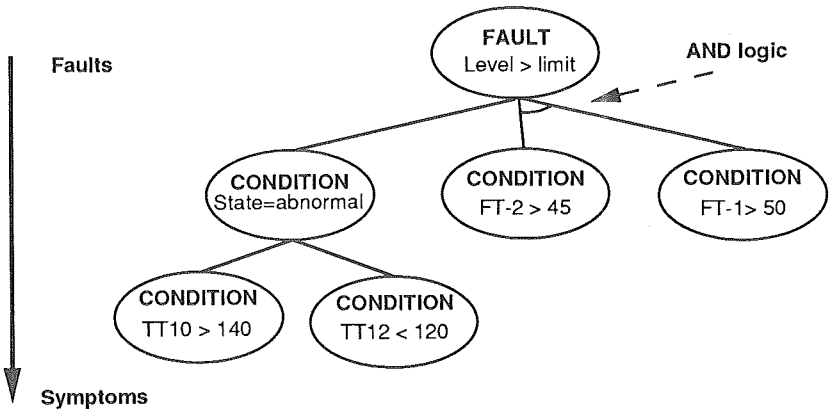


Figure 4.14 Fault tree

to locate the physical fault that has caused an alarm. Instructions and advice can be associated with the nodes.

4.6.2 Fault trees

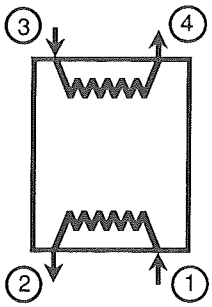
In a fault tree the root node represents a possible fault, e.g., that a certain variable is above some limit. The other nodes represent symptoms in the form of conditions on measured process variables which if fulfilled will cause the fault as shown in Fig. 4.14. Both AND and OR logic can be used in the tree.

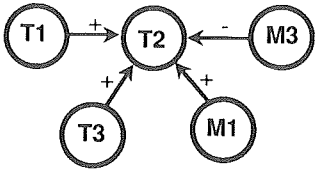
Fault trees can be used for on-line diagnosis. By successively generating and testing fault hypotheses against sensor readings according to the fault trees, alarms or warnings may be generated to the operator. Backward chaining rules match the fault tree structure well and can be used to implement it. However, in that case the inherent graphical structure of the tree is lost and replaced with a set of rules which quite often can be difficult to overview.

4.6.3 Digraphs

A digraph (directed graph), or an influence graph, is a set of nodes and connecting edges. Nodes in a digraph represent variables. If one variable affects another variable, a directed arrow or edge connects the independent and dependent variables. The directed edge may either be a normal edge which indicates that the relationship is normally true, or a conditional edge which indicates that the relationship is true only when another event (or condition) exists. Edges connecting a given pair of nodes are mutually exclusive; only one edge relationship is true at a given time.

The number $0, +1, -1$, when placed on the directed edge represents the gains (i.e., relationships) between the two variables. A gain of $+1$ from node X to





T = Temperature
M = Mass flow

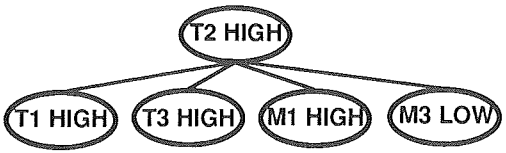


Figure 4.15 Heat exchanger example

node Y indicates that variable Y increases (decreases) when variable X increases (decreases). For -1 the opposite situation occurs.

Digraphs can be used as a basis for creating fault trees. Fig. 4.15 shows a heat exchanger where the hot stream enters at location 1 and leaves at location 2. The cooling stream enters at 3 and exits at 4. The digraph for the temperature of the output stream 2 is shown together with the corresponding fault tree. More or less automatic methods for creating fault trees exist (Allen and Rao, 1980). Special care must be taken for feedback loops.

Signed digraphs show the causal relations among variables and are commonly used as one way of expressing deep, model-based process knowledge. The main usage for digraphs is diagnosis. Faults are either associated with the HIGH and LOW states of the variable nodes or introduced in the digraph as separate nodes with edges to the other nodes. Faults may also remove some of the edges in the graph. The diagnosis is performed by locating the nodes which, if disturbed, would give a measurement pattern corresponding to the net. A drawback with digraphs is that they usually have no notion of time. The time it takes before a change is noticed in a variable is not represented. Also, digraphs only cover situations where the process is at an equilibrium point and faults show up as variations from this equilibrium. Transient behavior, start-up sequences, and running in different modes are normally not covered.

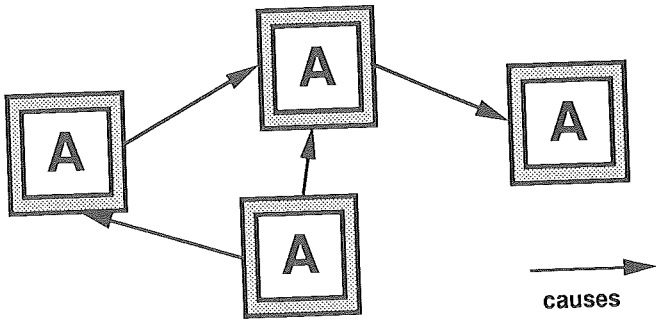


Figure 4.16 Event graph consisting of alarm events

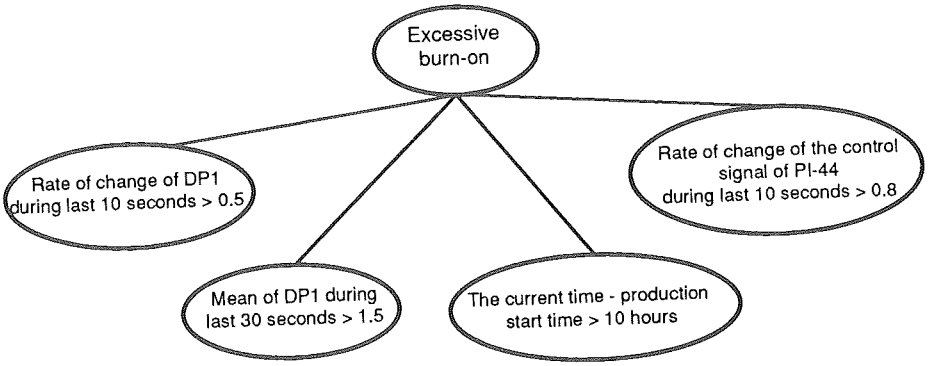


Figure 4.17 Burn-on fault tree

4.6.4 Event graphs

In event graphs, nodes represent events, e.g., alarms, and the edges represent different causal dependencies among the events. Possible dependencies could be *always causes, may cause, never causes,* etc. An example of an event graph is shown in Fig. 4.16. An event graph could, e.g., be used as the basis for an alarm filtering system that tries to find the primary alarm that has caused a set of secondary alarms.

4.6.5 Steritherm examples

An alarm tree for the high temperature alarm has been implemented in the G2 prototype described in Chapter 7. The Steritherm alarm trees are further described in Christiansson and Ericsson (1989).

A fault tree for burn-on in heat exchanger section I is shown in Fig. 4.17.

4.7 SEQUENCES

A special case of graphs are those which represent sequences of states. The sequences could, e.g., represent the sequential control logic in a process or a plan of production activities. Several special knowledge representation formalisms for representing sequential information exist. Some examples are Grafcet, Pert-diagrams, Gantt schemes, scripts, action plans, etc. The graphical representation and the object-oriented realization of these formalisms are similar to the case of trees and graphs.

4.7.1 Grafcet

Grafcet is a Petri-net based formalism for representing sequential and parallel activities (GREPA, 1985). Steps and transitions build up a sequential flow chart. A step represents a certain state of the process in which certain actions are performed. A transition contains a condition that determines when the process changes state. A sequence can be split up into parallel branches. Alternative paths can exist in the flowchart. Steps can have an internal structure of substeps and transitions, thus creating a hierarchical Grafcet structure.

Grafcet is a standard for representing sequence logic. It can, however, also be used as a graphical representation for more general algorithms.

4.7.2 Scripts

Scripts is a specialized knowledge representation formalism usually used to represent common, stereotyped sequences of events or activities. The method was developed by R. Schank (Schank and Abelson, 1977) to understand and reason about situations in the context of automatic story understanding. A usual example is a script that describes the normal sequence of activities and events that take place when a person visits a restaurant, i.e., entering the restaurant, ordering, eating, paying the bill, leaving, etc.

A script can be seen as an object that has the following set of special attributes.

Entry conditions Conditions that must, in general, be satisfied before the events described in the script can occur.

Result Conditions that will, in general, be true after the events described in the script have occurred.

Props Attributes representing objects that are involved in the events described in the script.

Roles Slots representing people who are involved in the events described in script.

Track The specific variation on a more general pattern that is represented by this particular script.

Scenes The actual sequences of events that occur.

Scripts have several usages. In the Eurohelp project (Breuker *et al*, 1989), scripts are used as the basis for a plan recognition system that monitor operator actions. In `(ihs)` (Larsson and Persson, 1987), scripts are used for a plan recognition system that monitors the commands given by an interactive user of a package for system identification.

4.7.3 Action plans

Plans can be represented as sequences of actions. The sequences may contain parallel parts. Associated with the actions are pre-conditions that must hold for the actions to applicable, post-conditions that will hold when the action is completed, and prevailing conditions that must remain fulfilled during the duration of the action. Several planning methods exist which use similar representations (e.g., Sandewall, 1988; Manna and Waldinger, 1987)

4.7.4 Steritherm examples

Grafcet has been used to represent the sequential logic for the Steritherm process in the G2 prototype. This is shown in Fig. 4.18. Each major operating mode consists of substeps. Fig. 4.18also shows the internal structure of the `Sterilization` step. The internal structure of the `Heating-sterilization` substep is shown in Fig. 4.19. In the heating substep water is heated until it reaches the sterilization temperature, $137°C$. This is the transition condition for entering the sterilize substep. This substep has two transitions. If the time elapsed since the substep was entered exceeds a specified sterilization time then the sterilization is finished. If, during that time, the temperature drops below the sterilization temperature the heating step is entered anew.

4.8 PROCEDURES

Although declarative representations are normally associated with knowledge-based systems, ordinary high-level language procedures are necessary to represent certain types of knowledge. Control algorithms are normally expresses as procedures. More advanced problem solving methods, such as planning algorithms, or constraint propagation algorithms, are best expressed in terms of procedures.

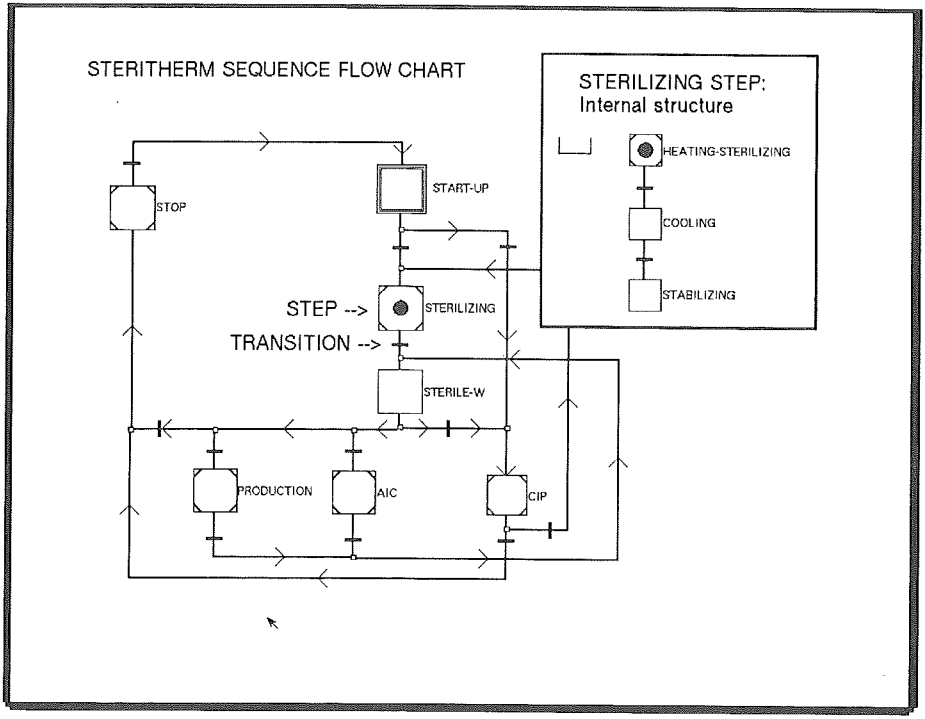


Figure 4.18 Main sequence in Steritherm process

In an object-oriented system a procedure can be seen as an object that has some kind of iconic representation. The statements allowed in the procedures should be the ones existing in languages like C, ADA etc., i.e., assignment statements, conditional statements, iteration statements, loops, etc.

4.9 FUNCTIONAL MODELS

A process such as the Steritherm can be modeled and described in several different ways. An operator, as well as a service engineer, and even the designer, very often reasons about the process in terms of its *goals* and the *functions* available for achieving those goals. The standard way of presenting the process for the operator is, however, with a process diagram, i.e., a formal description of the process topology. Therefore it is highly desirable to provide the operator with *functional models* of the plant, in addition to the topological ones. Multilevel flow modelling is one technique for making functional models of processes like

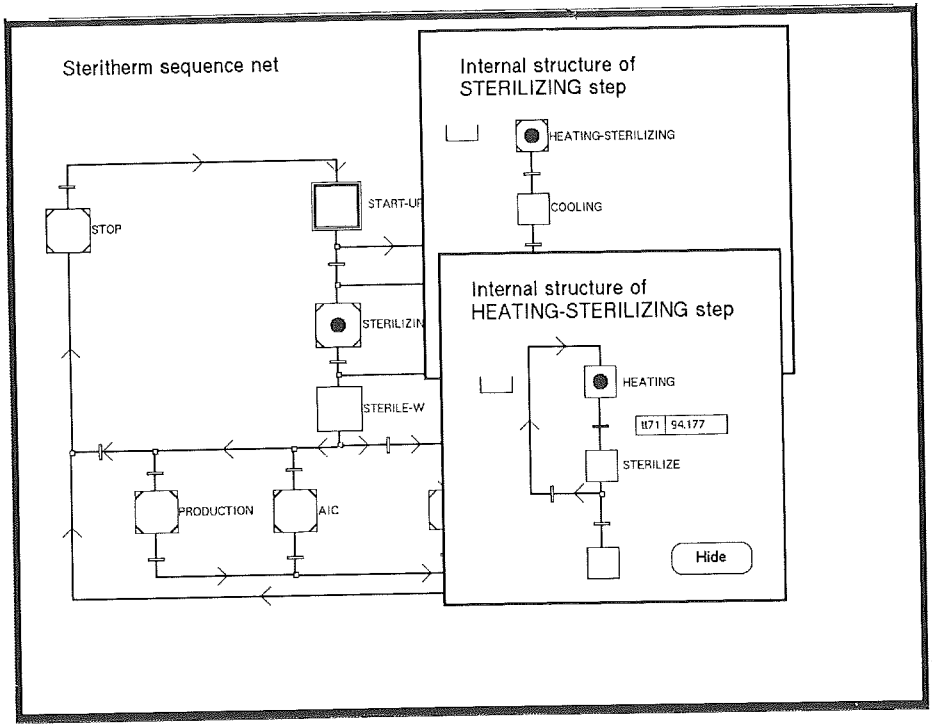


Figure 4.19 Heat and sterilize subsequence

Steritherm, and will be more closely described in the rest of this section. Other techniques also exist, see, e.g., the Travel notes from Japan in Appendix B.

4.9.1 Multilevel Flow Models

The Multilevel Flow Modeling technique, (from now on called MFM), has been developed from the analysis of the *functional* structure of complex systems such as nuclear power plants and chemical processing units. An early motivation can be found in Rasmussen and Lind (1981), while the basic principles and definitions of the graphical language are found in Lind (1987). In this section, we will only give a brief overview of the possibilities of using MFM techniques in the current project, together with a small example of a heat exchanger.

4.9.2 Basic Abstraction Principles

In MFM, there is a distinction between (at least) two different views of a process, with three discernible levels, see Fig. 4.20. The functional view represents the *goals* of the process and the *functions* provided. The goals describe the operational objectives of running the process, e.g., achieving production, efficiency, and safety. The goals are achieved by functions or networks of functions, and connected to these via *means-end* relations; thus, the goals and functions form a hierarchy of such relations.

The physical component view describes what components are present in a system and how these connect into subsystems. The relations between objects in this view are *connection* relations and the relations between systems and subsystems are *part-whole* relations. They all describe the topological structure of the physical system. The components are connected to the functions via *realize* relations. It should be noted that both achieve and realize relations may be of a complex and many-to-many correspondence nature, which makes the connections between the functional and the topological models non-trivial.

The main idea of multilevel flow models is to provide a formal way of representation goals and functions, i.e., the functional structure, of processes consisting of mass and energy flows. The MFM language has a formal syntax and can provide for several different formal semantics, e.g., schemes for knowledge-based diagnosis. However, the concept of a function is limited to that of a *flow* function. Many other functions are not treated. This is not as severe a limitation as it may seem, though. Most industrial processes can be described with flows of matter, energy, and information, all of which is captured in MFM.

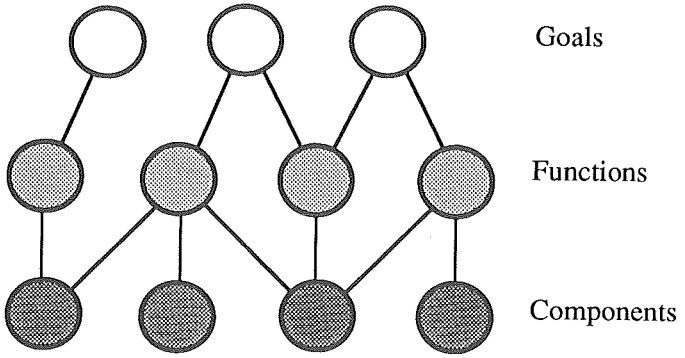


Figure 4.20 A process is modeled in three functional abstraction levels.

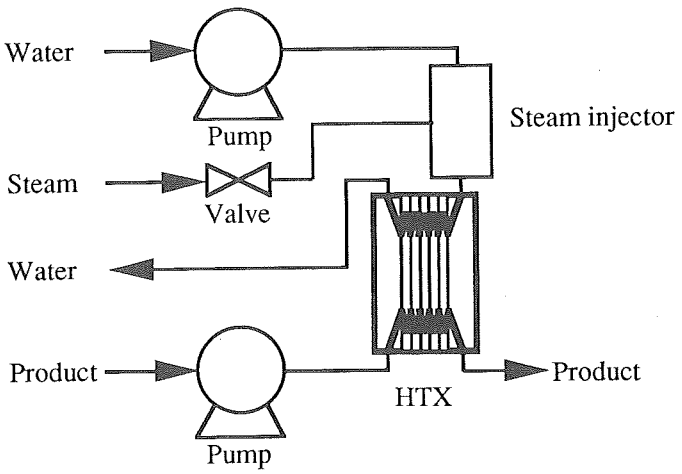


Figure 4.22 A heat exchanger system from Steritherm.

4.9.3 The Graphical Language

The basic MFM representation is one of objects such as physical components, connections, goals, flow functions, control goals, etc., and relations between these objects, e.g., connection of and aggregation into sub-systems of components and achieving a goal by a certain function. A graphical language has been developed for describing different types of objects and relations. In Fig. 4.21 there is an example of flow function symbols. It should be noted, however, that the basic MFM idea is not specifically connected to any certain graphical representation; the symbols may vary, as long as the syntax remains the same. In fact, a new set of symbols is currently being developed in the SIP project, Lind *et al* (1987). The symbols in Fig. 4.21 is only a selection of all symbols. More symbols are seen in the examples below. These symbols all adhere to the older standard of Lind (1987).

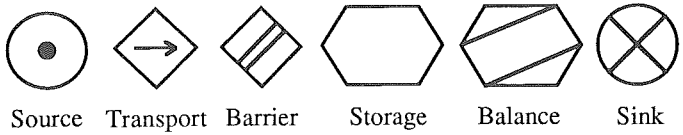


Figure 4.21 The basic flow function symbols.

4.9.4 A Heat Exchanger Example

In order to give a general idea of the expressive power, (and the limitations), of MFM, we will show a simple model. The target process used is, more or less, a small part of the Steritherm process, see Fig. 4.22.

4.9.5 An MFM Model of the Heat Exchanger

The MFM technique enables a graphical description of the physical structure of the heat exchanger process with its components, and of the material and energy flows in the process, in this case the water and product flows and the thermal energy flow from hot water to product. The resulting model is shown in Fig. 4.23.

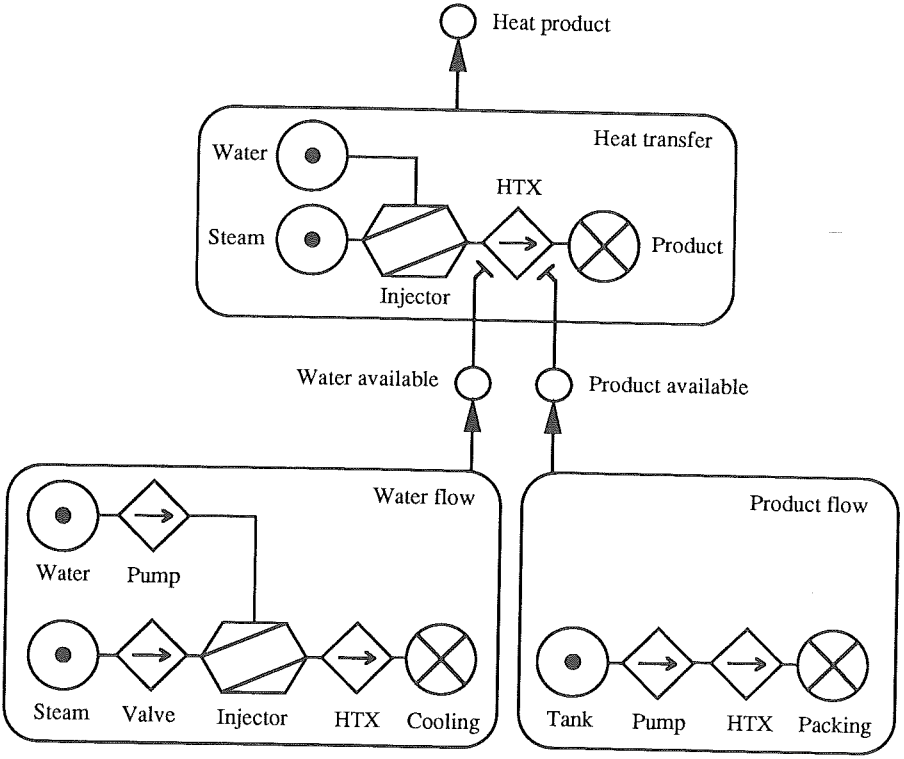


Figure 4.23 An MFM model of the heat exchanger system.

4.9.6 Comments on the Model

As an MFM model is used to describe how a process is functioning, the modeling decisions to a large extent depend on the view of the designer. This is even more the case than for other types of models, e.g., mathematical models such as differential equations and transfer functions.

- The level of detail may be arbitrarily chosen. In the above model, the piping has been completely ignored, while the steam valve is treated as a separate function. The packing is modeled as a single sink function. This reflects

the belief that the valve is more important than the piping and the packing machine. Such considerations will, of course, have a large impact on the level of detail of different parts of the process.

- Only parts of a process may be important. In the actual model, the control system for the steam valve has been completely ignored.

- The model obtained will depend on its assumed use. The demands for functional description, error diagnosis, measurement validation, and planning are quite different and may lead to the development of very different MFM models.

4.9.7 Uses for MFM Models

Basically, MFM models are nothing more than a description of structural and functional relations between objects, normally expressed in one of two graphical languages. But this representation may be used for several different tasks.

- *Functional modeling.* In order to describe a process in a systematic way, MFM can serve as a tool for documentation. This is very close to the original function of the MFM graphical languages.

- *Error diagnosis.* The classical use of knowledge-based systems in process control is to aid the operator of the process or plant to diagnose and remedy errors. This also was the original motivation for the development of MFM techniques. Here, the functional dependencies are explicitly represented, so when a certain control goal fails, i.e., an error occurs, the MFM model will provide information on which functions may be in error, and thus, in which component sub-systems the reasons for the failure can be found.

- *Measurement validation.* If all measurements are propagated into the net of flow functions, any inconsistent values of mass or energy flows can easily be found. Through back-propagation of consistent information, a subset of singularly inconsistent measurement points can be found.

- *Planning.* When the operator is planning different operations, he may use the MFM models to find out which goals depend on the function he plans to change or delete. If these goals may not be violated, something has to be done before the proposed action is performed. In this way, MFM may be useful in planning future control and reconfiguration actions.

- *Presentation.* As an alternative to process schematics that only shows the components of the process and their interconnections, the functional MFM language can be used to present process information to the operator with an emphasize on the process' goals and function.

Above, the most obvious uses for MFM modeling are described. These tasks have explicitly been mentioned in the development of the flow modeling technique. Of course, there may be other tasks that MFM can also handle well.

4.9.8 Diagnostic strategies

A MFM model gives a formal and explicit representation of the functional structure of a process. Thus, the relations between violated goals, functions no longer provided, and failed physical components are already represented in the data structure. Several diagnostic strategies may be defined, that work on this structure of relations. We will give two examples.

- Once a certain flow function fails, (a primary failure), the functions further down the flow will also fail, (secondary failures), as will the goals depending on that part of the function network. If the steam valve in the heat exchanger should get stuck, the heat exchanger water transport would also fail, as would the heat exchanger energy transport and, thus, the main goal of the model. This is an example of a *consequence propagation* strategy.

- If a certain goal is violated, it is possible to follow the achieve relations to the supporting function network, and check whether any of the functions in it have failed. If so, an explanation of the fault has been found. Also, the reason for a function's failure may be further investigated via any condition for its operation, i.e., by following the condition's relation to a subgoal and diagnosing that subgoal further. This is an example of the more or less standard *causal explanations* diagnosis scheme, as used in backward chaining diagnosis systems. Within the MFM framework, however, the search is governed by an explicit structure and not hidden in a maze of interacting rules.

4.9.9 Unclear Areas

However well developed MFM may seem, several questions arise after a thorough reading of the literature. Once one tries to build a few models, even more trouble comes up.

- The early MFM papers mention two kinds of goals, control goals and configuration goals. The control goals mean that certain constraints should be met, e.g., that a specified level of water in a tank should be held. The configuration goals mean that a certain function must be taken care of, but this may be done by one of several sub-systems. However, the problem of configuration and reconfiguration has largely been skipped during the latter stages of MFM development. Here, basic theoretical and practical contributions can probably be made.

- It is not uncommon that a goal may be achieved by a combination of functions. For example, it is possible that there is a decision tree structure among the conditions, i.e., a combination of *and* and *or* alternatives. The current MFM definition does not allow for this, and an extension is certainly needed.

- The MFM techniques do not demand any specified implementation, but obviously lends itself to graphical descriptions on a computer workstation. The whole problem of implementation and incorporation in an operator support tool must be looked into. In fact, the question of how the functional models shall be presented to the user is probably the most crucial question of all.

- The functional and topological structures of a process will usually be quite different. Thus, there exists a clear correspondence between flow functions and networks on one hand and physical components and subsystems on the other, only in certain cases. Generally, the realize relations are quite complicated and no one-to-one correspondences exist. This makes it necessary to provide the user with good facilities for navigating between the different functional and topological descriptions of the process.

4.10 TEXT AND PICTURES

Textual information, photographs, and drawings are important sources of knowledge that it must be possible to represent. In the Steritherm case the documentation consist of several volumes of information that it would be useful to have on-line access to. Some example are the Dairy Handbook, the Heat Exchanger Handbook, the Steritherm Instruction Manual, component data sheets, etc. Picture information considering Steritherm may consist of geographically correct drawing of the process, photos of the process and its components, mechanical assembly drawings, etc.

4.10.1 Hypermedia

Hypermedia techniques provide a possibility to include text and pictures in KBCSs. Hypermedia is a way of linking information using associative links. A hypermedia system can be likened to a database system with a graphical user interface. But instead of the typical database record and file structures, the information is freely structured and connected with a network of associative links. Nor is the data limited to text and figures, but can be graphics, photographs, sound, or video.

Hypermedia has shown itself to be an ideal method of accessing large amounts of information and hypermedia technology is developing hand-in-hand with optical disc technology to give cheap and easy to use information and documentation

systems. Documentation can be produced in hypermedia format from scratch or transferred to optical disc with document scanners. The documentation can then be easily integrated into a process control system with an advanced hypermedia user interface to help the operator to find relevant information.

Other words for hypermedia are hypertext and multimedia, although these are not strictly the same thing. Hypertext is for text only. Multimedia does not necessarily have associative links.

Good examples of hypermedia based on-line manuals with excellent browsing facilities are Symbolic's Document Examiner (text only), and DEC's CDA.

In an object-oriented setting a hypermedia unit (e.g., a HyperCard card) containing a text, a photo, or a drawing can be seen as an object.

4.11 CONCLUSIONS

Several commonly used knowledge and information representation formalisms have been described. It has been shown that most of the formalism can in some way be described in terms of objects and thus fit well into an object-oriented representation.

There is a strong interplay between the knowledge representation formalism chosen and what the knowledge should be used for. There is also an overlap among different formalisms in the sense that more than one formalism can be used to implement the same functionality. For example, a certain task can sometimes be solved both with rules and using procedures. Which formalism is used may depend on the personal preferences of the designer, performance considerations, etc. The KBCS language should provide the necessary flexibility and be rich enough to allow this.

5

The Main Knowledge Base

5.1 INTRODUCTION

The goal of this chapter is to describe the main knowledge base of the system concept. The chapter will concentrate on process knowledge structuring and the nature of the knowledge base language with a special focus on the class – object structure. The structuring of process knowledge will be exemplified on the Steritherm process. The class – object structure and the knowledge base language will be described in a tentative, discussive fashion. The description will focus on the most important design considerations and discuss the implications of different alternatives. Several references to and comparisons with G2, the current state-of-the-art in real-time expert system tools, will be given when the knowledge base language is discussed. The reader who is not familiar with G2 is referred to Chapter 8 for a thorough description.

5.1.1 Axioms

In order for the main knowledge base to be practically useful it must fulfill a set of basic requirements or axioms.

- It should be possible to store all necessary knowledge about the process in the knowledge base.

- The knowledge base should support different knowledge representation formalisms.
- Knowledge concerning one concept or object must be kept as a separate, localized unit in the knowledge base.
- Knowledge should not be duplicated.
- The knowledge base should allow a separation between knowledge that applies to classes of objects and knowledge that is unique for one object.
- The knowledge base should allow multiple, coexistent, hierarchical representation forms.
- The knowledge base should be flexible and support modifications.

As shown in the previous chapter, objects provide a flexible way of representing knowledge. Objects also encompass other representation forms such as rules, equations, procedures, etc. Therefore the main knowledge base uses objects as its main representational form. The requirement that knowledge concerning one object should be stored as a separate, local unit enforces modularization and makes it easier to avoid redundant knowledge. The separation between class and instance objects makes it possible to store all information needed about a concept in a class library from which instances can be created.

5.1.2 The contents of the knowledge base

The knowledge-base contains a variety of knowledge and information about the process and the control system. It may consist of objects, rules, equations, procedures, Grafcets, MFM models, photographs, drawings, text manuals, data sheets, etc., according to the description in Chapter 4.

The knowledge can be structured according to several criteria. According to one criterion the knowledge can be structured into

- application independent knowledge, and
- application dependent knowledge.

By application independent knowledge is meant the basic, general knowledge about the process and the control system that is needed for the normal operation of the process. It includes information about the process and its components and how they are interconnected; the control logic of the process (including continuous controllers, combinatorial logic, and sequential logic), drawings and photographs of the plant, etc. The application dependent part includes knowledge that is specific to some special application such as alarm-analysis, diagnosis, planning, simulation, etc. The application dependent knowledge contains both parts which

are process specific and parts that are generic. In some cases and for some applications, as discussed in Chapter 3, the generic parts are contained in the tools and in some cases they are stored in the knowledge base.

As an example consider the case of diagnosis according to the Diagnostic Model Processor method described in Chapter 4.5. The method is based on application dependent knowledge that consists of one process specific and one generic part. The process specific part is the network of interconnected model equations and fault assumptions. The generic part is Equations 4.2 – 4.4 that compute the satisfaction values of the model equations and the failure likelihoods of the fault assumptions. These equations make no assumptions at all about the nature of the model equations or the fault assumptions and, thus, apply to any type of process. It is natural that these equations are stored as a part of the knowledge base

The distinction between application independent and dependent knowledge is not always clear. One might argue that it is strange to include the control logic in the application independent part. After all, the control can also be seen as an application. The difference is that the control is necessary for the operation of the process.

Another criterion for partitioning the knowledge base is how specific the knowledge is to a certain process. Using this partition we get

- process generic knowledge, and
- process specific knowledge.

Process generic knowledge includes everything that is not unique for a specific process, for example methods for implementing specific control system functionalities such as different diagnosis and planning methods, component information and general product information.

Process specific knowledge is specific for a certain process. It encompasses process data, drawings, performance, components, process configuration, logic, etc.

The partitioning between specific and generic knowledge can be further extended. For example, also within knowledge that is specific to a certain process it is sometimes possible to structure this into generic and specific parts. In this case we get

- object specific knowledge, and
- object generic knowledge.

The specific knowledge contains things unique to an individual object. The knowledge may consist of object attributes, rules, procedures, simulation equations, photographs, drawings, etc. Knowledge of this kind is stored in connection with the object to which it pertain. Knowledge that is common to several objects

is described in connection with the class definition for these objects. The classes make up a library from which the designers can select appropriate objects.

5.1.3 Graphical knowledge base interfaces

The common knowledge base has two major types of graphical interfaces. These are

- the browser, and
- the user interfaces.

The browser

The interactive browser is available to all users and provides a means for them to navigate in, and inspect, the knowledge base. Through the browser, the graphical representation of the knowledge base language is seen. Objects are represented by graphical icons. Relations among objects also have a graphical representation. The browser should have built in operations for scrolling, panning, zooming, viewing an object in another context, etc.

A key element in the concept is that the contents of the knowledge base should be available not only to the designers but to all user groups. Operators, process engineers, etc., should have the possibility to go beyond their pre-defined user interfaces and browse through the contents of the knowledge-base. The browser constitutes a base interface that is the same for all users.

The user interfaces

The user interfaces constitute the users' individual interfaces to the KBCS. Different user groups need different information from the knowledge base. Different user groups also may need to access essentially the same information from the knowledge base but want it presented in different ways, e.g., on different abstraction levels. Finally, individuals within the same user group have personal preferences considering the way information is presented to them. This may include the look of the icons, colours, presentation form for numerical values (e.g., readouts, bar graphs, trend curves, meters, etc.), picture layout, the actions of the interaction objects, etc.

Descriptions of what parts of the knowledge base that should be presented for the different users, and how they should be presented are included in the common knowledge base. The descriptions can be seen as set-up files that have parameters that describe the interfaces.

The definitions of the user interfaces are a part of the common knowledge base. They include information of

Knowledge representation and presentation

The current trend in control systems such as Sattline, or real-time expert system shells such as G2 is that the representation of knowledge is integrated with the presentation of the knowledge. In Sattline objects are actually presentation objects that combine the attributes and behaviour of the object with how the object is presented for the operator. The G2 system has the same philosophy.

The integration of representation and presentation has many advantages. The object becomes a module that includes its own presentation. However, the requirement that knowledge about an object should be stored only in one place and that it should be available to several different users makes it necessary to separate the representation of the knowledge from the presentation. The user interfaces are defined independently from the representation of the knowledge that they present. The only exception from this separation is the browser. The graphical presentation of the knowledge base contents that is seen through the browser is the same for all users. It is also defined together with the knowledge, i.e., it is integrated with the representation.

5.1.4 Knowledge base distribution

The common knowledge base is not physically implemented in one place. As discussed in Chapter 3, e.g., real-time data can be stored on the local processing units where it is measured or calculated. The distribution that takes place are of two main types:

- distribution of real-time data, and
- distribution of the executable parts of the knowledge base language.

Distribution of real-time data is what takes place when certain parts of the common knowledge base are localized in local databases. This applies to real-time data, i.e., variables in the knowledge base whose values are computed and updated in, e.g., a local processing unit.

By distribution of executable code is meant the procedures, equations, rules, etc., that are extracted from the main knowledge base, converted into executable code, and distributed onto the processing units where they should actually execute.

Apart from these types, distribution also occurs in a few other cases. As discussed in Chapter 3, the main knowledge may itself be distributed. It is also possible that the user interface parameters are stored on the processing units where the user interfaces reside.

The knowledge distribution is defined in the main knowledge base. The distribution is determined by the designers. For example, PID controller object has an attribute that decides in which processing unit it should execute. Objects representing variables are physically in the local database of the processing unit where

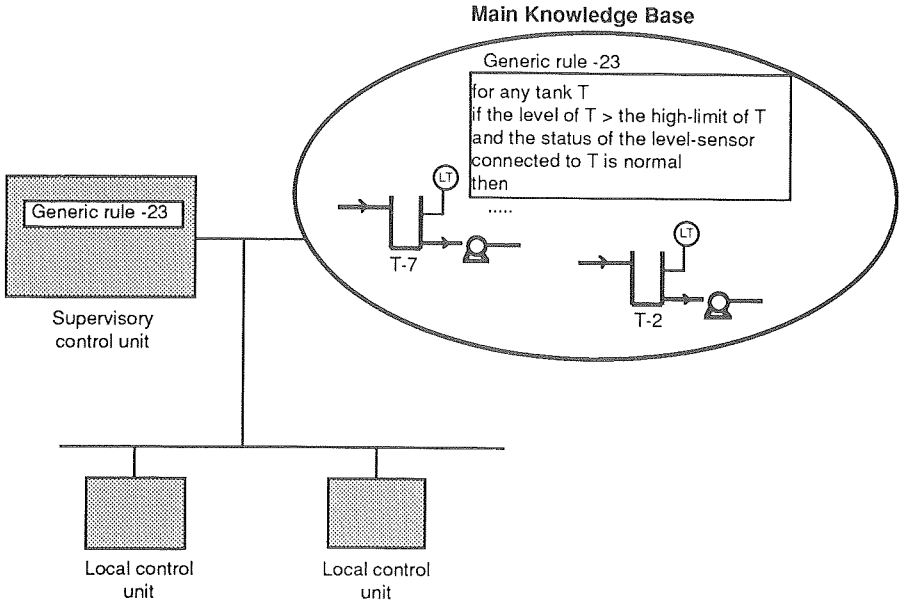


Figure 5.1 Distribution of generic rules: Alternative 1

they are measured or calculated. Rules have an attribute that determines where they should execute. For generic rules the situation becomes more complicated. Consider the following example of a rule.

```
for any tank T
if the level of T > the high-limit of T
and the status of the level-sensor connected
    to T is normal
then
  ....
```

This generic rule implicitly corresponds to a set of specific rules, each matching one occurrence of a tank with a connected level-sensor. By specifying a processing unit for the generic rule all the specific equivalents will be executed there. This is, however, not always desirable. One might want to spread out the specific rules on local processing units. In that case, this has to be done automatically by the realization tools based on the process schematic and the processing unit attributes in the tank and level-sensor objects on this schematic. The different possibilities are shown in Fig. 5.1 and in Fig. 5.2.

It is not always the case that a single object representing, e.g., a process component, resides completely in one local database. The object attributes might be spread out on different processing units. Consider the previous example. The level attribute of the tank object should be calculated from a level-sensor

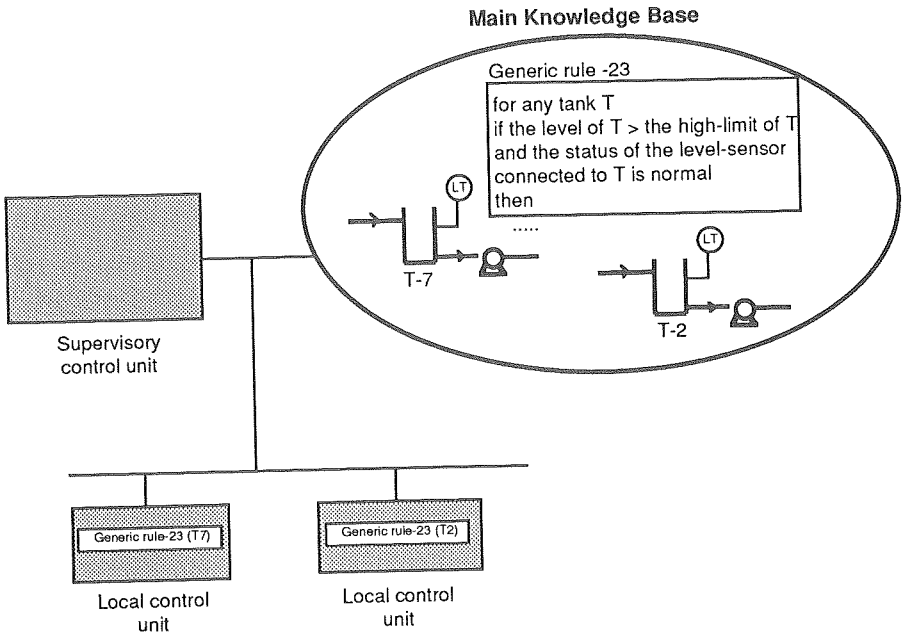


Figure 5.2 Distribution of generic rules: Alternative 2

value that is measured on a local processing unit. Hence, it is natural that this attribute resides locally. It is, however, also possible that the tank has a fault-status attribute whose value is calculated by a set of diagnosis rules that reside on a supervisory processing unit. Hence, it is natural that this attribute resides in the latter processor. The situation is shown in Fig. 5.3.

5.2 PLANT KNOWLEDGE STRUCTURING

In order for the common knowledge base approach to be practically useful, it is important to find a basic structure in which various types of knowledge about the process naturally fit in. This basic structure should be general enough to fit different types of processes, e.g., paper & pulp processes, dairy processes, power plants, chemical processes, manufacturing processes, etc., which could be either continuous, batch-oriented, or discrete in nature. The basic structure should also be flexible enough to allow the process to simultaneously be described from different aspects or views and allow different knowledge representation formalisms within an object-oriented environment.

The structure described here consists of two major elements: systems and views.

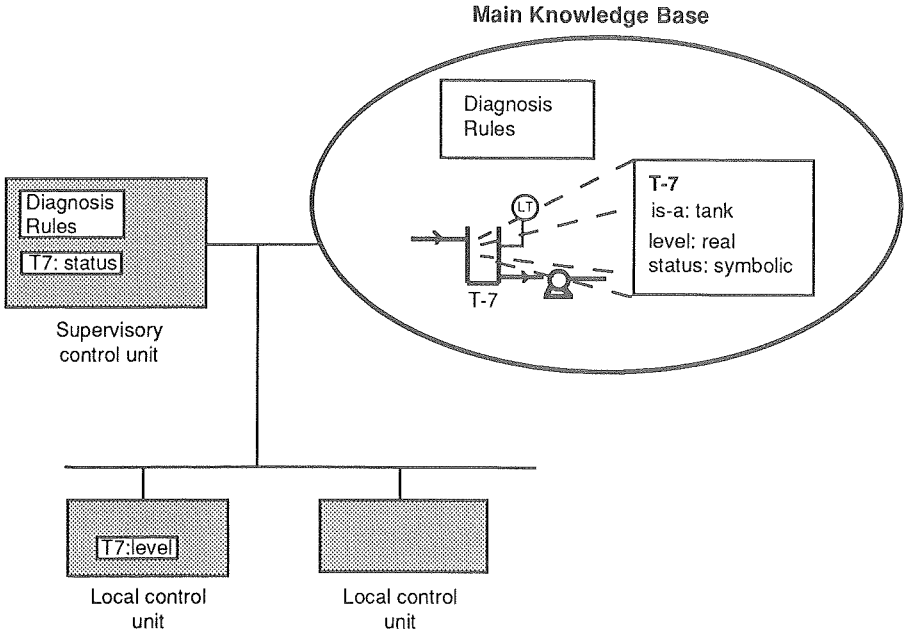


Figure 5.3 Distribution of a tank object

5.2.1 Systems

In general, a process can be decomposed into a set of systems. The systems correspond to the different flows of material, energy, or information in the process. The most important system is the main product system. In a pulp process this would correspond to the pulp system that describes how wood chips are converted to pulp, flowing through the process from the impregnation vessel, through the continuous digester, oxygen bleacher, diffuser bleacher, etc. In the Steritherm case the main product system corresponds to the product flow system.

The main product system is the raison d'être for the process. In order for this system to function properly a number of support systems must exist. Examples of support systems are the electrical system, the steam system, the pneumatic system, the hydraulic system, the control system, etc. In a pulp process the support systems would also include the white liquid system, the black liquid system, etc. In the Steritherm case the support systems are the warm water system, the cold water system, the steam system, the pneumatic system, the electrical system which consists of one 380 Volt system and one 220 Volt system, the control system, and the cleaning liquids system. This decomposition of the main knowledge base is shown in Fig. 5.4.

The reason for structuring process knowledge according to system is mainly that it reflects the situation of today in the process industry. Different user groups

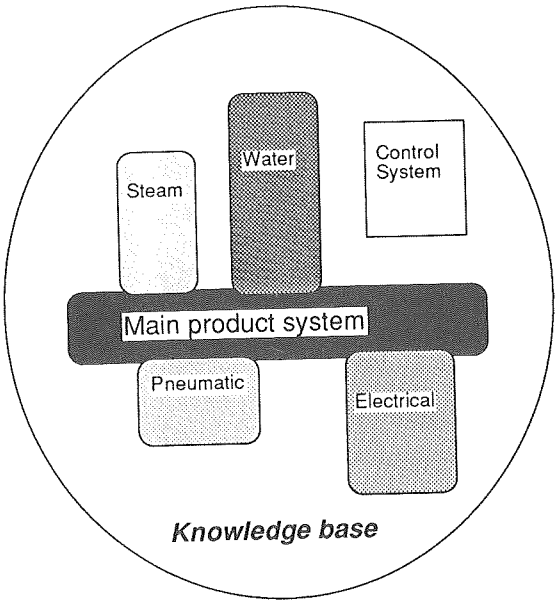


Figure 5.4 Steritherm systems

work with different systems in the process. For example, the main product system, the electrical systems, and the control system are the responsibility of the operators and process engineers, electricians, and the instrument engineers, respectively. Furthermore, different systems are documented separately.

A process component is often part of more than one system. A centrifugal pump may be a part of the main product system. However, the pump has a power supply and is therefore a part of the 380 Volt electrical system. The pump is also indirectly a member of the control system through the contactor that appears both in the 380 Volt system and in the control system. In the same way a simple pneumatic on-off valve is both a member of the main product system and the pneumatic system. It is also indirectly a member of the control system through the magnetic valve in the pneumatic system. A heat exchanger section that heats up the product using warm water belongs both to the main product system and to the warm water system.

A system may be hierarchically organized, i.e., it may be composed out of subparts which in turn are composed out of interconnected components.

5.2.2 Views

An object in the knowledge base such as the entire Steritherm object, a system in the Steritherm process, or a process component such as a centrifugal pump object can be described from different views. Describing an object from several views

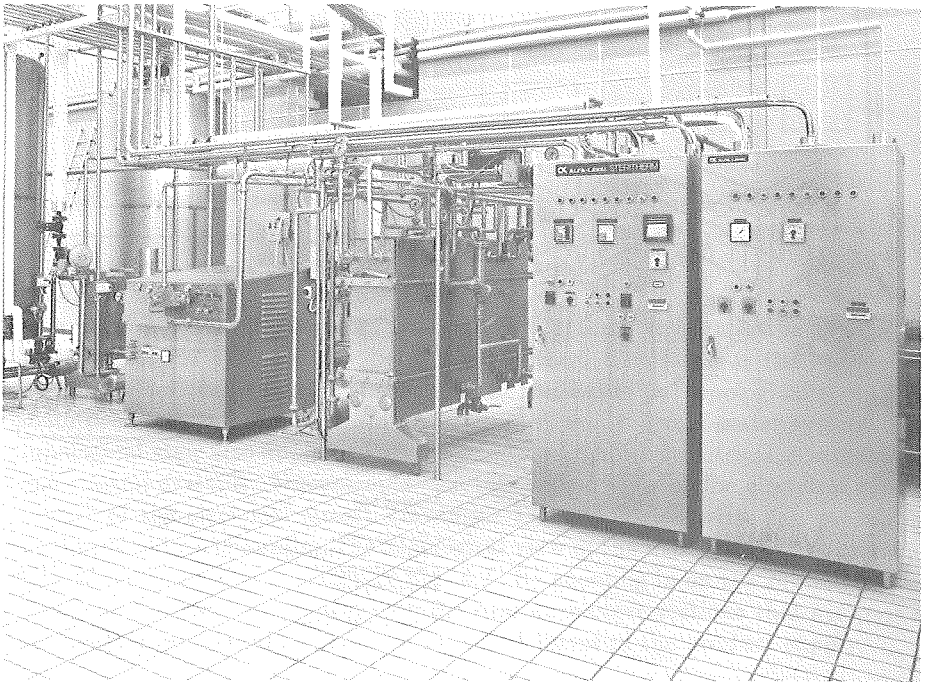


Figure 5.5 Geographical view described by a photograph

is a way of structuring the knowledge about an object into "natural" parts. The views in the knowledge base should not be confused with the user views defined by the user interface parameters.

Which views that are used to a large degree depend on the application. There are, however, a set of more general views that probably apply to processes in general, independently of the application. The standard views presented here can be seen as guidelines for how it might be wise to structure the process. The standard views are

- the geographical view,
- the topological view, and
- the functional view.

One should have in mind that these views are not always applicable, and that others exist.

The geographical view: The geographical view describes a physical object in a geographically and isometrically correct way. The description could be 3-dimensional or 2-dimensional. The description may consist of coordinate, size,

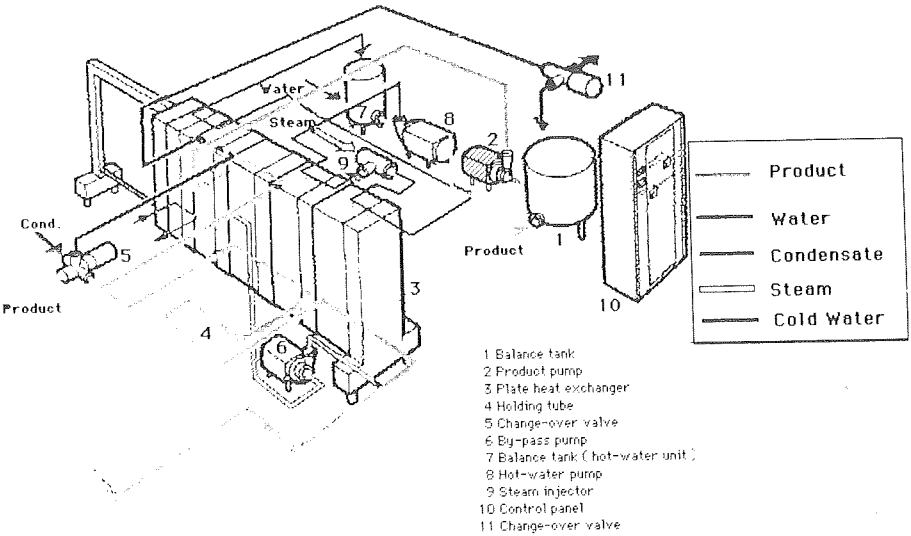


Figure 5.6 Geographical view of the main product system

and shape information that is the basis for automatically generated drawings; photographs possibly taken from different angles and with different degree of detail; various types of drawings about the process such as mechanical assembly drawings, etc. The geographical view has an internal structure. By selecting a part of, e.g., a drawing or a photograph a more detailed description of this subpart is shown. The geographical view associates the objects representing various parts of the process with their physical reality. A photographical description of the geographical view of the Steritherm process might look like Fig. 5.5. A geographical view of the main product system in terms of a 3-D drawing is shown in Fig. 5.6.

The topological view: The topological view contains the internal structure of a physical object, e.g., a system. Unlike the geographical view, the topological view has no geographical resemblance to the real process. The layout and contents of the topological view reflect in a sense the logical structure of the process. The topological view is closest to the way information is structured in today's control systems. The topological view is probably also the most important and general view needed.

The topological view of the entire Steritherm process consists of its different systems. The topological view of the main product system contains the balance

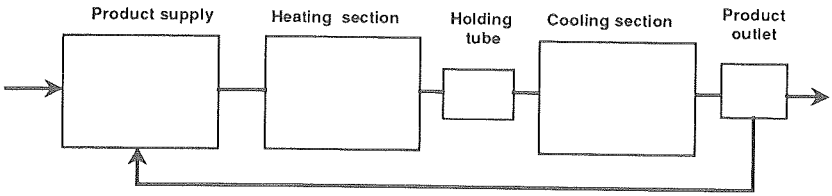


Figure 5.7 The topological view of the main product system

tank, the feed pumps, the heat exchangers, etc. The topological view is closest to a normal process schematic, although in a process schematic the topological views of several systems may be shown together. The latter is the case in the Steritherm process where the flow schematic consists of the topological view of the main product system, the warm water system, and the cold water system. The topological view also applies to other systems such as the electrical, pneumatical, and control systems. For example, the topological view of the electrical system consist of the electrical circuit drawings. The topological view can usually be decomposed into subunits. The topological view of the main product system at a high abstraction level is shown in Fig. 5.7. By zooming in on the subparts, more details in the view are shown, e.g., the heat exchanger sections that are part of the heating section, or the balance tank and valves of the product supply.

The functional view: The functional view describes the process in term of the goals that it should fulfill, the functions that are needed in order to fulfill these goals, and the process components that realize these functions. As indicated in Chapter 4, alternative formalisms for representing goal – function structures exist. Which one that is used is probably not so important. In the project we have chosen the MFM formalism described in Chapter 4.9 to represent the functional view. In the Steritherm process it is natural to have a functional view of the entire Steritherm process as well as functional views of some of its systems such as the main product system, the electrical system, the water system, etc. As in the case of the other views, the functional view has an internal structure. The functional views of the different Steritherm systems typically are subunits in the functional view of the entire Steritherm process. The functional view of the entire Steritherm process is described in Chapter 7.4.7.

A simple example

As an example, consider a simple electrical circuit consisting of one battery and one electric bulb. This can be seen as an object representing an electrical system that have all of the three described views. The geographical, topological, and functional views of the system are shown in Fig. 5.8.

The battery and the electric bulb are both objects. Each of them take part in more than one view. In the different views the objects have different iconic representations, are connected to different objects, and have different attributes.

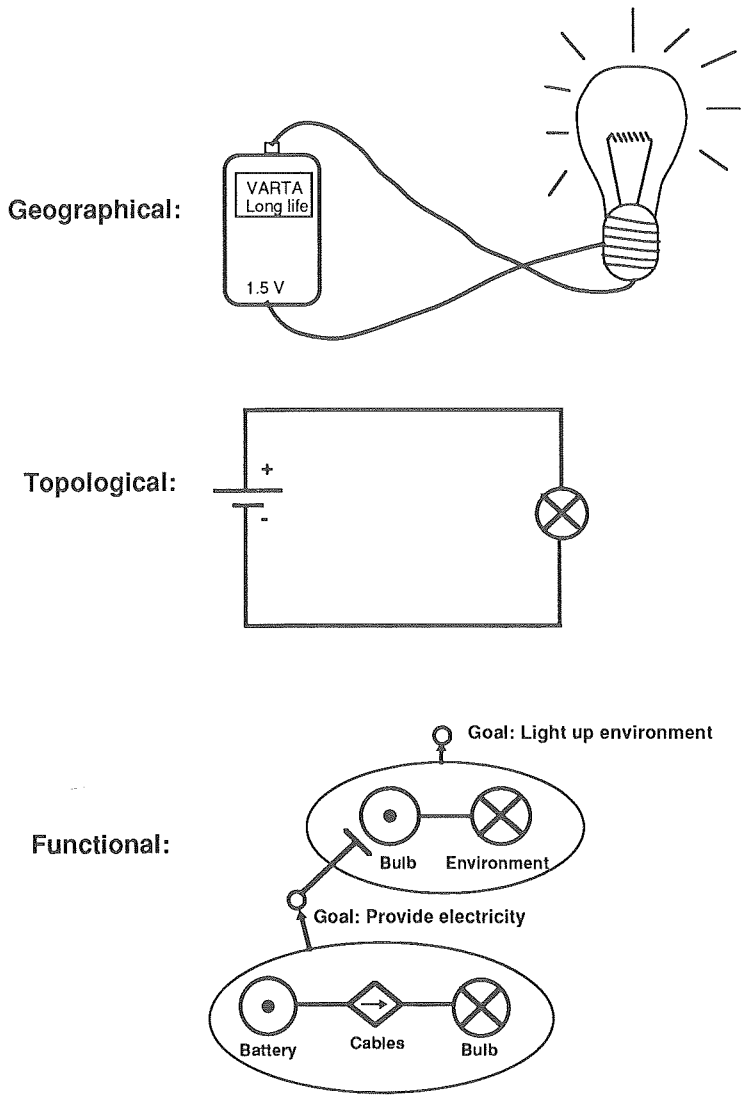


Figure 5.8 Different views of a simple electrical system

The situation for the battery object is shown in Fig. 5.9. An object can be seen as the sum of all its different views according to Fig. 5.10.

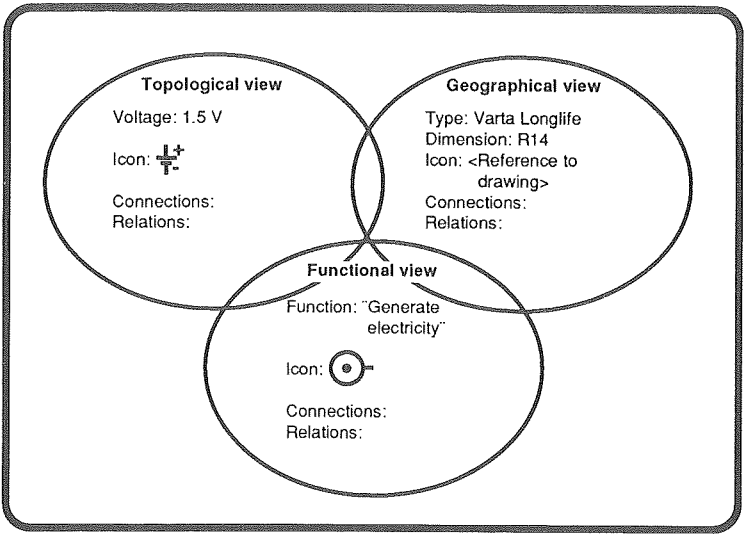


Figure 5.9 Battery views

5.2.3 Steritherm examples

The Steritherm process can be decomposed as described in Fig. 5.11. The topological view of the Steritherm process in a composite object that consists of the different systems, i.e., main product system, warm water system, electrical systems, steam system, pneumatic system, control system, etc. Each of the systems can be described according to one or several of the standard views. The possible views are shown in Table 5.1.

The functional views of the Steritherm systems are parts of the functional view of the entire Steritherm process. The control system of the Steritherm process can be described from several points of view. The topological view of the control system shows how the knowledge-based control system is decomposed into processing units, operator stations, knowledge bases etc. For each processing unit the topological view shows the inputs and outputs of that unit and what they are connected to, e.g., contactors, magnetic valves, circuit breakers, etc. The topological view of the processing unit itself contains the function blocks, PLC code, and rules that execute in that processing unit. The topological view of the control system in various degrees of detail could look like Fig. 5.12.

The control logic such as PID controller, PLC code, rules, etc., is described in the control system. It is, however, also natural to represent it in the topological views of the different systems. For continuous controllers it is also natural to have a special control loop view. The situation is shown in Fig. 5.13.

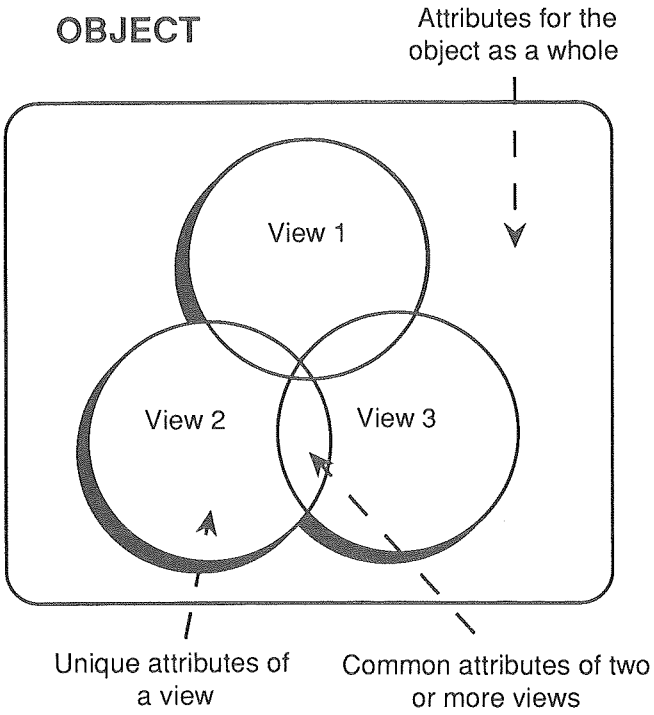


Figure 5.10 An object and its views

5.2.4 Other systems and views

The described systems and views are not enough to describe all the knowledge that we are interested in. Some structures that have no natural belonging are fault trees, influence digraphs, graphical representations of various production plans, maintenance planning, production, economy, etc. In order to handle these, new views and/or systems have to be created. If the new structure naturally apply to the existing systems, a new view can be added that perhaps only exist for one system. If the existing systems are inadequate, a new system may be created that can be described by already existing views or by newly defined views. If a new system is created, it is not necessarily so that this system has a physical correspondence, as is the case with the usual systems such as the main product system, the electrical systems, etc.

Fault and alarm trees can be seen as separate views that either describe the Steritherm as a whole or only exist for a certain system. Production plans can be described by a separate view for the Steritherm process as a whole. The plans in this view may be a subpart of the planning view of the entire dairy. The extension of the views and systems is indicated in Table 5.2.

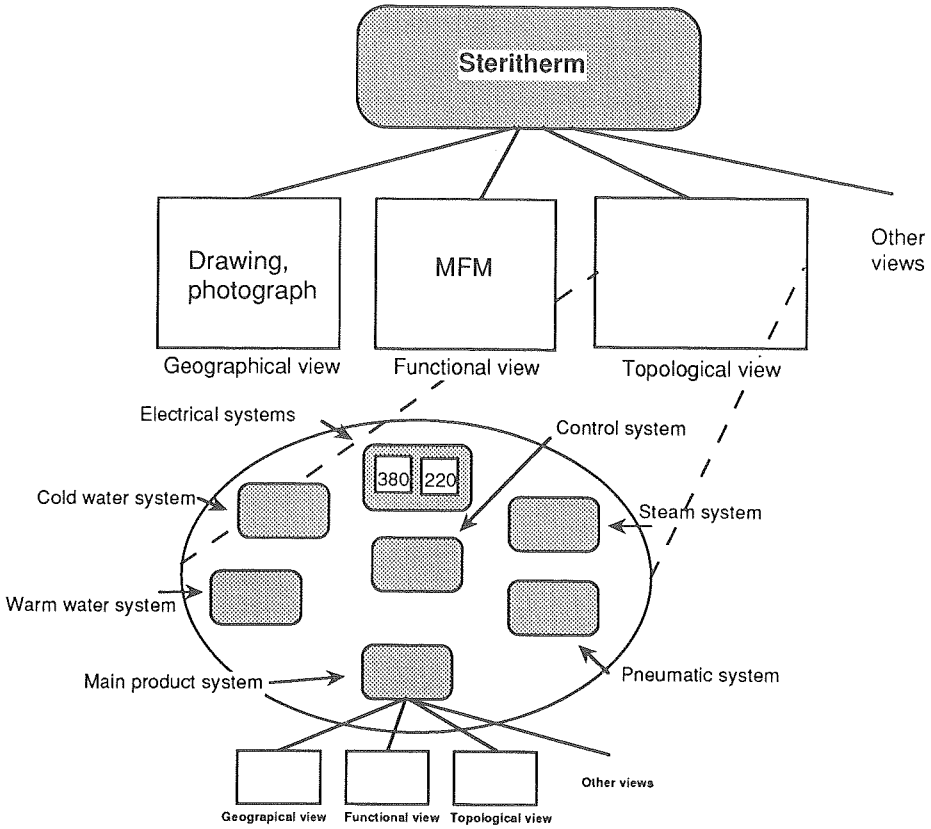


Figure 5.11 Steritherm views

VIEWS \ SYSTEMS	Main product	Warm water	Cold Water	Steam	Pneumatic	Electrical	Control
Geographical	X	X	X	X	X	X	X
Topological	X	X	X	X	X	X	X
Functional	X	X	X	X	X	X	(X)
Others							

X : Exists
(X) : Possibly exists

Table 5.1 Steritherm systems and their views

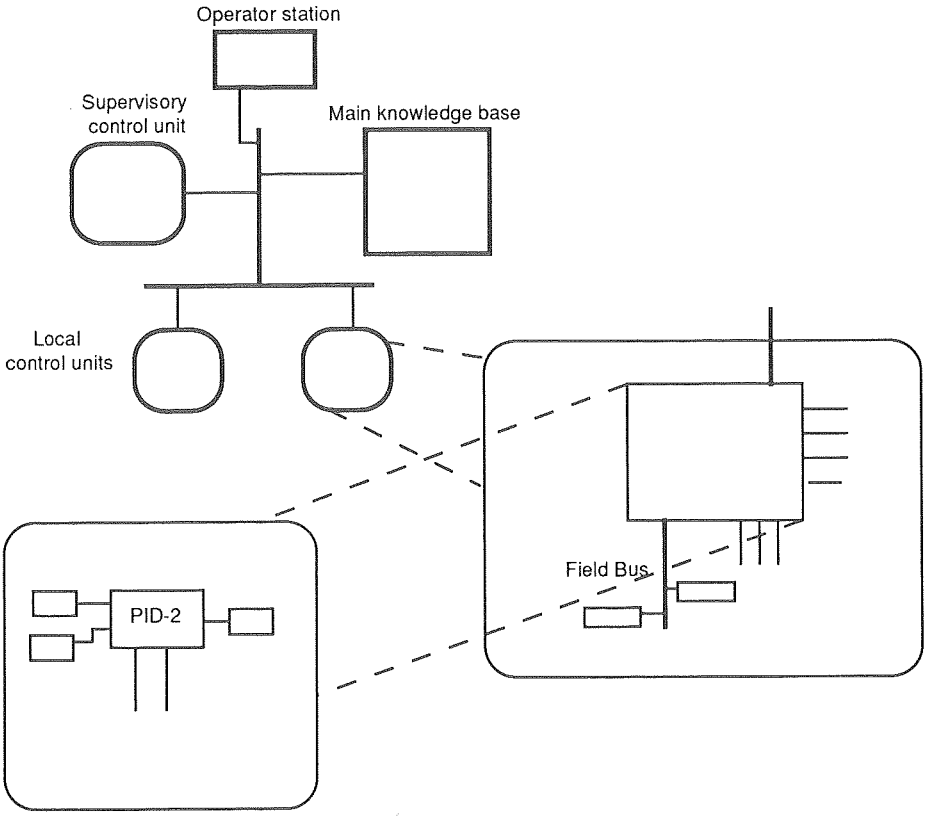


Figure 5.12 The topological view of the control system

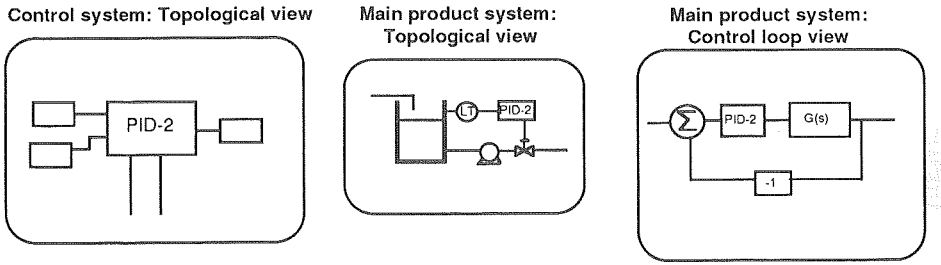


Figure 5.13 Control logics in views

5.2.5 User interface parameters

The systems and views together and the various types of knowledge associated with them make up the core of the common knowledge base. The user interface parameters determine how this knowledge base contents should be presented to

	SYSTEMS								
VIEWS	Main product	Warm water	Cold Water	Steam	Pneumatic	Electrical	Control	New system-1	New system-2
Geographical	X	X	X	X	X	X	X		
Topological	X	X	X	X	X	X	X	X	
Functional	X	X	X	X	X	X	(X)		
New view-1	X							X	
New view-2									X

Table 5.2 View and system extensions

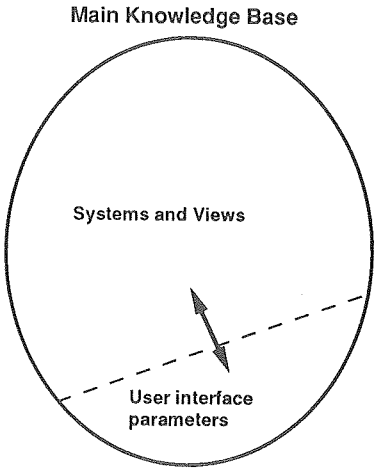


Figure 5.14 User interface parameters versus systems and views

the different users. This is indicated in Fig. 5.14.

The user interface parameters defines the different user interfaces in terms of how the contents of the main knowledge base should be presented. The complexity of the user interfaces ranges from very simple interfaces consisting of, e.g., a few dynamic data and some explanatory text shown on an alpha-numerical terminal to a full graphical interface with pan, scroll, and information zoom possibilities with multiple windows showing different information about the process.

In some cases the information in a user interface may have a direct mapping to the systems and views. For example, an electrician may want to see the electric circuit diagram which is represented by the topological view of the electrical system. In other cases a user wants to see several views in the same picture. This could be the case with the operator who perhaps would like to see a process

schematic of the Steritherm extended with some dynamic displays. The standard Steritherm schematic consists of the topological views of the main product system and the water system. In this case the user interface parameters should define that the information in these views should be combined together and presented to the operator.

It is also possible that a user wants information from different views in different pictures or windows and that different user wants information from the same system presented in different ways, e.g., in different level of detail. A production engineer is perhaps only interested in the status of the different substeps of the process in his process schematic, whereas the operator wants information about every process component. In this case the users want information about the same view, the topological view, but they want it presented at different hierarchical levels.

Furthermore, the user interface parameters should make it possible to specify how dynamic information should be presented, i.e., as dynamic readout displays, bar graphs, meters, gauges, trend curves, using colour changes, by animation, by changing the amount of information presented, etc. It should also be possible to specify that the presentation of the views and systems in the individual user interfaces should be different from how they are presented in the browser. The user may have preferences on icon shape, size, colour, and layout. Perhaps he wants the zooming between different hierarchy levels to behave in a special way, e.g., to skip certain levels. The actions of the various interaction objects in the user interface should also be defined. Interaction objects may be touch or mouse sensitive areas or buttons on the screen, type-in fields where the user may enter numerical, symbolical, or textual information, etc. The actions of the interaction boxes may affect the user interface itself, e.g., determine what picture that is presented, or affect the contents of the common knowledge base, e.g., change certain parameters, execute a certain part of the knowledge base language, etc.

5.3 THE KNOWLEDGE BASE LANGUAGE

In this section the structure of the knowledge base language will be discussed. The existing language that today is closest to a real-time, object-oriented language that combines rules, procedures, and equations is the language of G2. Therefore, G2 is used as a staring point in the discussions. Several references to and comparisons with G2 will be given.

5.3.1 Object structures

The system and view structures described previously can be implemented in an object-oriented system that has support for multiple views and composite objects.

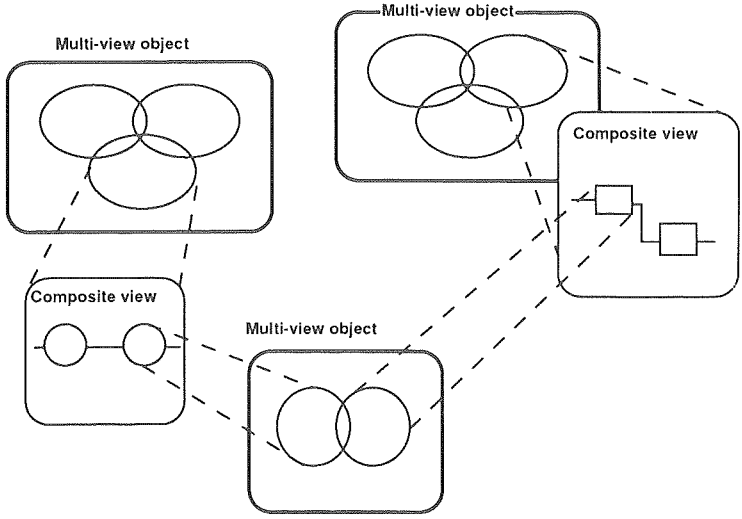


Figure 5.15 Composite views

The object instances can be of two different types: single-view objects and multi-view objects. The single-view objects include objects that are predefined in the language such as the basic data structures, rules, equations, procedures, etc., and user-defined objects.

Multi-view objects represent objects that show up in more than one view or context. In each view the multi-view object can be seen as an ordinary single-view object, i.e., the entire multi-view object is the sum of a set of single-view objects. Each single-view object within a multi-view object describes the multi-view object from a specific point of view. The single-view objects have unique attributes and attributes that are shared among several or all of the views of a multi-view object. Each single-view object of a multi-view object can have an associated icon and be connected to other objects. In that way, one view of a multi-view object can be a part of another composite object. In fact, each view of a multi-view object can be a part of a composite object. The situation is shown in Fig. 5.15.

Multi-view instances are instances of multi-view classes. A multi-view class definition describes a multi-view object according to

- the different objects that it should consist of,
- the attributes that should be common to all views in the object, and,
- if appropriate, the icon representation, attributes, and connections for the object as a whole.

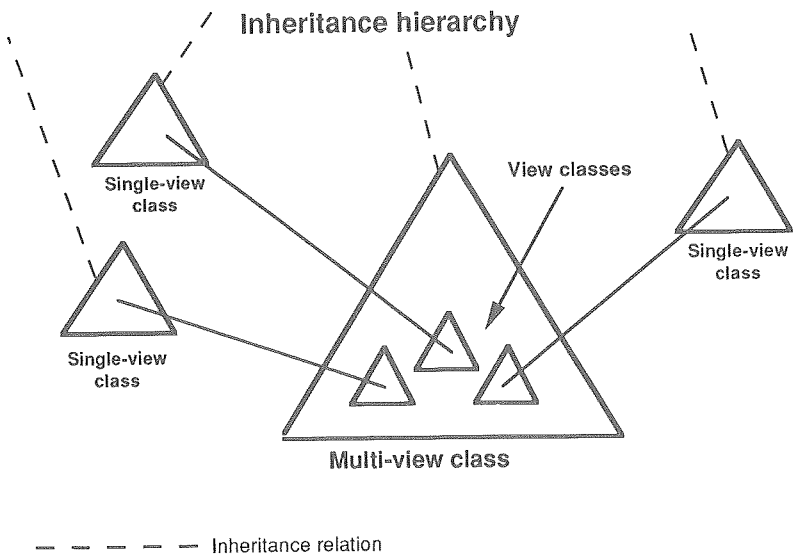


Figure 5.16 Multi-view class definitions

The views in a multi-view class definition are defined in terms of the classes that the views should be instances of. These, single-view, classes define the attributes, behaviour, icon, and internal structure of the different views in a multi-view object. The situation is shown in Fig. 5.16.

In the Steritherm example shown in Fig. 5.11, the entire Steritherm object is a multi-view object. The topological view of this object has an internal structure composed out of the different systems, i.e., the main product system, the warm water system, the electrical systems, etc. Each of these systems is a multi-view object, according to Fig. 5.17. The topological view of the main product system has an internal structure composed of interconnected process components such as the balance tank, pumps, heat exchanger sections, etc. These process components are single-view objects that show how the multi-view object representing the process component shows up in the topological view of main product system. This situation is shown in Fig. 5.17.

Inheritance

Both multi- and single-view classes inherit from their super classes. Whether multiple inheritance should be allowed or not is an open question. In a sense multiple inheritance is already used when a multi-view class is defined in terms of a set of single-view classes. However, one might also allow a single view in a multi-view object to inherit attributes and behavior from more than one super class. For most practical purposes it is probably sufficient to only allow single inheritance combined with multiple views.

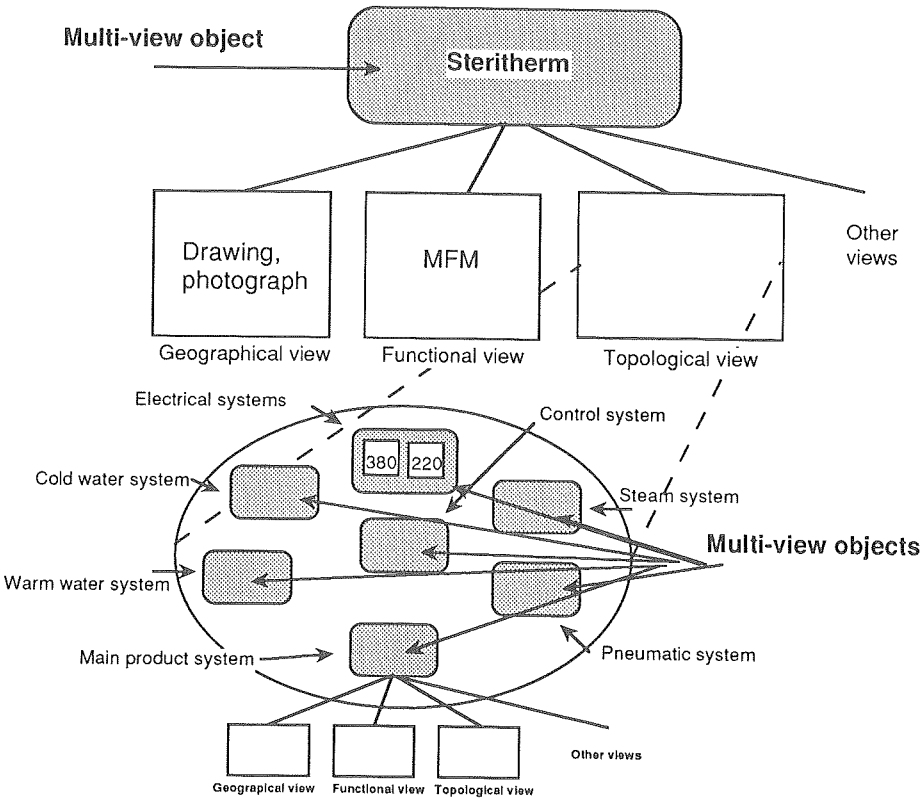


Figure 5.17 Multi- and single-view objects in the Steritherm

A multi-view class can inherit from another class. The specializations that can be made of a multi-view class only involve the attributes common to all views and the views the class should consist of, e.g., a multi-view subclass could add or delete attributes or views from its super class. One might think of the situation where the internal attributes of the different views also are affected by a specialization of a multi-view class. This functionality is probably not necessary. An exception from this are the attributes that should be shared between views and which not are shared by all views. The definition of which views an attribute should be visible in is included in the multi-view class definition.

Attribute inheritance: It is important to be able to control the nature of the inheritance that takes place from classes to subclasses and from classes to instances. For example, in G2 the class definition specifies what attributes an instance of that class should have. The attributes are the sum of the attributes that are specific to the class and the inherited attributes. Upon creation, an instance automatically receives its own set of attributes according to the class

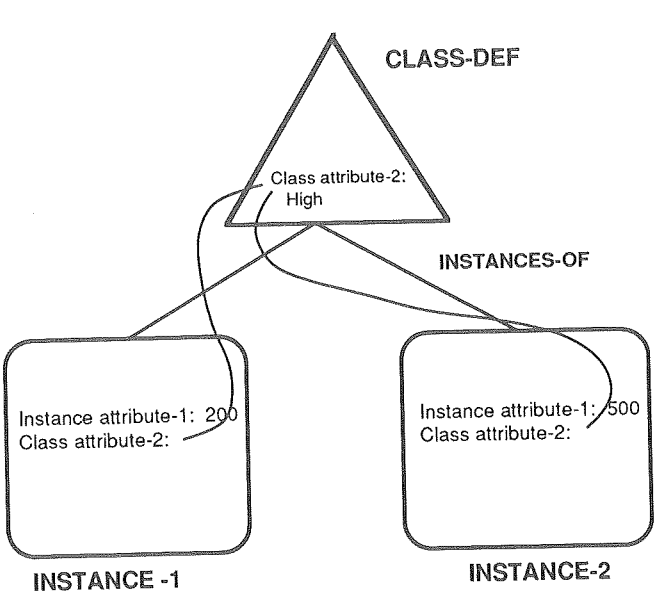


Figure 5.18 Class attributes and instance attributes

definition and with appropriate default values.

It is extremely important to also be able to specify that an attribute should be common to all instances of a class, i.e., be a class attribute. In that way information that is common to all objects of a class are only stored once, in the class definition. Examples of information that is common to many objects are installation and operation descriptions for a certain process component, fixed component data, etc. For these attributes, every instance contains references to its class definition according to Fig. 5.18.

Furthermore, it is important to be able to modify the exact nature of the inheritance of an individual attribute. One may want to specify that a class should inherit some but not all of the attributes of its superclass. Similarly, one may want to add an attribute to an instance that is not defined in its class definition.

Composite objects

The internal structure of a composite object is defined in its class definition. The "composed-of" relationship specifies that an object component should be of a specific class or of a subclass of that class. In the case of multi-view object components, the "composed-of" relationship also specifies the particular view of the corresponding multi-view object that the composite object should consist of. The definition also includes the local names for the object components and their interconnections.

Composite classes form a flexible way of specifying objects or functions which

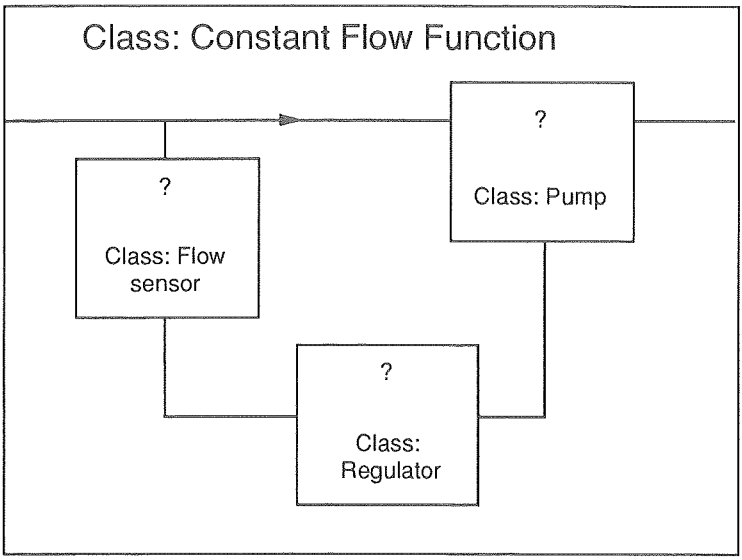


Figure 5.19 Composite class with empty "holes"

can be realized by replacing the class or parts of the class by a more specific class. For example, during design of an installation a composite class "constant flow function" can be defined and inserted into a structure, see Fig. 5.19. This class does not in itself define the particular implementation of the constant flow function and must, at a later stage in the design, be replaced by a composite class that does. The constant flow function may consist of several objects, for example a pump function, a sensor function, and a regulator function. Any one of these function classes may be replaced by a class derived from it (and so inherits the characteristics of the class). For example, the pump function class may be replaced by a class which specifies a particular pump, giving the manufacturer and model number. The completely specified "constant flow function" is shown in Fig. 5.20. This method allows changes later on in the design process. A particular implementation of a process function may be easily replaced by another implementation of the same function.

Predefined object classes

The objects in the knowledge base are either of a predefined, built-in class; of a class that belongs to a standardized, class library consisting of, e.g., standard process components, control blocks, etc.; or of a class specific to a certain process or application within the process.

The predefined classes are single-view classes that make up the basic primitive classes in the system. Examples are predefined classes for rules, equations, variables, etc.

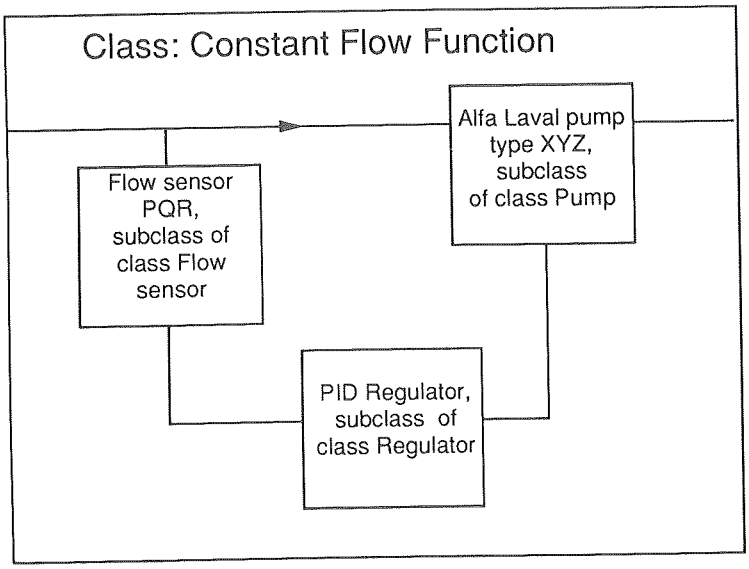


Figure 5.20 Completely specified constant flow function

G2 solution: In G2 everything is an item. Examples are rules, class definitions, objects, messages, equations, graphs, text boxes, etc. The items form a tree structure. G2 objects are part of this tree structure. The object part is the only part of the tree structure that the user has any control over, i.e., can specialize into subclasses, can reference in expressions, etc. Among the objects, a few are prespecified in the G2 language. These include *object* the root class in the inheritance tree, and classes for the different types of variables (quantitative, symbolical, logical, and text).

The separation between items and true, first-class objects in G2 is a problem. In the knowledge base language this separation will not exist. Everything is a first-class object and can be specialized and referenced to as appropriate.

The pre-defined classes will include

- rules,
- equations,
- procedures, and
- classes for the basic data structures in the language (which these are will be described later).

Static versus dynamic objects

The majority of the objects in the knowledge base will probably be static during run-time, i.e., the objects will be created during the design stage. It is, however, also necessary to be able to dynamically create and delete objects. An object representing an alarm may be created when an alarm has occurred. This object may have attributes representing the immediate cause of the alarm, the time of the alarm, whether the alarm has been acknowledged or not, etc. Similarly, objects representing operator actions may be created when an operator intervenes in the process.

Object location

The non-constant parts of an object may be distributed out to a local database during execution. This is specified for each variable. It should, however, also be possible to specify this at the object level, i.e., that the non-constant parts of the object should all reside on the same machine.

5.3.2 Connections and relations

Objects in the knowledge base have relations to other objects. Relations can represent a physical connection between objects such as a pipe between two flow objects or an electrical wire between two electrical components. Relations can also represent an abstract relation between two objects such as a cause-effect relation between a fault object and an alarm object, a dependence between a model equation and a fault assumption in the DMP formalism, or an influence between two nodes in an influence digraph. Relations can have a graphical representation. For physical connection relations, graphical representations are very natural. However, in other cases a graphical representation of the relation might not be relevant.

> **G2 solution:** In G2 a distinction is made between connections and relations. Connections always have a graphical representation and may represent physical as well as abstract relations. Relations have no graphical representation. Furthermore, connections are defined in terms of a connection class hierarchy where attributes can be associated with a connection class. Connections can be unidirectional or bidirectional. Connection type checking is performed to allow only connections of the same class to be connected together. Relations have no corresponding relation hierarchy and cannot have associated attributes. A relation can be specified as being one-to-one, one-to-many, many-to-one, or many-to-many. The inverse relation of a relation can be defined, as well as whether the relation should be symmetric (in which case the inverse relation is the same as the relation) or not. Both connections and relations can be used in expressions.

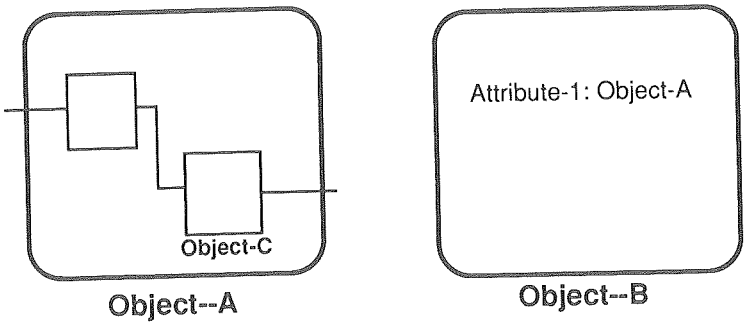


Figure 5.21 References between objects as relations

Connections are created manually by the designer by graphically interconnecting objects. Relations can only be created and deleted dynamically through an explicit rule action.

The distinction between connections and relations in G2 is unwise. The knowledge base language will only contain one concept: a relation. Relations are defined in a relation hierarchy and attributes may be associated with the relations. The relation definition specifies whether the relation should have a graphical representation or not, what the graphical representation should look like, what the name of the relation should be, the type of the relation (one-to-one, etc.), the name of the inverse relation, whether the relations may be created dynamically or not, whether the relation is symmetric or not, whether the relation is transitive, etc.

In a way similar to pre-defined object classes, the knowledge base language also contains a set of pre-defined relations. Some of these have already been described such as the "composed-of" relation, the "instance-of" relation, the "subclass-of" relation, and the "view-of" relation.

Relations is also the way in which references to an object will be handled. The multi-view object is an attempt to gather all knowledge about one object into a single unit. Even with this concept it is impossible to avoid the situation where an object is referenced from some other object. The simplest case where this occurs is when an object is a component of another composite object. The composite object must be able to reference its component objects as well as vice versa. It must also be possible for arbitrary objects to reference each other. One object could, e.g., be the value of an attribute in another object. This can be treated as a relation between the two objects. The situation is shown in Fig. 5.21.

In order to maintain the modularity of the knowledge base, it is important that

it contains tools that can show all the relations that an object has, and that inverse relations are established automatically. With this the designer has the possibility to check all the places where a certain object is used or referenced.

5.3.3 Basic data types

The basic data types needed in the system are

- real-valued variables,
- symbolic variables,
- logical variables, e.g, boolean variables,
- text variables,
- lists,
- vectors, and
- external data types.

The basic data types are available as pre-defined classes. Symbolic variables can be compared to the enumeration type of ordinary programming languages. Text variables contain text strings. Lists and vectors may be restricted to contain only objects of a certain class or subclass of that class. The external data types are used to represent objects that normally are outside the scope of the language. These may include pixel bit-maps to represent pictures, hypertext cards, programs written in external languages, etc. Associated with the external types should be information about how they could be referenced and manipulated.

The pre-defined classes contain a set of attributes that are common for all variables. These includes the current value of the variable, the specification of whether a history should be saved for the variable or not, the default update interval that determines how often a new value should be computed for the variable, the validity interval that determines how long the current value of the variable will remain valid (this will be further commented later), the data unit of the variable, the processing unit where the variable should reside, etc.

5.3.4 Logical systems

The underlying logical system that the knowledge base language is built upon is important. For boolean variables, the language can either be based on binary logic, three valued logic, or fuzzy logic. In binary logic the variable has either the value *true* or *false*. In three valued logic, the variable can also take the value *unknown*, or *none*. Fuzzy logic uses real-valued truth values between 0

and 1 where 1 means that the variable completely is described by a certain fuzzy statement such as `Very High` and 0 means that the variable is not at all described by the fuzzy statement.

G2 solution: In G2 the user can choose between different logical systems. This is determined for the individual variables by the value of the validity interval attribute. The validity interval can either be *indefinite* or specified by a time expression. If the validity interval is indefinite the variable will always have a valid current value. This is similar to what you have in an ordinary programming language. Variables with indefinite validity intervals are called parameters and form separate classes. For example, the possible values for a boolean logical parameter are *true* or *false*.

The possible values for a non-indefinite boolean logical variable are *true*, *false*, or *none*. Similarly, the possible values for a non-indefinite quantitative variable are any real number or *none*. The value is *none* when the last current value has expired. Any references to an expired variable will force G2 to try to establish a new value for the variable. This could be done by evaluating a formula for the variable, or by finding and invoking a rule which concludes a new value for the variable. Variables with non-indefinite validity intervals is an important way to solve the problem with non-monotonic reasoning and will be further discussed in that context.

In G2, fuzzy truth values are partly supported. Truth values can range from -1 to +1, where -1 indicates complete certainty that something is false and +1 indicates complete certainty that something is true. A fuzzy expression is defined by a logical "hysteresis" band such as

```
... tt-45 > 137 (+- 10) ...
```

The above expression defines a band of truth values that ranges from -1 to +1 as the value of tt-45 ranges from 127 to 147. In rules, conclusions assume the truth value of the antecedent provided that the truth value is higher than a specified truth threshold that lies between 0 and 1. When fuzzy truth values are used in a logical expression, the logical operators *and*, *or*, and *not* are equivalent to the minimum truth value of its operands, the maximum truth value of its operands, and -1 times the truth value of its operand. Using this fuzzy logic, simple, piece-wise linear fuzzy membership functions can be constructed as shown in Fig. 5.22.

In the knowledge base language all three logical approaches are needed. One can discuss if the selection between logical system should be done per variable basis as in G2 or on some higher level.

According to what is common in the fuzzy area, fuzzy truth values should range between 0 and +1 instead of between -1 and +1. In a complete fuzzy logic system

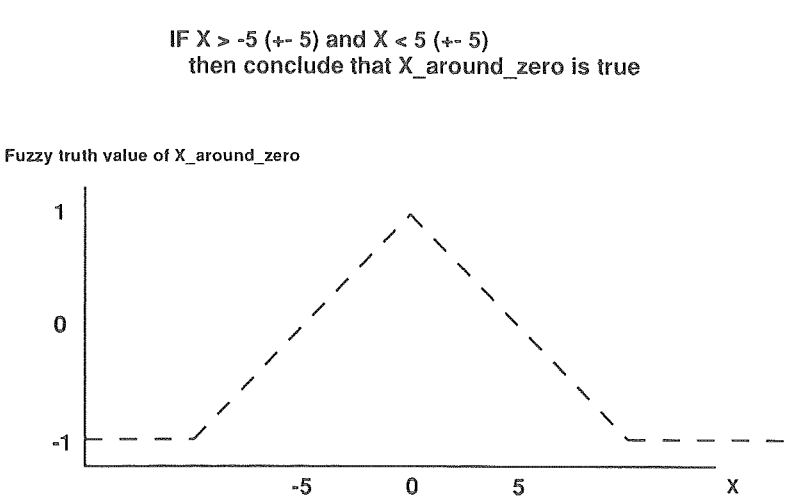


Figure 5.22 Fuzzy membership functions in G2

there must also be support for the fuzzy implication or fuzzy relationship `IF A THEN B`. This means that the rule system must be able to handle fuzzy conclusions in a correct way.

5.3.5 Rules

The internal nature of real-time inference engines can differ quite substantially. The nature of the inference engine also has a strong influence on what the rules look and, hence, on the knowledge base language. Among the commercial expert systems available that are used for real-time applications, two approaches dominate: the pattern-matching approach and the static link approach.

Pattern matching inference engines

Pattern-matching inference engines can be of two types: forward chaining production systems and Prolog type backward chaining systems. Among real-time systems the forward chaining solution is the usual. Examples of systems of this type are Chronos, Nemo, ART, and Muse, though in Muse a forward chaining production system can be used together with a Prolog style backward chaining system.

The antecedent part of a forward chaining production rule contains patterns that must match the contents of the database for the rule to be fulfilled. Variables in the patterns can match arbitrary symbols. When the same matching variable occurs in more than one place it must match against the same symbol in all occurrences. Matching variables can be used for several purposes. One example is for implementing generic rules that apply to all instances of some object class.

The reasoning is performed in a recognize-act cycle that has three phases. During the match phase, all rules that are fulfilled are collected into a conflict set together with the corresponding matching database elements. During the select phase, one rule is selected for execution according to some conflict resolution strategy. During the act phase, the right hand side of the selected rule is executed.

Conflict resolution strategies either select rules according to some pre-defined rule ordering or according to the state of the database. The most common of the latter bases the selection on the recency of the matched database elements. Rules matched by more recently added information are favoured.

Pattern matching is time consuming. Production systems usually use some incremental, network-based matching algorithm that saves information about partially matched rules. The most famous algorithm of this kind is the RETE algorithm (Forgy, 1982). High-speed pattern matching is a substantial research area within AI and several alternative algorithms, e.g., the TREAT algorithm, have been developed as well as parallel implementations of the algorithms.

Pattern-matching expert systems are event driven. The inference engine is activated when a new element is added to the database. This causes rules to be matched and selected for execution. During the execution, database elements are added, removed, or modified, causing new rules to be matched. In a process control system, the rule execution might be invoked by incoming process data, or by operator intervention.

Inference engines based on static links

Inferences engines that are based on static links do not perform any pattern matching. Instead, static links or references are maintained by the inference engine. The static links internally connect the variables and objects that the rules operate upon with the appropriate rules, and rules with other rules. During the inferencing, the inference engine follows the appropriate links to determine which rule should be tested next. The static link approach is used in G2. The static links are set up when an object or rule is created or modified.

> **G2 solution:** G2 maintains the following links. A variable has links to all the rules that mention this variable in their antecedents. Therefore, if the variable receives a new value, all these rules will be tested through forward chaining. Similarly, a variable has links to all the rules that may conclude a new value for the variable. Hence, if a value is needed for the variable those rules will be tested, through backward chaining. A class definition has links to all the instances of this class. Using these links, generic rules are substituted by a set of specific rules; one for each instance.

Inference engine selection

Comparing the efficiency of the different types of inference engines is difficult. The lack of pattern matching in the static link approach gives reason to believe that this approach is the most efficient. It is probably also the one that is easiest to implement. Both approaches are general and can be used for all applications including diagnosis, design, etc.

Ease of implementation and execution speed are important considerations since the inference engine will execute on various processing units ranging from local processing units to supervisory control units. The static link approach is therefore what is preferred in the knowledge-based control system concept.

The distribution of rules to different processing units creates interesting aspects. One such aspect is the links that must exist between rules and variables that reside on different processing units. For example, changes in some variable in a local processing unit may cause rules to be tested through forward chaining in a supervisory unit. For implementation purposes, one may want to restrict the forward and backward chaining to only take place within a processing unit.

Fuzzy rules: In addition to the G2 style, static link, inferencing, it is important that the user has the possibility to define that certain rules should behave as fuzzy rules. As mentioned earlier, this affects the underlying logical system. However, it also affects the inference engine.

Consider the following example. Two fuzzy rules both give a value to a fuzzy variable.

```
Fuzzy If X is High
      Then Conclude that U is Zero

Fuzzy If X is Normal
      Then Conclude that U is Large-Negative
```

The fuzzy antecedents of the two rules may both have fuzzy truth values, i.e., both rules are fulfilled to some degree. `Zero` and `Large-Negative` are defined as fuzzy membership functions. The result of the two rules is that the value of U is determined by a new fuzzy membership function. This new membership function is calculated from the membership functions of `Zero` and `Large-Negative` based on the fuzzy truth values of the rule antecedents, i.e., they are weighted together with the fuzzy truth values as weights. The fuzzy inference engine must also have a way to de-fuzzify U, i.e., to calculate a non-fuzzy number from the resulting membership function.

Certainty factors

It is an open question if certainty factors should be a part of the inference engine or not. If so, there should be built-in possibilities to associate a certainty value with every variable and with every rule, and to have these values propagated during the inference process.

G2 does not support certainty factors. However, many rule-based, off-line expert system tools use them. Systems with certainty factors usually allow a variable to have several values at the same time, each with a different certainty. This further complicates the inference engine. It is unclear if and how this can be incorporated into a real-time, distributed, static link inference engine.

Rule invocation

Except from forward and backward chaining it should be possible to invoke rules with a regular time interval, asynchronously when a certain event occurs, and explicitly.

> **G2 solution:** By associating a scan interval with a rule in G2, the rule will be regularly tested. The events which may trigger an asynchronous rule, or a *whenever rule*, are
>
> - when a variable receives a new value,
> - when a variable fails to receive a new value,
> - when a relation is created or deleted, and
> - when an object is moved.
>
> In G2 explicit rule invocation can be done in various ways. Rules can be associated with one or several classes and objects, and with user-defined categories. All rules associated with a class, an object, or a category may be explicitly invoked and tested. With this type of invocation the rules are only tested once. It is also possible to explicitly activate and deactivate rules and thereby start and stop their invocation. This is done by activating and deactivating the workspace upon which the rules are stored.

The above G2 functionality is almost sufficient. In addition to it one would like to be able to catch when an object is created, modified, or deleted; and to explicitly invoke rules on a specific processing unit.

Rule objects

Rules should be a pre-defined class with certain default attributes. The default attributes include the antecedent part; the conclusion part; the scan interval; the classes, objects, and categories that the rule is associated with; the rule priority; and the processing unit where the rules should execute.

It should be possible to define icons and relations for rules and to further specialize the pre-defined rule class.

Rule actions

The actions that should be available in the conclusion part of a rule should be actions for concluding new values for a variable; creating and deleting objects and relations; explicitly invoking other rules; calling functions or starting procedures; and for enabling and disabling various parts of the knowledge base.

These rule actions are similar to those in G2. In addition to this G2, has several rule actions that are used to affect the user interface. These actions include changing the colours of objects; revealing and hiding windows; moving and rotating objects; and writing information messages to the operator. In the knowledge-based control system these functions will be defined by the user interface parameters.

5.3.6 Syntax considerations

The syntax of the knowledge base language could be of the natural language style or of a conventional programming language style. Both approaches have their advantages. A problem with natural language syntax is that expressions tend to become very long and thus cumbersome to manually type in.

> **G2 solution:** G2 is based on a natural language syntax. Expressions in G2 look like
>
> ```
> .. the status of the pump connected to V-44 ..
> .. the status of any pump ..
> .. the rate of change per minute of tank-4 between
>    1 hour ago and 3 hours ago ..
> ```
>
> The user is supported by a syntax-oriented, mouse-driven editor which prompts the user for the syntactically correct possibilities and lets him select among those with the mouse. In that way the user does not have to type in the expression.

An advantage with natural language style syntax is that it, for short expressions such as rules, feels very natural. However, in larger expressions such as procedures a programming language syntax that, e.g., uses dot notation to refer to attributes, is more compact. Instead of the natural language style syntax

```
If the level of tank-4 >
   the high-level of tank-4
Then conclude that the status of tank-4 is high,
```

one would like to be able to say

```
If tank-4.level > tank-4.high-level
Then tank-4.status := high.
```

Therefore, it is probably wise to have the possibility to choose between both syntaxes.

References to object and attributes

The language must provide possibilities to reference objects and their attributes either by their names or through their relations with other objects. For multi-view objects it must be possible to specify which view of the object should be referenced, as an alternative to referencing the object as a whole.

Language expressions

Expressions in the knowledge base language are phrases that have a value which is either a number, a truth value, a symbol, or a text string. The language must support a large variety of different expressions including arithmetic expressions, symbolic expressions, logical expressions, text string expressions, relational expressions, fuzzy expressions, conditional expressions, history expressions, expressions over sets of objects, list and vector expressions, formatting expressions, etc. The history expressions will be commented further in the section on real-time aspects.

5.3.7 Languages versus meta-languages

One may consider having a knowledge base language with meta-language properties. For example, this would mean that the language and the syntax of the language could be defined in the language itself and modified in the language. This is something which is possible in, e.g., Lisp based expert system languages. Considering the desire to write interpreters for the language that may execute also on quite small local processing units this functionality is probably unwise.

A related issue is the question of how complete the language should be. Should it be possible to easily implement all things that a user needs within the language or should there be hooks in the language so that external languages such as, e.g., C++, may be called? The latter is probably the case. If so, the interface between different languages is an important issue.

5.3.8 Functions and procedures

The language will have a set of built-in functions. These includes the standard arithmetic functions, functions that convert between different time expressions, string conversion functions, etc.

User-defined functions should be defined by the user. They include analytic and tabular functions. Tabular functions makes it possible to define, e.g., non-linear functions between variables as a table of numbers.

Analytic functions should be defined in terms of procedures that return values. Procedures should be defined in terms of a block structured, sequential programming language. Procedures could either be called as subroutines or started as parallel execution tasks. It should be possible to associate procedures with objects and to call them using message passing, i.e., the procedures can be used to implement methods. Procedures should be able to have attributes, local variables, and they might call or start other procedures. The language constructs that are needed are the usual ones found in any conventional programming language. Examples are

- iteration statements that iterate over numbers, lists, vectors, all instances of an object, etc.;
- conditional statements such as if-then-else, case, etc.;
- loop statements such as while-do, repeat-until, etc.; and
- statements for waiting a certain time or until a certain condition is fulfilled.

Rule actions may be executed as statements in procedures.

> **G2 solution:** G2 includes the above functionality for procedures and functions. For historical reasons they distinguish between functions and procedures. Procedures can only be stated explicitly. Functions can only be called by the inference engine when evaluating a language expression in, e.g., a rule condition, a formula, or a conclusion statement. Further, functions may only contain one, possibly conditional, statement.

The above division between functions and procedures should be avoided.

It should be possible to associate a scan interval with a procedure. In this way the procedure will be called or started at a regular time interval. Procedures should have an attribute that determines in which processing units that the procedure should execute. If a procedure is started explicitly through, e.g., a rule action it should be possible to specify in which processing unit it should execute.

5.3.9 Equations

Equations should be a pre-defined class similar to rules. Instances of this class may be used as the value of other object attributes. Attributes of equations could be the type of the equation, i.e., differential equation or boolean; the different parts of the equation; etc.

5.4 REAL-TIME LANGUAGE CONSTRUCTS

State-of-the-art commercial real-time expert system tools such as G2, Talos, and Chronos contain different methods for partially solving the problems that exist when an "intelligent" system should reason in time about a changing environment. The problems, which were discussed in the Feasibility Study, include non-monotonic reasoning, temporal reasoning, reasoning under time constraints, and handling of asynchronous events. A KBCS needs to combine the expert system methods for solving these issues with conventional methods such as, e.g., fast sampling.

5.4.1 Validity intervals

A problem with reasoning about dynamic environments such as processes is that measured sensor data is only valid for limited time. This time is basically determined by the dynamics of the process. Hence, dependent variables computed or concluded from these data will also have a limited validity. When the data changes, dependent variables may no longer remain valid and their values have to be retracted, i.e., the system must have a non-monotonic behaviour.

The conventional solution to this problem is to sample everything as fast as possible and to recompute dependent variables each sampling interval. The solution applied, e.g., in G2, is to associate a validity interval with measured variables and to propagate this validity interval to dependent variables. When the validity interval of a variable has expired the variable looses its value. References to a variable that has a valid value will return that value. References to a variable whose value has expired will cause a new value to be computed either by performing a new measurement or by recomputing the value for the variable. The method is exemplified in Fig. 5.23.

If, for some reason, measurements cease to arrive to the system, all time dependent variables will eventually expire. Validity interval is one way to partially solve the non-monotonicity problem which can be reasonably easily implemented also in real-time systems, as shown by G2. A drawback with the approach is that it does not maintain explicit dependencies between a concluded variable and the variables that it depends on. Instead, only timeouts are propagated between the variables. It is impossible to represent that a particular conclusion is dependent

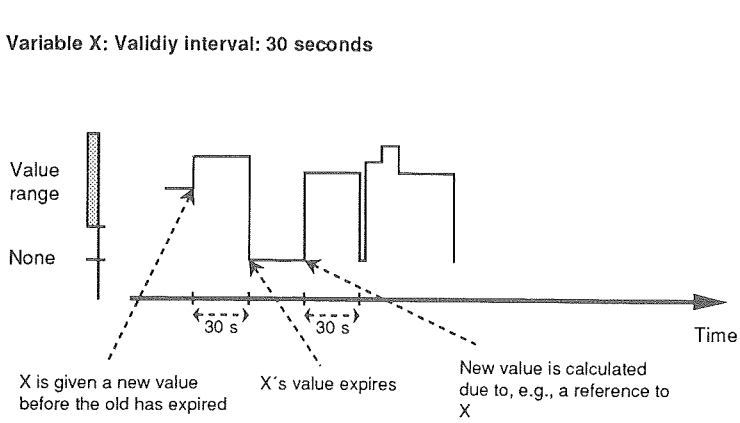


Figure 5.23 Validity interval based truth maintenance

on the specific value of some variable, and that it should be retracted immediately if the variable changes its value.

A truth maintenance system (TMS) is a system that automatically maintains dependency links between concluded variables. TMSs operate as huge book-keeping systems that keep track of everything that is being done in the system. Large hybrid expert system tools such as KEE and ART have TMS facilities built-in. To add them to a real-time system is, however, not realistic.

Therefore, the validity interval method is the only viable method apart from the usual fast sampling. However, it has been pointed out by (Pavek, 1990) that if the inference engine is fast there are indications that it may be more efficient to recalculate the conclusions every time they are needed instead of caching them. Validity intervals can be seen as a way of caching information. Maintaining the cache may cause more overhead than is saved by a fewer number of calculations. If so, validity intervals should be used only on measured variables or on variables that are transferred over the network in order to reduce communication.

5.4.2 Timed actions

In the G2 system, actions in rules or procedures are executed at the time when the rule or procedure is executed. It may be useful to have the additional possibility to perform an action after a certain delay or at a certain time according to the following example.

```
If ...
Then after 2 minutes
  conclude that the status of pump-4 is normal
```

or

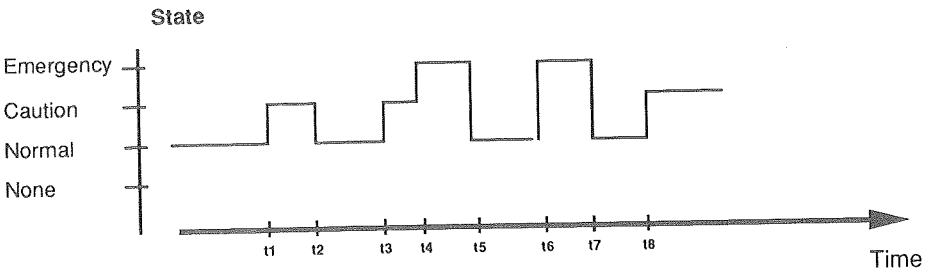


Figure 5.24 Symbolical time history

```
If ...
Then at 3.00 PM
  conclude that ....
```

5.4.3 History-based reasoning

Reasoning about old values of variables and over the time history of variables is important. By monitoring rates of changes and variations, a skilled operator can gain much knowledge about the process operation.

Old variable values

It should be possible to store histories for all variable. The processing unit in which the history is actually stored is determined by where the variable executes. The language should have built-in functions for referencing old variable values.

Numerical history functions

The language should have built-in functions operating on time histories of numerical variables. These should include functions for the minimum and maximum values of a variable over a certain time, and the standard deviation, rate of change, and integral of a variable over a selected period of time.

5.4.4 Time intervals

Reasoning about how a variable behaves over a time interval is important. Symbolic variables in combination with validity intervals makes it possible to reason about the state of a variable at a certain time. Consider the following example. Assume that a symbolical variable `state` has been defined. This variable can take three possible values, `normal`, `caution`, and `emergency`. `State` has a certain validity interval and is recalculated regularly. The time history for the `state` variable is described by Fig. 5.24.

Using references to old values of the variable it is possible to reason about the state of the process at a certain past time. However, it is also necessary to be able to reason about the time intervals during which `state` has had a certain value. For example, during the interval between `t1` and `t2` the state was caution, during the interval between `t1` and `t8` the state has been emergency two times, etc.

Two possible alternatives for representing time intervals exist. First, one can base the representation on time histories and provide them with a set of appropriate symbolic history functions. Second, intervals can be represented as separate objects.

In the time history solution, some of the appropriate functions are exemplified below.

```
the number of times that state has been emergency
between 2 hours ago and 1 hour ago ==> 2

when did state first become emergency
between 15.00 and 17.00 ==> 15.23


the time when state received its value
as of 1 hour ago ==> the start of the time interval
for state as of 1 hour ago

the time when state lost its value
as of 1 hour ago ==> the end of the time interval
for state as of 1 hour ago

etc..
```

In the second alternative, advocated by Pavek (1990), special time interval objects are defined. A time interval object is created dynamically when the start condition for the interval is true. The object has default attributes for its starting time, ending time, and duration. When the end condition for the interval is true the ending time is given a value. Interval objects are deleted when, e.g., `current time - the ending time > 4 days`. As with all objects it should be possible to define new interval classes with extra properties.

One can further extend the idea of support for reasoning about time intervals by considering the following types of expression in the language.

```
If ever(X = 0) before time T
```

```
If always(X = 0) after time T

If ever(X > 5) during time [L,R]

If always(X > 5 & voltage = low) during time [L,R]
```

5.4.5 Events

Events are important entities in a temporal reasoning system. A true temporal reasoning system must support reasoning about events and sequences of events. Language support for the expressions exemplified below are important.

```
If (X > 0) before (Y < 10) in last 10 minutes

If (X = 1) after (Y > 5) in last 10 minutes

If (X = 2) while (Y > 5) in last 10 minutes

If (X > 0 & Y = 1) before (Z = 0) in last 10 minutes

etc.
```

Similarly to intervals, it is possible to represent events as objects that are created dynamically and which can be referenced in language expressions.

5.4.6 Priorities

It should be possible to associate priorities to rules, rule groups, procedures, etc. The priority determines the relative importance of the corresponding activities in time critical situations.

5.5 SUMMARY

Knowledge about a process is available in a variety of different forms. The main knowledge base should be able to represent this knowledge in an as structured and in the same time flexible way as possible.

This chapter has discussed two major issues: how to structure knowledge about a process, and the nature of the language needed to represent this knowledge. Two ways of decomposing a process has been described. The systems of a process corresponds to the different flows of material, energy, or information. In the

Steritherm process the systems are the product system, the warm water system, the cold water system, the pneumatic system, the electrical system, etc.

The views represent different ways in which an object, e.g., the entire process, a system in the process, or a process component, can be described. Which views that are used depends on the application. However, three views are of a more general nature. The geographical view describes an object in a geographically correct way. It may consist of 3-D, or 2-D descriptions, photographs, etc. The topological view describes the internal structure of an object and is close to what we today mean by a process and instrumentation diagram or a flow schematic. The functional view describes an object in terms of the goals that it should fulfill, the functions that are needed to fulfill these goals, and the components that realize these functions.

A language for describing the main knowledge base has been discussed. The language is heavily influenced by G2. In comparison, the language has a more complex object structure that includes multi-view objects, has extended support for reasoning about events and situations, and can be executed in a distributed environment.

Several issues are still unclear. The separation between knowledge representation as described in the knowledge base by objects representing systems and views and how this knowledge should be presented for different users is one such area. Distributed inference engines are another.

6

Requirements and Limitations

6.1 INTRODUCTION

This chapter describes the requirements for implementing the KBCS concept and the limitations of current technology. In some areas, current technology is not advanced enough to be able to implement the concept, but future development of technology can be foreseen to a certain extent.

In order to implement the KBCS concept, certain hardware requirements must be fulfilled, for example - CPU power, primary memory capacity, mass storage capacity, and user interface hardware. We must also consider various aspects of software technology.

6.2 REQUIREMENTS

The technology requirements can be divided up into the following points:

- Hardware technology
  - Distributed or parallel processing
  - CPU power

- Primary memory capacity
- Mass storage capacity
- User interface hardware

- Software technology
  - Programming Languages
  - Operating Systems
  - Software Standards
  - Graphical User Interface Software
  - Language technology for structuring knowledge
  - Object-oriented Databases
  - AI Technology
  - Computer Aided Software Engineering

6.2.1 Hardware Technology

Distributed or Parallel Processing

It is not unlikely that KBCS concept will be implemented on distributed and/or parallel computers.

Distributed Processing Distributed computer technology is well developed from a hardware point of view. To a certain extent, one can increase the computing power available by adding more computers to the network. However, some functions cannot easily be distributed, and therefore require a certain minimum CPU power in a single computing node. From another point of view, the more powerful the single CPU nodes can be, the simpler the network can be.

One must also take into consideration that the network will be heterogeneous, i.e. it will contain many different computers with different CPUs, different instruction sets, etc. The computers may even run different operating systems. Also to be taken into account in the configuration of the network is the amount of primary and mass memory available at each node, the presence of file servers, compute servers, and specialized high-level servers such as inference engines, etc.

Parallel Processing There are many types of parallel processing hardware. The type that is relevant to the KBCS concept is called Multiple Instruction Multiple Data, or MIMD. (Single Instruction Multiple Data or SIMD, such as vector processors used for number-crunching in super-computers, are not suitable).

MIMD processing is a less well developed area of technology than distributed processing. There are a number of different approaches to MIMD parallelization, from coupling together conventional microprocessors on a bus architecture, to a point-to-point network architecture of microprocessors specially designed for parallelization (such as transputers) with the associated problems of optimal network topology.

CPU Power

Judging by the performance of current knowledge-based systems on state of the art workstations and guessing the requirements of the KBCS concept to be significantly higher, one comes to the conclusion that today's microprocessors do not have adequate performance. The typical CPU power of a state of the art microprocessor today is about 20 MIPS. It would not be unreasonable to estimate that roughly 10 times that (about 200 MIPS) is required for the KBCS concept. Further increases above that will certainly be beneficial because they will enable simpler computer configurations to be used.

Increases in CPU speed must of course be matched by corresponding improvements in primary memory access speed, etc.

Primary Memory Capacity

Primary memory capacity of today's computers is inadequate for the KBCS concept. The complex software of the KBCS will require large primary memory in order to execute effectively, both for the program code and for large amounts of data that need to be in primary memory. In addition to the KBCS software, there is a large memory requirement for the standard software for the graphical user interfaces.

Typical primary memory capacity of a state of the art workstation today is about 20 megabytes. It would not be unreasonable to estimate that an increase of about 10 times (about 200 megabytes) is likely to be needed for the KBCS concept.

Mass Storage Capacity

The Main Knowledge Base integrates all of the knowledge about a complete installation, and therefore requires a large mass storage capacity.

Typical storage capacity of a state of the art hard disk today is about 600 megabytes. The mass storage capacity required for a KBCS, that will store all of the data for a complete installation, could easily be about 10 gigabytes or more.

User Interface Hardware

The hardware necessary for the user interface is available today, i.e., megapixel colour displays and interaction units such as keyboard and mouse.

6.2.2 Software Technology

Software technology is very difficult to predict. It is possible to implement the KBCS concept using existing software technology (for example with Unix and C++), but it likely that future advances will make it very much easier.

Different aspects of software technology are discussed below:

Programming Languages

It is much more difficult to implement distributed systems than to implement systems with a single CPU. This is one area where improvements in programming languages can have a large impact on the implementation. The ideal language for implementing the KBCS concept would have the following features:

- Object-oriented
- Parallelism and multi-tasking and distributed processing built into the language.

Examples of these individual features exist today. For example, C++ is an effective object-oriented language. Occam is a language with parallelism/multi-tasking/distributed processing built into the language in a natural way, and there are also similarly capable versions of C available that have been inspired by Occam. As yet there is no language which combines object-orientation and parallelism/multi-tasking/distributed processing, but the most likely development is that a new variant of C will emerge to meet this need. It is better to have a variant of a mature language that is as far as possible compatible with the original language than to have a completely new and immature language. It is nevertheless very difficult to predict when a suitable candidate will appear.

Operating Systems

When using multi-tasking/parallel/distributed programming languages and parallel and/or distributed hardware, it is essential to have an operating system that is specially designed to handle multi-tasking, parallelism and distributed processing and effectively utilizes the hardware.

As with programming languages, it is better to have a variant of a mature operating system rather than a completely new operating system. The most interesting alternative in this case is the Mach kernel from Carnegie Mellon University. This is a Unix kernel which is specially designed to handle both parallel and distributed processing and is hardware-independent.

Software Standards

Standards lead to portability, increased market competition, lower prices, and better system performance. Examples of recent standards that are relevant to the KBCS concept are Unix standards (POSIX, X/Open, OSF, AT & T System V version 4), graphics standards (X Window System, Display PostScript), and networking standards (Ethernet, NFS).

Graphical User Interface Software

Graphical user interface software is currently maturing, and a number of industry standards are becoming established: For example, window systems (such as the X Window System), graphics standards (such as PostScript), and "look and feel" definitions (such as OSF/Motif).

Language technology for structuring knowledge

Language technology is quite old but nevertheless still immature, as it is a difficult technology. There are, however, a number of tools available for building compilers (such as the Unix programs "yacc" and "lex"). With the current explosion of object-oriented programming languages, object-oriented versions of these tools should soon become available, and these should be easier to use.

Object-oriented Databases

The new technology of Object-oriented Databases is likely to prove very useful in the KBCS concept. There is currently a considerable amount of research in this field, and it is likely to increase enormously in commercial importance over the next 5 years.

AI Technology

There was a great deal of interest in Artificial Intelligence and Expert Systems at the beginning of the 1980's. Interest declined when it proved difficult to integrate the technology with conventional systems. Since then, the technology has improved, particularly with the introduction of model-based expert systems and the influence of object-oriented techniques. Object-oriented technology makes it much easier to integrate AI techniques with conventional software technology.

Computer-Aided Software Engineering

CASE technology has been much talked about, but has not yet matured. The CASE tools available are usually very specific to particular languages and operating systems, which has limited their use.

6.3 FUTURE OF HARDWARE TECHNOLOGY

Some aspects of the development of hardware technology can be predicted with reasonable accuracy. Thus we can estimate when the technology will be mature enough to implement the KBCS concept.

6.3.1 CPU Power

Future development of microprocessor speed is reasonable predictable, partly being a question of semiconductor technology, which is improving all the time. Additionally, there are advances in CPU design, such as RISC, which further increase the speed of CPUs, but which are less predictable.

Typical CPU power today: 20 MIPS.

Desired CPU power: 200 MIPS.

Current rate of increase: 10 times every 5 years.

Technology available: 1995.

6.3.2 Primary Memory Capacity

Primary memory capacity is largely a question of price, which is in turn determined by the size of memory that can be accommodated on a single chip. It is therefore less meaningful to talk about when the capacities required are available, and more meaningful to discuss when they will be not unusual.

Fortunately, the future development of technology in this area is fairly predictable for next few years, being almost entirely a question of improvements in semiconductor technology.

Typical capacity today: 20 megabytes.

Desired capacity: 200 megabytes.

Current rate of increase 10 times every 6 years.

200 megabytes not unusual: 1996.

6.3.3 Mass Storage Capacity

Like primary memory capacity, mass storage capacity is largely a question of price, which is constantly decreasing as technology advances.

Rewritable optical disk technology has recently come onto the market. This technology promises to give the disk capacities required by the real-time knowledge-based control system concept. Current optical disks are relatively slow, but are expected to improve rapidly in the next few years.

Typical capacity today: 600 megabytes per drive.

Desired capacity: 10 gigabytes.

Current rate of increase of disk capacity for a given price: 10 times every 7 years.

10 gigabytes not unusual: 1997.

6.4 CONCLUSIONS

Hardware-wise, the critical factor is CPU power. The other hardware factors are mainly a question of price.

Software-wise it is very difficult to come to any conclusions. Today's software must be considered adequate for the KBCS concept because one can, in principle, program anything with existing software technology. To look at it another way, it is not possible to say that any particular technology (for example, a parallel version of C++) is an absolute requirement to implement a KBCS.

What one can say is that future developments in software technology can make the programming task much easier, and it is also possible to a certain extent say what functionality future software technology will have. It is nevertheless difficult to say when this functionality will be available. Software functionality is a very much complex subject than hardware functionality (which is often just a question of price and performance).

If one knows that a particular breakthrough will come in five years time, then one could suggest waiting until then. But because software technology is unpredictable, one cannot usually tell whether a breakthrough will come in 20 years time, or in one months time. Such unpredictability means that it is probably best to proceed with the software technology that exists today, rather than wait an unpredictable period of time for something that may not come.

7

Prototypes

7.1 INTRODUCTION

A major part of the first phase of the project has been the implementation of two prototypes. The prototypes are used to visualize some of the different aspects in the system concept.

In the first prototype, the hypermedia tool Plus (1989) has been used to visualize the functionality of a future operator station in a knowledge-based control system. The prototype uses drawings, scanned-in pictures, graphics, colours, and sound on a Macintosh II computer. The prototype was implemented as a master thesis project by Peter Höjerback (1989).

The second, and largest, prototype uses the commercial real-time expert system tool G2 from Gensym Corporation (Moore *et al*, 1987) for experimenting with the structure of the main knowledge base. The areas of the two prototypes with respect to the system concept are shown in Fig. 7.1.

G2 has been used to build up a model of a KBCS that controls, monitors, and diagnoses a real-time, quantitative simulation of the Steritherm process which also has been implemented in G2. The connection between the prototype and the simulation model consists only of sensors used in the process and through control signals to valves and pumps. This structure is shown in Fig. 7.2. The KBCS contains

- PID controllers,

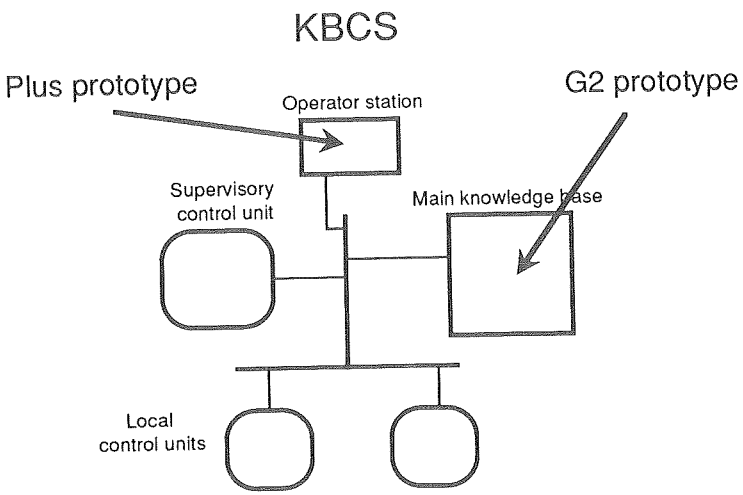


Figure 7.1 The prototypes and the system concept

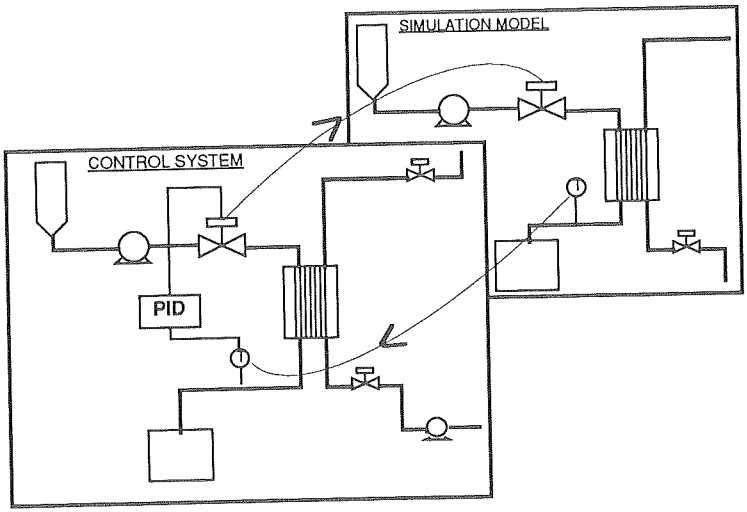


Figure 7.2 The G2 prototype architecture

- sequence logic implemented using the Grafcet formalism,
- combinatorial logic,
- simple rule-based burn-on monitoring,
- model-based diagnosis according to the Diagnostic Model Processor method developed by Thomas Petti,
- alarm tree based alarm analysis,

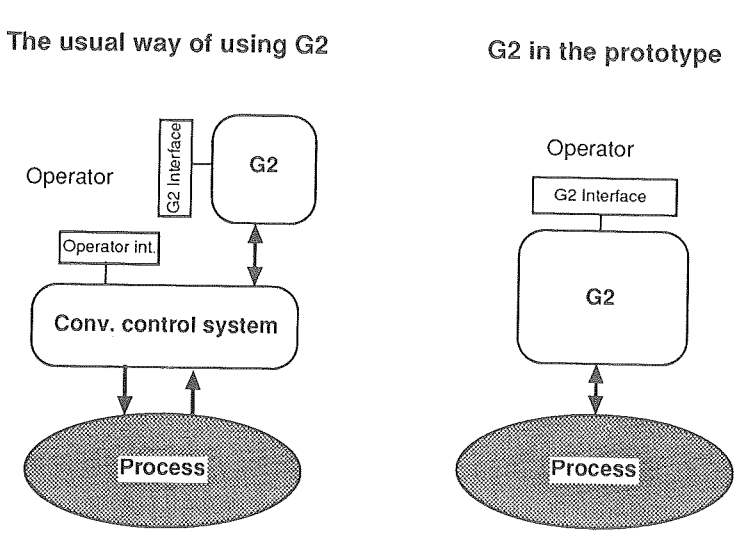


Figure 7.3 Different ways of using G2

- MFM (multi-level flow model) based diagnosis, and a
- product flow following system.

The core of the prototype was developed as a master thesis project by Michael Christiansson and Pär Ericsson (1989). The prototype has since been continuously extended. The system was originally implemented on a monochrome Symbolics Lisp machine and has since been ported to, and further extended on, a colour Sun Sparcstation.

It may seem strange that the expert system tool G2 has been used for implementing the prototype. G2 represents the on-line expert system architecture of today where an external expert system is used on top of a conventional distributed process control system with all the problems of multiple user interfaces, communication bottlenecks, and redundancy previously pointed out. However, even if not intended to be used in this way, G2 is powerful enough with respect to representation to also allow the implementation of the conventional controllers and logic. In that way G2 is a good system for experimenting with integration along the lines of this project. The way we use G2 and the usual way of using G2 are compared in Fig. 7.3. The G2 prototype has another interesting property. It can, with moderate effort, be modified to control and monitor an actual Steritherm process instead of the real-time simulator.

The two prototypes were developed before the system concept had stabilized. Therefore, the terminology in the following description differs somewhat from the system concept. This is specially the case with the term view. In the Plus prototype, view refers to the user views defined by the user interface parameters.

In the G2 prototype, the views correspond closer, though not exactly, to the concept,

7.1.1 Outline of the chapter

The Plus prototype is described in Section 8.2. The various parts of the G2 prototype are described in Section 8.3. The general conclusions from the prototypes are summarized in Section 8.4.

7.2 A HYPERMEDIA OPERATOR INTERFACE

An important aspect of the real-time knowledge-based control system is the user interfaces. There are many potential user interfaces to the system: interfaces for designers, operators, maintenance personnel, and administration personnel. In order to explore the user interface aspect, it was decided to make a prototype of an operator interface to a KBCS.

7.2.1 The hypermedia tool "Plus"

The pre-prototype was made using the hypermedia tool "Plus", which runs on the Apple Macintosh. Plus has the two advantages that it offers the freedom and possibilities of hypermedia technology to integrate text, pictures, and sound in a free-form structure, and it has good tools for building programs.

Programs produced by Plus are structured as a number of full-screen pictures, called "Cards". The cards are linked together by associative links, which are in turn coupled to icons called "Buttons". By clicking a button with the mouse, a new card is displayed with new information and graphics.

Plus contains an interactive graphics editor, and can import graphics and sounds from other Macintosh programs. Plus also contains a powerful language called HyperTalk for animating the graphics and coordinating the display of pictures and use of sound.

The lack of windows or of pictures smaller than a full screen limits the flexibility of Plus, but it was judged to be the tool at the time that would give the best results in the prototype considering the limited time available.

7.2.2 The prototype

The prototype was designed to show as many aspects of the operator interface of a KBCS as possible. Using the capabilities of the hypermedia program "Plus" the different aspects were to be emulated in order to demonstrate the possibilities of different ways of interaction and to evaluate their usefulness.

The aspects of the operator interface to be shown were:

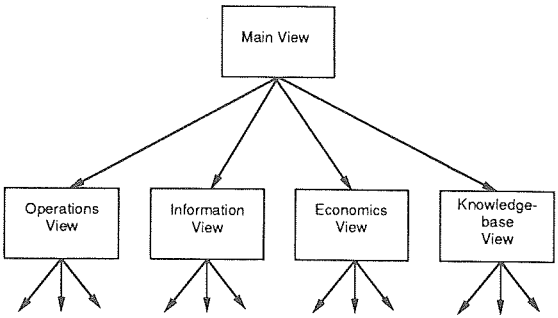


Figure 7.4 Organization of screens in the hypermedia prototype.

- Process flowchart
- Product information
- Economic information
- Process design information
- Process simulation
- Sequence control
- Expert system diagnosis
- Information zooming
- Different abstraction levels (What, How, Why)
- Alarm handling
- Operator advice
- Animation
- Visual effects
- Sound effects

In most cases only one example of each aspect was implemented. As in the rest of the project, Steritherm was chosen as the demonstrator process.

Views

The interface screens are organized in a structure that is basically hierarchical, but with extra associative links that cut across the hierarchical structure directly to relevant screens. There are five main screens:

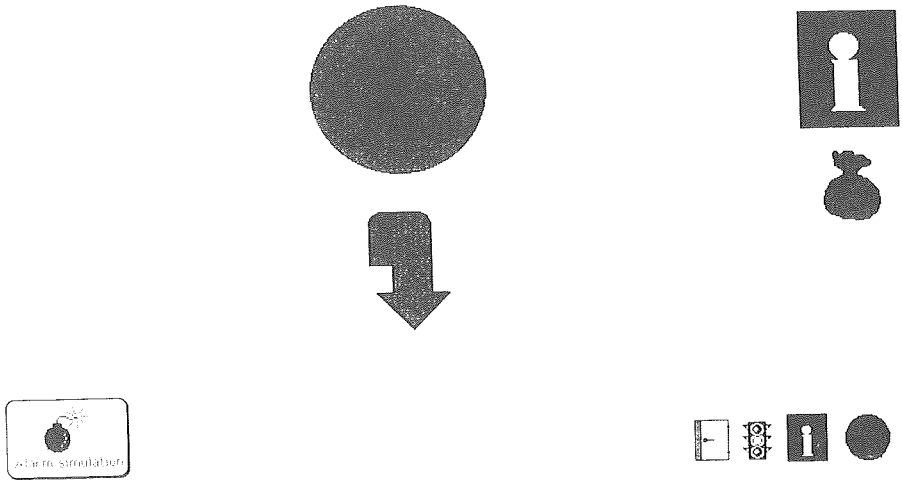


Figure 7.5 The Main view

- Main View
- Operations View
- Information View
- Economics View
- Knowledge-base View

In addition to these five main screens, there are a number of other screens further down in the view hierarchy. The organization of the main screens is shown in Fig. 7.4.

Main View: The main view shows the highest level of abstraction. In the centre of the screen is a Chernoff's Face, which indicates the general health of the process. In addition there is a number of icons which are linked to the other four main screens, the Operations View, the Informations View, the Economics View and the Knowledge-Base View, and to other important screens. The Main View is shown in Fig. 7.5.

Operations View: From this screen it is possible to start and stop sequences such as sterilization, production, rinsing, etc., and to check alarms. There are also several icons which are linked to further information on the sequences, e.g.,

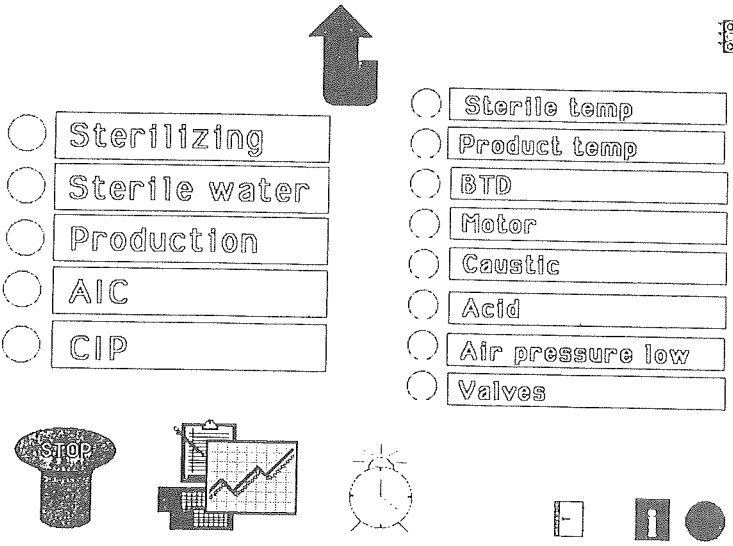


Figure 7.6 The Operations view

flow charts and adjustment of process parameters, such as temperature, pressure, and sterilization time. In short, one is in control of the dynamics of the whole process from this view and its subviews. The Operations View is shown in Fig. 7.6.

Information View: The Information View gives access to static information about the process. From a screen showing a card register, one is able to pick the subview one is interested in. The Information View is shown in Fig. 7.7.

One of the subviews of the Information View is a flow schematic of the process, which can be animated to show active pumps and colour-coded temperature indications. The Flow Schematic View is shown in Fig. 7.8. From this subview, further subviews show 3-D perspective views of the physical construction of the process and individual components. The 3-D Perspective View is shown in Fig. 7.9.

Other subviews of the information view give component information, electrical diagrams and product information.

Economics View: The Economics View indicates with the aid of charts how efficiently the production runs.

Knowledge-Base View: The Knowledge-Base View emulates an interface to the knowledge browser of the main knowledge base.

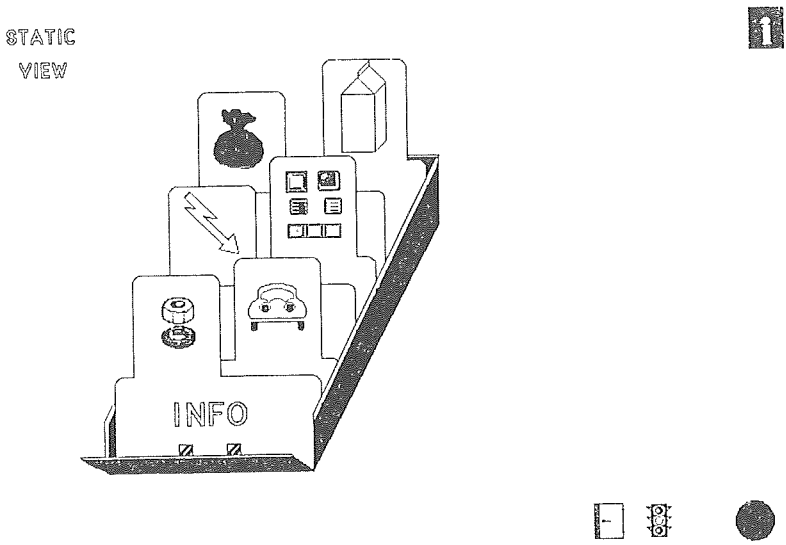


Figure 7.7 The Information view

Sound effects

Sound effects have been introduced in many places in the program. The sound effects available have been limited, so the ones used are not always the most relevant, but are used to demonstrate the principle that sound can be an important means of conveying information to the process operator.

Alarm Simulation

The alarm area is located at the top of the screen. There is an alarm simulation button on the Main View which causes an alarm message to be flashed across the top of the screen, accompanied by an audible alarm signal. The alarm also affects the appearance of Chernoff's Face on the Main View. The alarm signals continue until the alarm is acknowledged by the operator. The simulated alarms are motor faults.

Expert System fault diagnosis

There is a button on the Main View for the operator to activate expert system fault diagnosis in order to find the possible causes of an alarm. This function is not implemented in the prototype.

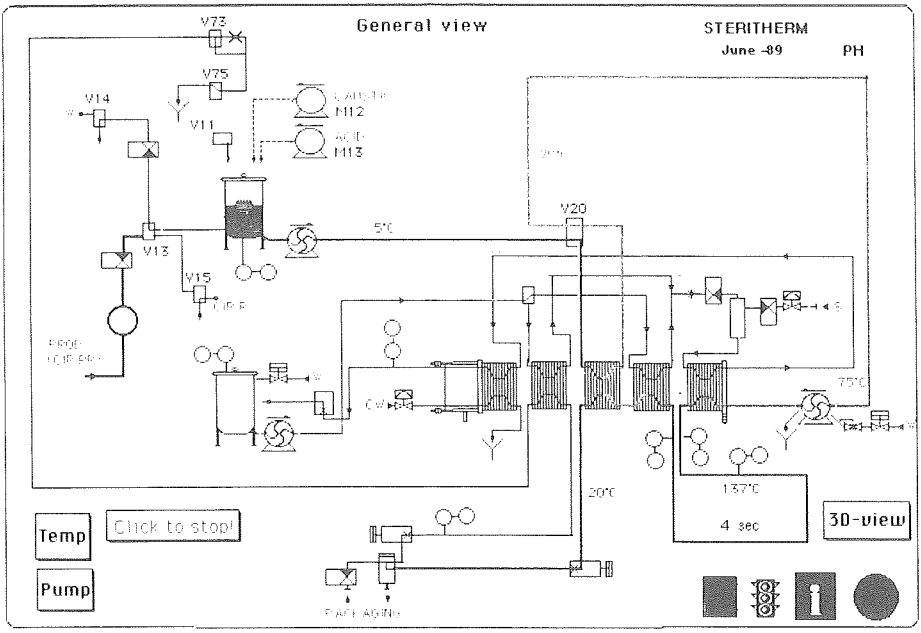


Figure 7.8 The Flow Schematic view

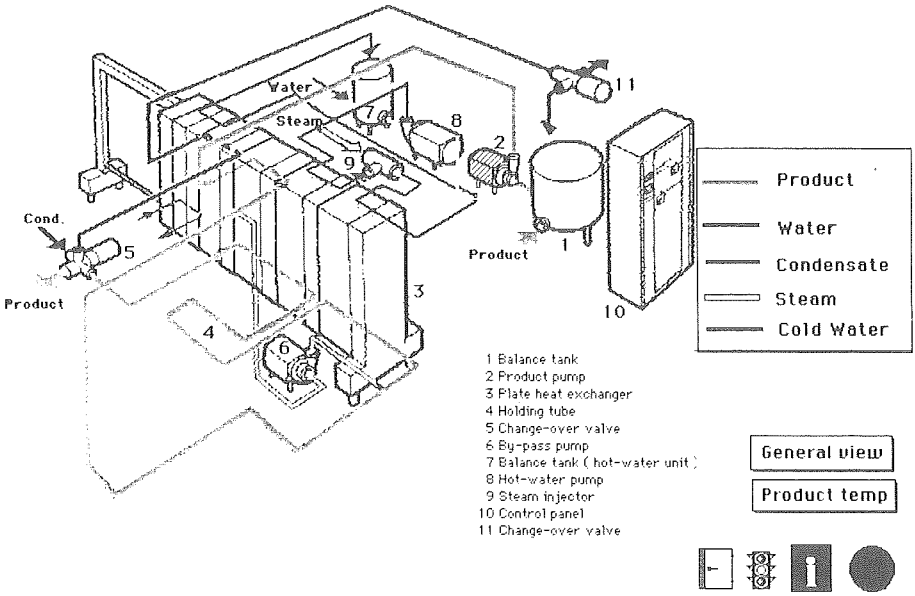


Figure 7.9 The 3-D Perspective view

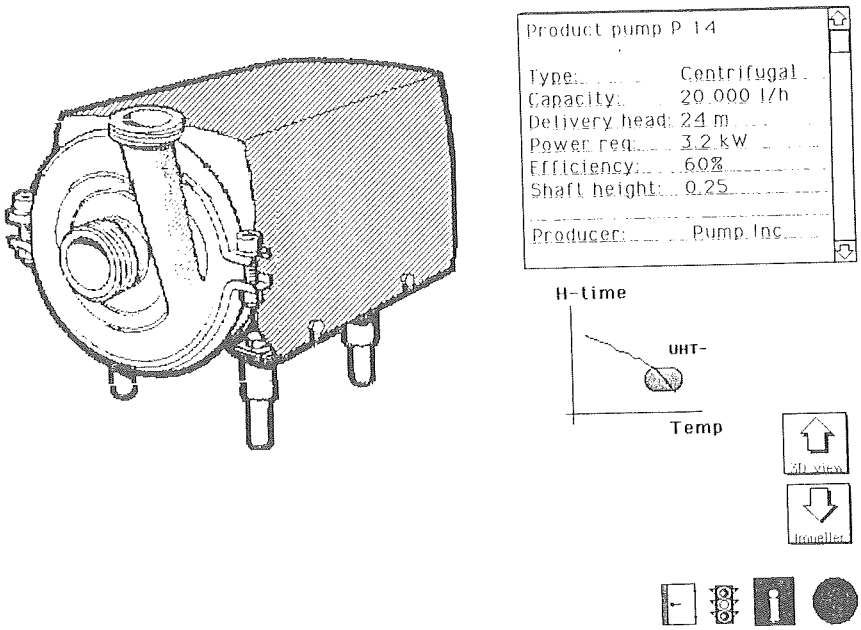


Figure 7.10 Detailed view of pump

Zooming

From the 3-D Perspective View of the process, under the Information View, individual components, such as pumps, can be selected and "zoomed" in order to examine them in detail. Information about the components, such as pump charts, is also available in the zoom views. There are several zoom levels showing different levels of detail. Fig. 7.10 shows an example of a detailed view of a pump which has been "zoomed" from the 3D Perspective View.

7.2.3 Conclusions

The hypermedia technology made it possible to quickly develop a "mock-up" of an operator station. The prototype makes strong use of scanned-in drawings, colours, and sounds, in order to visualize the possibilities that exist. The prototype is not so deep. Most of what is seen predefined, static screens.

7.3 THE G2 PROTOTYPE

The main purpose of the G2 prototype is to examine the inner structure of the knowledge-base. In order to do this, a G2 prototype of the main knowledge base in a KBCS has been developed. The prototype controls and supervises a numerical real-time simulation model of Steritherm that also has been developed

within G2. Those readers that are not familiar with G2 are referred to Chapter 8 which contains a thorough presentation of G2.

The G2 prototype began as a master thesis project. The contents of the master thesis are described in subsections 7.3.1 to 7.3.4 beginning with a description of the real-time simulator in subsection 7.3.1. The main hierarchical structure of the control system is described in subsection 7.3.2. This subsection also includes the process schematics and the PID controllers. The Grafcet implementation of the sequence logic is described in subsection 7.3.3. The monitoring and diagnosis parts of the thesis project: the rule-based monitoring and the alarm tree based alarm analysis are described in subsection 7.3.4. The model-based diagnosis is described in subsection 7.3.5. Subsection 7.3.6 is devoted to the MFM model of Steritherm and its implementation in G2. Subsection 7.3.7 contains the product following system as well as some other more or less implemented ideas.

7.3.1 The Simulation model

The Steritherm real-time simulator is based on G2's internal simulator. The simulator is object-oriented with objects representing the process' components, i.e., heat exchanger sections, valves, pumps, balance tanks, sensors, etc. The simulation schematic composed of the interconnected simulation objects is stored on a separate G2 workspace. This is shown in Fig. 7.11.

Each object has numerical attributes representing flows, pressures, temperatures, simulation constants, etc. The flow and pressure simulation is static and the temperature and level simulation is dynamic. The heat transfer partial differential equation in the heat exchangers is approximated by a set of ordinary differential equations by discretizing the space variable into three equally sized segments on the hot and cold side of the heat exchanger respectively. The burn-on that occurs in the heat exchangers is also simulated. The basis for the simulation models of pumps, valves, and heat exchangers has been taken from Åström (1974; 1989).

The mechanical regulators, such as constant pressure and constant flow valves, that are part of the Steritherm are simulated. Various sensor faults can be introduced in the simulator. The simulation consists of around 80 simulation objects and 400 algebraic and differential equations which have been statically tuned against real process data.

The main operation phases of Steritherm, i.e., production, sterilization, etc., are made up of several subphases that sometimes can differ a bit from plant to plant. The activation chart that shows the phases that are simulated in the model is described in Appendix A.5.

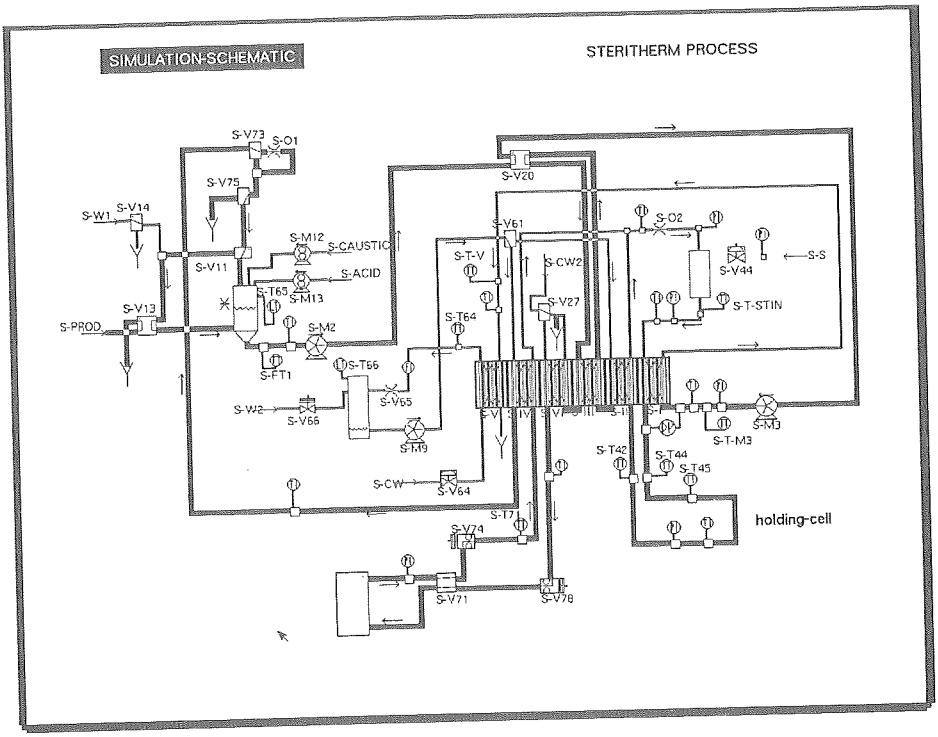


Figure 7.11 Steritherm simulation schematic

The structure of the simulation model

The objects that are part of the Steritherm model are pumps, valves, heat exchangers, tanks, a steam injector, indicators, transmitters, pipes, sources, and finally a packing machine. The objects are organized into three class hierarchies: one for the process components, one for the indicators and transmitters, and one for the pipes. All of the classes and objects connected to the simulation model have the prefix *s-* to distinguish them from the declarations of the corresponding classes in the control system. The root classes in the hierarchies are *s-process-equipment, s-indicators,* and *s-connections.* The class s-process-equipment is the most interesting. It has the subclasses s-plate-heat-exchanger, s-pump, s-valve, s-balance-tank, s-source-and-sink, and s-steam-injector.

The simulation objects have various different numerical attributes related to the simulation. In common for all simulation objects are attributes representing flows, pressures, and temperatures. The equations and attributes for the heat exchanger sections will be described in detail.

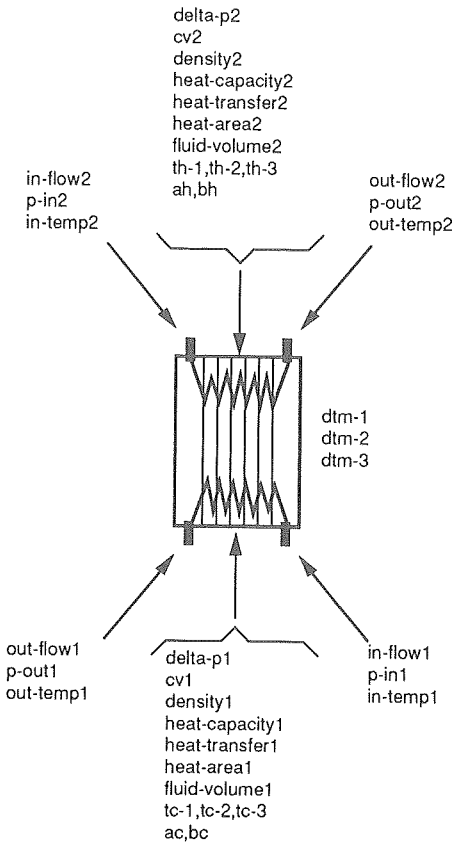


Figure 7.12 S-plate-heat-exchanger

S-plate-heat-exchanger: The icon of this class and its attributes are shown in Fig. 7.12. The suffixes 1 and 2 are used to differentiate the two sides of a heat exchanger. The suffixes 1, 2, and 3 in dtm, th, and tc are used to split the heat exchanger into three sections which is necessary for the temperature equations in the simulation.

The attributes concerning temperature, pressure, flow, density, heat capacity, heat-area, and fluid-volume are rather self explanatory. In the rest of this section only the attributes that may be difficult to understand will be explained.

The cv attribute corresponds to the flow resistance, i.e., the corresponding valve equivalent of the heat exchanger. The $delta - p$ attributes are the pressure drop across the heat exchanger. The heat-transfer attributes are the total heat transfer coefficients from one side to the other. The attributes th, tc, dtm, ah, ac, bh, and bc are used in the temperature equations for the heat exchangers. The simulation equations for this class are generic and can be found in its subworkspace.

There are three subclasses of s-plate-heat-exchanger. The reason for this is that some heat exchangers have water connections at both sides, some have product connections, and some have one of each type of connection. These subclasses have no attributes of their own but they inherit the attributes of the superior class.

With respect to pressures and flows the heat exchanger is modelled as a valve, i.e., the flow through the heat exchanger depends on the pressure drop across the heat exchanger and a constant, the *cv*-number. The simulation formula for the pressure drop is identical to the pressure drop of a valve, that is

$$P_{in} - P_{out} = \frac{\rho q^2}{C_v^2}$$

The simulation formulas for the temperature in the heat exchangers are a bit more complicated. A dynamical model for counterflow heat exchangers taken from Åström (1989), described by the following equations has been used.

$$\frac{\partial T_h}{\partial t} + a_h \frac{\partial T_h}{\partial (x_h/l)} + a_h b_h (T_h - T_c) = 0$$
$$\frac{\partial T_c}{\partial t} + a_c \frac{\partial T_c}{\partial (x_c/l)} + a_c b_c (T_c - T_h) = 0$$

where

$a = q/V$ inverse residence time

$b = \dfrac{kA}{q\rho c}$ number of heat transfer units

$q =$ flow, m^3/s

$V =$ fluid volume

$A =$ heat transfer surface

$k =$ total heat transfer coefficient

$\rho =$ density, kg/m^3

$c =$ specific heat capacity

$l =$ the length of the heat exchanger

$x =$ the space variable

The indices c and h correspond to the cold and the hot side of the heat exchanger.

To simulate the model with reasonable effort the partial differential equation is approximated by a set of ordinary differential equations by discretizing the space variable. The discretizing is done by segmenting both the hot and cold side of the heat exchanger into three equally sized parts. The heat exchanger in Fig. 7.13 shows the hot and cold side and how the segmentation is done.

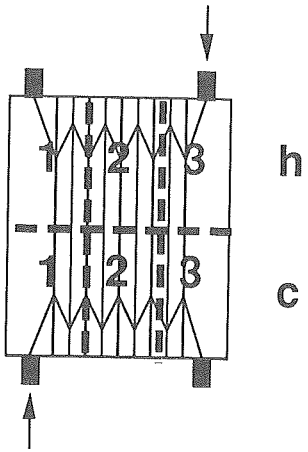


Figure 7.13 Heat exchanger discretization

The approximation now becomes

$$\frac{dT_{hi}}{dT} = a_{hi}(3(T_{h(i+1)} - T_{hi}) - b_{hi}\Delta T_{mi})$$
$$\frac{dT_{ci}}{dT} = a_{ci}(-3(T_{ci} - T_{c(i-1)}) - b_{ci}\Delta T_{mi})$$

where

$$\Delta T_{mi} = \frac{\Delta T_{chi} - \Delta T_{ch(i-1)}}{\ln(\Delta T_{chi}/\Delta T_{ch(i-1)})}$$
$$\Delta T_{chi} = T_{h(i+1)} - T_{ci}$$

Finally, the last part is further approximated as

$$\Delta T_{mi} = 0.5(\Delta T_{chi} + \Delta T_{ch(i-1)}).$$

The index i refers to one of the three segments that the heat exchanger is divided into.

When implementing the equations in G2 the following notation was used.

$$a_{hi} = Ah$$
$$a_{ci} = Ac$$
$$b_{hi} = Bh$$
$$b_{ci} = Bc$$
$$\Delta T_{mi} = Dtmi$$
$$T_{hi} = Thi$$
$$T_{ci} = Tci$$
$$i = 1, 2, 3$$

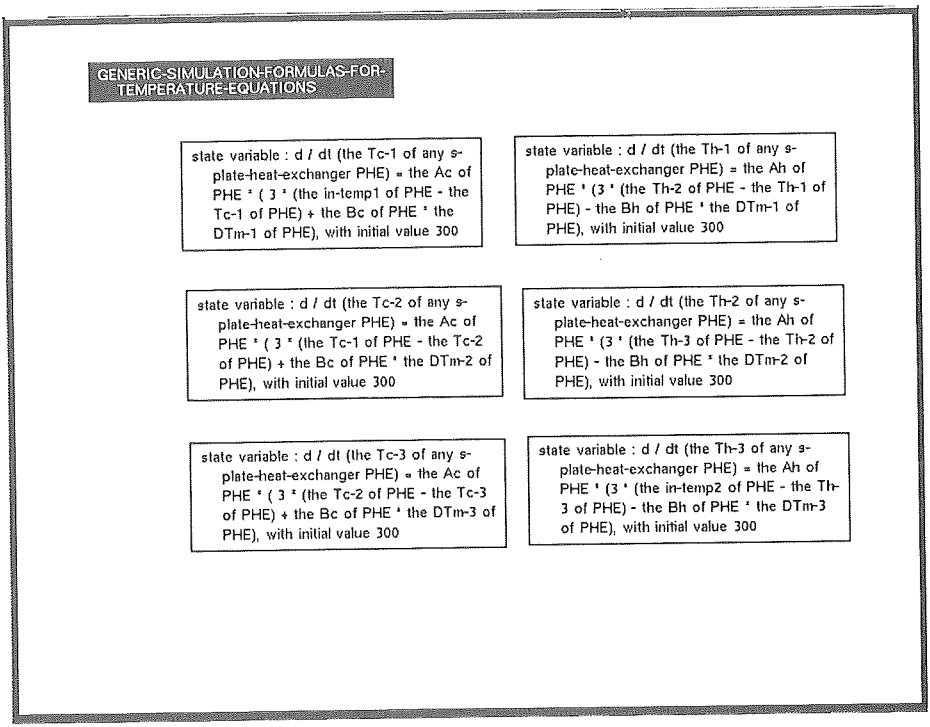


Figure 7.14 G2 generic heat exchanger temperature equations

Notice that there are no indices in the first four lines. This is because the volume, the heat transfer surface, the flow and the heat transfer coefficient are the same for all segments of one side. The above equations are represented in G2 as generic simulation equations that apply to all instances of s-plate-heatexchanger. Some of the above equations are shown in Fig. 7.14.

It may seem to be a rough approximation to divide the heat exchangers in only three parts, but it proved to be sufficient to give a realistic behaviour of the temperature in the heat exchangers. Reducing the load on the simulator is another reason for having as few segments as possible.

When the flow on one side of the heat exchanger is zero, the temperature on the other side is the same in all three segments, i.e., the heat exchanger is inactive when simulating the temperature.

When tuning the heat exchangers with respect to the temperature, the flow and the pressure, the heat transfer coefficient (k) and the cv-number were adjusted until the simulated temperatures agreed with "real" values. This was a rather

cumbersome and time consuming procedure. One reason for this was that the temperature equations easily turned unstable. The equations are only tuned for the production phase because it is the most interesting phase to simulate. This means that in the other phases, the temperatures may not agree fully with the correct temperatures.

A solution to the problem of the equations turning unstable would be to include a set of equations for each phase. Each set of these equations can be tuned for one particular phase and they would agree with the correct temperatures in every phase. There are two reasons why we have not done this. First, the deviation from the correct temperatures is rather small and does not affect the simulation in any important way. Second, the number of equations would be very large.

Other simulation objects: The other important simulation objects are s-pumps, s-valves, and s-balance-tanks.

The s-pump class has two subclasses: centrifugal pumps and positive displacement pumps. The centrifugal pumps are simulated with a predefined increase of pressure which is assumed to be flow independent. The displacement pumps are modelled as constant flow sources. Neither of these pumps affect the temperature of the liquid.

S-valve has three subclasses: s-1-to-1-valve, s-1-to-2-valve, and s-2-to-2-valve. S-1-to-1-valve is a valve that has one inport and one outport. The flow through the valve can either be regulated manually or automatically. S-1-to-2-valve is a valve with one inport and two outports. It is used to control the direction of the flow. The incoming flow is directed to one of the two outports. S-2-to-2-valve is a valve with two inports and two outports. This valve also controls the direction of the flow. The two outports can be connected with either one of the two inports.

The s-1-to-1-valve class has the following three subclasses: s-constant-flow-valve, s-constant-pressure-valve, and s-regulator-valve. S-constant-flow-valve is a regulator valve that tries to maintain a constant flow through the valve. This valve has the attribute flow-ref which is the reference flow. S-constant-pressure-valve is a regulating valve that tries to maintain a constant pressure at the inport. It has the attribute p-in-ref which is the reference pressure. S-regulator-valve is a valve where the throttle that controls the flow through the valve is controlled by an external regulator.

The regulator valves are modelled as ideal valves with a pressure drop that depends on the flow through the valves. The model for the pressure drop is given by

$$P_{in} - P_{out} = \frac{\rho q^2}{C_v^2}$$

where C_v is the flow coefficient. The flow coefficient corresponds to the flow

resistance, R, through a valve. The flow coefficient is given by:

$$C_v = \frac{1}{\sqrt{R}}$$

By changing the cross area of a valve you change the flow coefficient (i.e., the flow resistance) and thereby the flow and pressure drop across the valve.

The s-balance-tank class describes the two tanks that are used in the process. The model of the balance tanks includes dynamic simulation of the tank levels and the mechanical controller that controls the inlet flow.

Implementation of the simulation model

The Steritherm process consists of two main lines. The water line is used for heating and cooling of the product. The product line is where the product flows during the production phase. The lines contain several valves that have different positions in the different phases. This means that the flow goes through different objects depending on the current position of the valves. Furthermore, there are some valves that sometimes are active and control the flow or pressure in the lines, and sometimes are inactive and act as if they were not there. These are some of the reasons why the simulation equations are difficult to write.

When simulating flow and pressure some problems arise. To calculate the flow one has to look at the process from a global point of view. One must know all the different objects that are involved in a particular line (compare with resistances and currents in an electrical circuit), which objects increase the pressure, which objects decrease the pressure, and so on. To be able to handle these problems, a few global functions that calculate the flow in the different lines and during the different phases have been implemented. By computing the flow (through a call to a global function) at the first object in a line, for example the out-flow attribute in the product tank in the product line, the flow can be propagated together with the calculated pressure and temperature through every object in the line. With this approach, the components in the lines can easily calculate their pressure drops which also means that the pressure at different points in the process can be calculated. The different lines in the process can be seen as lines where the pressure at at both ends are known, with pumps that increase the pressure and valves and heat exchangers that decrease the pressure. If the pressure at both ends and the total flow resistance in the line is known the flow can be calculated.

Burn-on simulation

The burn-on is only simulated in heat exchanger I, because the effect of the burn-on is most observable there. The burn-on simulation is only active in the production phase. The burn-on is simulated in two different ways. First, through

manipulating the heat transfer coefficient of the heat exchanger, and second, through manipulating the *cv*-number.

The heat transfer coefficient is manipulated through subtracting a state variable, temp-burn-on-state-variable, from the coefficient. The state variable increases with time. The increase rate is given by another variable, temp-burn-on-characteristic. The value of this variable depends on the actual product in the process. For a product that causes fast burn-on, this characteristic variable has a greater value than for a product that causes slower burn-on. Decreasing the heat transfer coefficient will require a higher temperature on the heating side of the heat exchanger to keep the temperature on the other side of the heat exchanger constant.

The *cv*-number of the heat exchanger is manipulated in the same way as the heat transfer coefficient (except that the variables now are called pressure-burn-on-state-variable and pressure-burn-on-characteristic). Decreasing the *cv*-number is the same as if the cross area of the pipe in the heat exchanger gets smaller, which is what actually occurs during burn-on. This will result in an increase in the pressure drop.

As mentioned before, the burn-on is only simulated in the production phase; but, in all phases after the production phase the effect of the burn-on remains. The effect is still there until the process has been cleaned. There are two different kinds of cleaning in the Steritherm process, an intermediate cleaning (AIC) and a final cleaning (CIP). After an intermediate cleaning the effect of the burn-on is reduced by 50%. This is managed in G2 by using a whenever rule that tests when the current phase is intermediate cleaning. After a final cleaning the effect of the burn-on has vanished.

7.3.2 Hierarchical levels and views

Hierarchical levels are used to represent knowledge about the Steritherm process at different degrees of resolution. The highest level in the hierarchy represents the whole plant, which in this case is assumed to be a dairy. The lowest levels represent the basic entities of the process, e.g., pumps. valves, heat-exchanger sections, etc. With hierarchical levels, knowledge can be stored at the most appropriate level in the knowledge base.

An object, e.g., the entire Steritherm process or a pump, contains knowledge and information of different kinds that belongs to the object. This knowledge or information can be the internal structure of the object, a description of the object in terms of its functions and goals, a photo of the process, or textual information about the process. These different types of information about a single object are stored in different views.

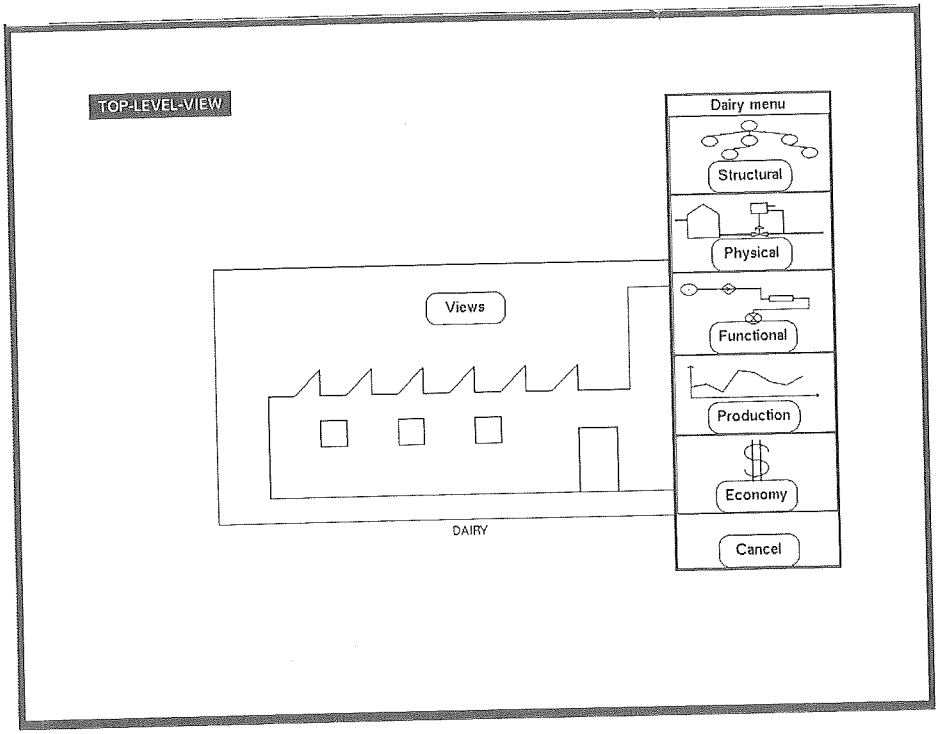


Figure 7.15 The dairy object and its view menu

The object-oriented model of Steritherm

At the highest level in the hierarchy there is a single object representing the whole dairy. On top of the icon representing this object there is a button named "views". When clicking on this button a graphical view menu appears on the screen. When clicking on the icon, outside the button, the attribute table of the object appears. This table could contain attributes which are common to all views of the object. Fig. 7.15 shows the dairy object and its view menu.

The view menu contains menu choices for the different views of the dairy object. The menu choices are represented as icons with a button on top of them. When clicking on the button, the view that the icon represents pops up. When clicking on the icon, outside the button, the attribute table of the view object shows up. This table contain attributes concerning this particular view only. The attribute table of an object contains references to the different view objects associated with the object.

On every view (except the top level view) there are two buttons. Clicking on

the "Superior view" button is equivalent to moving up one level. Clicking on the "Superior menu" button causes the view menu of the above object to be shown and from this one can move around horizontally between different views of the same object.

The menu of the top level object (the dairy) consists of the following menu choices:

- Structural: This view gives a more detailed description of the inner structure of the dairy. The structural view is equivalent to what is called the topological view in the system concept.
- Production: This view may show information concerning raw product consumption, production of the different end products, process or line utilization and status, historical statistics, production plans, etc.
- Economy: This view may show the economical aspects of the dairy, e.g., return on investment, profit, down-times, consumption of electricity, steam, water, etc.
- Physical: This view may show photographs and drawings of the plant, room and floor layouts, etc. This view is similar to the geographical view in the concept.
- Functional: This view will give a functional description of the plant using the MFM formalism.

Of these views, only the structural view is fully implemented in the system. The other views are only present to show the possibilities. The structural view consists of a number of different objects, that give a more detailed description of the dairy. Fig. 7.16 shows the structural view of the dairy. The objects at this level are briefly described below.

- Administration: This object consists of information regarding the staff of the dairy, e.g., employments, shifts, names, addresses, wages, etc.
- Silo tanks: In this object, there is information about the different raw products used in the dairy.
- Spare part inventory: This object represents a database about all spare parts that are available in the dairy.
- UHT line: This object represents the production line for UHT treated products.
- Pasteurizer line and cream line: These two objects represents two other kinds of production lines in the dairy.

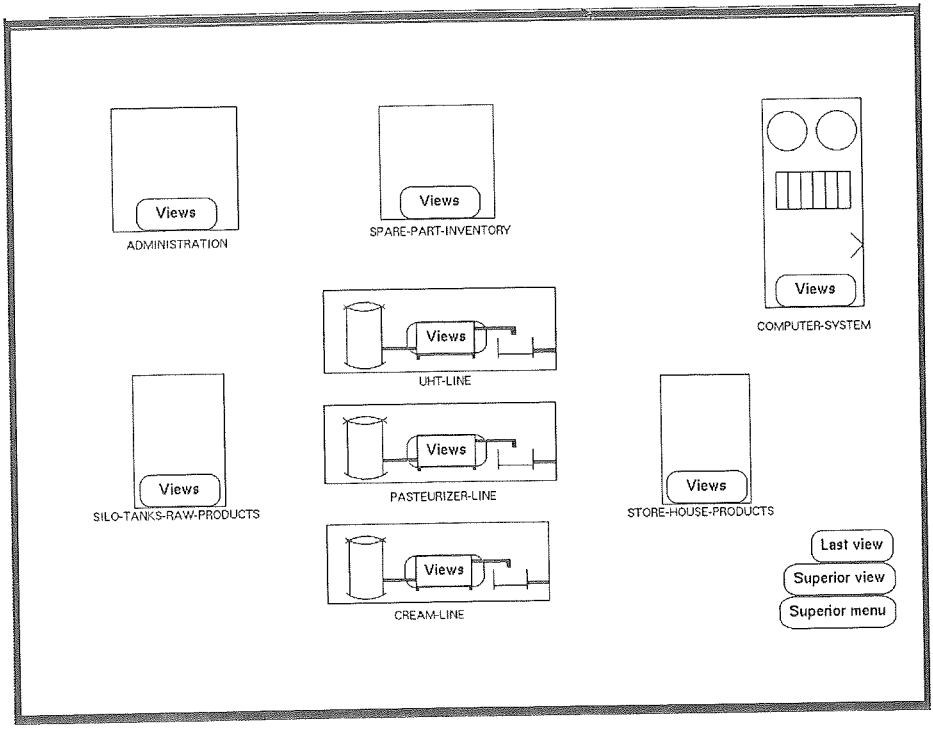


Figure 7.16 The structural view of the dairy object

- Computer system: This object contains information about the whole computer system used in the dairy, e.g., networks, terminals, process control units, operator consoles, etc.
- Store house products: This object consists of information about the end products.

Of these objects, it is only the UHT-line that has further internal structure. The menu of the UHT-line contains the same type of views as the dairy menu. Of these menu choices it is only the structural view that is implemented. The structural view of the UHT-process consists of a silo-tank, the actual Steritherm process, and the packing machine. Again it is only one object, the Steritherm process, that is implemented. This object has the same views as previously described and the additional menu choice *operation*.

The Steritherm structural view: The Steritherm structural view contains the process schematic. The schematic, shown in Fig. 7.17, contains intercon-

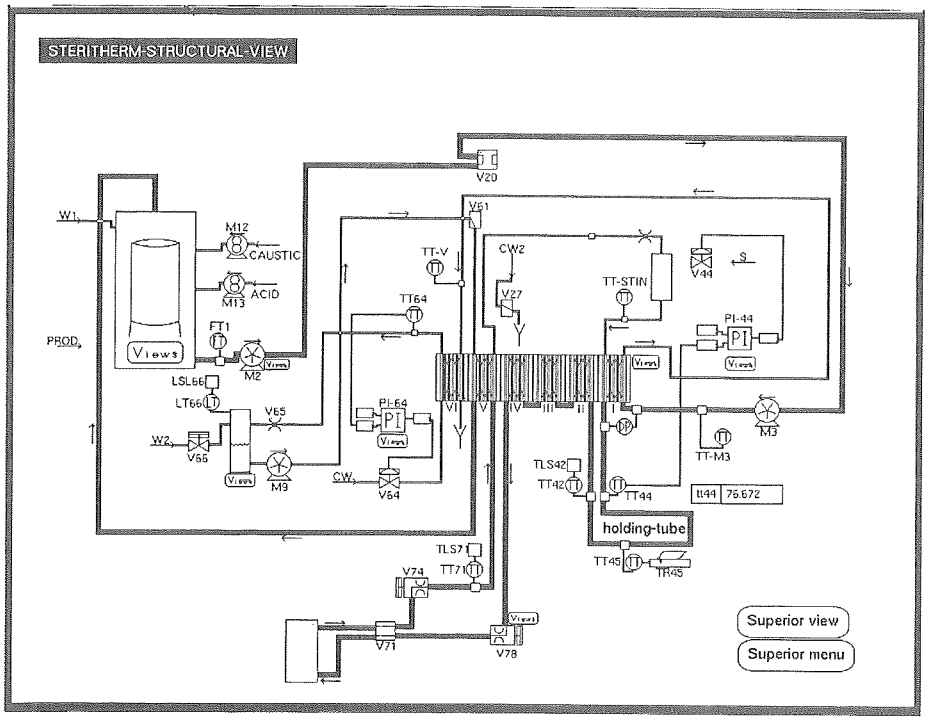


Figure 7.17 The Steritherm structural view

nected objects representing the process components and the control system components (PID controllers and guards). The process component objects may contain various kinds of information such as qualitative information about the state of the components, textual information, data sheets, maintenance information, etc. Some of the process components have an internal view structure in the same way as previously described. The Steritherm structural view in the G2 prototype is roughly equivalent to the sum of the topological views of the main product system, the warm water system, the cold water system, and the steam system.

Animation is used to indicate current valve position, tank levels, and whether the pumps are in operation or not. Colours are used to indicate the current media in the product line (water or product). Pipes that are not active in the current phase, i.e., they have no flow, are hidden. The use of animation in the knowledge base does not agree with the system concept. The graphical interface to the knowledge base in the G2 prototype corresponds to the knowledge base browser. In the concept, animation should only be used in the different user interfaces.

The sensors have graphs showing their history values on associated workspaces. The PID controllers are implemented as generic discrete simulation equations which are executed by the G2 simulator. G2 version 2.0 allows the use of procedures to represent the PID algorithm. This would give a more realistic implementation for the code of a PID controller. The regulators have different views for showing the trend curves of the controller, inspecting the controller code, and changing the controller parameters.

The Steritherm functional view: The functional view will contain the MFM model of the Steritherm process described in Section 7.3.6. At the moment the MFM model has not been integrated with the rest of the G2 prototype.

The Steritherm production view: The production view contains a graph showing the different process phases during the last 24 hours.

The Steritherm operation view: The operation view contains a menu choice for selecting between the sequence net view, the alarm tree view, the model-based diagnosis view, and the product following view. These views are further described in the rest of the chapter.

7.3.3 Sequential logic using Grafcet

The sequence net view contains the sequential logic portion of the control system. The sequence net is implemented using the Grafcet formalism (GREPA, 1985).

The Grafcet formalism consists of steps, transitions, and parallel bars. The steps contain the actions that should be performed in a certain phase, i.e., opening and closing of valves, starting and stopping pumps, etc. Steps could be structured, i.e., consist of an internal step-transition sequence. Transitions contain the conditions for going from one step to the next. Parallel bars are used to split up a sequence into parallel branches and to subsequently join them together again. Markers are moved around in the sequence net indicating the current active step. The Grafcet net for the Steritherm process is shown in Fig. 7.18. The sequence net contains the four major phases of the plant: sterilization, production, intermediary cleaning (AIC), and final cleaning (CIP). It also contains the phase `Sterile-w`, where the process waits for the operator to decide the next phase after sterilization. Each of the major phases contain several substeps.

G2 implementation

The Grafcet implementation is based on G2's activatable subworkspaces. Steps have subworkspaces containing either an internal structure or rules that decide the actions that should be performed in the step. The workspace is activated when the step begins and deactivated when the transition following the step is true. Actions that should only be executed once when the step is started are

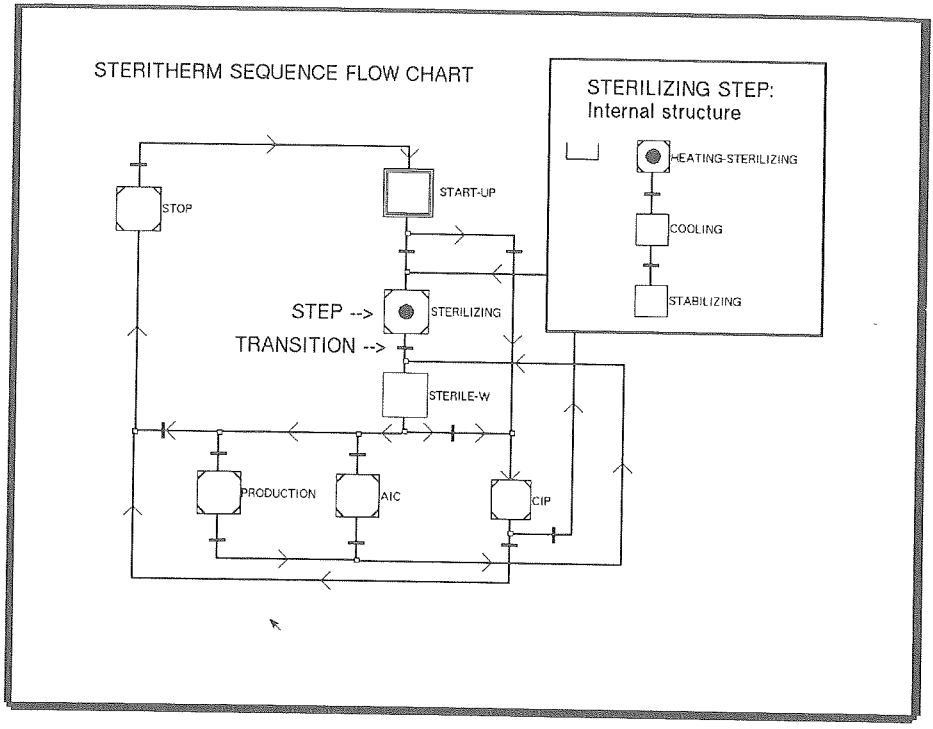


Figure 7.18 The sequence net

implemented as G2 *initially* rules. Actions that should be performed continuously when the step is active are implemented as scanned rules. Since rules that are executed immediately before a workspace is deactivated are not available in G2, there is no possibility to specify actions that should be executed before the step terminates.

Transitions also have activatable subworkspaces. Each transition subworkspace contains one rule that tests the transition condition. The rule is scanned every second. A transition is only active, i.e., its rule is scanned, when it should be so according to the Grafcet standard. A rule condition can test on the time spent in the previous step.

The Grafcet net is executed by means of 12 generic rules, mainly of the *whenever* type. The rules can be divided into two categories. Rules in the first category are activated when the transition condition of a transition becomes true. The second category is invoked whenever a marker object is moved. A rule in the first category performs the following actions.

1. The marker is moved from the current step to the next step after the transition that just has been fulfilled.

2. The subworkspace of the "old" step is deactivated.

3. The subworkspace of the "new" step is activated.

Additional actions are required for structural steps and parallel bars. The rules in the second category performs the following actions whenever a marker has been moved.

1. Sets the time-of-enter attribute of the step that has received the market to the current time.

2. Deactivates the subworkspaces of all transitions connected directly after the step from which the marker was moved.

3. Activates the subworkspaces of all transitions connected directly after the step to which the marker was moved.

As before, additional actions are required for structured steps and parallel bars.

The small numbers of generic rules that are needed to implement the Grafcet formalism show the power of G2's generic rules. The Grafcet formalism has many applications apart from the pure sequential control for which it is mainly used here. It could, e.g., be used to structure monitoring and diagnosis rules after which process phases they should be active in. This is partly done in the prototype. The rules for burn-on monitoring are placed on the subworkspace of the production step. Thus, they are only active in that process phase.

Additional features

It is desirable to be able to connect other types of knowledge and information to steps and transitions. To the steps it should be possible to associate textual descriptions of what is happening in that step, information about normal conditions or constraints that should hold in the step, information about faults that may occur in the step and which the operator should be prepared for, etc. This has not been implemented.

7.3.4 Monitoring and alarm analysis

Within the master thesis project two monitoring and diagnosis style applications were implemented: a simple rule-based system for on-line monitoring of the burn-on and a more advanced alarm-tree based alarm analysis system.

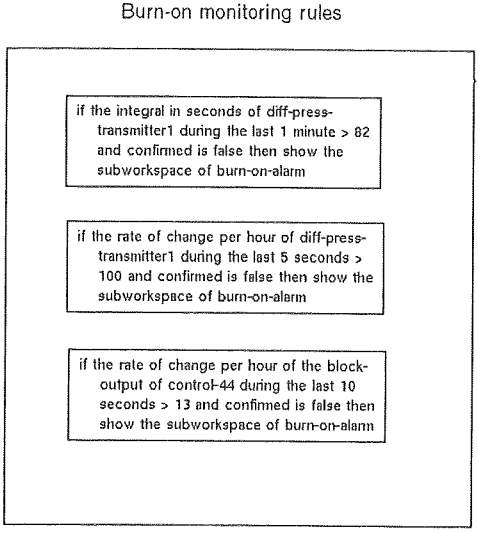


Figure 7.19 Monitoring rules

Rule-based burn-on monitoring

The burn-on that occurs in the heat exchanger sections cannot be directly measured. In a real Steritherm process the time between cleanings is predetermined depending on the product. Products that are more sensitive to burn-on or for which the burn-on rate is high have shorter production times.

The motivation behind the rule-based burn-on monitoring is to continuously monitor the trend curves of variables that indirectly give indications of burn-on. The two variables used are the differential pressure on the product side of heat exchanger section I and the control output signal of PID controller PI-44 that controls the sterilization temperature in the heating tube. Burn-on causes fouling on the inside of the heat exchanger pipes which decreases the cross section area and thus increases the pressure drop over the heat exchanger. Burn-on also decreases the heat transfer coefficient in the heat exchanger. To maintain the sterilization temperature, the temperature of the hot water must be increased. PI-44 achieves this by opening the steam valve V-44.

The supervision is performed by three rules. Two rules supervise the differential pressure, one using the mean value and the other using the rate of change. The third rule supervises the controller output using the rate of change. The action of the rules causes a workspace to appear which contains a recommendation to the operator to manually stop the production. The rules are shown in Fig 7.19. It would have been natural to also include a rule that warns the operator when

the production time exceeds a certain specified maximum-allowed-continuous-production-time parameter.

The rules are stored in the production step of the Grafcet sequence net. Alternatively or additionally one would like to store the rules in a rule block on the process schematic with graphical connections to the differential pressure transmitter and to the controller output.

Alarm-tree based alarm analysis

The motivation for the alarm analysis system is to assist the operators and service personnels with finding the physical fault that has caused an alarm. The Steritherm process control system generates alarms to the operators when, e.g., the temperature in the holding cell is lower than the sterilization temperature, 137°, or when the level in the product balance tank falls below the low level limit.

The most serious alarm is the low temperature alarm in the holding tube. The control system has built-in alarm logic that ensures that no unsterile product is packed in the packing machine. The alarm logic includes a sequence that involves closing valve V71 and thus recirculating the product to the balance tank. This alarm logic has been implemented in the G2 prototype. When the process is back in a safe state it is the task of the operators and the maintenance personnel to find out what might have caused the alarm.

The fault localization in the alarm analysis system is based on alarm trees originating from the Steritherm designers at Alfa-Laval. The low temperature alarm part of the tree which is the only part that has been implemented in G2 is shown in Section A.5.

The alarm tree is made up of a number of nodes, where the top node represents the alarm and where the leave nodes represent either a physical fault, e.g., a broken pump rotator, or a functional fault, e.g., the steam boiler is not working. The intermediate nodes describe only functional faults. These functional faults differ from the physical faults in the leaves in the sense that it is possible to further encircle the fault from these nodes. There are two kinds of functional faults.

In the first case, the system knows for sure that the fault must be found in the subtree of this functional node. A node like this can represent the fact that the pressure in the holding tube is too low. This in turn is a consequence of a question to the user, e.g., "Is the pressure in the holding tube low?". By knowing this, it is possible to prune all other parts of the tree and only concentrate on this subtree. If it turns out that no fault could be found, this does not mean that the fault may be found in another part of the tree. Instead this subtree is incomplete and has to be reconstructed if it should be able to locate the real fault.

The second type of functional fault in the intermediate nodes is used to divide the analysis into smaller and smaller parts. In these nodes, the system cannot

know if the fault is in the subtree of the node. The system tries to verify the fault hypothesis of the node using backward chaining. If it fails, the system will try to locate the fault in other nodes in the rest of the fault tree.

To verify whether a specific leaf node represents the fault or not, a question is asked to the user. In the question, the possible fault is described. If the answer is positive, the fault analysis is over, otherwise the analysis continues. The order in which the nodes are examined is determined by depth-first traversal. The nodes are arranged so that the nodes representing the most probable fault are examined first.

The questions in the leaf nodes differ from the questions in the intermediate nodes. The questions in the intermediate nodes are used to lead the system in the right direction, and by that, exclude the other parts of the tree. The questions in the leaf nodes are used to verify whether the node represents the actual fault or not.

The alarm analysis is implemented with fault objects and rules. The objects represent the nodes in the tree, i.e., the alarms, the functional faults, and the physical faults. The fault objects are graphically interconnected to form the alarm tree shown in Fig 7.20. The rules are used to traverse the alarm tree, ask questions to the user, and take appropriate action when an answer has been given. The rules are invoked using a combination of explicit invocation, backward chaining, and forward chaining. The analysis is started on demand, when the operator clicks on a button.

Extensions: Several extensions to the alarm analysis are possible. The rules that traverse the alarm tree are not general. They build up a structure parallel to the fault tree. It is desirable to combine these structures. Animation and colours could be used to show the status of the alarm tree traversal. Explanations to the questions asked by the system have not been implemented. The physical faults are associated with process components, e.g., they represent some fault in a specific valve or pump. This connection is not explicit in the knowledge-base. Establishing the connection would make it possible to go between the fault object in the fault tree and the process component in the process schematic. It would also be possible to, using, e.g., colours, dynamically indicate the possible fault areas in the process schematic as the physical faults are encircled and finally localized.

7.3.5 Model-based Diagnosis

After the master thesis project was finished, Thomas F. Petti from the University of Delaware joined the project and added a G2 implementation of his approach to model-based diagnostics – the Diagnostic Model Processor method (DMP) described in Section 4.5 – to the G2 prototype.

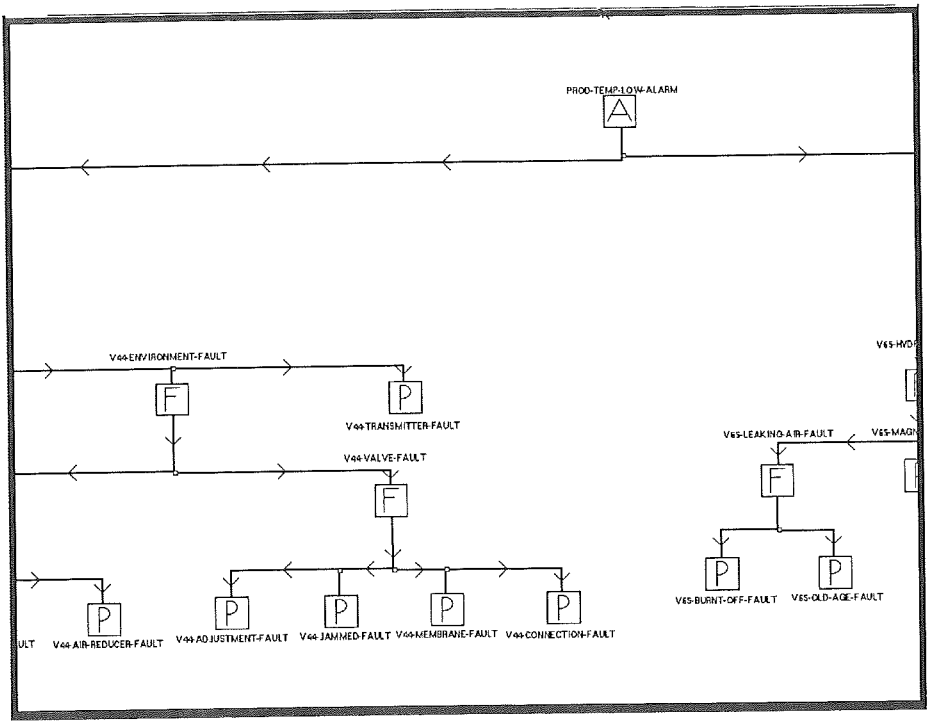


Figure 7.20 A part of the alarm tree

Object and Formula Definitions

The diagnostic model processor is implemented in G2 using three basic objects. The first is the model equation, the second is the assumption, and the third is a dependence which characterizes the relationship between the equations and assumptions. Figure 7.21 shows the object icons and their relationships; the dependence object is the connection between the equations and assumptions. The table for the model equation object is also shown. The attributes for this object include the name of the equation, a formula used to calculate the residual, a formula to calculate the tolerance limits, and the satisfaction value (sf). As described in the figure, the value of sf is calculated using Equation 4.2 for all model equations through a single generic formula. This generic equation is shown in figure 7.22; the equation is written so as to perform the calculation for all instances of the object "model-equation".

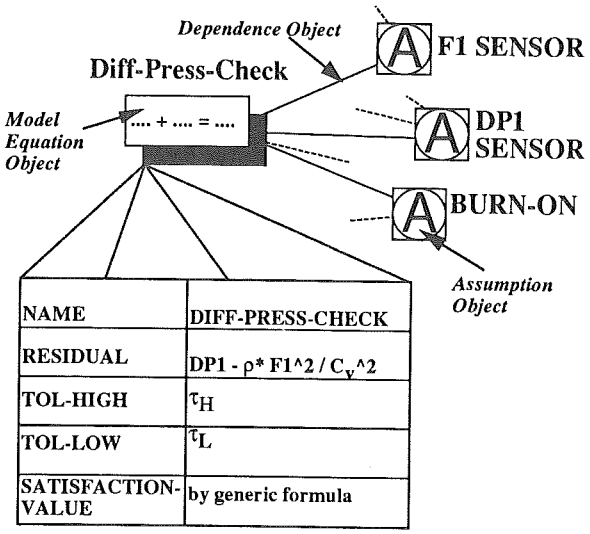


Figure 7.21 G2 objects and their relationships.

```
let the satisfaction-value of any model-
    equation M = (if the residual of M > 0
    then (the residual of M / the tol-high of
    M) ^ n / (1 + (the residual of M / the tol-
    high of M) ^ n) else -1 * (the residual of
    M / the tol-low of M) ^ n / (1 + (the
    residual of M / the tol-low of M) ^ n))
```

Figure 7.22 Example of a generic formula, calculation of *sf*.

The other object types have similar tables which describe the attributes associated with the object. The dependence object (connection) has attributes which characterize the type of relationship (implicit or explicit), a formula to calculate the partial derivative of the equation with respect to the assumption, and a sensitivity attribute which is handled by a generic representation of Equation 4.3 for each dependence. Each assumption object has a name and a failure likelihood attribute. The failure likelihood is calculated for each assumption by another generic equation (Equation 4.4). The G2 representation of this equation is shown in Fig. 7.23. A significant feature of the implementation is the fact that the process model can be examined and maintained through these objects with no concern for the underlying methodology, which is handled by the generic formulas.

A variation of the model equation object described above is also used which allows

Calculation of failure-likelihood

```
let the failure-likelihood of any assumption
    AS = (the sum over each dependence D
    connected to AS of (the sensitivity of D
    * (if there exists a model-eq-type at an
    end of D and the status of the model-eq-
    type at an end of D is active then the
    satisfaction-value of the model-eq-type at
    an end of D else 0))) * (if the sum over
    each dependence D1 connected to AS
    of (abs (the sensitivity of D1)) /= 0 then
    (1 / the sum over each dependence D2
    connected to AS of (abs (the sensitivity
    of D2))) else 0)
```

Figure 7.23 Example of a generic formula, failure likelihood.

for operator interaction with the methodology. Since in the Steritherm process there are many sensors which are not connected to the control computer, these values could be used if the operator supplied them. A type of model equation which can be activated when certain quantities are supplied is therefore also available. These can be used as additional evidence when a failure is detected and further discrimination between faults is necessary.

A total of 18 model equations are used on the Steritherm process with connections to 17 assumptions. Additionally, 7 model equations which are activatable with supplied values are available. Some of the assumptions which could be applied to these equations were not considered (e.g., piping leaks); it should therefore not be expected that the analyzer be capable of diagnosing these failures.

The model equations and fault assumptions are stored on a model-based diagnosis subview under the operation view of the Steritherm process. The graphical dependence connections are normally hidden. Model equations and fault assumptions have menu choices that by which the dependencies of that object can be shown and hidden. The workspace is shown in Fig. 7.24 with most of the dependencies hidden.

Presentation of Faults

Failure conditions are indicated on the process control schematic through the creation of a dynamic object called a fault alert. A fault alert is created when the magnitude of any failure likelihood exceeds 0.5. The fault alert appears as an

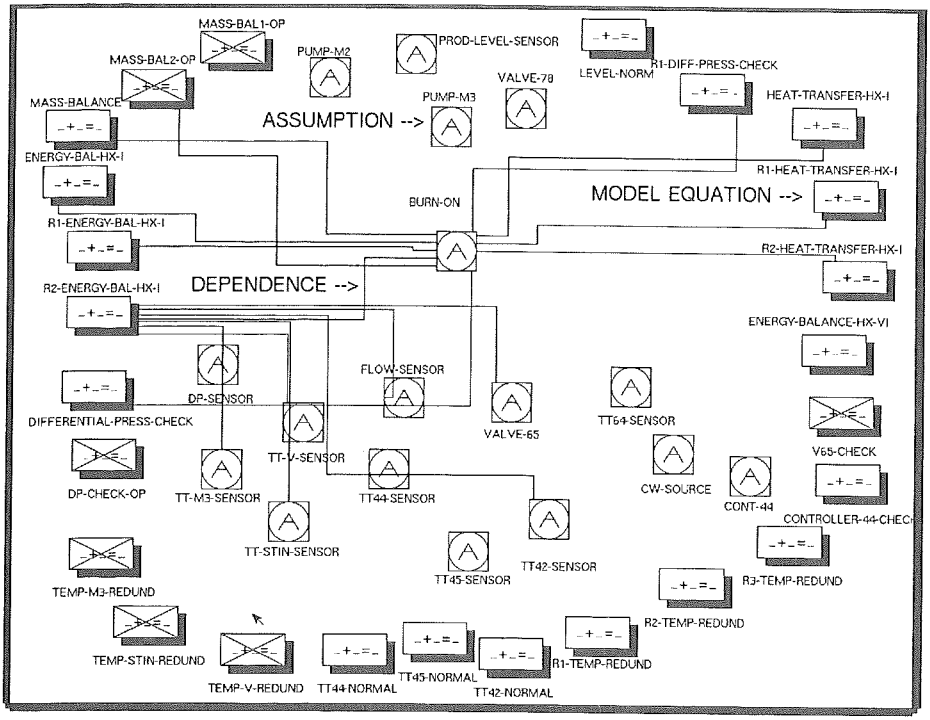


Figure 7.24 The model-based diagnosis workspace

exclamation mark on the schematic near the possible fault occurrence. The fault alert is orange if F is between 0.5 and 0.8 in magnitude and red if it is greater than 0.8. Associated with each alert is a source (the assumption it is indicating a problem with) and a menu access to a graph of the F value. Figure 7.25 shows a schematic with an indication of a failure of the temperature sensor TT42 near the pointer, also shown is a graph of F for the TT42 assumption.

Improved fault discrimination

As previously discussed, the diagnostic model processor allows for the detection of non-competing multiple faults. This often leads to several assumption's failure likelihood exceeding the 0.5 presentation limit when a single failure has occurred. This is illustrated by the simple example in Fig. 7.26. If a_1 is the true fault, and all sensitivities are similar, both equations, c_1 and c_2 would have large values of sf. This would lead to high failure likelihoods of both a_1 and a_2, simply due to the connection scheme. More complicated, but similar, cases often occur.

To limit the number of faults presented to the operator and to direct attention

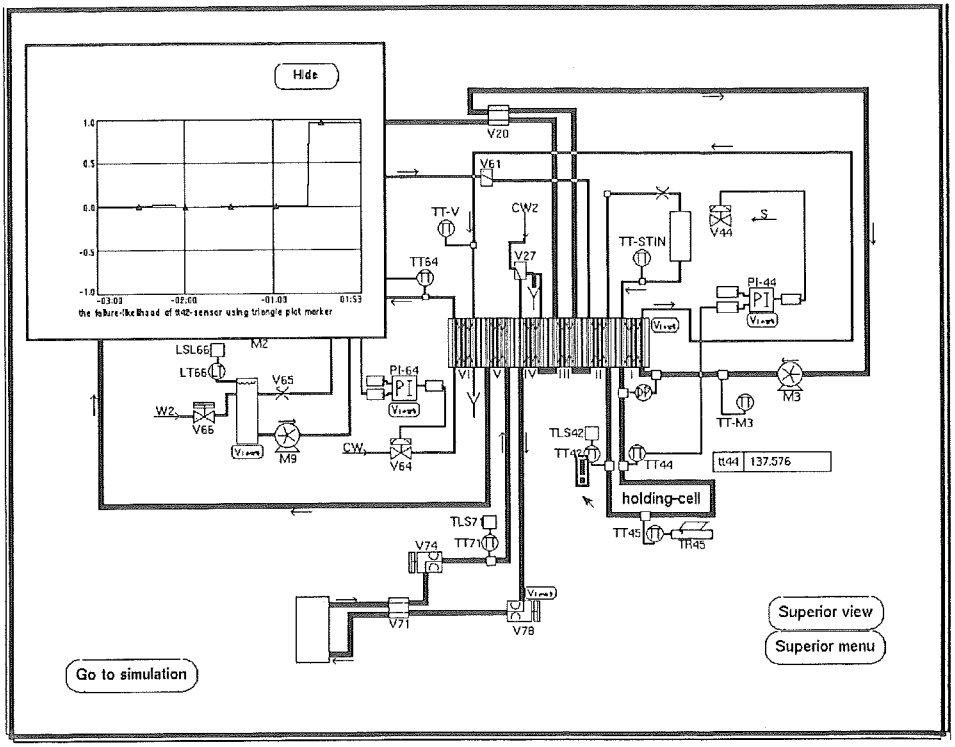


Figure 7.25 Process schematic showing a failure (high) of sensor TT42.

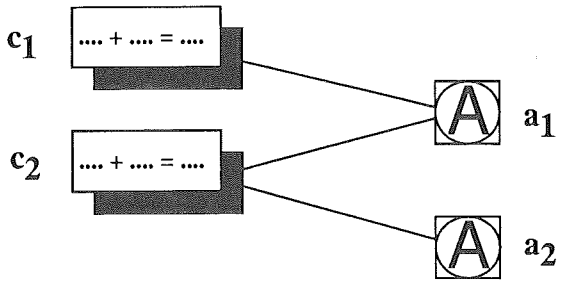


Figure 7.26 Equations and assumptions illustrating discrimination problems.

to the most probable conditions, a procedure is used to check the causal relationships between the assumptions. The procedure basically assumes a fault condition and checks the expected behavior of the model equations to see if other failure likelihoods should exceed 0.5. If this is the case, the assumption is said to

"explain" the appearance of the other assumptions.

The procedure call is made with two assumptions as arguments (a_1 and a_2) whose F values exceed 0.5. The purpose is to try to establish the relation a_1 is "explaining" a_2. The procedure uses the most sensitive model equation which is connected to a_1 to estimate the magnitude of the deviation of a_1, assuming that the deviation of the equation is caused solely by a deviation of a_1.

$$DEV(a_1)_{est} = \frac{e_j}{\frac{\partial c_j}{\partial a_1}}$$

where c_j is the most sensitive equation connected to a_1. Using this estimate of the deviation and the partial derivatives of the connections surrounding a_1, the residuals and satisfaction values of all equations connected to a_1 can be estimated. The residuals are estimated based on $DEV(a_1)_{est}$:

$$e_{j_{est}} = DEV(a_1)_{est} \frac{\partial c_j}{\partial a_1}$$

and from these values, Equation 4.2 is used to estimate satisfaction values, sf_{est}. Finally, the vector sf_{est} is used to calculate a failure likelihood of a_2 using Equation 4.4. If the calculated value of F_2 is close to the actual value of F_2 ($|F_2 - F_{2_{est}}| < 0.2$), then a_1 is said to "explain" a_2.

If any assumption is not "explained" by any other assumption, it is considered top-level and is marked by a green rectangle around its fault alert. Also, if assumption a_1 "explains" a_2 and a_2 also "explains" a_1, they are both considered top-level and are indicated as such. Finally, if any assumption cannot "explain" itself, then presentation of this failure is suppressed completely (no fault alert).

Relating this procedure to Fig. 7.26, if a_1 is the actual failure (all sensitivities being about the same), the procedure would identify assumption a_1 as "explaining" the appearance of assumption a_2. a_2 would not be able to "explain" a_1 back, so only a_1 would be considered top-level and marked with the green rectangle. Conversely, if a_2 were failing, the failure likelihood of a_1 may exceed the 0.5 presentation threshold. Since, model equation c_1 would remain satisfied, assumption a_1 would not be able to "explain" the appearance of itself, and presentation of its fault alert would be suppressed.

This procedure limits the number of fault alerts on the schematic and the green rectangles show the most likely fault conditions, although not necessarily the only conditions. This approach to the problem improves the discrimination between faults without discarding any information (we still have the original F values for all assumptions). There are cases, however, where perfect discrimination is still not possible. In these cases the use of operator input and the activatable model equations can greatly improve the diagnosis.

Results from Steritherm

Many of the failure possibilities in the Steritherm process were examined to determine the effectiveness of the diagnostic model processor. The method is quite capable of identifying the correct fault condition. Only in a few cases is the correct result accompanied by the possibility of failures which are not occurring. These cases, however, are as expected because the model equation relationships are identical for a few of the assumptions.

Two types of fault simulations are included in the Steritherm simulation to allow for step and ramp changes of certain values. Some of these failures do not affect the process, such as non-controlled sensor values, and others do, such as controller or pump failures. Figure 7.27 illustrates the various equations and assumptions considered in the Steritherm process and the relationships between them. Within G2, the various connections can be made visible and hidden on command so that relationships between the objects can be easily handled and inspected.

Table 7.3 shows the ultimate results from the various experiments that were performed on the process. The table is arranged such that each row represents a different simulated fault, and the columns show the response of each assumption considered (also refer to figure 7.27). If the box is blank, this indicates that the F value for that assumption remained less than 0.5. If the failure likelihood is greater than 0.5, but no fault alert is presented, then the approximate value is recorded in the box. If a fault alert is presented then an exclamation mark is recorded in the box (open for orange and filled for red). If a green rectangle is also presented, then the exclamation mark is shown with a box around it. None of the activatable model equations were used in these experiments, so these represent the worst case results.

All of the failures were identified with proper emphasis (red exclamation mark, and green box). Some results show cases where other possible failures are also indicated; but, the correct failure is never omitted.

Advantages of the G2 Implementation

The advantages gained by using G2 to implement the diagnostic model processor are discussed in contrast to a procedural language implementation.

Object-oriented representation: Because G2 relies on an object-oriented knowledge representation, the model equations, assumptions, and their relationships are easily visualized and manipulated. All objects are graphically represented (see Fig. 7.21) and therefore construction and maintenance of the model can be clearly handled. As previously discussed, the diagnostic methodology can be ignored after the initial programming, as it is handled through the use of generic formulas which perform the calculations for all instances of the classes of objects.

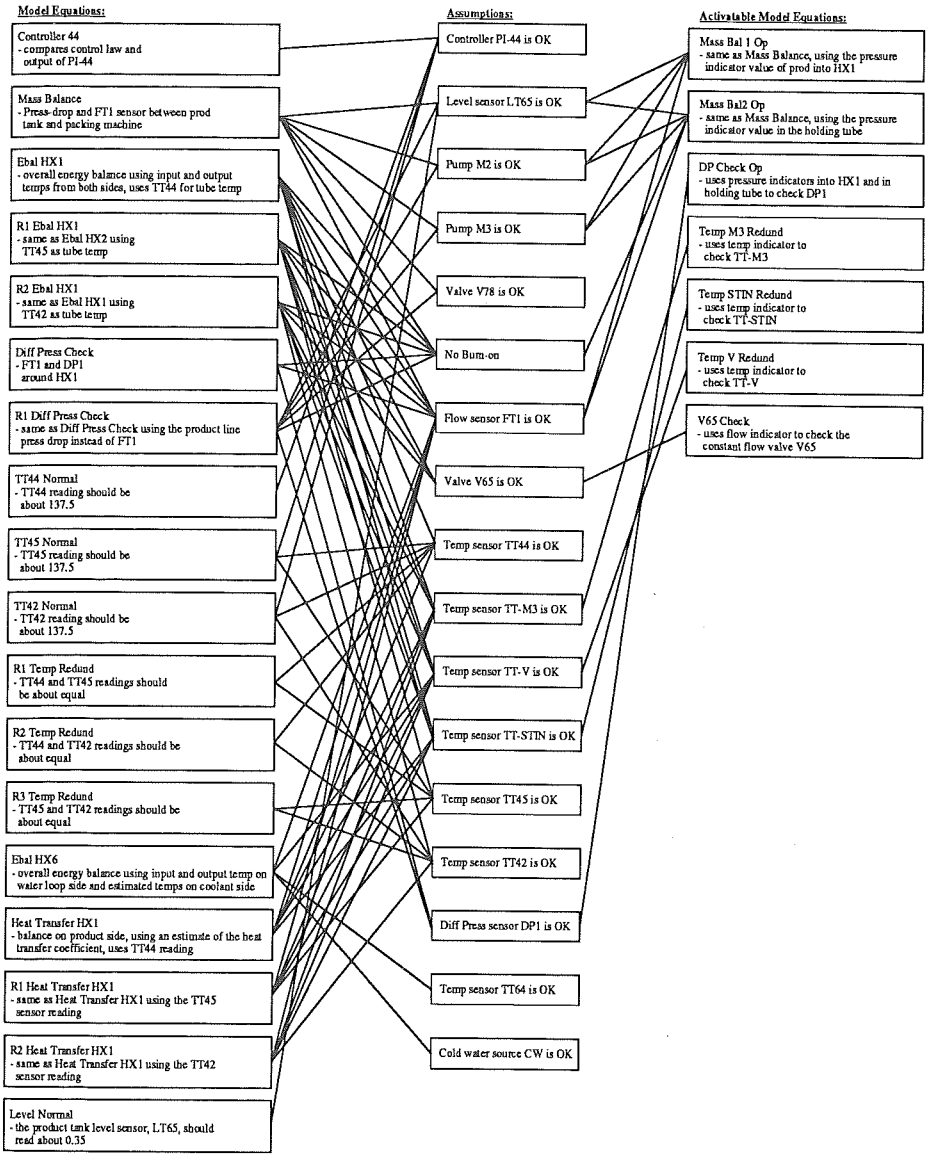


Figure 7.27 The model equations and assumptions used in the Steritherm process.

Another advantage of the graphical object environment is the ease with which methods can be developed to present (to the operator) the results from the analyzer. This work uses a dynamic object called a fault alert (see Fig. 7.25) to draw attention to possible fault conditions. Other methods, however, can be used and

Fault \ Assumpt.	PI-44	LT65	M2	M3	V78	BO	FT1	V65	TT44	TT-M3	TT-V	TT-STIN	TT45	TT42	DP1	TT64	CW
LT65 400% step																	
FT1 25% step												-0.6					
TT-M3 25% (C) step											-0.7						
TT-V 25% (C) step																	
TT-STIN 25% (C) step																	
TT45 25% (C) step																	
TT42 25% (C) step																	
DP1 -25% step			-0.5	-0.5	-0.5												
PI-44 1% (SP, K) ramp, 60s																	
M2 -20% (P-out) step																	
M3 25% (P-out) ramp, 60s															0.7		
V78 -35% (P-ref) step																	
Burn-on											-0.7						
TT64 2% (K) ramp, 120s																	
CW -45% (P-out) ramp, 180s																	
TT44 1.5% (K) ramp, 120s																	

Table 7.3 Failure test results from the Steritherm experiments.

easily programmed.

Real-time aspects: The ability of G2 to update values asynchronously allows all the important quantities to be calculated independently. In a procedural implementation each calculation is made at a regular time interval. In G2 each quantity is updated according to its own schedule. This schedule can be easily adjusted to improve the analyzer's performance after it has detected an event. In this manner, real-time performance can be assured with the ability to focus attention when necessary. This ability also allows for easy implementation of activatable model equations using asynchronous operator input.

Additionally, the procedural call to refine the diagnosis is made only after the value of F for some assumption exceeds 0.5 in magnitude. Although this can be handled in a procedural language, it fits in G2 very naturally using a rule. The idea of detection followed by refinement is an important concept in real-time applications. Since G2 is built for real-time use these concepts are easily handled.

As previously mentioned, G2 allows the use of rules in the knowledge base. This ability can be used as a further refinement device, if certain heuristic knowledge regarding the possible fault conditions of the process is available. Rules can easily be included to constrain the fault possibilities which are presented to the operator.

We see that the advantages of using the G2 environment are primarily important from an ease of use standpoint, and although many points discussed could be duplicated in a procedural language, it would require much greater effort.

Extensions

The implementation of the DMP method could be extended in various directions. The fault assumptions should have links to the process components to which they refer. If the control system contains simulation equations for the process components, which is not the case in the prototype, it is plausible that some of the model equations could be, at least, semi-automatically generated from the simulation equations and the process schematic.

7.3.6 A Multilevel Flow Model of Steritherm

In Chapter 4.9, the basics of MFM techniques were described, together with an example of a heat exchanger system. The Steritherm process is, of course, much more complex. Still, it is a good target system for testing MFM, for the following reasons:

- The Steritherm process is of moderate size. It is large enough not to be trivial, and at the same time small enough to allow it to be modelled in its entirety.
- Steritherm is a rather typical matter and energy flow process, and there is even a barrier function, that of not allowing the treated medium to come in contact with the environment. MFM was designed for describing precisely such things.
- Steritherm can be configured and reconfigured in many ways. Thus, it may well be an ideal process on which to develop theory and practise for the configuration goals in MFM.

An MFM Toolbox in G2

In order to provide computerized support for the building and utilizing of MFM models, a set of definitions, rules and procedures has been written in the G2 language. These aids are intended to be developed into a toolbox for functional modelling in the near future. A first sketch of models of the Steritherm process have already been produced, together with facilities for a primitive consequence propagation.

The flow functions, conditions, goals, and networks have all been implemented as G2 objects, and models are built by cloning new MFM objects and connecting them in the standard G2 way. According to the MFM syntax, there are no connections between flow functions and conditions; this has been handled by making the particular type of connection invisible when execution is started.

The connections between objects in different views are handled with lists. Every flow function has a list of corresponding objects, e.g., in the functional model of the steam injector, (a transport function), there is a list containing a reference to the topological representation of the injector. In the same way, the topological objects have lists of the corresponding flow functions.

The functions provided by the MFM toolbox have been implemented with general rules, and thus, they are all available once the flow function symbols have been connected into systems and the lists of corresponding objects have been initialized. Currently, the toolbox performs two kinds of actions; it transfers the working status between corresponding objects in different views and it can perform a consequence propagation. The status of an MFM object is shown with colours; it is in this way the results of the consequence propagation are shown. The number of rules in the implementation is shown in Table 7.4. The initialization rules are used to give values to the lists of corresponding objects in other views. Thus, their number depends on the size of the flow model.

Consequence propagation	22 rules
Graphics	8 rules
Topological view connections	3 rules
Initialization	80 rules

Table 7.4 Number of general rules in the MFM toolbox.

An MFM Model of Steritherm

The following models were developed mainly for the purpose of testing the G2 toolbox. However, they also point to what a more conclusive functional model of the Steritherm might look like. The specific details of the models should be seen in light of this. It is very likely that the descriptions will change, generally as well as in detail.

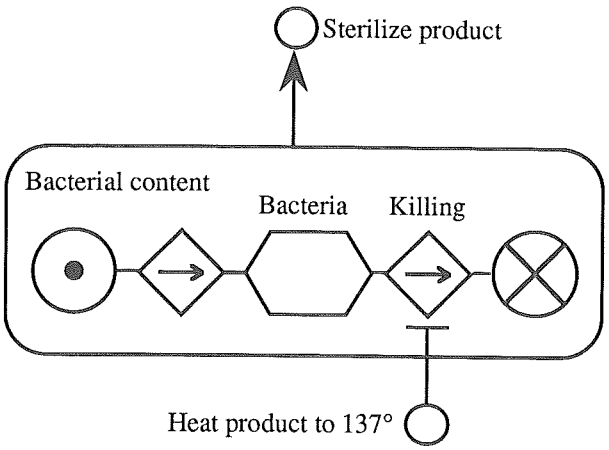


Figure 7.28 The first top-level goal of Steritherm is to sterilize the product.

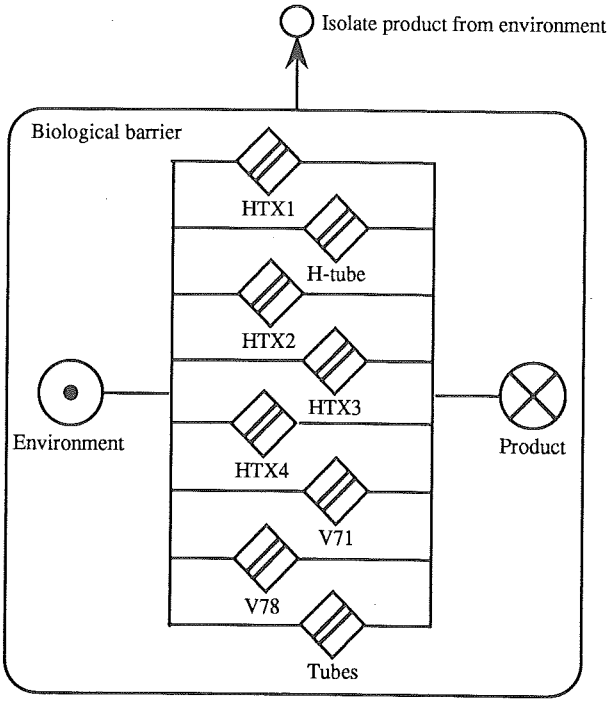


Figure 7.29 The second top-level goal is to keep the product sterile.

The Steritherm process was modelled in the production phase only. In this phase there are two main goals. The first is to sterilize the product, something which is ensured by keeping the temperature of the product in the holding tube above

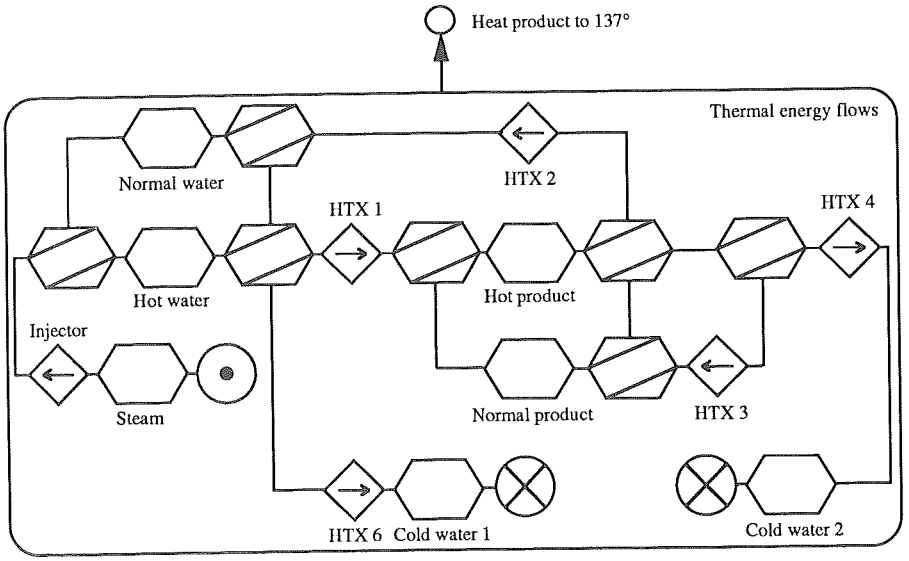


Figure 7.30 The thermal energy flow diagram.

137° C, see Fig. 7.28. In the topmost network, the bacterial content of the product is modelled. The first transport function represents 'life-giving' and the second killing of the bacteria through heating. On this level, there are no clear correspondences of the flow functions to physical components. The goal of heating the product is provided by the thermal energy flow network, see Fig. 7.30.

The other, equally important, main goal is to keep the product sterile during cooling, transportation, and packing. Most of the barrier functions of Fig. 7.29 correspond to a physical object, a heat exchanger or valve that must not leak. The tubes have been gathered into one barrier, however.

The Steritherm is quite cleverly designed when it comes to reuse of heat, as can be seen from the thermal energy network in Fig. 7.30. There are two explicit thermal feedback loops, via the heat exchangers `HTX 2` and `HTX 3`. In this flow diagram, the media at different temperatures are modelled as energy storages, while the heat exchangers and steam injector are described as transport functions.

The energy enters the Steritherm system via the hot steam, used to heat the secondary (water) flow. The hot water transfers its energy to the product, but some of it remains in the water flow, and another part leaves the system via the ice water cooling in heat exchanger 6. Energy also leaves the system when the product is cooled by ice water, in heat exchanger 4. Both these energy exits have been modelled as sinks. Energy is also lost via packing of the product and radiation and transfer to the environment. Every heat exchanger, e.g., gives off heat to the surrounding air. These sinks have, however, all been ignored.

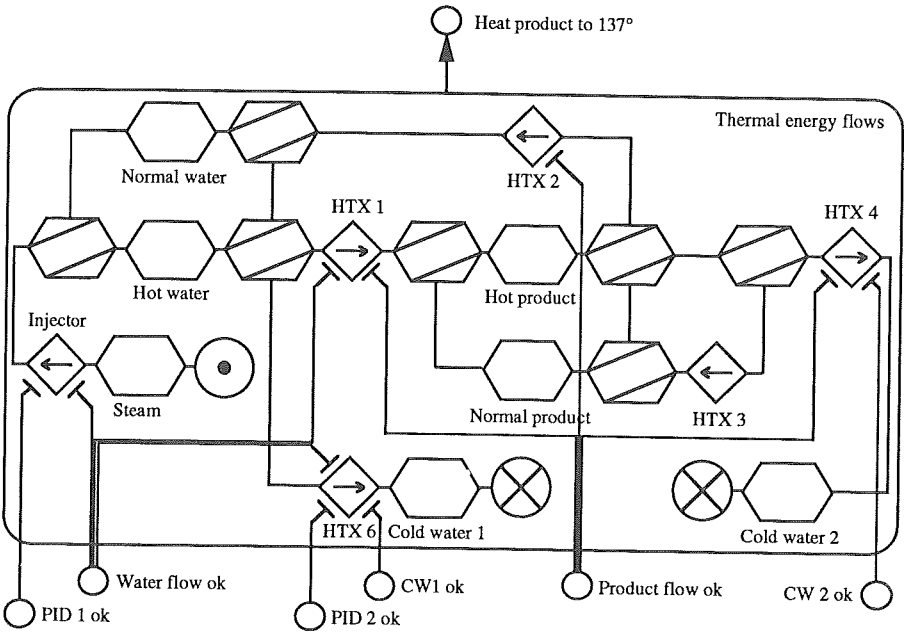


Figure 7.31 The thermal energy flow network with conditions shown.

In the model presented, the product and water media have been modelled as being in either 'normal' or 'hot' condition. It would be possible to make a finer grading and describe the product in more phases, such as, e.g., 'normal', 'pre-heated', 'fully heated', and 'pre-cooled'. This would give finer detail at the expense of a more complex diagram. Choices like this are good examples of the decisions that must be considered in the construction of MFM models.

In Fig. 7.30 only the flow functions appear, but. of course, the transport functions all depend on other subsystems. In Fig. 7.31 the conditions are shown too, together with the subgoals necessary for the energy flow to work. It is via these subgoals that the functional dependencies continue down into the mass flow networks.

The primary flow in the production phase is that of the product itself. The flow network is shown in Fig. 7.32. The product goes through a lot of pumps, valves, tanks, and heat exchangers, thus this network has been given a hierarchical decomposition. The heating, cooling, and recirculation steps are modelled as single transport functions, each with an inner structure. There is no physical motivation for this kind of decomposition, instead it is a matter of using hierarchy to provide an better presentation of the model.

The water flow network is somewhat simpler than the primary flow, and so may be shown in a single network, see Fig. 7.33. It should be noted that steam and

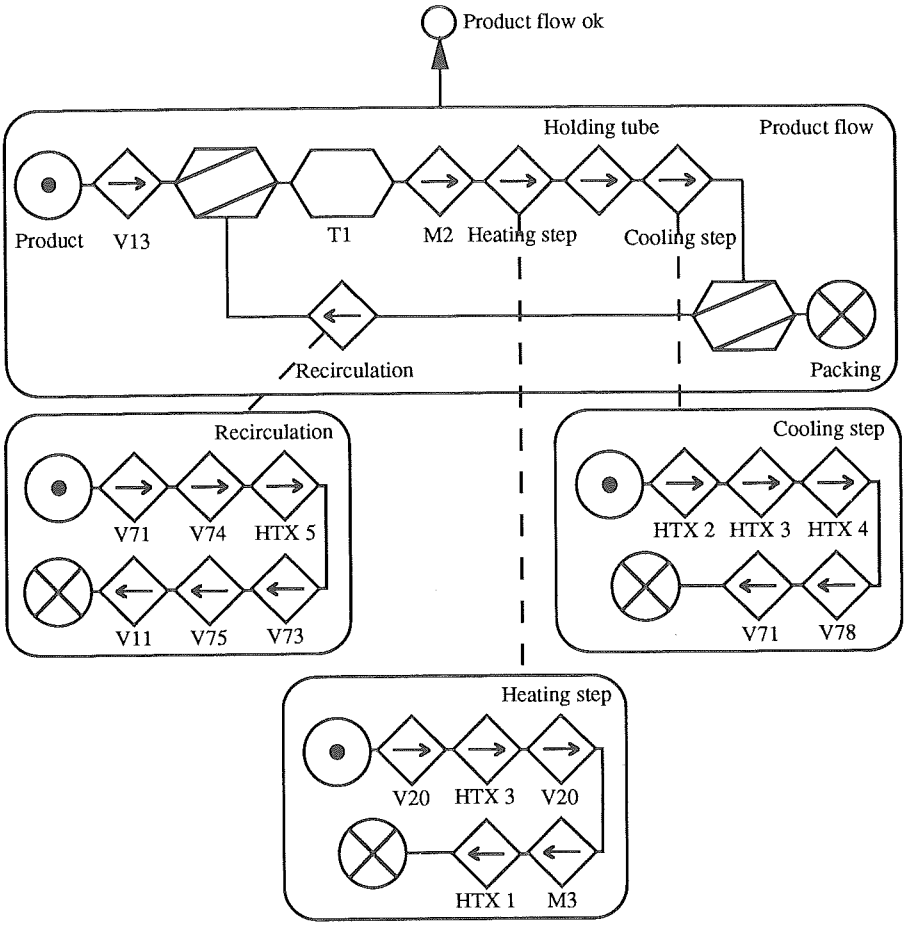


Figure 7.32 The (hierarchical) product flow network.

liquid is not separated, but treated as a single medium.

In the secondary flow, water is heated to above 137° C. The water must be cooled before it is recirculated to the balance tank, and this cooling is done by `HTX 6`. The small flow of ice water necessary for this is modelled in Fig. 7.34. The valve `V64` is the same valve that can be found in the second control system network.

The product must also be cooled before it is packed. This cooling is done with `HTX 4`, which uses a small flow of ice water, shown in Fig.7.35. This flow is not actively controlled, but governed by a two-way valve, `V27`, which can be in either bypass or active position. The latter has the effect that the valve is modelled as two transport functions.

The Steritherm process has only two active control loops. The most important

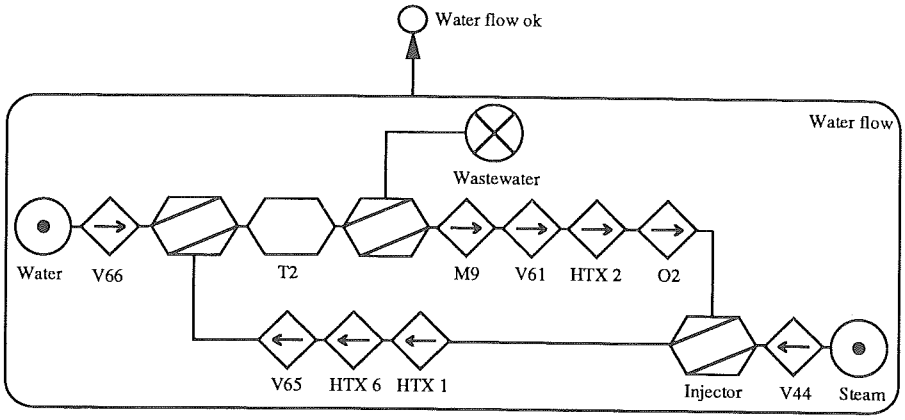


Figure 7.33 The water flow network.

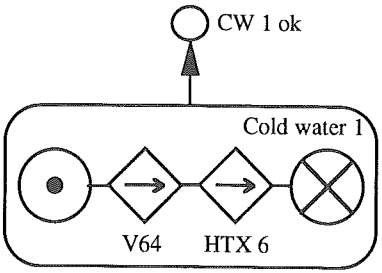


Figure 7.34 Ice water flow for cooling of return water.

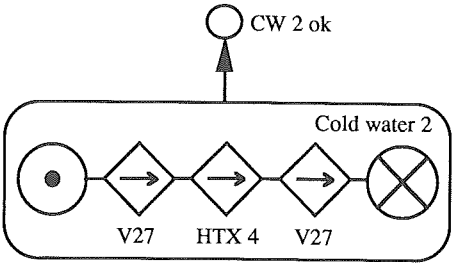


Figure 7.35 Ice water flow for cooling of product before packing.

one is used to keep the temperature of the holding tube at approximately 137 degrees Celsius. The temperature is measured by the sensor `T44` and the valve `V44` is used to control the flow of steam to the steam injector, as shown in Fig. 7.36.

The second loop controls the temperature of the return water by adjusting the flow of ice water through `HTX 6`. The temperature is measured by the sensor

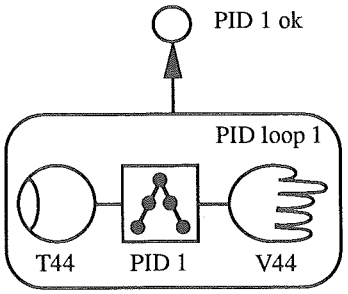


Figure 7.36 Steam injector control system.

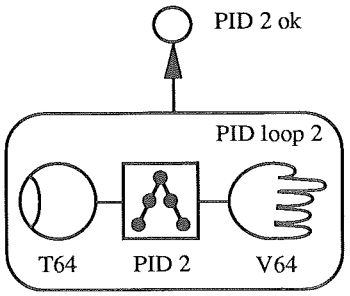


Figure 7.37 Return water temperature control system.

T64 and the ice water flow is controlled with the valve V64, as can be seen in Fig. 7.37.

Connecting MFM to Other Models

The project uses several modelling techniques to present the same process to the user. This makes it necessary to give the user help in navigating in and between the different models. The problem is acutely visible in the case of MFM's connections to the standard topological representation, because a large part of the functional structure does not have a clear physical equivalent. Instead, each function may be realized by several different components, and each component can be used to realize several functions.

In order to remedy this, development has been started on the following presentational possibilities. First, there is a possibility to select a physical component or a flow function, and highlight all its corresponding representations. For example, selecting the valve V27 in the process diagram will highlight both transport functions in Fig. 7.35. In the same way, selecting a flow function will highlight the physical components that realize it. Secondly, the mass flow networks can be highlighted in the process diagram.

It is possible that parts of the MFM models on the topmost levels will be connected to other models, e.g., alarm trees and model-based diagnostics. This is a

subject for further research, however. It is also important to observe that some of the properties of the Steritherm process is clearly shown *only* in the flow models, e.g., the thermal energy flow with its feedback. In these cases, new ways of presentation must be developed. Possibly, the MFM graphical language can serve as a base for this.

Preliminary Conclusions

In conclusion, MFM is an interesting modelling technique, and the Steritherm process seems to fit quite well in the basic ideas and representations used in MFM. A functional model of the process should most certainly be at the heart of any knowledge-based control system.

7.3.7 Other features of the G2 prototype

The Product following system

A simple product following system has been developed. Its intended use is quality follow-up. The idea is to associate bulks of product with the treatment with regard to different process parameters such as temperature and flow, that it has been exposed to during the processing. Product slice objects are dynamically created at the inlet to the product balance tank, with the creation rate depending on production. The product slice objects have attributes concerning temperatures, etc., which originally have no values. The product slice objects are graphically moved along the production line and their attribute values are recorded from the actual values of the sensors. The attributes which are used are:

- The temperature before the pre-heater.
- The product flow
- The temperature in the holding tube.
- The production time and date.
- The time since the last cleaning of the process.
- The differential pressure over the final heat exchanger section. The last two measurements give indications on the burn-on.
- The temperature before the packing machine.

It would have been natural to also record the serial number of the packing and the order number. In a realistic setting the product slices would finally be stored in a relational database.

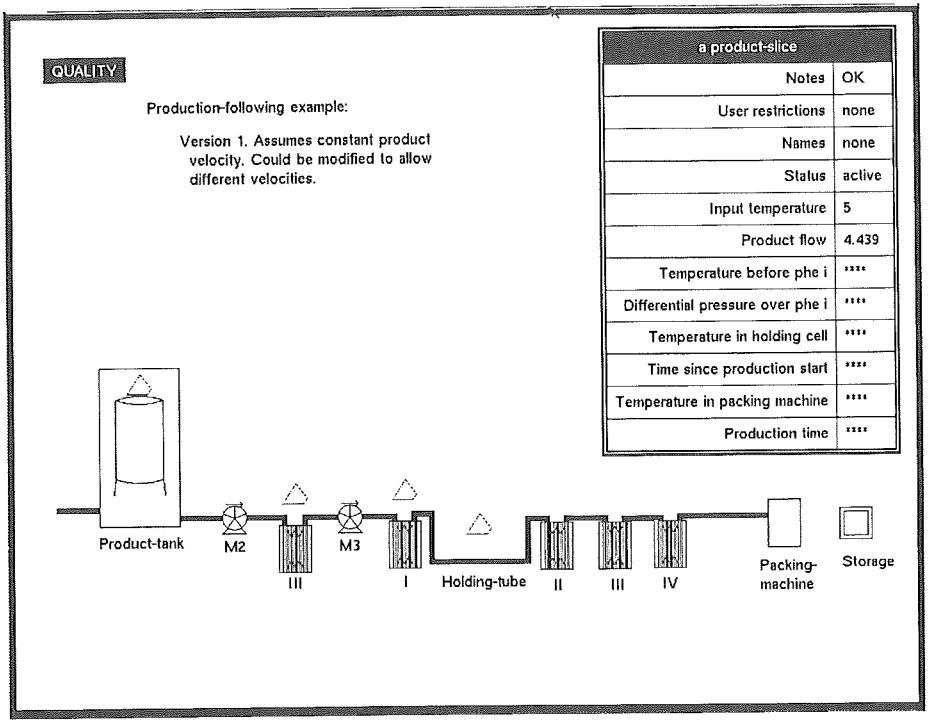


Figure 7.38 Product following system

The product following system, shown in Fig. 7.38, uses a simplified process schematic. This is stored under the quality control subview of the Steritherm operation view.

The current implementation assumes constant product velocity throughout the production line. This is obviously not true. The system could, however, easily be extended to accomplish different velocities in the different parts of the process.

Multiple systems

Topological views of the 380 V power supply system and of the control system have been partially implemented. It is possible to move between the process schematic description of the product pumps, M2 and M3, and their electrical descriptions as shown in the power supply system shown in Fig 7.39. It is also possible to move from the contactors and circuit breakers in the power supply system to their appearance in the control system.

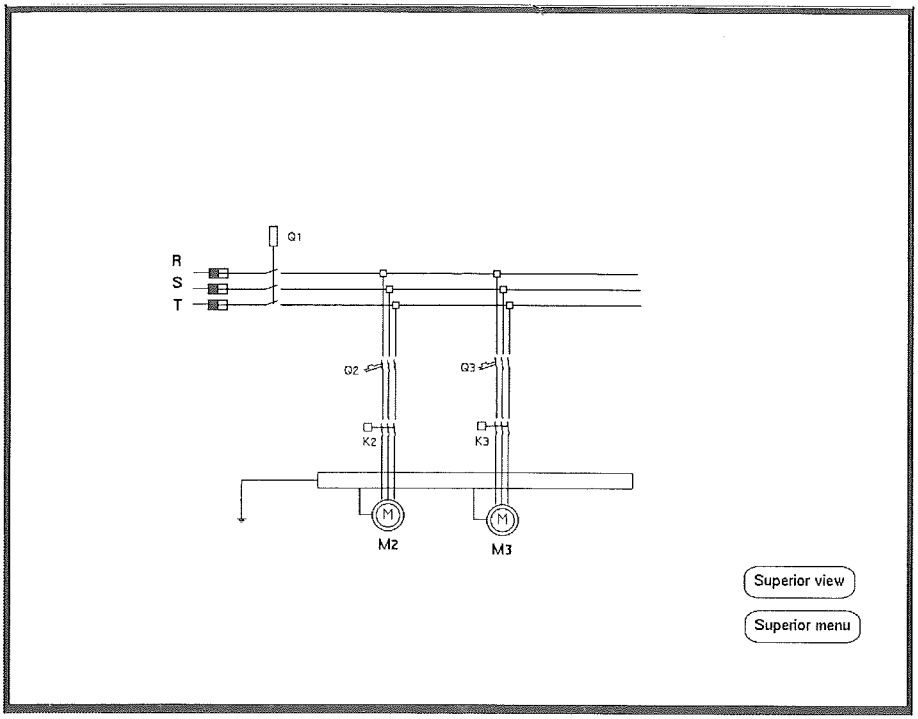


Figure 7.39 Part of the topological view of the 380 V power supply system

7.4 CONCLUSIONS

The G2 prototype is one of the most important results of the first project phase. Also the Plus prototype has helped highlighting some of the features of the concept. However, with the arrival of version 2.0 of G2 most of the features of the Plus prototype can also be implemented in G2. The major exception is scanned-in pictures.

Both prototypes were developed before the system concept had stabilized. Therefore there are some discrepancies between the prototypes and the concept.

The multi-view object structured cannot be directly implemented in G2. Also the structuring of the knowledge base into different systems and views does not follow the concept. Animation and colours are used in several places in the G2 prototypes, e.g., in the DMP implementation; in the product following system; in the process schematic to indicate valve positions, tank levels, and pump operation, and to indicate the media in the product line pipes. These are things

that should be placed in the user interfaces of the various user groups. The G2 interface should only implement the knowledge base browser.

The knowledge included in the prototypes only cover a small part of what is needed. Knowledge about raw products, recipes, process components, etc., is not included at all. Due to restrictions in G2, knowledge is also duplicated. For example, the fault assumptions in the DMP method and the fault objects in the alarm tree are separate objects even though they are very closely related to each other. Furthermore, the objects are also separated from the objects representing the process components. In a KBCS all information about a process components including its possible faults should be kept as one unit.

The major contribution of the prototypes is the role they play in visualizing the concept. For example, the G2 prototype shows how both conventional and knowledge-based techniques are integrated together in one knowledge base.

8

Technical Survey Update

In the Feasibility Study (IT4, 1988) a chapter was devoted to a survey of some of the techniques that are essential for KBCSs. Also, the study contained a chapter on international research programmes relevant for KBCSs. This chapter is an update of those chapters that reflects the development between 1988 and 1990. Special attention is given to the commercial real-time expert system tools available now.

8.1 EXPERT SYSTEM TOOLS

The market for expert system tools specially oriented towards real-time, on-line applications within the process industry, and the aerospace industry has increased substantially since 1988. Several commercial tools have emerged, with G2 from Gensym Corporation as the most mature and widely spread. This section contains an overview of some of the most interesting real-time tools. Since G2 has an important role in the project, a more detailed description is given. Plexsys, an off-line tool from Intellicorp, is also described.

Apart from the systems described several tools have been developed within the European Esprit projects. However, these tools are not yet of a commercial nature. Also during the last quarter of 1989, two Japanese real-time tools were announced from Meidensha and Toyo Information Systems. According to the information we have, both of these tools are modelled on G2.

8.1.1 G2

G2 from Gensym Corporation in Cambridge, MA, is the most widely spread and technically most advanced real-time expert system tool available on the market. The number of licenses sold is over 300, with at least 50 of them including on-line licenses.

Gensym Corporation

Gensym Corporation was founded in 1986 by the group at Lisp Machine International which previously had developed the PICON system. By January 1990 the company had grown to 42 employees. Of them the development team constitutes 5 to 7 persons. The rest of the company is divided between applications development, training, G2 consulting, and administration. G2 consulting is an area that is expanding.

The quarterly revenue of the third quarter 1989 was 1.2 MUSD. The company has opened three additional offices in USA, and are planning its first European office for the end of 1990, possibly located in Germany or the Netherlands. Gensym has also value-added-resale contracts with several companies.

Market situation

G2 has established itself as the de facto standard for real-time expert systems in the process control area. One symptom of this is that standard interfaces have been developed between G2 and the major process control systems. The initiative to this has come either from Gensym, external consulting companies, or from the control system developers themselves. Gensym also has many applications in the aerospace industry, network management, manufacturing, etc.

USA is the largest market for Gensym with Du Pont as their most important customer. Japan and Europe are about equal in market size. In Sweden G2 is available at Lund Institute of Technology, ABB, Uppsala University, STFI, Linköping Institute of Technology, Forsmark II, and The Army Technical School.

During 1989, several interesting research projects have selected G2 as their expert system developing tool. Space Biospheres uses a network of G2s to monitor artificial biospheres. The French oil company ELF will use G2 for a project where they plan to totally automate their North Sea oil platforms during the pumping phase. NASA is using G2 for space shuttle monitoring. The Japanese nuclear energy research centre NUPEC has chosen G2 for a large project on nuclear safety.

Availability

G2 is implemented in Common Lisp and runs on Sun 3 and 4, HP, Apollo, Vaxstation, Decstation, VAX 8600, TI Explorer and MicroExplorer, Symbolics,

Compaq 386, and Mac II. For most machines, 16 MB RAM memory and X-Windows is required. The price ranges from $ 18,000 to $ 36,000 depending on computer.

Technical Description

The main parts of G2 are: the knowledge-base, a real-time inference engine, a procedure language, a simulator, the development environment, the operator interface, and optional interfaces to external on-line data servers.

Classes and objects: In G2 everything is an item, i.e, rules, objects, procedures, graphs, buttons, text boxes, etc., are all items. The items are organized into a hierarchy. All items have a graphical representation through which they are manipulated by mouse and menu operations. Operations exist for moving an item, cloning it, changing its size and colour, displaying its attribute table, etc. One part of the item hierarchy is the G2 objects. The object is the only part of the item hierarchy that the user has full control over, i.e., can specialize into subclasses, can reference in expressions, etc.

Objects are used to represent the different concepts of an application. They can represent arbitrary concepts, i.e., both physical concepts such as process components and abstract ones. The objects are organized into a class hierarchy, i.e., only single inheritance is allowed. The class definition, or using G2 terminology, the object definition defines the attributes that are specific to the class and the look of the icon. Icons can be created with an interactive icon editor. The attributes describe the properties of the object. The values of an attributes may be

- constants,
- variables,
- lists, or
- other objects.

Constants can be numbers, symbolic values, i.e., the G2 correspondence to the enumeration type, logical values, i.e., true or false, and text strings. Under run-time, constants can only be changed explicitly by the user. Variables are used to represent entities whose values change during run-time. Variables are defined from four basic predefined classes:

- quantitative variables, i.e., real-valued variables,
- symbolical variables,
- logical variables, and

- text variables.

The predefined variable classes can be specialized by the user. With the predefined variable classes come a set of default attributes. These include attributes that determine whether a history should be saved for the variable or not; the current value for the variable; what should be the source of the variable's value, e.g., the inference engine, the simulator, or some external data server; the validity interval of the variable, e.g., how long the current value of the variable should remain valid; etc. The validity interval could be specified to be indefinite, given by a fixed time interval, or dependent on the validity intervals of the variables that were used to calculate the value of the variable. Variables with indefinite validity interval are called parameters and form separate classes. Parameters always have a current value and their initial values can be specified.

Lists may contain arbitrary values. The allowed values in a list can be specified. It is possible to have objects as the values of attributes in other objects. In that case, the attribute objects have no iconic representation.

Objects can be static, i.e., they are explicitly created by the developer, or dynamic, i.e., they are created dynamically during run-time. Dynamic objects can also be deleted during run-time. The G2 language contains actions to move, rotate, and change the colour of an object. Using this, animations can be created.

Composite objects, i.e., objects that have an internal structure composed of other objects, can be created using objects as the value of attributes. It is, however, not possible to at the same time have a iconic representation for these objects. If such a representation is desired this has to be implemented using the subworkspace concept. In G2 each object and most items may have an associated subworkspace. In this (sub-)workspace arbitrary items may be positioned. The internal structure of an object can be represented on its subworkspace. It is also possible to connect together objects on the subworkspace with object connected to the subworkspace's superior object. It is, however, not possible to define that an object should have an internal structure of this type in the class definition.

Connections and Relations: G2 has two ways of defining relations between objects: connections and relations. Connections are primarily used to represent physical connections, e.g., pipes or wires. It is, however, also possible to let connections represent abstract relations among objects. Connections have a graphical representation and may have attributes. They are defined in terms of a connection hierarchy. Both unidirectional and bidirectional connections are allowed. Type checking is performed to allow only connections of the same class to be connected together. Connections can be used in G2 expressions for reasoning about interconnected objects in a variety of ways. A connection is attached to an object either at a pre-specified location, a *port*, or anywhere on the object.

Connections are static. They cannot be created during run-time. In order to make possible relations also between dynamically created objects, *relations* are

used. Relations can only be created at run-time and have no graphical representation. They have no corresponding relation hierarchy and cannot have attributes. Relations can be specified as being one-to-one, one-to-many, many-to-one, and many-to-many. The inverse relation of a relation can be specified, as well as whether the relation should be symmetric or not. In the latter case the inverse relation is the same as the relation. Relations can be used in G2 expressions in a similar way as connections.

The inference engine: G2 rules are used to encapsulate an expert's heuristic knowledge of what to conclude from conditions and how to respond to them. Five different types of rules exist.

- If rules
- When rules
- Initially rules
- Unconditional rules
- Whenever rules

When rules are a variant of ordinary 'If' rules that may not be invoked through forward chaining or cause backward chaining. Initially rules are run when G2 is initialized. Unconditional rules are equivalent to 'If' rules with the rule conditions always being true. Whenever rules allow asynchronous rule firing as soon as a variable receives a new value, fails to receive a value within a specified time-out interval, when an object is moved, or when a relation is established or deleted.

The rule conditions contain references to objects and their attributes in a natural language style syntax. Objects can be referenced through connections with other objects. G2 supports generic rules that apply to all instances of a class. The G2 rule actions makes it possible to conclude new values for variables, send alert messages, hide and show workspaces, move, rotate, and change colour of icons, create and delete objects, start procedures, explicitly invoke other rules, etc. G2 rules can be grouped together and associated with a specific object, a class of objects, or a user-defined category. This gives a flexible way of partitioning the rule-base. The following is an example of a G2 rule,

```
for any water-tank
 if the level of the water-tank < 5 feet and
 the level-sensor connected to the water-tank is working
 then conclude that the water-tank is empty
 and inform the operator that
```

```
"[the name of the water-tank] is empty"
```

The real-time inference engine initiates activity based on the knowledge contained in the knowledge base, simulated values, and values received from sensors or other external sources. In addition to the usual backward and forward chaining rule invocation, rules can be invoked explicitly in several ways. First, a rule can be scanned regularly. Second, by a focus statement all rules associated with a certain focal object or focal class can be invoked. Third, by an invoke statement all rules belonging to a user defined category, like safety or startup, can be invoked. The scanning of a few vital rules in combination with focusing of attention is meant to reflect the way human operators monitor a plant. It is also an important way to reduce the computational burden on the system.

Internally the G2 inference engine is based on an agenda of actions that should be performed by the system. The agenda is divided into time slots of 1 second's length. After execution, scanned rules are inserted into the agenda queue at the time slot of their next execution. Focus and invoke statements causes the invoked rules to be inserted in the agenda at the current time slot. Rules being invoked by forward chaining is treated in the same way.

A rule is invoked by backward chaining if the rule actions of the rule includes a conclude statement that gives a variable a new value, and if a new value for the variable is needed, i.e., the variable has a default update interval that specifies that the value should be recalculated regularly, the value is needed in a rule condition, or the value is needed in a display. Depth or breadth first backward chaining may be specified as well as the precedence order of the rules.

Simulation: G2 has a built-in simulator which can provide simulated values for variables. The simulator is intended to be used both during development for testing the knowledge base, and in parallel during on-line operation. In the latter case, the simulator can be used, e.g., to implement filters for estimation of signals that are not measured.

The simulator allows for differential, difference, and algebraic equations. The equations can be specific to a certain variable of apply to all instances of a variable class. Each first-order differential equation is integrated individually with individual and user-defined step sizes. The numeric integration algorithms available are a simple forward Euler algorithm with constant step size and a fourth order Runge-Kutta algorithm, also with fixed step size. GSPAN, an interface between G2's simulator and external simulators is available as a separate product.

Procedures: G2 contains a Pascal-style procedural programming language. Procedures are started by rule actions. Procedures are reentrant and each procedure invokation executes as a separate task. Procedures can have attributes and return one or several values. Local variables are allowed within a procedure.

The allowed procedure statements include all the rule actions, assignment of values to local variables, *If-then-else* statements, *case* statements, *repeat* statements, *for loop* statements that can either be numeric or generic for a class, i.e., they execute a statement or set of statements once for each instance of the class, *exit if* statements to exit loops, *go to* statements, and *call* statements to call another procedure and await its result. Procedures can be temporarily halted with a *wait* statement. A wait statement causes G2 to stop executing the procedure until either a specified amount of time has passed or a condition is met. It is possible to specify that two or more statements should be executed in parallel and that all iterations of a loop should be done in parallel.

Procedures are executed by G2's procedure interpreter. The procedure interpreter cannot be interrupted by other G2 processing, i.e., the inference engine or the simulator. Other processing is only allowed when the procedure is in a wait state. A wait state is entered when a wait statement is executed, when the statement *allow other processing* is executed, and when G2 collects data from outside the procedure for assigning to a local variable.

Real-time issues: Unlike the majority of expert system tools, G2 is designed for real-time operation. This shows in a number of different ways. The inference engine is based on the link approach described in Chapter 5 instead of using pattern matching. The inference engine automatically sends out requests for variables that have become invalid and waits for new values without halting the system. Priorities and scan intervals can be associated with rules.

Regular scanning of rules and thus updating of information in combination with variables with time-limited validity gives a partial solution to the problem of non-monotonic, time-dependent reasoning. The validity interval of a variable specifies how long the current value of the variable should remain valid. By the possibility to propagate validity intervals to dependent, concluded variables their values will also eventually expire if for some reason sensor values cease to arrive to G2.

Whenever rules can be used to catch asynchronous events such as the arrival of un-requested sensor data from a passive sensor, e.g., representing some alarm in the underlying control system, or that a requested sensor value has failed to arrive to G2 within a specified time-out interval.

G2 has some facilities for temporal reasoning. It is possible to save histories of old values for all variables. Functions for referencing old variable values are available. G2 also has built-in statistical functions operating on quantitative variable histories. These are functions for computing the integral, standard deviation, rate of change, maximum, and minimum values over some time interval.

G2 also has possibilities for reasoning about whether a variable has a current value or not, and can refer to the time when a variable received its current value.

To avoid garbage collection, G2 takes care of the dynamic memory allocation and reallocation internally during run-time.

Development interface: G2 has a nice graphics-based development environment with windows (called workspaces), popup menus, and mouse interaction. Input of rules, procedures, and other textual information is performed through a structured grammar editor. The editor prompts all valid next input statements in a menu. Using this menu the majority of the text can be entered by mouse-clicking. It is, however, also possible to use the keyboard in an ordinary way. The editor has facilities for Macintosh style text selection, cut, paste, undo, redo, etc.

The Inspect facility allows the user to search through the knowledge base for some specified item. The user can go to the item, show all matching items on a temporary workspace, write them out on a report file, highlight them, and make global substitutions.

G2 has facilities for tracing, stepping, and adding breakpoints. The internal execution of G2 can be monitored using meters.

End-user Interface: G2 has facilities for building end-user interfaces. Colours and animation can be used. An object icon is defined as a set of layers whose colours can be changed independently during run-time. The meta-colour *transparent* makes it possible to dynamically hide objects. Different user categories can be defined and the behaviour with respect to which menu choices that are allowed can be set for each category. It is also possible to define new menu choices.

G2 contains a set of predefined displays such as readouts, graphs, meters, and dials that can be used to present dynamic data. G2 also has a set of predefined interaction objects that can be used for operator controls. Radio buttons and check boxes can be used to change the values of symbolical and logical variables by mouse clicking. An action button can be associated with an arbitrary rule action which is executed when the button is selected. Sliders can be used to change quantitative variables and type-in boxes are used to type in new variable values.

External interfaces: G2 can call external programs in four different ways: using foreign function calls, and using GFILE, GSPAN, and GSI. On some platforms, external C and Fortran functions may be called from within G2. GFILE is an interface to external data files that allows G2 to read sensor data from the files. GSPAN is the interface between G2 and external simulators. GSI is Gensym's standard interface. It consists of two parts; one part written in Lisp that is connected to G2 and one part written in C to which the user can attach his own functions for data access. On the same machine, the two parts communicate

using interprocess communication media such as pipes or mailboxes. On different machines, TCP/IP – Ethernet is used.

GSI is the base for several off-the-shelf interfaces between G2 and conventional control systems, PLC systems, relational databases, etc. The Travel Notes in Appendix B contains reports from User Group Meetings where details about the existing interfaces are presented.

Networking: Several G2s can be used in a network exchanging data over Ethernet. Telewindows is a separate product that allows multiple users to access the same G2 system, giving each user his own window into the application. It is also possible for one Telewindows user to simultaneously access several G2s in a network.

Drawbacks

The main problems with G2 stem from the fact that G2 is a closed system. G2 can only be interfaced with other program modules through the predefined interfaces. The G2 environment in itself is also a quite closed world. It is impossible to modify the that G2 operates internally. If what G2 provides in terms of, e.g., graphics, class – object structures, etc., is insufficient nothing can be done about it.

G2 can not be modularized. Hence, it requires quite powerful computers even if only a small subset of the functionality is used within an application.

Although G2 is fast compared to many expert system tools, it can be too slow for certain applications. The smallest time unit is one second. For applications that require faster response, G2 is inadequate. Gensym claims that G2 is capable of running between 300 and 500 medium-sized rules per second depending on the machine that is used. These figures are difficult to verify. If the simulator is used, the speed decreases substantially. Gensym are currently developing a run-time version that uses compiled rules and procedures instead of interpreting them, as is currently done.

G2 versus the KBCS concept

G2 is one of the main sources of inspiration in the project. The knowledge base language discussed in Chapter 5 has borrowed many features from G2. However, there are some important differences.

Although it is possible to use networks, G2 is in essence a centralized system without any distribution. The G2 knowledge base executes only on a single processor. G2 can be seen as having three realization tools: the inference engine, the procedure interpretor, and the simulator, which all operate on the knowledge base as shown in Fig. 8.1. The tools have one module in common, the expression evaluator.

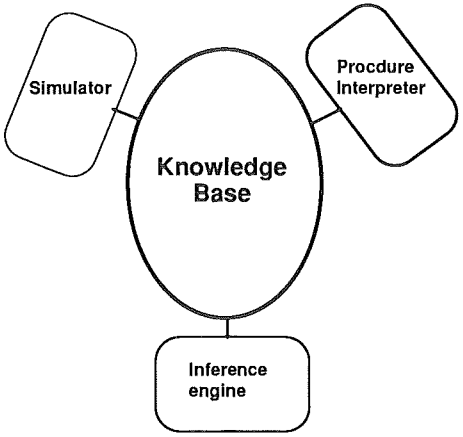


Figure 8.1 G2 realization tools

The interactive interface to G2 is basically equivalent to the knowledge base browser. The only thing similar to design tools is the editor.

In G2 the representation of knowledge is integrated with the presentation. If an object should be presented differently to different users, e.g., using different icons, two G2 objects have to be used. It is also impossible to have multiple views of an object in the knowledge base. If an object has to be represented in two different contexts with different connections and slightly different attributes, two G2 objects have to be used. All relations between these two objects have to be defined explicitly through, e.g., G2 relations.

The support for composite models and information zooming is limited. For example, it is not possible to define that an object should have an internal iconic structure on its subworkspace in the class definition.

G2 is a closed system. It is impossible to include external data structures such as, e.g., pixel bit-maps, or hypermedia cards.

8.1.2 Talos–R*TIME

R*TIME is a set of modules for real-time data acquisition, data analysis, data distribution, and message/data display from Talarian Corp., Mountain View, CA, announced to be released in the first quarter of 1990. Talarian Corporation was founded in October 1988 by a group from Lockheed and Stanford University that previously had developed Lockheed's L*STAR, a distributed knowledge-based architecture designed for real-time applications. When R*TIME first was announced it was called Talos. For some reason the name has changed since then. The intended applications for R*TIME are intelligent real-time monitoring, analysis, display, and control of complex systems. The system includes a real-time inference engine which is claimed to be "at least an order of magnitude

faster than its nearest competitor" and capable of executing "thousands of rules each second, working with very large rule sets", a colour graphic man-machine interface, and data acquisition and management modules.

The three main modules in R*TIME: the inference engine, the man-machine interface, and the data acquisition module, operate independently, can be distributed across processors on a LAN, and communicate via message passing using standard protocols.

The major application so far for L*STAR – R*TIME is the NASA Hubble Space Telescope monitoring system.

Availability

R*TIME will be available on DEC, SUN, and 386 workstations and have interfaces to IBM PCs for data acquisition and control. Each R*TIME module is designed so that it is possible for it to be embedded within other existing hardware and software. R*TIME allows rules to call out to procedures written in C, Fortran, and ADA. Furthermore, hooks are supplied so that external procedures can archive and retrieve data from R*TIME's real-time database.

The data acquisition module supports data archiving to permanent storage devices such as disk or tape. Data collected can be played back into R*TIME.

Knowledge Representation

Knowledge representation in R*TIME is based on frames and rules. Rules can be invoked in three different ways:

- at a fixed time interval, i.e., using scanning
- by forward chaining, and
- by backward chaining.

The below scanned rule is used to detect a power battery anomaly in a satellite application.

```
RULE          :"Inadequate battery voltage"
CONTEXT       :{ Maneuver };
PRIORITY      : 100;
TEST INTERVAL : 10 seconds;

IF voltage ?B < 27.5
THEN status ?B := abnormal;
     ALERT(NODE,MADMAX,"?B",
     "Current voltage of battery ?B inadequate");
```

?B is a matching variable that allows the rule to be applied to any battery in the application.

R*TIME's frame representation language (FRL) supports inheritance and has been extended in the temporal dimension by allowing slots to point to ring buffers where a series of values and their associated time tags are kept. A variety of methods for trend analysis such as linear and nonlinear regression analysis, least squares, moving averages, rate of change, mean, standard deviation, minimum, maximum, correlations, and exponential smoothing are available. It also possible to define new methods.

Real-time Aspects

R*TIME has facilities for reasoning about past, present, and future events. It can compare current data with past as well as reason about the sequence in which the events occurred. Event sequences are reasoned about using the operators *BEFORE, AFTER*, and *DURING*. Reasoning about future events is achieved by asserting values into the frame database at some upcoming time. A rule condition using the sequence operators looks as the following.

```
IF DURING( (voltage ?B > 35.5),
           (temperature ?B < 40),
           10 minutes )
AND ..
```

The condition checks if the voltage of a battery is > 35.5 at the same time its temperature < 40 in the last 10 minutes.

R*TIME provides focus of attention capabilities through

- changing the set of sensors the system is currently investigating,
- bringing a new set of rules to bear, and
- changing the sampling rate or compression scheme of the data being analyzed.

Similar to G2, R*TIME ensures a garbage free execution.

End-user Interface

The man-machine interface receives sensor data from the data acquisition module, derived data and text messages from the inference engine, updates its graphical display in real-time, and acts as the interface to the operator. The user is supplied with a DRAW program to develop his displays and define their interactions. The DRAW program has built-in graphic primitives (line graph, strip chart, dial, bar chart, etc.). Relationships between displays and hierarchies of displays can be defined. Animation facilities are available.

Communication

The communication between R*TIME modules or between R*TIME and user-defined processes is built on a user-extensible set of message primitives (Info, Alert, Warning, Numeric_Data, etc.). Messages may be sent to a particular process, to all processes of a given type, to all processes on a given node, to all processes receiving messages from a certain datagroup, and to all processes who know about a given object.

Comments

The open architecture and real-time support in R*TIME are promising. Since it is not released yet it is difficult to evaluate it any further.

8.1.3 Nexpert Object

Nexpert Object, which was described in the Feasibility Study, has gained increasing popularity for various applications including on-line, process control applications. Not really a real-time tool, the main reason for its popularity is the excellent facilities to embed Nexpert within other programs, i.e., to call-in Nexpert routines from other programs and to call-out to external procedures from within Nexpert. Another reason for its popularity is the wide range of machines that it is available on, also including smaller machines such as the Macintosh.

Nexpert has interfaces to data presentation programs such as Dataviews and Ease+ and to Hypercard programs with which end-user interfaces easily can be built up.

Within Sweden, Nexpert Object is being used in two process industry applications. At Skoghall in Karlstad, Stora Teknik and The Karlstad Institute of Technology cooperate in a DUP project concerning monitoring of a continuous digester. Here Nexpert Object will be used together with Dataviews, Oracle, and Matlab. The Royal Institute of Technology (Produktionsteknik) has a Nordic project together with, among others, SSAB where Nexpert Object will be used together with relational databases for monitoring and control of blast furnaces.

SHERPA

BHP Central Research Facilities in Newcastle, Australia has developed SHERPA, a facility that integrates modules for knowledge base development, signal processing, operator displays, on-line numerical models, and data base storage.

SHERPA is an open architecture tool that simplifies the integration of commercially available software into one framework. Presently SHERPA can integrate Nexpert Object , Dataviews, and Oracle. SHERPA includes SHERPATALK, a high level interpreted language including simplified calling sequences for the more common routines in the different programs and over 200 numerical routines for, e.g., data pre-processing and recursive identification.

SHERPA has been applied to different operator guidance applications mainly within the Australian steel industry and is currently being used in the Skoghall project.

Comments

Also here, it is the open architecture that is the strong feature. Nexpert Object can easily be embedded with conventional software on a range of different machines.

8.1.4 Chronos and Nemo

Chronos and Nemo are two French rule-based expert system tools developed by Sagem and Euristic and by S_2O respectively. Since they have similarities, only Chronos, the most real-time oriented of them, will be described.

Availability

Chronos is written in ADA and runs on IBM PCs, VAX, and UNIX machines. The system costs between 60.000 and 120.000 FF. Chronos can be used as a stand alone system or as an ADA package that can be embedded into an existing ADA application.

Knowledge Representation

In Chronos knowledge is represented as facts and rules. Facts are represented as object–attribute–value triplets. To handle time, four dates can be associated with a fact: creation time (the time when the fact is entered in the database), starting time (the time when the fact becomes valid), ending time (the time when the fact ceases to be valid), and the obsolescense time (the time interval, beginning at the ending time, after which the fact is removed from the database). The dates can be given in absolute form, i.e., in terms of day and time, or be relative to, e.g., the current time.

Chronos has two main rule types: "*as soon as* conditions *then* actions" and "*as long as* conditions *then* actions". The first type is equivalent to an ordinary *If* rule. In the second type, a link is created between the time stamps of the conditions and the time stamps of the conclusions. Any modification of the time stamps of the conditions is propagated to the time stamps of the conclusions. This leads to a way of handling non-monotonic reasoning.

The rule action part consists of a structured procedural language with conditional branching and loops. Within the actions, facts can be manipulated, rules can be explicitly fired, and external functions can be called. All actions can be postponed for a defined time period.

Chronos is a strict forward chaining production system based on a modification of the RETE pattern matching algorithm. Priorities can be associated with rules.

Real-time aspects

The different dates that can be associated with a fact give interesting possibilities to represent temporal facts. For example, it is possible to assert that a fact should not become valid until some future date.

An example of a rule using some of the features is shown below.

```
rule name_1;
priority := 4;
uninterruptible;

as soon as
  !x:=100;
  temperature(!reactor) >= !x [!t1,!t2];
  clock >= !t1 + 30;
  no(exists associated_valve(!reactor)=!valve [!t3,!t4] such
     that state(!valve)=opened, !t3 =< !t1+5.0, !t4 > clock);
then
  put_line("Warning problem with reactor"); put(!reactor);
  call "action_1.exe" (!reactor:in,!t1:in);
end rule;
```

The rule expresses that as soon as the reactor temperature is greater or equal to 100 degrees for the last thirty seconds and none of the valves associated with the reactor were opened within the five following seconds, then the operator should be informed and the external procedure action_1 be executed. Pattern matching variables are preceded by an exclamation mark. `Clock` returns the current time. The notation $[t1, t2]$ denotes the starting time and the ending time of the fact validity.

Chronos also supports the notion of *situations (événements)*. A situation is a way of grouping together facts that are true over subsequent time intervals. Consider the following example:

```
state(process) = normal [100,200]
state(process) = normal [200,300]
state(process) = normal [300,400]
state(process) = normal [400,500]
```

This constitutes a situation that begins at time 100 and ends at time 500. The beginning and end of the situation that a fact belongs to can be referenced using the following syntax.

```
state(process) = normal [!t1,!t2] {!t3,!t4}
```

with `!t3` and `!t4` referring to the beginning and ending time of the situation.

Development Interface

The development interface is based on multiple windows and mouse and menu interaction. It allows for rule editing and display of the rule flow chart with zoom and scroll possibilities.

During execution four windows are used to display acquired and deduced facts, justifications of deduced facts, and execution trace. The command window allows an interactive dialogue during execution.

Comments

Chronos is a purely rule based system and, thus, has limited applicability. The real-time constructs are, however, interesting as well as the possibility to embed Chronos in other ADA systems.

8.1.5 Cogsys

COGSYS (Cognitive System) is a real time expert system product aimed at the process industry. COGSYS is supported by a collaborative club which was launched 1987 in the UK. COGSYS is a continuation of the Alvey project RESCU. The club consists of about 40 members. The members are mainly from the process industry, and suppliers of process control equipment. ABB Automation is one of the members of the club. System Designers - Scicon is the system contractor.

COGSYS is divided into a generator system and a run-time system. A knowledge representation language has been developed. COGSYS is a frame based system with inheritance of attributes. The rules can be generic and the rule scheduling can be controlled in real time. The knowledge base can be divided into blocks to support real time scheduling. High targets for speed benchmarks have been set up – the goal is to develop a fast system. COGSYS currently runs on VAX/VMS in POPLOG.

The base development of COGSYS is nearly finished. Two test site installations are under commissioning. The next step is to establish a marketing organization and to work with the support and future development of COGSYS.

8.1.6 Domain-specific tools

Expert system tools specially developed for a single type of application such as, on-line diagnosis, are beginning to emerge. These systems are often based on a single knowledge representation formalism, e.g., fault trees. This can make the systems quite inflexible. They often run on smaller hardware such as IBM PCs.

RES-D2

RES-D2 (Real-time Expert System Shell – Diagnostic Domain) from ARS in Italy is one example of a small on-line diagnosis tool. RES-D2 is based on two concepts: fault trees and observables. The fault trees describe the possible faults and their interrelations. Observables represent the entities that the system can measure, e.g., analog process values, digital bit values, etc. Associated with each observable is a measurement procedure that is fired when it is necessary to get a new value for the observable.

RES-D2 runs on IBM PC/AT and has been developed in Golden Common Lisp.

GDS

Combustion Engineering, Inc, has developed GDS, "Generic Diagnostic Shell", a software shell for real-time diagnosis (Neuschaefer *et al*, 1987). GDS includes a module for automating knowledge acquisition through an automated fault tree construction tool. Both heuristics and fault trees that describe critical safety functions are used. Critical safety functions are defined as high-level process functions that must be maintained to ensure safety. GDS operates in two stages. First, it uses heuristic knowledge to interpret plant data for symptoms of casualty. If one exists, the affected critical safety functions are identified. Then GDS examines the fault trees of those safety functions to identify the cause of the casualty.

8.1.7 Plexsys

The Plant Expert System (Plexsys) development tool is an add-on to KEE from Intellicorp specially designed for off-line power plant applications. Plexsys has been developed in cooperation with EPRI (Electric Power Research Institute).

Plexsys is designed to aid the electric power utilities in the development of knowledge-based application for specific nuclear applications. In addition to the facilities of Plexsys, the user has full access to KEE and Lisp. Plexsys is built around the idea that the understanding and description of processes is centered around graphical forms such as Piping and Instrumentation diagrams (P & IDs) and electrical diagrams. Such diagrams define a graphical "model" of the plant knowledge that is common to many applications such as analysis of system reliability, the evaluation of valve and component configurations during maintenance, and the predictive analysis of operational transients and accidents.

The basic components of Plexsys are described in terms familiar to plant personnel: valves, tanks, motors, pipes, pumps, etc. These elementary components are more than just simple pictures on the screen – they also encapsulate the knowledge that describes the constituents of an actual component and, more important, how it behaves as a part of a functioning system. Plexsys supports

composite components, i.e., the process can be hierarchically decomposed into components, units, etc.

The Plant Model Editor

The major component of Plexsys is the Plant Model Editor. This is a graphical editor for creating a model by creating, moving, grouping, and deleting components using a mouse and window interface. Components are selected for addition to the model from graphical menus which display libraries, groups of devices, and components. Relations between components are modeled as graphical connections. Schematics are used to group components together in order to break a diagram into conceptually more manageable parts. The functionality of a group of components can also be represented by a single component called a composite. A composite has ports that allows it to be connected to other components or composites, and provision for equations to model the relations between its ports.

Plexsys allows multiple views in the knowledge base. Components that have functions in several different contexts can be represented as a single object, but with distinct properties, icons, and interconnections in the different contexts.

The Plexsys Browser

The Plexsys Browser is designed to assist an end-user in

- determining what diagrams that contain specified components or component classes, and display those diagrams,
- displaying the associated working area or *canvas* of those diagrams,
- determining what components and component classes are viewable in any of those diagrams and list them, and
- determining and indicating the name of a component from its picture.

The Network Inspector

The Network Inspector is a tool within Plexsys to assist the user in interrogating, validating, and analyzing the plant model. It allows the user to search the plant network starting at a given component, then moving along connections between components. The Network Inspector is used to answer questions such as:

- "What are all the paths, or the optimal path, from one component to another?"
- "Are all the components along a path operable, or available?"

- "What is the hydraulic isolation boundary for performing maintenance of a component?"

From a technical point of view Plexsys contains several interesting ideas which are very similar to ours. For example, Plexsys is one of the few systems that support multiple views. Plexsys is, however, not intended as an on-line tool, one of the reasons for this being KEE which is difficult to apply in real-time. To our knowledge Plexsys has not had any commercial success. A few electrical utilities have tried it and also some oil refinieries.

8.2 PROCESS CONTROL

The technical development of process control systems is too large and diversified to be described in a single chapter. Here, a small selection of some trends and new systems that we believe are relevant for the development of KBCSs will be presented.

8.2.1 Intelligent process components

There is currently a trend towards more intelligent process components. The trend is most observable for sensors. "Intelligent sensors" combine the actual measuring with basic numerical pre-processing of the measurements.

However, also other process components have built-in intelligence. Pumps that include local monitoring systems for condition monitoring and detection of abnormal operating conditions is one example.

8.2.2 Distributed control systems

New generations of distributed process control systems are beginning to adopt programming techniques, e.g, object-orientation, similar to what is found in KBSs. A good example of this is Sattline from SattControl AB.

SattLine

SattLine is a distributed control system from SattControl AB. It consists of both hardware and software with integrated communications. The Sattline software has a uniform graphical language, using a common programming environment and distributed execution. It is hardware independent and uses distributed intelligence linked by transparent communications. The system software is object oriented, supports module libraries so that control solutions can be reused, and is self-documenting. The operator interface supports information zooming and windowing and is created along with the control software.

Object oriented programming: SattLine includes a uniform graphical object oriented programming language, which reaches from the simplest local controller to the plant supervisory level.

Modules in a program correspond to physical objects in the plant: valves, pumps, or complete subprocesses. Each module contains everything which relates to the object, including control program, operator interface, management information, and diagnostics.

Modules can be duplicated if more than one object of a certain kind is needed. These copies, or instances, are related to each other in the way that when a modification is done in one of them, the same change occurs in all of them. However, there is no superior instance or class definition, and therefore no sub-/superclass structure. However, a module can consist of one or more submodules. In other words, SattLine does not support class structures and inheritance, but does support "consists-of" structures.

A module instance can be "decoupled" from the other instances of the same definition, and thereby a new definition is created. After this operation, modifications in this instance does not any longer affect the other instances.

When sequential functions are required, for example for the control program of a module, these are represented using the Grafcet notation, which is especially powerful when displayed with dynamical status on the operator station. Interlocking and logical functions can be specified using ladder diagrams and/or functional blocks.

After the modules needed have been created, the control program is constructed using cut and paste operations on these modules. Modules are connected graphically with a link that can pass any type of data - single bits, numerical values, recipes, batch data, production parameters, etc.

Control program language: As mentioned before, each module can have its own control program. This control program can be implemented using the Grafcet notation, where each transition and state is an equation block. The control program could also be implemented as a single equation block.

An equation block is a list of statements. There are four kinds of statements: equations, if-equations, procedures calls, and comments. Equations have a left hand part that must be a variable, and a right hand part that is an expression. IF-equations are used when different sets of equations are conditionally valid. An if-equation may contain any number of branches. Predefined procedures can be called from equation blocks. Procedure calls consist of a procedure name, followed by a parameter list enclosed in parentheses.

On evaluation, an expression will yield a simple value: boolean, integer or real. Operands may be variables (local or global), or literal values. Operators available

are boolean (AND, OR and NOT), arithmetic (+, -, * and /), and relation operators (=,<>, >, >=, < and <=).

All variables used in a program must be declared. In the declaration the variable is given a name, and is associated with a data type. Optionally, an initial value may be specified.

The language supplies the predefined simple data types boolean, integer, and real. Variables of these types may be used in expressions. The data types string and timer are also predefined. In addition to these, the user can create his own data types, as records. A record is an aggregate of variables. It can contain any number of components, and each component can be either of a simple type or of another, previously defined, record type.

Libraries: A large amount of standard modules exist. These are grouped into libraries based on functionality, for example communication, control, calculation, I/O, and report libraries. The users can use these modules by duplicating them or define new modules by customizing them. The users can create libraries of their own, with standard as well as user-defined modules.

Operator interface: SattLine employs the SattGraph operator interface. The software in a SattLine system consists of modules, each of which has its own operator interface, i.e., graphical representation and interaction objects.

"Information zooming", one of the basic features of SattGraph, together with window techniques give the operator split vision – the ability to monitor detailed operations whilst keeping watch on the plant overview. By pointing at an object it can be zoomed in. The size of the object is chosen as large as possible while still being contained in the window. Any part of a window can be zoomed in in the same way. It is also possible to zoom up or down a whole window. When objects are enlarged, hidden information is shown and icons are replaced by more detailed graphical information. There can be many layers of information on top of each other in the same module.

Distributed environment: A Sattline system may consist of several control systems, personal computers, and minicomputers. But there is only one software program, "the global program", for the whole system. After the software is developed the designers tell the system were different parts of the software should execute. The system then automatically carries out the down-loading of the software and sets up communication links between the different parts. If the system is reconfigured, new instructions can be given and a new, and different, distribution will take place.

As a true distributed system, SattLine is always in control. Should, for example, a supervisory system fail, then the plant controllers continue their local operation unaffected.

Communication: Communication is an integral part of the SattLine system, with conformity to international standards throughout. At the heart of the system is a carrier-band MiniMap network (the real-time implementation of the MAP protocol). At higher levels Ethernet is used for communication with VAX-stations and other computers.

Open system: SattLine is based on an open architecture. There is a well defined interface which allows integration with other software. E.g., for connection to relational databases, SattLine provides a SQL interface.

8.2.3 Intelligent controllers

A clear trend towards local control units with increased abilities for automatic tuning, adaptation, and diagnosis can be seen. The research area of expert control uses KBS techniques to implement the knowledge involved in these systems. ECA 400 is a good example of a commercial controller along these lines.

ECA 400

ECA 400 from SattControl is a PID controller with relay-based auto-tuning, continuous adaptation, adaptive feedforward compensation, and the possibility to have gain-scheduled controller parameters based on the set-point, measured variable, or the control signal. The relay method automatically finds a set of PID parameters for the process. Three different sets of parameters can be stored in a gain-schedule, each set obtained through tuning in a different operating region. If, e.g., the process is non-linear, the controller changes parameters depending on operating region. For processes that have time varying dynamics, the continuous adaptation makes it possible to continuously adjust the parameters in accordance with the changes in dynamics.

8.3 OBJECT-ORIENTED DATABASE SYSTEMS

Object-oriented database systems (OODBSs) is currently a very active research area (Atkinson *et al*, 1989). OODBSs aim to combine the strong features of conventional, e.g., relational, database management systems (DBMS) with object-oriented ideas.

Conventional DBMS provide support for data persistence, disk management, sharing of data between multiple users, data reliability and security, and simple *ad hoc* query languages such as SQL. In OODBS this is combined with support for complex object structures; encapsulation of both data and behaviour, i.e., code, into objects; classes or types; inheritance; extensibility; etc. Several first generation OODBS products have been released during the last few years including Gemstone, G-base, and Statice.

Gemstone: Gemstone from Servio Logic was released as a product in 1988. It target markets are office automation and CAD. Its is implemented in C and uses Smalltalk as its method language.

G-base: G-base from Graphael, France, allows multi-media applications and has a user interface with hypertext facilities. G-base has a set of optional tools including a graphical browser, a menu-based query interface, and a PROLOG-like programmable query language. G-base is implemented in Lisp and available on Sun, Apollo, and different Lisp machines.

Statice: Statice from Symbolics, USA, allows database management facilities for object-oriented applications on Symbolics machines. Statice is implemented in Lisp.

Prospects

Although commercial products are emerging, OODBSs should still be considered as a research area. The available products follow no accepted standards and are, to a large degree, still prototype systems. Most of them require powerful computers and several are implemented in Lisp on Lisp machines.

OODBSs are important for the development of knowledge-based control system since they can be seen as one possible future way of implementing the common knowledge base concept. However, there is still a long way to go before this may come true. Up to now, OODBSs have not been focussed on real-time applications. For this project distribution is a specially important criterion. Here, the technique is not yet mature even for conventional relational databases.

8.4 OMOLA

Omola is a general language for representing models of dynamic systems. The language is based on ideas from object-oriented programming and the name is short for *Object-oriented MOdelling LAnguage.*

Omola is one of the outcomes from a larger project in computer aided control engineering – CACE (Mattsson, 1989) – that is performed at the Department of Automatic Control, Lund Institute of Technology. In this project it was realized that models play an essential role in engineering and in particular in the design of control systems. Most simulation languages and model representations used in various design tools are too specialized and inflexible to be used as a general modelling language. Omola has been designed to overcome these deficiencies.

One of the main goals of the CACE project was to design an integrated environment of cooperating tools supporting the various stages in process and control

system design. Omola contributes as a common ground, or a core model representation, around which the tools may be arranged. The core model representation serves as a common database and a communication channel between the tools.

The design of Omola reflects the following important properties of a modelling language.

- The language should support a number of mathematical and logical frameworks for representing model behaviour. For example, differential algebraic equations (Mattsson, 1989b), transfer functions, state space descriptions, discrete events, and qualitative behaviour.
- It should include concepts for structuring of large models, for example, hierarchical submodel decomposition as in Dymola (Elmqvist, 1978).
- It should be modular in order to support reuse of parts of models in other models.
- It should be possible to include "redundant" information in models for the purpose of documentation and automatic consistency check.
- It should be generally useful as an input language for different control design tools and simulators. It should also be useful for model documentation and as a standardized exchange language between users and tools. This means that it must fit within an interactive CACE environment.

8.4.1 Data modelling in Omola

Omola is designed to describe structure and behaviour of dynamic systems. However, it is based on a few very general concepts of object-oriented data structuring. This makes Omola generally useful as a data modelling language.

The basic entity that can be defined in Omola is called a class. A class defines a data type which has a name and a number of attributes. The attributes defines the properties of the class and they can be ordinary named variables of a defined type (real, integer, string, etc.) or they can be other class definitions, so called components.

Classes are arranged in a hierarchy such that every class has one super-class. A class will inherit all attributes present in its super-class. An inherited attribute belongs to a class in the same way as if it was defined locally in the class. If a local attribute is defined with the same name as an inherited attribute, the local definition will override the inherited one.

The general form of an Omola class definition looks like:

```
<name> IS A <super class> WITH
```

```
  <body with local attribute definitions>
END
```

where the class bode may contain other class definitions or variable definition on the following format.

```
<name> TYPE <type name> := <binding expression>
```

The binding expression in a variable definition is optional. It binds the variable to a specific value or an expression.

Let us now regard a few examples of how Omola can be used to represent structured data. Suppose we want to represent cars and start by defining the Omola class `Car`:

```
Car IS A Class WITH
  parts:
    body     IS A Car_body;
    engine   IS A Combustion_engine;
  properties:
    prize    TYPE real;
    fuel     TYPE string := "gasoline";
END;
```

The super-class of `Car` is `Class` which is predefined in Omola. The `Car` defines four attributes: `body` and `engine` which are components and `prize` and `fuel` which are variable attributes. The words '`parts:`' and '`properties:`' are keywords dividing the attributes into *categories*. Categories are used to structure the attributes into groups according to their different roles in the model. For example, in this case we can ask questions to the data base like: "What are the *parts* of a car?" and "What are the *properties* of a car?".

Now we can use inheritance and specialize the `Car` class into a class representing a special type of cars (a subclass) called `Mercedes`:

```
Mercedes IS A Car WITH
  fuel := "diesel";
  parts:
    stereo IS A Car_stereo;
END
```

The `Mercedes` class rebinds the `fuel` attribute inherited from `Car` and adds another component called `stereo`.

Composition and specialization are two most important concepts for structuring process models and other kinds of data. In the examples we have seen how these concepts are supported by Omola. Composition is accomplished by classes that have other classes as attributes. Specialization is accomplished by subclasses and inheritance.

8.4.2 Model representation in Omola

We will now see how models of dynamic systems can be represented in Omola. The discussion is based on a basic set of concepts for model structuring covered in more detail in (Mattsson, 1988) and (Andersson, 1989).

A *model* is the main structural entity. A model contains a description of its interface to the environment, its dynamic or static behaviour and its parameters. Models can be developed and tested, saved in libraries, and reused as submodels in different contexts.

A model is an encapsulated module where the interaction with the environment is limited to certain variables called *terminals*. A model can have any number of terminals. Very often a model represents a physical component such as a pump, a valve, or a regulator, and the terminals represents physical quantities like mass flow, electric voltage, etc. Models can be parameterized in order to make them more flexible and adaptable to different circumstances.

Here follows an example of a simple model definition in Omola. It defines a tank model with two terminals and two parameters.

```
Tank IS A Model WITH
  terminals:
    inflow IS A Terminal;
    outflow IS A Terminal;
  parameters:
    tank_area := 5.0;
    outlet_area := 0.05;
END
```

The tank model is defined as a subclass of Model with the terminals and parameters defined as local attributes. The terminals are component attributes, i.e., they are classes defined as subclasses of the predefined class `Terminal`. The parameters are variable attributes bounded to some default values. A parameter default value can be changed in a subclass of Tank or in an instance involved in a simulation. Because these attributes are defined in the parameter category, a tool using this model (e.g., a simulator) may assume that they are time invariant.

The Tank model defines only the model interface and not the model behaviour. Behaviour definitions are considered as model components called *realizations*. Inheritance is used when defining a new tank model with non-linear behaviour specified as two equations:

```
NL_tank IS A Tank WITH
  realization:
    Re IS A  Primitive WITH
      variable:
        level := 0.0;
```

```
      equations:
        tank_area * dot(level) = inflow - outflow;
        outflow = outlet_area * sqrt(level);
    END;
END;
```

This new tank model will inherit the terminal and parameter attributes from `Tank` and add a realization component. The realization is a subclass of `Primitive`, which is another predefined class, and it has a variable and two equations as attributes. One advantage of separating the tank model into two different classes, one defining the interface and one defining the behaviour, is that we can define alternative tank models with different realizations but with identical interfaces. When a tank is used as a part of large plant model, it is easy to exchange tank models with different realizations. This is a good example of how the object-oriented approach achieves both modularity and reusability of models.

The `NL_tank` was an example of a primitive model, i.e., it had a realization that was based on differential equations. Models can also get their behaviour definition from a set of connected submodels. Such a model is called a *structured* model. For example, we can define a new model composed of two connected tanks:

```
TankSystem IS A Model WITH
  terminals:
    inlet  IS A Terminal;
    outlet IS A Terminal;
  realization:
    TankStructure IS A Structure WITH
      submodels:
        tank1 IS A NL_tank;
        tank2 IS A NL_tank;
      connections:
        inlet AT tank1.inflow;
        tank1.outflow AT tank2.inflow;
        tank2.outflow AT outlet;
    END;
END
```

The realization of this tank is a class with two categories of components: submodels and connections. The submodels are subclasses of the previously defined `NL_tank`, while the connections are written in a special syntax relating terminals of the submodels and of the tank system.

We have seen Omola used for representing models in a modular way. Models can be defined as classes which can be specialized in various directions and used as components in other models. Also terminals can be structured in a similar way. The terminals used in the examples here have been on the very simplest form. We

can also define terminals with additional attributes defining the physical quantity, unit of measure, range, etc. Interaction between model components involving a set of quantities can be represented by structured terminals. For example, a terminal modelling a pipe connection may have pressure, temperature and flow as component terminals.

8.4.3 An interactive environment

Omola is intended as a textual format for models represented in a core data base in an environment of cooperating tools for control systems design. The environment will include tools for manipulating models graphically, and for browsing libraries of models and model components. Most of the time the user of such modeling tools will not have to write or modify Omola code directly, but rather, he defines and modifies the models incrementally by using mouse and menus and graphical editors.

Simulation is only one way to use a model represented in Omola. Other tools in the environment may use the models for other purposes. Here are some examples:

- Generating a graphical picture of the system structure, for example, a block diagram.
- Generating text descriptions of the system for documentation or user information.
- Generating special purpose code, for example, regulator code or simulation code in other simulation languages.
- Generating standardized system descriptions in order to communicate with other control engineering packages.
- Derivation of different kinds of systems properties like stability margins, loop gains, etc.
- As input to various control design tools.

8.5 INFORMATION PRESENTATION SYSTEMS

Graphical information presentation systems have two possible architectures, as shown in Fig. 8.2.

The most common alternative today is to have a user interface toolkit and a number of user interface tools. The other alternative, which is more advanced and less common, is to have a user interface management system. Both architectures

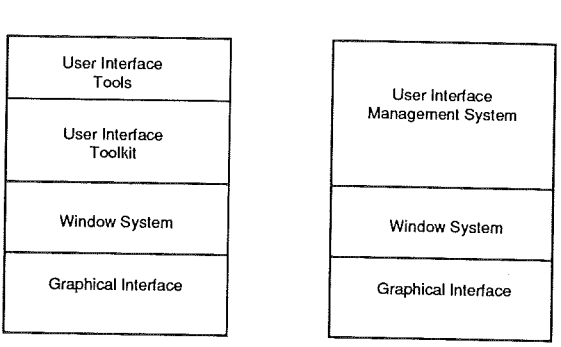


Figure 8.2 Two alternative architectures for graphical information presentation systems

have a graphical interface and a window system. The architectural components are discussed below.

Also relevant to real-time systems are interactive editors for editing dynamically updated pictures. These are also discussed below.

8.5.1 Graphical Interface

There are a wide range of standards for the graphical interface to display terminals. The standard interfaces give access transparently to many different hardware devices. The same application code can use different devices.

The most important graphics standards are:

- Graphics Kernel System (GKS)
- Programmer's Hierarchical Interactive Graphics (PHIGS)
- PostScript, from Adobe Systems

8.5.2 Window System

The most important window systems today are network-based, which allow different windows on one terminal to be simultaneously connected to different computer nodes. Some network window systems allow different types of operating systems and hardware to be used in the same network.

The most important window systems are:

- X Window System, from M.I.T.
- NeWS, from Sun Microsystems

Of these, the X Window System is more widely accepted, although NeWS is technically more advanced.

8.5.3 User Interface Toolkit

User Interface Toolkits are program libraries for programmers to build user interfaces. This area of technology is not yet mature and there are many products with different approaches and varying degrees of sophistication. They can be roughly divided into object-oriented and non object-oriented. The object-oriented toolkits are designed to be used with object-oriented languages such as C++.

The most important non object-oriented user interface toolkits are:

- OSF/Motif, from Open Software Foundation
- Open Look, from Unix International

Object-oriented user interface toolkits exist (for example, InterViews from Stanford University, CommonView from Glockenspiel and NextStep from NeXT Computer Systems), but it is too early to say which will become the most important standards.

8.5.4 User Interface Tools

User interface tools are not the same as user interface toolkits. A toolkit is program library to be linked with the application. Tools are separate standalone programs to be used in conjunction with a toolkit. User interface tools are not essential, but they can reduce program development times.

Examples of user interface tools are:

- Bitmap editors

Interactive graphical editors for designing icons.

- Presentation Editor

A visual tool to interactively build screen layouts containing dialogue boxes, scroll bars and other interactors or widgets. For example, Interface Builder from NeXT Computer Systems.

- Presentation Description Language

Also for building screen layouts, but using a language instead of a visual editor. An example is User Interface Language (UIL) from DEC.

- Dialogue Description Language and Interpreter

A language to describe the dialogue paths in the user interface and an interpreter to execute the description.

8.5.5 User Interface Management System (UIMS)

A user interface management system is more advanced than the toolkit/tool approach because the application and the user interface are more completely separated. The connections between the application and the user interface are described by an Application Interface Model. UIMSs also contain Presentation Description and Dialogue Description tools.

There are very few commercially available products that use the UIMS technology (separation of UI and application functions). Many manufacturers call their products UIMS, in the same way as many products some years ago was called DBMS, although they were not real DBMSs.

An example of a UIMS is TeleUSE, which is recommended by the Open Software Foundation. There are no standards for UIMSs yet.

8.5.6 Editors for Dynamic Pictures

There are a few products in the market for implementing full graphics pictures that are dynamically updated from application data.

The reason why the products are called editors is that there is normally an interactive editor in the products. In some products there is also a philosophy that the application uses the same "interactive" commands to define and update pictures.

Examples of editors for dynamic pictures are EASE+, DataViews, and Sherrill-Lubinski Editor. There are no standards in this area yet.

8.6 FUZZY CONTROL

As pointed out in the travel notes, the activity in fuzzy control is currently very high in Japan. LIFE (Laboratory for International Fuzzy Engineering Research) was started last year with 48 of Japan's largest companies as sponsors. One of the main focuses of LIFE is fuzzy control. In Japan over 100 industrial fuzzy control application have been reported including coordination control of elevators, brake control systems for subway trains, etc. Several large international conferences on fuzzy techniques have been announced lately.

Special chips for execution of fuzzy rules are becoming available. One example is DFP (Digital Fuzzy Processor) FC110 from Togai. FC110 is single chip, VLSI co-processor that can be added to IBM PCs, Suns, and Apollos. FC110 has a capacity of over 100.000 fuzzy rule evaluations per second. DFP also allows fuzzy logic rules to be compiled into portable ANSI C code.

8.7 RESEARCH ON KBS AND PROCESS CONTROL

The research and development concerning knowledge-based applications within the process industry is concentrated to four different sources: the process industries, AI companies, control system suppliers, and academic institutions. Much of the activity going on is taking place within some of the national research programmes such as DUP in Sweden or within the international research programmes such as ESPRIT.

8.7.1 Process industries

The interest among the process industry for KBS applications continues to be high. Examples of industry branches where the activity is high are the chemical industry, the steel industry, the pulp and paper industry, the power industry, and the manufacturing industry. Within the nuclear industry alone, 298 expert systems were reported by June 1989 (Bernard and Washio, 1989). Which industry branch that dominates varies from country to country. In USA, the chemical industry has a strong position with Du Pont (Rowan, 1989) as the leader in KBS applications. In Japan, the power industry and the steel industry have the highest activity. In Sweden, the pulp and paper industry dominates.

8.7.2 AI companies

AI consulting companies specially directed towards process industries continue to emerge. Most of them also develop their own KBS tools with G2 from Gensym as a good example. Other examples of AI companies are Cambridge Consultants, Framentec, PA Consultants, Infologics, Epitec, Stone and Webster, Scicon, and SIRA Ltd.

8.7.3 Control system suppliers

All major control system suppliers are active in the field. So far, the work being done is restricted to "interfaced solutions". However, Japanese companies like Toshiba, Hitachi, and Yokogawa have gone one step further. They develop their own expert system tools with a tight interface to their existing control systems. In some cases the systems share end-user interfaces.

Honeywell: Around 1986, Honeywell was involved in the Cooker project, see the Feasibility Study, a knowledge-based system for monitoring of batch processes. According to the information we have this has been further developed into a commercial product with the possible name of "Conchshell". The system was supposed to be released last autumn. The group at Honeywell doing KBS research consists of around 40 persons. Honeywell has also implemented the forward chaining system OPS83 within TDC 3000.

Foxboro: As described in the travel notes of the Feasibility Study, Foxboro have made it possible to include Personal Consultant Online in their new control system I/A (Intelligent Automation). Foxboro has been involved in KBS activities for a long time. They were one of the participators in the Falcon project. They have also looked at model-based systems and alarm analysis.

Siemens: Siemens was a member of the German Research programme TEX-I (Technische Expertensysteme zur DatenInterpretation, Diagnose und Prozessfuhrung) together with Bayer AG, Elektronik System Gesellshaft, Interatom, Krupp Atlas, Fraunhofer Institut, and GMD. In that project the German tool Babylon was used in a Symbolics environment. Partly as a result of this project, Siemens, through Interatom, has developed an interface between G2 and Simatic S5.

Combustion Engineering: Combustion Engineering, now a part of ABB, has developed the previously described GDS system for on-line diagnosis. Combustion also work with G2 and have developed a special system towards the pulp and paper industry based on G2.

Bailey Controls: Bailey has developed Expert 90 (Oyen *et al*, 1988), a rule-based module that can be embedded within Bailey's Network 90 control system. The rules share the memory of the the other control blocks and is processed by the same processor. The rules can use simple temporal expressions and allow for fuzzy logic. Some examples of how rules may look like are:

```
IF CAVITATING
THEN FOR 30 SECONDS
 INHIBIT_PUMP_ON
END

IF CERTAINTY PUMP_FAILURE > 30%
THEN PUMP_SUSPECT
END
```

Bailey also use expert systems within the organization.

Toshiba: Toshiba activities on expert systems are described in the Travel Notes. Toshiba's solution consists of three parts: a process computer (G8050), a special purpose Lisp processor (IP704) running the TDES3 expert system tool, and a separate engineering workstation for off-line development of the knowledge base. TDES3 uses facts, production rules, schemas, and procedures for knowledge representation. The process computer and the Lisp processor communicates

through a VME bus interface. The operators use the same console to both the process computer and the KBS.

Yokogawa: Similarly to Toshiba and Hitachi, Yokogawa has developed their own tool, XL/AI, that can be interfaced to the CENTUM system. Japanese Gas Company has developed a G2 interface to Centum.

8.7.4 Research programmes

The research programmes that are most central to our project is DUP in Sweden and Esprit in the European Community. They will be described in some detail here.

DUP

DUP (Development of User-friendly operation systems for the Process industry) is a Swedish research programme funded by the Swedish National Board for Technical Development. The programme is focussed on the end-users of modern control systems within three selected branches: the chemical industry, the pulp and paper industry, and the food engineering industry. DUP is an interdisciplinary programme composed of Computer Science, Automatic Control, Process technology, Cognitive Psychology, and Environmental Sciences. Important areas with DUP are knowledge-based systems for operator guidance and support and the use of simulation techniques within the process industry.

Several projects within DUP are related to this project.

KE2000: SCA/Teknik and Uppsala University have a DUP project where they develop a real-time expert system for monitoring of the pulp line at the Östrand plant. The real-time expert system is based on Prolog and the operator interface is built in Supercard. The system executes on Macintosh II computers.

A fundamental part of the project is the organization of a future process control room. The expert system part of the system will run on a separate operator station, the KE-station 2000 (Knowledge – Experience) which should be well integrated with the rest of the equipment in the control room.

Billerud/Skoghall: The previously discussed project at Skoghall where Stora Teknik and The Karlstad Institute of Technology apply knowledge-based systems to the monitoring of a continuous digester is a part of DUP.

Frövifors: At Frövifors, the expert system tool Epitool is used for closed loop control of a pulp washing process.

Knowledge techniques in the Food Engineering Industry: AB Felix performs a study together with Infologics on how multi-media based knowledge system can be applied in a mashed potato plant.

HAZOP: Nobel Chemical together with the Department of Chemical Engineering, Lund Institute of Technology, and Masic AB take part in a Nordic project where qualitative methods are applied for various process applications. The DUP part of the project considers qualitative methods for HAZOP based risk analysis of chemical processes.

ESPRIT

The ESPRIT programme (European Strategic Programme for R & D in Information Technology) was defined after an analysis undertaken in close liaison with EC industry in 1982 and 1983. The main reason for the programme was to improve the competitive ability of the European Community. ESPRIT has the following three objectives:

- to provide European Information Technology (IT) industry with the basic technologies to meet the competitive requirements of the 1990s,
- to promote European industrial cooperation in IT, and
- to pave the way for new standards.

The first phase of the programme started in 1984 with ESPRIT I. The total budget of ESPRIT I amounted to 1,500 MECU (about 10,500 MSEK) with 50% coming from EC and 50% from industry. About 3000 full-time engineers and scientists have been engaged in 226 projects. About 526 organizations have been involved.

The second phase, ESPRIT II, is a five year programme from the 1st December 1987. It is a larger scale programme than ESPRIT I and the budget is 3,200 MECU (about 22,400 MSEK).This figure represents about 5% of the R & D expenditure in the IT industry in EC. Relative to the long term R & D, the percentage is much higher. The catalytic effect of ESPRIT has played an important role. In the fact the IT industry R & D has grown to the same level as US IT companies in terms of percentage of turnover.

The first call for proposals to the ESPRIT II programme resulted in 156 project contracts with 585 participating organizations. This represents about half of the ESPRIT II programme. The second call for proposals is scheduled for January 1990.

The Bidding for Research contracts

ESPRIT projects can be started only in competition with other interested groups. This is done by a bidding procedure. An ESPRIT work programme describes in detail different areas of interest for EC. Project proposals are prepared from the different groups in order to meet the requirements of a specific work programme description. Normally, only one proposal is selected for contracting in a specific domain. In the second call of the ESPRIT II, the statistic chance to get a specific contract is estimated to 10 - 30%.

In a group bidding for a contract, there has to be at least two industrial partners from different countries. Large companies (more than 500 employees) and universities dominate the ESPRIT programme.

The ESPRIT work programme

The work programme is focused on three strategic sectors:

- Microelectronics and Peripheral Technologies.
- Information processing system.
- IT Application Technologies.

Two areas of application technologies were identified:

- Office and Business Systems.
- Computer Integrated Manufacturing (CIM).

In CIM, all kinds of manufacturing are included – workshops, chemical plants, etc. CIM also includes activities like design, product preparation, control, etc.

Microelectronics and Peripheral Technologies

The microelectronics and peripheral technologies sector was judged to be strategic because microelectronics will influence all advanced technology areas. The market for integrated circuits (ICs) is expected to approach 60 BECU (about 420,000 MSEK) by 1992 and about 20% to 30% of this will be for application-specific integrated circuits (ASICs) required by electronic systems producers to reach optimized solutions.

In ESPRIT I, 49 projects were launched. In ESPRIT II, at least 30 projects have started. In very general terms the sector can be classified into four sub-areas:

- High-Density Integrated Circuits.
- High-Speed Integrated Circuits.
- Multifunction Integrated Circuits.

- Peripheral Technologies.

Technology, CAD, manufacturing, and material issues are treated in each sub-area.

Information Processing Systems

Information Processing Systems is a fast growing commercial sector and is therefore of high strategic importance. For instance, the market volume for packed software has been estimated to 35 BECU by 1993. The number of deployed expert systems in the US jumped from 50 to 1400 between 1987 and 1988.

The sector has more than 137 projects and in ESPRIT II the sector is broken down in five (four in some documents) sub-areas:

- System Engineering.
- Knowledge Engineering.
- Advanced System Architectures.
- Human-Computer Interfaces.
- Sensor-Based Systems.

As knowledge-based systems are of special interest to us, we will go into more detail on Knowledge Engineering.

Knowledge Engineering: In the first call to ESPRIT II the key priorities were for real-time and integration issues. The following projects were started (state in July 89):

Key words	Projects
Real-time	AITRAS
Expert Agents	ARCHON
Front-ends to existing systems	KBSSHIP, ITSIE, FOCUS
KB and database integration	KIWIS, STRETCH
MMC and KBS	PROMISE, MMI2
Knowledge acquisition	ACKNOWLEDGE
Temporal qualitative reasoning	EQUATOR
KBS validation and verification	VALID
Learning	MLT

ESPRIT I did not use the same classification in sub-areas and the projects are in their finishing stages. The ESPRIT I projects of special interest to our project are GRADIENT, QUIC, KRITIC, ESB, and EUROHELP, which all were described in the Feasibility Study. Another ESPRIT I project of interest is SKIDS, Signal and Knowledge Integration with Decisional Control for Multi-Sensory Systems.

In the second call for proposals within ESPRIT II, real-time knowledge-based systems has been selected as a special area of interest (no. I.2.1). The objective is "to develop new hardware/software systems that will integrate the traditional approach to embedded computer systems based on numeric processing with more novel approaches based on the processing of knowledge". The interest area involves identification of factors involved in real-time systems, i.e., "reasoning in specified time constraints, reasoning with incomplete data, reasoning with changing facts, time dependent / spatial reasoning, ability to respond to interrupts, and real-time updating of knowledge bases". The developed system should be demonstrated and evaluated in real, demanding applications (i.e., process control, autonomous agents).

At the present time it is not clear which proposal that will be selected for this interest area.

Application area – Office and Business Systems

The world market for Office and Business systems is estimated to be around 240 BECU by the early 1990s. Under ESPRIT I, 48 projects were started and under ESPRIT II, 39 projects have so far started. The area can be divided in:

- Office Document Architecture.
- Application Systems Engineering.
- Business Systems.
- Human Factors and Human-Machine Interfaces.
- Workstations.
- Networks and Distributed Systems.
- Storage and Retrieval Systems.

A wide range of technologies including KBS is used in the different projects. Standardization issues are also addressed.

Application are – CIM

The Computer Integrated Manufacturing markets are growing fast with about 15 to 25% per year and represents an important market. Especially important is the ability of CIM to improve the productivity and international competitiveness of the manufacturing industry. CIM is an key area for ESPRIT as the potential market is large and the area is not dominated by overseas suppliers.

The ESPRIT strategy in CIM is based on four lines of actions. These are:

- to identify integration paths based on open system concepts and to develop the associated methods and tools,
- to develop sub-systems capable of exploration within this framework,
- to demonstrate the success of this approach and its benefits by early implementation in a wide range of production environments, and
- to build on achieved ESPRIT results.

The CIM sector has at least 74 projects and it may be divided into the following sub-domains:

- CIM Architecture and communications.
- A Manufacturing Systems Design and Implementation.
- Product Design and Analysis Systems.
- Management and Control of Manufacturing Processes.
- Robotics and Shop Floor Systems.

There is an ambition to coordinate and integrate all work in the sector with mainstream CIM-OSA and CNMA developments. All kinds of technologies, including KBS, are used in the projects.

The Basic Research Programme

A Basic Research IT programme has also been added to ESPRIT. At present this programme is quite small with a budget of 63 MECU (about 440 MSEK) during a 30 month period. This programme is supposed to equal similar ones in the US, i.e., basic IT programmes of about 200 MUSD from the US Congress and from DARPA. The Basic Research is divided in three main areas:

- Microelectronics.
- Computer Science.
- Artificial Intelligence and Cognitive Science.

Almost all of the work is done at Universities and Research Establishments. The projects are called "actions" in this programme.

Microelectronics: At least 26 actions are started with the following main targets:

- Low-Noise and High-Speed Devices using super-conduction, etc.

- Fundamental In-Depth Studies of Structures exploring quantum effects or tailoring of electrical properties.
- Nanometer-Scaling of Circuits using organic molecule assemblage, etc.
- Super-fast and Parallel Computing using optical devices.
- Next Generation Design Systems, Methods, and Algorithms.

Computer Science: At least 15 actions have started in Computer Science. The sub-area of architectures is not covered in a proper way. The areas are:

- Formal Systems.
- Concurrent Systems.
- Specification and Verification.
- Algorithms and Integration of Programming Styles.
- Dependability, Data Bases, and Distributed Computing.

Artificial Intelligence and Cognitive Science: At least 20 actions have been launched in the following domains:

- Robotics and Vision.
- Neural Networks.
- Knowledge Representation.
- Speech and Natural Language Processing.
- Formal Theories of Automated Manufacturing.
- Human-Computer Interaction.

Participating Companies

An interesting question is: "Which companies are involved in ESPRIT projects?" We have put together some statistics to answer that question. The most engaged companies seen from the number of projects are:

Company	No of projects
THOMSON	83
PHILIPS	67
BULL	63

GEC	62
SIEMENS	60
AEG	49
ICL	35
OLIVETTI	32
PLESSEY COMPANY	28
NIXDORF COMPUTER	24
BRITISH TELECOM	24
CAP group	21
STC	20
KRUPP	16
CGE	16
MATRA	15
TITN	15
CRI-COMPUTER RESOURCES	13
SYSECA LOGICIEL	13
ALCATEL	12
BRITISH AEROSPACE	12
MARCONI	11
OCE-NEDERLAND	11
SEMA	10

As indicated in these figures, the leading European electrotechnical, electronical and computer industry play a big role in ESPRIT. The automobile industry (Fiat, Renault, Daimler-Benz, Volkswagen) take part only in a few projects, probably because of the existence of a special R & D program outside ESPRIT for car manufacturers.

More interesting is the low profile of the process industry. The Chemical process industry in Europe is supposed to be the biggest in the world. BASF, for instance, is only involved in three projects. BP is not represented at all. The US companies – Dow Chemicals and Du Pont – keep themselves informed through participating subsidiaries. One explanation is maybe the lack of process orientation in the working program. The use of the word "CIM" to cover process control, is a good indicator. Part of the US information technology industry is kept informed by participating subsidiary companies. Bell has 7 projects, Digital has 4, Hewlett Packard also has 4 projects, and IBM has 3 ESPRIT projects. There are rumours that Motorola will open an office just for the purpose of taking part in ESPRIT.

8.8 SUMMARY

Since the time of the Feasibility Study, the development of real-time expert system tools has increased. Several commercial tools are now available. However,

all of them follow the "interfaced solution" architecture described in Chapter 1 with all the problems that follow.

Conventional control system suppliers have increased their activities in the field and several have products on the way. The development involves both connecting expert system to existing control systems and new generations of control systems based on modern programming paradigms, such as object-orientation. The best example of the latter is Sattline from SattControl that implements some of the ideas behind the KBCS concept.

The development of object-oriented database systems is important for KBCSs. OODBS will probably be of value for future KBCS implementations.

9

Description of Future Activities

9.1 INTRODUCTION

This chapter will discuss which activities could form the major part of a continuation of the project. A number of possible alternatives will be suggested and discussed, and some of them recommended for the project.

9.2 ALTERNATIVE COURSES OF ACTION

Possible future activities are as follows:

- Further specification of the KBCS concept.
- Specifying a language for knowledge base representation.
- Further prototyping of the KBCS concept with G2.
- Prototyping of the KBCS concept with other environments such as Smalltalk or C++.
- Coupling the G2 prototype to a control system.

- Prototyping of real-time KBS diagnosis, planning, etc.

9.2.1 Further specification of the KBCS concept

This report presents the framework of a concept for a KBCS system. In order to further develop the concept, further specification work is essential.

9.2.2 Specifying a language for knowledge base representation

Some form of formal description or language (as discussed in Chapter 5.3) would be very useful for implementing the knowledge base. Such a language must be capable of completely describing the knowledge base.

Development of such a language would take place in the following stages:

1. Write a Requirements Specification for the language.
2. Specify the syntax of the language.
3. Implement an interpreter or compiler for the language.

The language can be tested "on paper" before any implementation of an interpreter or compiler is done.

A language is also a suitable means of transferring the knowledge base between different future implementations of the knowledge based control system. It is reasonable to expect that real implementations of knowledge-based control systems in the future will have some form of language.

The language OMOLA, developed at the Department of Automatic Control, has some of the characteristics required for the knowledge based control system. The specification of a new language based on OMOLA is a possible course of action.

9.2.3 Further prototyping of the KBCS concept with G2

G2 has certain limitations that prevent important features of the KBCS from being implemented, particularly the multiple-view object structure of the main knowledge base. However, the prototyping work so far done with G2 has shown that G2 is a very powerful tool, and it can be used to prototype some very important aspects of the KBCS concept.

Aspects of the KBCS concept that are suitable for implementing with G2 are:

- the operator interface,
- diagnosis methods, and
- the knowledge base browser

Typical aspects that are difficult to implement in G2 are:

- the structure of the main knowledge base,
- the design tools, and
- a language representation of the knowledge base

9.2.4 Prototyping of KBCS concept with other environments

There are a number of environments that can be used for prototyping work, e.g. Smalltalk and C++. These environments are not as productive as G2, but on the other hand they are more flexible and can be used to prototype aspects of the KBCS concept that cannot be implemented with G2.

9.2.5 Coupling the G2 prototype to a control system

G2 can be coupled to a conventional control system, which can in turn be coupled either to a real process or to the G2 Steritherm simulator. This configuration would not implement the proposed internal structure for the main knowledge base, but it would test the concept of realization tools for extracting knowledge from the main knowledge base and distributing control system functions to executing units. Realization tools are one of the key features of the KBCS concept.

The conventional control system used can be an ABB or SattControl system or some other system. Whichever system is used, it must have an interface to G2 (i.e. the G2 GSI interface). If this does not already exist it must be implemented especially for the project.

Using a real process would not prove any new principles when compared with the G2 simulator, but would require more effort. The G2 simulator is therefore the best alternative. This configuration is shown in Fig. 9.1.

9.2.6 Prototyping of real-time KBS diagnosis, planning, etc.

This alternative ignores the concept of a main knowledge base and concentrates on various individual knowledge-based aspects of control systems such as diagnosis or planning. Although it is possible to achieve interesting results in these areas, the results will not contribute to the development of the key aspects of the KBCS concept.

9.3 SUMMARY OF ALTERNATIVES

Of the above alternatives, the following appear to be the most promising:

- More detailed specification of the total concept.

More specification work is essential to the project in order to come closer to realizing the KBCS concept.

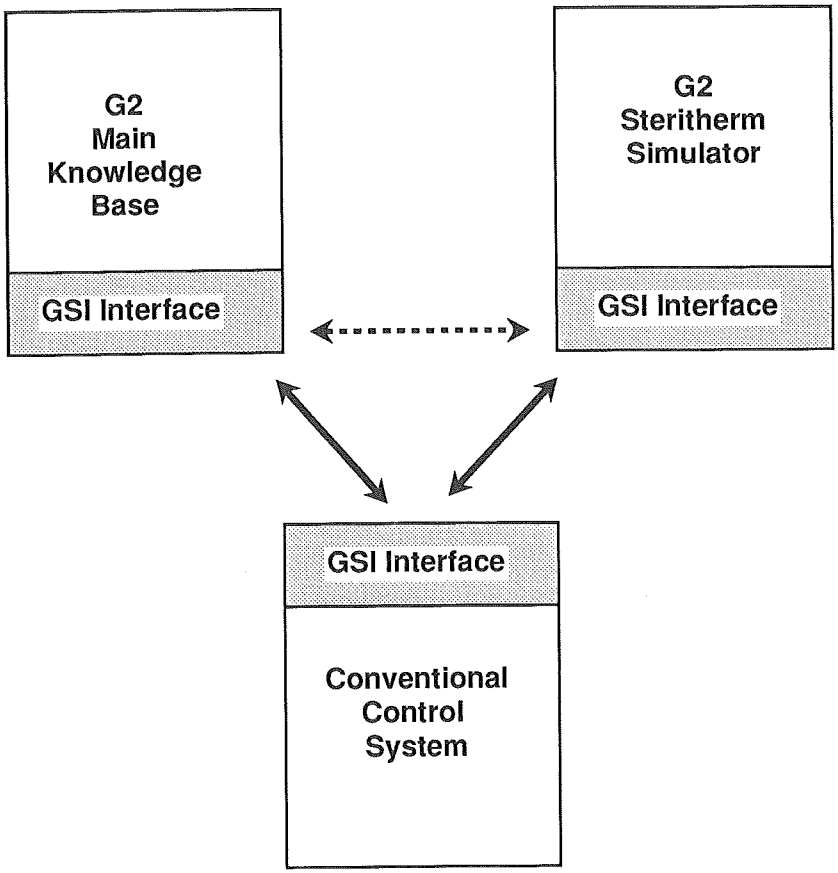


Figure 9.1 System configuration

- Requirements Specification of a language to describe the knowledge base.

It is very important to specify a language, and we must therefore take the first step towards this goal.

- Further prototyping of the the KBCS with G2.

Further prototyping can give important insights into the KBCS concept.

- If possible, connecting the G2 KBCS prototype to the G2 Steritherm Simulator via a conventional control system

This would test the concept of realization tools, one of the key concepts of the KBCS.

9.4 SOFTWARE/HARDWARE REQUIRED

To proceed with the selected prototype, a conventional control system is needed. This could be a SattControl or ABB system, or some other system. The control system selected must have a G2 GSI interface already implemented or specially written for this prototype.

9.5 CONCLUSION

The future activities recommended will give the best possible use of the resources available in the project. Further specification of the concept and of the knowledge representation language are extremely important for the project, and coupling of the G2 prototype and simulator to a conventional control system will allow a key part of the concept (realization tools) to be tested.

10

Summary

The control systems of today are very good at handling quantitative knowledge, expressed as control logic, sequential logic, and procedures. Control systems are, however, poor in representing qualitative knowledge, such as functional knowledge, heuristics, etc.

In the IT-4 Feasibility Study (IT4, 1988), a basic concept was introduced in order to solve the problem of representing qualitative knowledge. The concept is based on knowledge based systems, an area within AI that focuses on the representation and utilization of qualitative knowledge. In contrast to the traditional approach of KBSs in process control, the concept is aiming at an integration of KBSs and conventional techniques. The integration is essential in order to avoid data redundancy and create consistent man-machine interfaces.

The concept implies a new generation of process control systems and influences several parts of their design, e.g., what hardware and software units the system consists of, the network architecture and communication protocols, the control system language, the user interfaces, etc.

The kernel of the concept of Knowledge Based Control Systems (KBCSs) is a knowledge base containing an object-oriented, multi-perspective model of the process components and the control system. The knowledge base is surrounded by tools. The tools build up the user interface for the different user groups and implement the different functions of the system.

The work in the first phase of the main project, which is described in this report, has basically been concentrated on verifying and broadening the original concept.

There has not been any reason to change the original concept. However, we have found the complexity of the proposed system to be much higher than expected.

The reason for this is mainly the complex structure and dependencies in an industrial plant. In general, a process can be decomposed into a set of systems that correspond to different flows in the process. The systems may be composed of subsystems which in turn are composed of interconnected components. The components are very often parts of several systems, which gives a very complex structure.

The object-oriented, model of the Feasibility Study has been deepened into an object-oriented, hierarchical, multi-view model. The basic views will be geographical, topological and functional. It should be possible to add other, more application dependent, views to the model. The multi-view approach will also be valid for the object structure, where multi-view objects are composed from single-view objects. The structure could also be described as multi-view objects consisting of several views, each with unique attributes, appearance etc.

Methods for representing knowledge of different kind such as sequences, rules, and functional knowledge are essential. Several methods are discussed and proposed.

The contents of the knowledge base is described in terms of a language. The language combines conventional programming techniques such as procedures and equations, with knowledge-based techniques, e.g, objects and rules.

The model-tool concept has been retained and extended in order to secure real-time performance and to avoid redundancy problems. The tools will extract knowledge from the knowledge base and generate executable code. The code could perform the basic continuous and sequential control, rule-based monitoring, diagnosis, simulation, troubleshooting, etc. All interaction with the knowledge base will be through tools. The tools will have different user interfaces depending on the user's demands.

Two prototypes have been developed. A hypermedia prototype models an operator interface to a KBCS and a G2 prototype concentrates on the structure of the knowledge base. The prototypes have been extremely important for visualizing various aspects of the concept.

We have found the concept to be well suited for solving many of the problems in today's process control systems. Since the concept has proved to be more complex than originally expected, more specification and prototyping will have to be done. With this in mind we do not believe that it will be possible to introduce a commercial product that fully implements the concept for another 10 years. However, there will be products that include ideas from the concept long before that.

A

Steritherm

A.1 INTRODUCTION

This appendix contains a description of the Steritherm process which is used as a demonstrator in the project. The reason for having a demonstrator process is to try the system concept on a "real" process which is fairly representative for a large class of industrial processes.

Section A.2 describes the process, its operation, and the current automation level. Even though the Steritherm process mainly is used as one example out of the large class of industrial processes for which the knowledge-base control concept is of interest, the process has a set of specific problems that might be solved with knowledge-based techniques. These are discussed in Section A.3. Finally, the selection criteria for chosing the Steritherm process as a demonstrator are described in Section A.4. Finally, Section A.5 contains the process schematic which the two prototypes are based on, and the sequence activation chart.

A.2 PROCESS DESCRIPTION

Steritherm is Alfa-Laval's full-scale process for indirect UHT (Ultra High Temperature) sterilization of liquid food products.

A UHT product is a liquid that has been subjected to a continuous flow heating process at a high temperature for a short time, normally 135-140 degrees C for

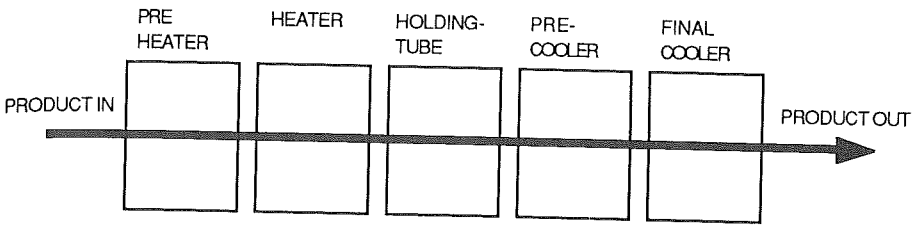


Figure A.1 Block diagram of a UHT process with indirect heating

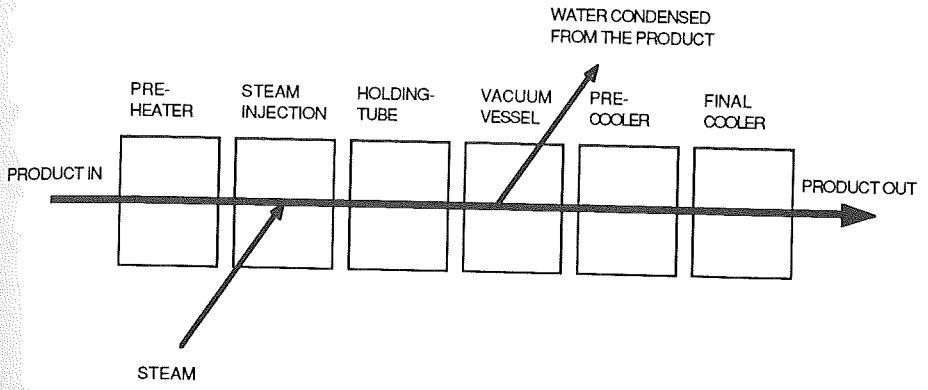


Figure A.2 Block diagram of a UHT process with direct heating

a few seconds. The purpose of the process is to kill all micro-organisms in the product. If the sterilized product is packed under aseptic conditions, it can be stored at room temperature for months or longer. The most common product is milk but it is also possible to process cream, coffee, dressings, sauces, etc.

UHT processing may be done with direct heating or indirect heating. In direct heating, steam is injected into the product and condensed water is removed in a vacuum vessel after cooling. In the case of indirect heating the product is heated in a heat exchanger. Indirect heating is by far the most common method due to legal restrictions in many countries against adding components to, e.g., milk.

Figs. A.1 and A.2 show the basic block diagrams for indirect and direct heating. For some products the UHT process is complemented with a homogenizer.

A normal capacity for a full scale UHT-plant is somewhere between 1,000 and 30,000 liters per hour. Such volumes are obviously too large for product development. Sterilab is a laboratory scale UHT process intended for product development. It has a capacity of about 100 liters per hour and can be configured both for indirect heating, i.e., as a Steritherm process, and for direct heating.

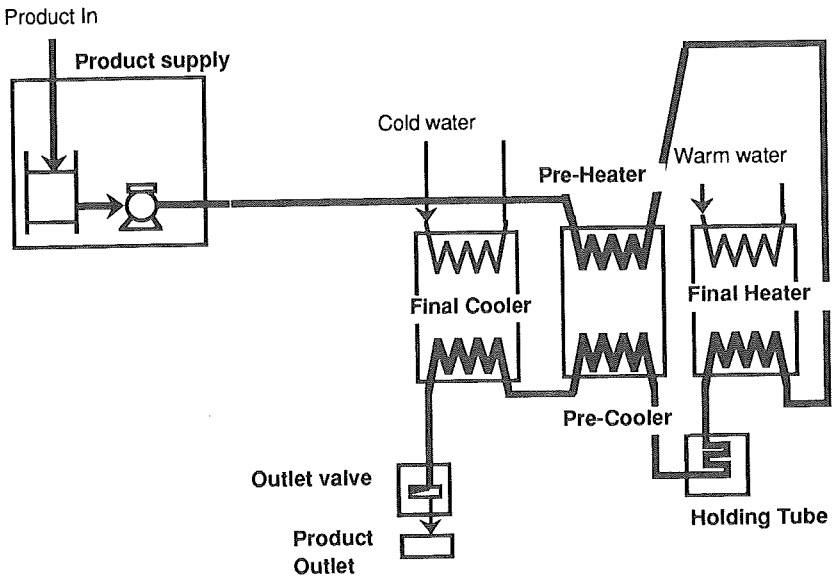


Figure A.3 Steritherm block diagram

A.2.1 Steritherm configuration

Fig. A.3 shows a block diagram for a Steritherm process without homogenizer. The process equipment is:

- Product supply.
  Balance tank and feed pump.
- Pre-heater.
  Incoming cold product is heated with warm product from approximately 5°C to 75°C in a heat exchanger.
- Final heater.
  The product is heated with hot water from 75°C to 137°C in a heat exchanger.
- Holding tube.
  The product is held at sterilization temperature, 137°C, for approximately 4 seconds.
- Pre-cooler.
  The product is cooled with cold product in a heat exchanger to 75°C. This is the same heat exchanger that is used for pre-heating.
- Final-cooler.
  The product is further cooled with water or ice-water in a heat exchanger down to filling temperature, 20°C or 5°C depending on the product.

- Outlet valve.
  In the case of a disturbance in the plant, e.g., too low temperature, this valve is activated to protect the filling machines from an infected product.

- Product outlet.
  To filling machines or an aseptic storage tank.

In addition to this, the process contains several valves, and additional pumps. The hot water system consists of a balance tank, a pump, several valves, and a steam injector through which steam is added to the hot water in order to increase the water temperature. A complete flow diagram of the Steritherm process used in the prototypes is shown in Section A.5.

A.2.2 Process operation

A normal production cycle in a Steritherm process can be split up in the following steps:

- Plant sterilization

  The plant is sterilized in order to kill all micro-organisms that may be present in the process equipment. The sterilization is done by circulating water at a temperature of approximately 140°C in the plant for a certain period of time, typically 0.5 – 1 hour. When the sterilization is completed, the plant runs with circulating water waiting for the production to start.

- Production

  When production starts, a valve to a product tank is opened at the same time as the circulating water is led to the drain. The product pushes out the water and, after a certain time, when the product has reached the outlet valve the production starts. Production normally continues for, at most, 8–15 hours depending on the product and the rate of disturbances. After that production period, the efficiency of the heat exchangers has decreased due to burn-on of the product, and a cleaning is required.

- Intermediate cleaning

  Intermediate cleaning is a way of extending the production period. Intermediate cleaning is done under sterile conditions and does not require a re-sterilization of the plant. Intermediate cleaning is not as efficient as a normal cleaning, which implies that it can only be done a limited number of times before a ordinary cleaning is performed.

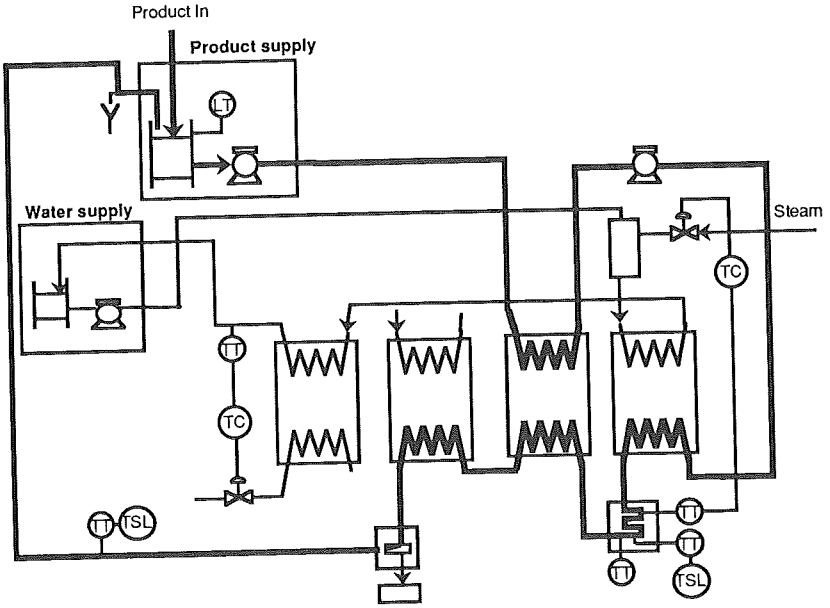


Figure A.4 Steritherm flow diagram with control objects

- Cleaning

  After the production period described above, possibly including some intermediate cleanings, there is a need for a proper cleaning. This cleaning is done with all equipment in place (Cleaning In Place, CIP). The cleaning program depends on the product and could vary quite a lot. Except for temperature and flow, the cleaning program can vary in time and with respect to what types of detergents that are used. After a CIP the plant must be re-sterilized.

A.2.3 Control and supervision

The basic control objective of a Steritherm process is to control the sterilization temperature. This can be done in many ways with different levels of automation.

A normally equipped Steritherm has an automation system that includes a programmable control system to control the sequence logic and carry out the alarm monitoring. The operator interaction is basically to start sequences like sterilization, production etc, and to supervise alarms. The operator communicates with the control system via LEDs, pushbuttons, and keyboard.

Control objects

Steritherm includes the following control objects shown in Fig. A.4:

- 2 Temperature controllers.
  Used to control the product heating temperatures and the recirculated warm water temperature.
- 5 Temperature transmitters.
- 2 Temperature guards.
  Covering production temperature and sterilization temperature.
- 1 Timer.
  For measuring the sterilization time of the plant.
- 5 Pumps.
  Two of the pumps are used in the product line, one is used in the warm water system, and the remaining two are used for the cleaning liquids.
- 15–20 Automatic valves.
  Includes regulator valves, constant pressure and constant flow valves, and on–off valves.
- 2 Level transmitters.
  To indicate high and low level in the balance tank.
- Differential pressure transmitters (optional).
  To measure the pressure drop in a heat exchanger affected by burn-on.
- Flow meters (usually mechanical).

A.3 PROBLEM AREAS

The Steritherm process is not very complex compared to other types of industrial processes. The problems that do exist are not especially complicated. Production with the Steritherm often runs quite smoothly. However, there are some problems which may be more important than first expected. This is indicated by the reject flow in the production of the order of 3–7% and by the fact that the non-aseptic alternatives to Steritherm products still seem to have a better reputation among customers.

The quality aspects are good starting points for an analysis of the problems of, and the weaknesses in, the process concept. However, first the goals of the process will be specified.

The purpose of the Steritherm process is to produce sterilized liquid food products from of unsterile products. In the Steritherm process the bacterials in the raw product are killed by the means of heat. However, it is not possible to heat up

only the bacterials – the bulk of the material is also heated and affected by the heat.

The main problems of the process have to do with the key question: "How to kill the bacterials in the raw product without affecting product properties like taste, consistency, stability, structure, and production volume in a negative way?"

In our case, we want to use information in a more efficient way in order to solve the Steritherm problems. Different interest groups like the production management, the operators, the service personnel, and, indirectly, the customer need different types of information. The information needed is not only dependent on the user group, but also on the current situation.

The KBCS concept may give us a means to solve some of the problems found in the Steritherm process. We have found the following examples of problem areas, which will be analyzed in further detail:

- Representation of design knowledge
- Alarm analysis
- Quality control
- Production optimization
- Raw material
- Auxiliary treatment

A.3.1 Representation of design knowledge

The present design of the Steritherm process is a result of more than 20 years of continuous improvement of a basic process concept. For most people involved in developing and using the system it is important to be able to trace and add knowledge of reasons and criterias behind the basic design as well as behind the different improvements done on the way during these years. By easily accessible explanations or references to where such knowledge could be found, the users should be able to adapt and optimize the process in a much better way to their own specific demands and conditions. Mistakes could also be avoided when modifying and adjusting the design of the process. The problem today is that the relevant knowledge is hard to elicit, represent, merge, and unify, and, furthermore, difficult to make available in a convenient way. Information that it should be possible to access or reference by the control system is mainly of the following types:

- Documented information in instruction manuals, process diagrams, design criterias, field reports etc.

- Un-documented design knowledge and experience about Steritherm, now in the brains of several "experts" such as designers, service and maintenance personnel, operators, aseptic product developers, etc.
- Important textbook knowledge of, e.g., aseptics, raw materials, process theory, control theory, etc.

A.3.2 Alarm analysis

A correct alarm analysis can, under certain circumstances, be a problem for the operator. This is typically the case for unskilled operators without deeper understanding of the process. The major sources of alarms are the temperature guards. These alarms can be caused by several differents faults such as problems with the flow pump, leaking pipes, problems with the heating water supply, broken temperature sensors, etc.

A standard Steritherm process is equipped with two temperature guards, a level guard in the filling tank, and motor fault logics. It is not unusual that customers add more guards. One such example is the Steritherm process at Pågen in Malmö which has also had an alarm analysis system installed. The system was based on additional measurements of the product pressure before and after heating, steam pressure, product pressure at homogenization, hot water temperature, and water flow. The reason it was removed was that it gave erroneous alarms when the Steritherm process was used for product development.

A.3.3 Quality control

The main problem with the Steritherm process is to keep the quality of the product high. To maintain a high quality the Steritherm has to be cleaned and sterilized according to complex procedures.

How to kill bacteria

The main quality factor is to produce a sterile end product. The decay of the number of bacteria in the sterilization process is probably close to a theoretical first order decay reaction with the constant of speed and the half-time factor governed by the Arrhenius temperature law.

This means that the number of live bacteria in the raw product decreases logarithmically with time and that the mean bacterium survival time decreases in an exponential way with higher sterilization temperature.

This is, however, only an ideal way of looking at the sterilization problem. The transportation mechanism of heat to the particular bacteria is also important. A particular bacterium may be well insulated and thus survive the fast sterilization sequence.

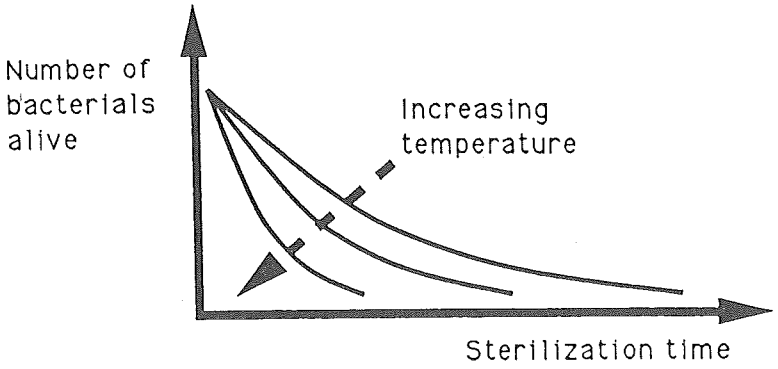


Figure A.5 Logarithmic decay of number of alive bacterials

The stirring of the product is important, as heat transport by conduction is a slow process. The natural flow turbulence in the Steritherm takes care of the stirring, unless the viscosity of the product is very high.

The product breakdown

It is not only the bacteria that decay in the Steritherm process. The product itself decays according to similar theoretical models. The sensitivity for quality breakdown is very dependent on the type of product, its material content, and its internal physical structure.

Milk products are sensitive because the calcium content will be saturated and less solvable at higher temperatures. The proteins tend to coagulate. Burn-on coatings will be the result, especially on heated surfaces of the process equipment. These coatings may be the source of changes in the taste of the product. Other product properties like the fat droplet sizes and the long term physical stability may also be affected.

Sterilization and quality

The sterilization process is governed by the least affected bacteria and the product quality is governed by the highest temperature points. These factors have to be well balanced in the process in order to obtain good products. A lot of the knowledge about product quality comes from the designer, but the experience and the actual values have to come from the people involved in the production.

Control parameters at the plant site: The following control parameters of interest are changeable to a greater or lesser extent at the plant site:

- Production flow rate.
- The preheating temperature.
- The final heating temperature.
- The volume of the holding tube.

Sterilization time: The sterilization holding time depends on the flow rate and the volume of the holding tube. The back pressure or the speed of the pump could be used to control the flow rate, while the tube volume has to be changed by changing the piping of the Steritherm.

The back pressure also controls the boiling temperature in the holding tube and prevents infection in the aseptic part of the process.

Sterilization temperature: The final heating temperature controls the sterilization temperature in the holding tube. Another important factor to control is the undesirable burn on. One way of minimizing this is to have the right time dependent balance between the pre- and the final heating. Large local temperature gradients in the contact area between the product and the heating surface should be avoided in the high temperature zones. Another possibility is to optimize the time sequences of production and cleaning.

Measurements: Direct, on-line measurements of quality factors are not possible today. All quality factors have to be estimated out of indirect measurements and historical experiences.

For instance, the taste depends on the burn-on level, which can be estimated in two ways. Firstly, the heat-insulating properties of the coating can be used and secondly, the burn-on material blocks the flow in the heat exchanger and acts as a measure of the fouling.

A.3.4 Production optimization

Optimization of production is an important issue. The production is determined by the product flow which is normally constant. The flow rate depends on the size of the heat exchangers and and the volume of the holding tube. The flow is determined by the flow pump and by pre-sized orifices that are inserted into the flow pipes. Different flows are used for production and cleaning. This is achieved by bypassing the orifices. Due to the different flows, the parameter settings of the temperature controllers are usually not correct in the cleaning phase. The tuning of the temperature controllers is also important. They must be tuned to ensure that the product temperature is also above the sterilization temperature level during transients caused by switches in the product recirculation.

In some cases the process is run with a variable product flow in order to adjust the production to the capacity of the following filling machines. To maintain product quality the heating temperatures should then also be varied. This is, however, seldom done.

The washing and cleaning of the process due to fouling and burn-on also constraints the process availability. The cleaning takes place both between product changes and during normal production. For products that are highly disposed to burn-on the cleaning must be performed as often as two or three times a day.

The decision of when to interrupt production and clean the process is taken by the operator. The standard procedure is to have pre-scheduled cleaning interrupts where the schedule is product dependent. The schedule is based on experience of how fast burn-on occurs for different products. The operator can, however, decide to override the cleaning intervals if, e.g., there is only a short time left to a normal process stop.

A way to estimate the burn-on is to measure the pressure difference over the heat exchangers. Burn-on is detected as an increase in the pressure difference. These differential pressure sensors are, however, not part of the standard Steritherm process.

With more advanced methods for burn-on detection and with product and production dependent cleaning decision thresholds there is potential for increasing the intervals between cleaning and thus increasing process availability.

A related area, where large amounts of heuristics are involved, is the actual cleaning. The sequence and concentration of the different cleaning detergents, and the cleaning flow and temperature are highly product dependent. The same is true for the overall cleaning scheme: whether to make many intermediary cleanings or a few complete cleanings.

A.3.5 Raw material knowledge

The properties of the raw materials used in the Steritherm process are crucial for its efficiency and for the quality of the products produced in it. Milk-based products dominate ranging from plain milk to all types of flavoured and fermented products, but products based on vegetables and fruits are also used. As the raw materials are usually biological, the quality can vary due to where and how they have been produced and treated.

Some examples of important parameters for the raw materials are viscosity, acidity, air content, disposition to burn-on, fat content and if it is vegetable or animal fat, sweetened or salted, with or without pulp or fibre, heat stability and sensitivity, heat resistance of spores, and whether it has been pre-treated and thus already depleted of some of the bacterials and enzymes.

Knowledge of these parameters and their influence on the process efficiency and the final product is, the same as for design knowledge, scattered about in different process documents, in the heads of experts, in textbooks, etc. Possibilities to access this knowledge in a unified way and through the control system should be an important support for the designers and the different users in their efforts to further develop the process, to adapt it to specific conditions, and to run it more efficiently.

A.3.6 Auxiliary treatment

Incoming product

The Steritherm process works on final products, e.g, after mixing in the case of sauces, brewing in the case of coffee, or after standardization in the case of cream. This indicates that the quality of the incoming product could vary quite a lot depending on how long it has been stored, the air content, etc.

Outgoing product

After the product has been processed it is normally forwarded to one or several filling machines for packing. Due to the construction of the filling machines they often require a somewhat higher inflow of product than they can pack. The normal solution to this problem is to have a higher flow rate in the Steritherm than the filling machine can handle. The "overproduction" is circulated via the balance tank back into the Steritherm.

Excess volumes of recirculated product may cause problems with a burned taste in the product. This may occur if, e.g., one filling machine suddenly stops. The ideal solution to this problem is to use an aseptic buffer tank between the UHT-process and the filling machines. This solution is often rejected for economical reasons. Another solution is to have variable flow in the UHT-plant and adjust the flow to the capacity of the filling machines. This solution gives problems with controlling the temperature when the product flow is not steady.

Knowledge about auxiliary processing, before and after the Steritherm, is needed in order to optimize production and product quality. It is therefore essential that this knowledge is made available in the operational system.

A.4 SELECTION CRITERIA

A.4.1 Generality

One interesting question concerning the Steritherm process is: "How general is the process and is it possible to extend research results from the Steritherm to industrial processes in general?"

It is impossible to give an answer to the second part of the question, but if the Steritherm is "similar" to other processes we have reason to believe that the results are possible to generalize.

The generality of the Steritherm process will be judged according to the following points:

- Products.
- Production organization.
- The type of process.
- The identified problems.

Products

The Steritherm products are food products, which are sold to the consumer without any further processing. Compared to many non-consumer products, the delivery is fast. The number of different products produced by a particular Steritherm process is about average for a process oriented industry.

As the product is delivered to the end user, there is limited relevance to products that are an integrated part of other products. The very high quality standard aspects which originate from advanced non-consumer products are not present.

Production organization

At Pågen in Malmö we found that the product development was integrated with the production organization. However, this is not true in general. Arla (a Swedish dairy corporation), e.g., has a separate development organization.

The production planning seems to be quite simple and the role of the process engineer seems to be well integrated and non-distinct in the organization.

The process operator has a very important position as the main supervisor of the process. The status and the advancement level of the service personnel varies from plant to plant. In general, they have less process knowledge than the operators. Instead, they have more detailed knowledge about the individual process components.

The type of process

The Steritherm process has as input one single raw product and as output only one product. The process may be judged as continuous with one main process function, which is heat treatment. The frequency of the product changes varies from installation to installation.

The sequence control of the Steritherm is advanced. There are sequences for start-up, close-down, and for change of the sub-function of the process. The continuous control problems seem to be quite easy. Today the control is restricted to the base level – pressure and temperatures – and there are severe problems with measuring the quality factors.

There are many types of process problems which are not present in Steritherm. Missing from the process flow point of view are batch, assembly, positioning, separation, major reaction, and storage problems.

The identified problems

Several of the previously identified problems with the Steritherm process are quite general. Representation of design knowledge is one example. Representation of knowledge about raw materials and auxiliary treatment are others. The alarm analysis problem is perhaps the most well-known motivation for knowledge-based systems in process industry. The complexity of this problem is, however, in our case, relatively small. The long-term monitoring of burn-on has strong similarities with condition monitoring problems in other industries.

Summary

The Steritherm process is quite simple from a continuous control point of view. It has advanced sequence control. The main problem is to measure the quality factors.

There are many general process aspects missing, but still, all aspects present in the Steritherm process are relevant from a general point of view. The different kinds of physical process problems missing would probably not have influenced the major structure of an integrated process model.

The subset of problems in the Steritherm process is big enough to serve as a generator for ideas and for demonstrators. To use a process with a greater complexity would probably just have led to confusion in the project. It has not been possible to work with the whole Steritherm in detail – certain aspects have been selected.

A.4.2 Sterilab accessability

The Sterilab process is available for experiments during most of the year at the Department of Food Technology, Lund Institute of Technology. There are also two Sterilabs at Alfa-Laval Food Engineering AB which probably can be used if planned well in advance.

A.4.3 Access to experts and documentation

Expertise on process technology, the process design, and most of the included main components (e.g., the heat exchangers) is available at Alfa-Laval Food Engineering AB. They also have people with a lot of experience of installation, service, and maintenance of the process. Key people are Bengt Palm, the designer of the process, and Lennart Alkskog, head of process development.

Among the customers we have contacted Pågen Produkter AB. Lennart Persson, site manager, is willing to share their experience of using the Steritherm process.

Other customers of interest in Sweden are Arla in Alingsås and Ekströms in Örebro who have modern and fully automated Steritherms.

A.4.4 Access to full scale processes

It is probably unrealistic to think that we will be allowed to make any extensive experiments on full scale processes at customers.

A.5 DOCUMENTATION

The process schematic that has been the base for the prototypes is shown in Fig. A.6. The sequence activation chart used in the G2 prototype is shown in Fig. A.7.

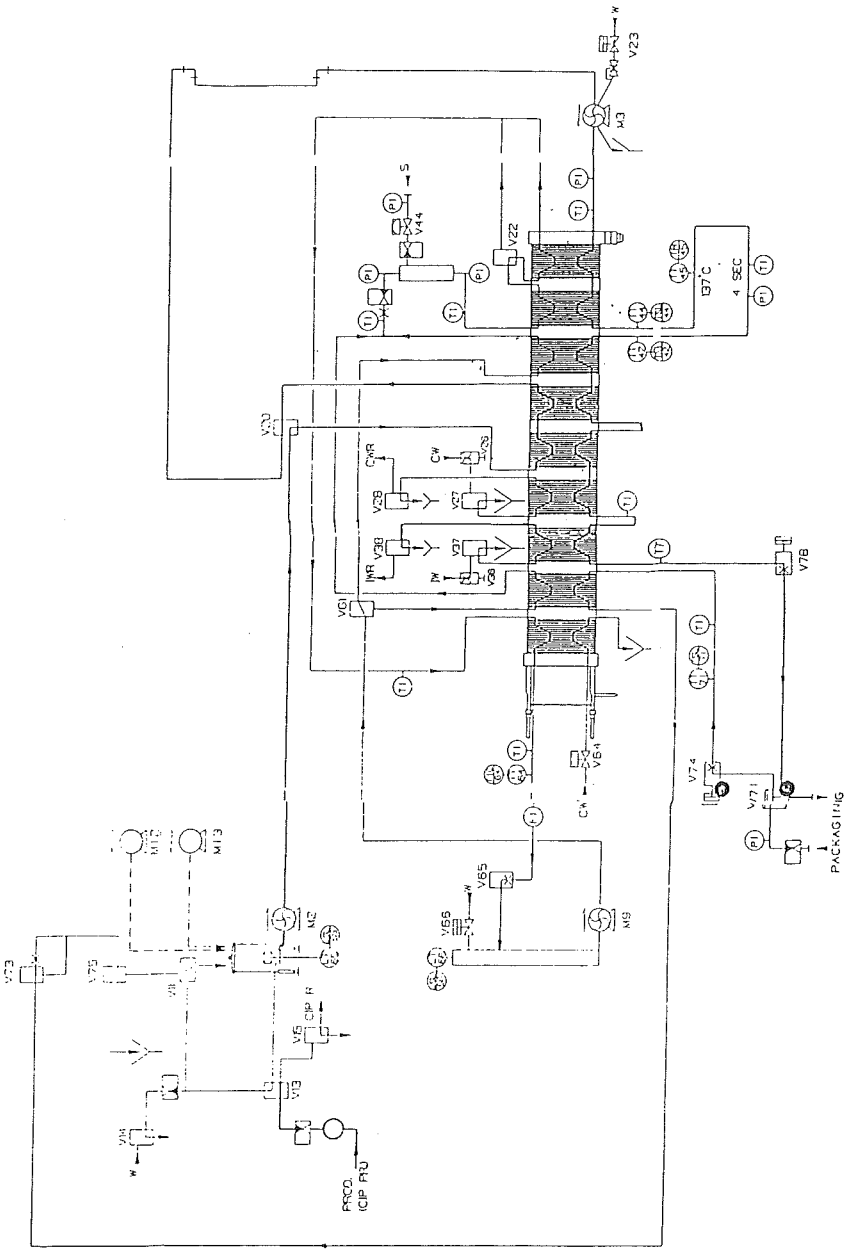


Figure A.6 Steritherm process schematic

PAGE 1 (2)

STEPNAME	STEP	COND	M2	M3	M9	M11	M12	M13	V11	V13	V14	V20	V27	V44	V61	V64	V66	V71	V73	V74	V75	V78
	STERILE WATER																					
HEATING	112	TS71	X	X	X					X		A		X	A	X		X	X		X	
STERILIZ	112	T 1	X	X	X					X		A		X	A	X		X	X		X	
COOLING 1	113	T 2	X	X	X					X		B		X	B	X		X	X			
COOLING 2	114	T 3	X	X	X					X				X	B	X		X	X			
STERILE W	115		X	X	X	X				X				X		X		X		X	X	X
	PRODUCTION																					
BTD EMPT	117	LS65	X	X	X	X								X		X		X		X		X
PROD IN	118	T 4/5	X	X	X	X			X					X		X		X		X		X
PRODUCTION	119		X	X	X	X			X					X		X		X		X	X	X
BTD EMPT	120	LS65	X	X	X	X								X		X		X		X		X
PROD OUT	121	T 6/7	X	X	X	X					X			X		X		X		X		X
RINSING	122	T 8	X	X	X	X					X			X		X		X		X		X
STERILE W	115		X	X	X	X					X			X		X		X		X	X	X
	ASEPTIC INTERMEDIATE CLEANING																					
RINSING	132		X	X	X	X					X			X		X		X				X
CAUSTIC DOS	133	T11/12	X	X	X	X	X				X			X		X		X			X	X
CAUSTIC CIRC	134	T 13	X	X	X	X								X		X		X			X	X
BTD EMPT	135	LS65	X	X	X	X								X		X		X				X
RINSING	136	T 14	X	X	X	X					X			X		X		X				X
	STOP																					
BTD EMPT	142	LS65	X	X	X											X		X				X
RINSING	143	T 21	X	X	X						X					X		X				

STEPNAME	STEP	COND	M2	M3	M9	M11	M12	M13	V11	V13	V14	V20	V27	V44	V61	V64	V66	V71	V73	V74	V75	V78

PAGE 2 (2)

STEPNAME	STEP	COND	M2	M3	M9	M11	M12	M13	V11	V13	V14	V20	V27	V44	V61	V64	V66	V71	V73	V74	V75	V78
	CLEANING																					
RINSING	152		X	X	X	X					X			X		X		X				X
CAUSTIC DOS	153	T31/50	X	X	X	X	X				X			X		X		X				X
CAUSTIC CIRC	154	T 33	X	X	X	X								X		X		X			X	X
BTD EMPT	155	LS65	X	X	X	X								X		X		X				X
LEVEL ADJ	156	T 34	X	X	X	X					X			X		X		X				X
CIRC V13	157	T 35	X	X	X	X			X			B		X		X		X			X	X
BTD EMPT	181	LS65	X	X	X	X								X		X		X				X
RINSING	158	T 36	X	X	X	X					X			X	B			X				
ACID DOS	169	T45/51	X	X	X			X			X			X	B			X				
ACID CIRC.	170	T 46	X	X	X						X			X	B			X			X	
BTD EMPT	171	LS65	X	X	X									X	B			X				

RINSING	168	T 44	X	X	X						X			X	B			X				

STEPNAME	STEP	COND	M2	M3	M9	M11	M12	M13	V11	V13	V14	V20	V27	V44	V61	V64	V66	V71	V73	V74	V75	V78

Figure A.7 Sequence activation chart

B

Travel Notes

B.1 PALO ALTO, 22/4 – 1/5 1989

Between April 22 and May 1, Karl-Erik Årzén visited Palo Alto in California. The main purpose of the visit was to participate in a G2 user group meeting, April 24 – 25. In connection with this, Intellicorp, Neuron Data, Stanford University, Santa Clara University, and FMC Corporation were visited.

B.1.1 G2 users meeting

The user group meeting took place at the Holiday Inn in Palo Alto. Monday morning was devoted to presentations of the current status of Gensym, their plans, and new products. Monday afternoon and Tuesday morning contained user presentations. The meeting ended with demonstrations of some of the new features of G2 version 2.0 intended to be released in September 1989.

The meeting gathered around 80 participants including 20 persons from Gensym. The majority represented American industrial users. Peter Pavek from Uppsala University was the other Swedish participator. His travel was sponsored by FMV. The only other European participator came from Siemens. Japan was represented by Mr. M. Yokoyama from C.Itoh Techno-Science.

A Sun, a HP, and a Vaxstation were available in the meeting room for G2 demonstrations.

Gensym presentations

Gensym's presentations began with Lowell Hawkinson and Robert Moore. They gave a very positive picture of Gensym. The company has grown from 17 to 32 employees over the last year. The yearly revenue is 4 MUSD. They have currently 80 customers world-wide and 150 G2 licenses. Of these, 40 include licenses for the on-line interface GSI.

The major current event for Gensym is the announcement of G2 version 2.0. With this version Gensym aims to bring out G2 in the operator rooms. This intention shows up in the improved possibilities to build up end-user interfaces within G2, including the use of colour. The user requirements, specially from Du Pont, on integrating G2 with external simulators, have led to the development of GSPAN that allows G2 to call external simulators through the G2 simulator. Du Pont uses GSPAN as an interface between G2 and their own simulator written in C++.

The development of off-the-shelf interfaces to major control systems continues. The initiative is shared between Gensym, consultant companies, and control system suppliers. Fisher Controls has developed an interface to Fisher PROVOX, JGC Engineering has developed an interface to Yokogawa CENTUM, and Interatom is developing an interface to Siemens S5. Gensym already has an interface to HP 48000 and is in the process of developing interfaces to Honeywell TDC 3000, Allen Bradley PLC, HP RTAP, Modicon/AEG PLC, and to the relational database Oracle.

G2 is currently available on Symbolics, Texas Explorer and micro-Explorer, Sun 3 and 4, HP 9000/3XX and HP 9000/8XX, Vaxstation 2000 and 3000, Vax series 6000 and 8000, Macintosh II, and Compaq 386. Versions for Apollo DN 3500 and Decstation 3100 are being developed. They have also considered IBM PS/2. The usual memory requirement is 16 MByte.

Gensym have three new offices; in Texas, California, and Illinois. SIRA in the UK, ORSI Automazione in Italy, Cognitech in France, and C. Itoh Techno-Science in Japan sell G2, provide user support, and give G2 courses. Gensym also has value-added resale contracts with ESIA in France, Interatom in Germany, JGC Corporation and NTT Data Communications in Japan, and Scicon in USA. Companies that do G2 consulting include AI Systems in Belgium, Insiders in Germany, Computas in Norway, and Coopers & Lybrand, Badger Engineers, Management Analysis Company, Combustion Engineering, and Kaman Sciences Corporation, all in the USA.

User presentations

Sarat Chandra from FMC Corporation presented a project where G2 was used for diagnosis of an armoured tank. The ultimate goal for the project was to

embed G2 inside the tank. Due to the nature of the computers that G2 runs on, this is currently not possible. The presentation was ended by a video film.

Albert Brown from Lockheed described a G2 application for monitoring and diagnosis of a propellant manufacturing process. At this stage they were still running against a simulated process. However, they had money granted to eventually install it in a real plant.

Clint Whitaker from Scicon gave a general talk about Scicon's reasons for using G2. He described one project for optimization of a fluid catalytic cracking unit. Scicon has also developed a G2 demonstration of a car painting process and is currently looking upon using G2 in the manufacturing industry.

Karl-Erik Årzén gave a talk were he presented his experiences of using G2 for modelling and simulation. The main focus of the talk was on the need for hierarchical objects and why this is difficult to implement in G2. The talk was very well received.

Karlene Kosanovich from Du Pont described GSPAN, the interface between G2's simulator and external simulators that Gensym has developed in cooperation with Du Pont. Gensym will also market G2 with GSPAN as a front-end to commercial simulator packages like, e.g., ACSL.

Lee Thompson from Fisher Controls gave an overview of their interface between G2 and Fisher PROVOX. The name of the interface is CGI (CHIP G2 Interface) and connects GSI to the CHIP real-time database in PROVOX.

Makoto Yokoyama from C. Itoh in Japan presented an icon editor that they have developed. The editor operates on G2 knowledge-base files and modifies their icon descriptions. The editor was only available on Sun. G2 has been selected by MITI as the official tool in a nuclear safety research programme. In connection with this a spectacular G2 demonstration has been developed including presentation on a 70" video back-projection system and voice output using DECTalk.

Finally, Karlene Kosanovich discussed the possibilities to develop realistic end-user interfaces in G2. She was very critical. Du Pont's original specification of the interface was described as a "Cadillac" and the interface they ended up with as a "Yugo". Du Pont is Gensym's largest single customer and is currently installing one G2 system each month in operation. From what was shown of the new features in version 2.0, it is evident that Du Pont has strong influences on Gensym's development.

Version 2.0

The meeting ended with a presentation of the new features in version 2.0 and four demonstrations. The new features are: transient object, i.e., the possibility to create new objects and connections dynamically from rules; sets and lists to represent variables with more than one value; connections between workspaces;

improved presentation possibilities including new and better graphs and alarm messages; polychrome icons; user-definable menus; scroll windows; a generalization of the object concept to also include other G2 items such as operator controls, alarm messages, workspaces, etc.; an integrated icon editor (possibly in ver. 2.1); separate compiled run-time versions of G2 (possible in ver. 2.1); Telewindows; and G2 networks.

Procedures have been expected and talked about for long and will now finally be available. The earlier discussions had indicated that the procedural language would be a graphical language of Grafcet type. This is, however, not the case. The procedures resemble a conventional Pascal-ADA type of language. Procedures are activated with a `START` action and executes as parallel activities. Procedure arguments are typed and local variables are allowed. Procedures can be procedure arguments and recursive procedure calls are allowed. Primitives that interrupt the procedure execution for a certain time or until a certain condition is true exist. Gensym have plans to use procedures to implement object methods. This will not be available in version 2.0.

Gensym demonstrated their new end user interface features. Polychrome icons contains multiple layers with different colors that can be changed dynamically. Combined with the possibility to move and rotate objects this gives quite powerful animation possibilities. What is still lacking is the possibility to dynamically change the size and shape of icons. The improved graph facility was shown.

Telewindows makes it possible for multiple users to simultaneously access a single G2 over a network. It provides the user with a viewport into G2. Telewindows runs on smaller workstations and requires less memory (4 MByte) than G2. G2 networking allows several G2 systems to be interconnected and exchange information.

Conclusions

The impression of the meeting is that Gensym goes very well at the moment. It is interesting to note that G2 is becoming a de facto standard for real-time expert system tools that most control system suppliers want to provide interfaces to.

G2 is also used in several interesting projects. One example is MITI's nuclear safety project. The French oil company ELF has chosen G2 for a project where they plan to totally automate their North Sea oil platforms during the pumping phase. The american venture capital project Space Biospheres uses G2 to monitor complete artificial ecological systems. In two years they plan to seal their first biosphere for a period of two years. The biosphere will contain humans, animals, different climates, etc.

B.1.2 Center for Integrated Systems

The G2 meeting included a visit to the Center for Integrated Systems at Stanford University. The Center is a research institute for semiconductor manufacturing research. The visit was organized by Don Gardner who also was the local organizer of the G2 meeting and uses G2 to monitor the semiconductor manufacturing process.

The visit consisted of a tour of the lab and presentations of the different research groups. One of the groups, headed by Jay M. Tenenbaum from Schlumberger, works with integrated systems for planning, administration, control, and monitoring using model-based approaches.

Don Gardner gave Karl-Erik Årzén the opportunity to present the IT-4 project at one of the Center's weekly meetings. The response was very positive. Tenenbaum was enthusiastic and interested in maintaining the contact.

Karl-Erik also had the opportunity to discuss with Jeff Y-C Pan and Jay Glicksman from Tenenbaum's group. Pan has developed PIES (Parametric Interpretation Expert System), a system for interpretation of parametric test data and model-based fault diagnosis of the semiconductor manufacturing process. They are currently including PIES in MKS (Manufacturing Knowledge System), a knowledge-based CIM system (Pan *et al*, 1989). Many of the ideas in the MKS project are similar to our's.

Tenenbaum's group was using the HyperClass tool from Schlumberger. HyperClass is an object-oriented expert system development environment similar to KEE or Knowledge Craft. It includes objects, rules, graphics, etc. It is written in Lisp and runs on Sun workstations under SPE (Symbolic Programming Environment). HyperClass was developed as an internal product within Schlumberger in the beginning of the eighties. On top of Hyperclass they were developing graphical editors and hypertext systems.

In parallel with Tenenbaum, a Norwegian guest researcher from SINTEF developed a similar system using HyperCard on a Mac II. HyperCard was used for rapid development of a user interface to a semiconductor manufacturing process. The system included pictures of the process layout, process components, production plans, etc., and was connected to a relational database containing on-line process data. This project also coincided very much with our project. The impression was that the small HyperCard project well matched the much larger HyperClass project.

B.1.3 Intellicorp

During Wednesday, Karl-Erik visited Intellicorp to see Plexsys. Plexsys is an add-on package to KEE, aimed for power system applications, developed by Intellicorp together with EPRI (Electric Power Research Institute) in Palo Alto. Plexsys

has been further described in Chapter 8. The Plexsys system was demonstrated in a not too convincing way. There are rumours that EPRI has cut down their work on KEE and Plexsys to instead concentrate on Nexpert Object. The overall impression of Intellicorp was not very good. The building was very empty and quiet with very little activity of any kind.

B.1.4 Neuron Data

On Thursday, Karl-Erik visited Neuron Data to see their HyperCard interface to Nexpert Object. Here the general impression was the opposite from that of Intellicorp. The activity was very high, telephones were ringing, etc.

The HyperCard interface was not ready. They gave a very primitive demonstration. The interface will probably be announced at the AAAI in August 1989. Neuron Data have interfaces available to Dataviews and Ease+ which both are graphical information presentation programs. Neuron Data also has its own end-user interface to Nexpert, called AIVision, with some hypertext features.

B.1.5 Other visits

On Wednesday Karl-Erik visited H. Chris Tseng and D. Siljak at University of Santa Clara where he gave a talk about the department's activities in knowledge-based systems.

On Friday a planned visit to Advanced Decision Technologies was cancelled. Instead he visited C.W. Chen and R. Walker at FMS Corporation in Santa Clara. They worked mainly with conventional control theory.

B.2 MARYLAND, 2/5 1989

In connection with a review meeting for the Systems Research Center at University of Maryland, Karl Johan Åström had the opportunity to visit the Human Computer Interaction Laboratory at University of Maryland. Professor Shneidermann gave a an overview of the laboratory and their activities. Hyperties, a new hypertext system was demonstrated (Marchionini and Shneidermann, 1988). The system is available on IBM PC and IBM PS/2 including compatibles and will later also be available for Sun workstations.

B.3 ESPRIT CONFERENCE, 27/11 – 1/12 1989

Anders Åberg and Börje Rosenberg attended the ESPRIT research conference in November – December 1989 in Brussels. The purpose was to check up on the ESPRIT programme.

The five day conference had three days of paper presentations, workshops, and panel discussions with about 12 parallel sessions. The fourth day was an ESPRIT policy day with EC R & D policy presentations and panel discussions. The fifth day was a free activity day, which was spent at the huge ESPRIT project exhibition. The exhibition was running in parallel to the rest of the conference during five days.

The general impressions from the conference were:

- ESPRIT means a lot for the process of gluing together research organizations and, especially, large companies in the EC.
- There are political forces which are really trying to design an R & D programme based on an analysis of the future needs of the European industry.
- ESPRIT creates an European spirit and a general sense of belonging together. "Inside and outside" of EC was a subject which was stressed at several occasions. Some lecturers wanted to build quite high walls in order to protect Europe. Especially, the US was pointed out as a dangerous technological competitor. We got a strange feeling that Japan was regarded as less dangerous and as a potential cooperation partner
- Some projects in ESPRIT I are not very goal-oriented and seem to have been put forward to get financing for each of the consortium members and not for the common projects. ESPRIT II is growing much more strict in this aspect.
- The administration of the projects is expensive. Through the cooperation of many partners and the multiplication effect, it is still regarded as beneficial to participate in ESPRIT. Some companies have as a business idea to start up and to administrate ESPRIT projects as the main contractor.
- In the application sectors, the strong standardization efforts are probably very important. The position in CIM (CAD, job shops) is quite impressive and EC may have a chance to get a world leadership, if they can succeed in marketing.
- The information processing systems sector (IPS) is quite influenced by the academic world. The IPS sessions had a high number of academic visitors and this also influenced the discussions. The industrial people were found mainly in the application sessions. There seems to be a gap between the industrial and the theoretical people. But there is a dialogue, which may bridge this gap.
- The quality of the sessions was very uneven. We believe that fewer sessions but higher quality would be a good goal for future conferences.

B.3.1 Interesting Projects

The interesting ESPRIT I projects are described in detail in the Feasibility Study, (IT4, 1988). Some ESPRIT II projects have been started and these will be described in more detail.

AITRAS (2167)

The AITRAS project is supposed to deal with real-time expert systems. Nothing else is reported.

ARCHON (2256)

"Architecture for Cooperating Heterogeneous On-line Systems" is a five year project with about 14 MECU (about 100 MSEK) in the budget and 14 partners. It aims to develop an architecture for cooperating expert systems in industrial applications. This will be done by defining and implementing a Knowledge Interchange Protocol (KIP), which will ensure re-usability of existing systems, versatility with respect to different techniques and paradigms and, finally, openness in the information interchange with conventional systems (databases and process control systems).

Demonstrators: Electrical power generation and transmission, Electrical distribution, CERN control, Cement kiln control, Robot arm control.

Prime contractor: Krupp Atlas Elektronik.
Others: CERN, Framentec, Queen Mary College, etc

KBSSHIP (2163)

"Shipboard Installation of Knowledge Based Systems" is a 3 1/2 year project, which continues the ESPRIT I project (1074). The objective is to optimize several aspects of ship operation. This is done by a number of integrated expert systems. The System Manager Expert System coordinates five sub-expert-systems. The "Expert Voyage Pilot" helps in the voyage planning. The "Expert Loading System" provides advice for ship loading and unloading. The daily maintenance planning is supported by the "Expert Maintenance System". The "Expert Diagnosis System" monitors and pin-points equipment failures. Finally, the "Statutory requirements and Classification Expert System" gives advice according to international safety legislation, to port entry requirements and to Classification Society Rules.

Prime Contractor: Danish Maritime Institute.
Others: Søeren T. Lyngsø, Krupp Atlas Elekronik, Lloyd's, etc.

ITSIE (2615)

"Intelligent Training Systems in Industrial Environments" is a project which looks at the possibilities of using AI techniques to develop sophisticated training systems for industrial personnel concerned with the operation and maintenance of complex industrial processes. In a generic adaptive architecture for training purposes, multiple representations of the physical system must be utilized. Abstract teaching based on qualitative reasoning comes first. Teaching details will probably utilize numerical modelling.

Prime Contractor: Consortium Marconi Simulation.
Others: Heriot-Watt University, AXION A/S, CISE SpA, etc.

FOCUS (2620)

Classified as a front-end to an existing system. Nothing else is reported.

KIWIS (2424)

The "Advanced Knowledge-Based Environment for Large Database Systems" project is developing a knowledge based management system which supports both a sophisticated stand-alone "personal knowledge machine" and an integration of information coming from a wide variety of sources. The KIWI system has a layered architecture with a object virtual machine at the bottom. The next layer is the heart of KIWI – the Basic Language machine, which combines logic and objects in an integrated way. An abstraction layer provides application oriented languages and on the top we find the user interface development system and the cooperation manager (for communication). The project originates from ESPRIT I (641, 1117).

Prime Contractor: Philips.
Others: CRAI, ENIDATA, SISU, etc.

STRETCH (2443)

The purpose with "Extensible KBMS for Large Knowledge-Base Applications" project is to provide a physical and conceptual object manager in the form of a knowledge-based management system. The system will support a rule based language and an object-oriented language in a multiuser environment. The demonstrator will be an intelligent training system.

The only contractor reference: Laboratoires de Marcoussis.

PROMISE (2397)

The main objective of the "Process Operator's Multi-media Intelligent Support Environment" project is to develop techniques for enhancing man-machine interfaces to knowledge-based systems in the real-time and the process control area.

The system will provide several interaction channels to the system such as graphics, sound, voice and animation, which make use of KBS techniques. Subjects such as dialogues in real-time systems, user modelling, and timing and synchronization aspects will also be covered. The "ESB" (ESPRIT I 96) results will be used as a base in the project. A nuclear plant oriented demonstrator and a chemical plant application are planned.

Prime Contractor: Tecsiel S.p.A.
Others: Algotech, Dow Chemicals, Scottish HCI Centre, South of Scotland Electricity Board, etc.

MMI2 (2474)

"A Multi-Modal Interface for Man-Machine Interaction with Knowledge-Based Systems". Nothing is reported.

ACKNOWLEDGE (2576)

The "Improving the Knowledge Acquisition Process" project will achieve its goal by constructing a Knowledge Engineering Workbench (KEW). The system will assist knowledge engineers in their tasks and partially automate the tasks. Learning mechanisms are planned to support automatic deduction and the knowledge is re-organized if new information is entered. Tools will harvest, refine, and reconstruct knowledge. One application mentioned is fault isolation and corrections in the telecommunication domain.

Prime Contractor: Cap Sesa Innovation.
Others: Marconi, Telefonica, Computas ES, Sintef, etc.

EQUATOR (2409)

The "Environment for Qualitative Temporal Reasoning" project aims at developing a toolkit and environment for industrial and commercial applications in which Time Dependent Reasoning (TDR) is required. The TDR system structure will be based on a system model consisting of the real world model, the computer world model and the operator world model. A General Representation Formalism (GRF) for events and temporal relations will be used in the environment. The GRF representation is translated into an executable common representation language. The EQUATOR environment will be evaluated on two demonstrators.

Prime Contractor: ERIA.
Others: CENA, CISE, Ferranti, Imperial College, etc.

MLT (2154)

The "Machine Learning Toolbox" objective is to build a toolbox of machine learning algorithms. These algorithms will use a common knowledge representation language to allow different algorithms to be used with the same data. The

toolbox will be evaluated on several applications. The MLT-Consultant will be an expert system helping the user to represent the knowledge.

Prime Contractor: Nixdorf.
Others: The Turing Institute, Intellisoft, INRIA, British Aerospace, etc.

IPCES (CIM 2428)

The "Intelligent Control by Means of Expert Systems" project has the objective of developing a set of modular building blocks, combining new and already available technologies, which can be tailored and assembled to perform a range of control systems functions in small and medium sized manufacturing enterprises. Expert systems for control, for diagnosis of product defects, and for future system behaviour are suggested. The project is a continuation project from ESPRIT I (1653).

Prime Contractor: Philips.
Others: ELTEC Elecktronik, RTC, MINIWATT, CNRS, Dornier, etc.

B.4 JAPAN, 2/12 – 12/12 1989

Karl-Erik Årzén from the Department of Automatic Control, Claes Rytoft from ABB, and Christer Gerding from SattControl visited Japan, December 2–12 1989. The goal of the visit was to study Japanese activities concerning real-time applications of expert systems. Hilding Elmqvist from SattControl and Arne Otteblad from The Swedish National Board for Technical Development (STU) accompanied the travel group. Visits were made to Yokogawa, Nippon Kokan Steel, Toshiba, LIFE, JAERI, Hitachi, Tokyo Institute of Technology, Japanese Gas Company, and Petroleum Energy Centre. A G2 users group meeting at C. Itoh Techno-Science Co. was also attended. The majority of the visits were organized by The Swedish Technical Attachée Office (STATT) in Tokyo.

B.4.1 Yokogawa

On Monday morning we visited Yokogawa where we were received by Mr. Yasuro Hirata, manager of the AI section of the application engineering section. Yokogawa has been active in the field of AI for four years.

XL/AI, an expert system tool for diagnosis and alarm reduction, has been developed by Yokogawa and seems to have been their main AI project during recent years. However, what they were working on now was a carefully avoided subject of discussion.

XL/AI is implemented in C and executes on a 68020 based UNIX workstation. It took 10 man years to develop. XL/AI is intended to be used together with

Yokogawas control system CENTUM. However, it is not integrated with the control system, but put on top on of the conventional system with a separate database and user interface. Concerning the user interface, the separation is deliberate since, according to Mr. Hirata, the conventional user interface has totally different demands on what information to present and how to present it.

XL/AI is purely rule based and mainly uses backward chaining. The rules are divided into rule classes of about 40 rules each. The reason for this is to make verification and consistency checks easier. Except from this, there is no support for verification. Verification is considered a problem that the customer and the developer of the expert system should solve by themselves according to Mr. Hirata. There is a graphical rule interface and it should be easy to add, delete, and modify rules, as it is mainly the customer, not Yokogawa, that is the user of XL/AI. The tool does not have real-time functionality as it cannot handle interrupts, i.e., changes in data or new data, while executing rules.

Yokogawa and Shin-Daikyowa Petrochemical Corporation Ltd have jointly developed a process diagnostic expert system for a cumene plant using XL/AI. The system is aimed at the reduction of operators' load by providing guidance for the causes of plant abnormalities. The system consists of about 1250 rules and 500 items and was installed in May 1989.

Before developing XL/AI, Yokogawa has worked with BRAINS, KEE, and similar systems. However, even if many of these systems had a high functionality they did not have the right functionality. That is the reason why XL/AI was developed.

Some fields that Yokogawa found interesting for the future were mentioned. These include model based diagnosis, combining XL/AI and fuzzy control, and planning and scheduling. Yokogawa already has a fuzzy control package that runs as a separate program on the same computer as XL/AI.

Although Yokogawa did not show much of their recent development, our impression was that the combination of a control system and an expert system tool from the same company is an interesting beginning of something that could come close to some of the ideas in our IT4 project.

B.4.2 Nippon Kokan Corporation

On Monday afternoon we visited NKK, a company active in steelmaking, shipbuilding, and engineering and construction. Our host was Mr. Masanori Itoh, manager of the Electronics Research Center. The different research areas of the centre are AI, fuzzy control, neural networks, CIM, CAD/CAM, robotics, micro computers, measuring instruments, and electronic devices.

NKK develops expert systems and tools for internal use as well as for customers. Expert systems have been used in NKK plants for five years and the main purposes are to reduce the number of operators and to transfer skill from old to new

operators. Two expert systems were presented: Expert System for Blast Furnace Operation Control and Refuse Incinerator Operation Guidance Expert System.

NKK has applied expert system techniques to the furnace condition diagnosis system, which is one of the most important systems of the operation control system for the blast furnace. The Expert System for Blast Furnace Operation Control consists of two systems. The first is the Abnormal Furnace Condition Prediction System, which predicts the occurrence of burden slip and channeling in the furnace. The second is the Furnace Heat Monitoring and Control System, which judges the in-furnace heat level and instructs the operators. The functions of the system are separated into two parts. One is the preprocessing part, which uses conventional techniques. The second is the inference part which is performed on an AI processor, which so far only is implemented as software, not as separate hardware. The knowledge structure is mainly production rules, and a blackboard architecture is used for the inference part. The knowledge base is divided into several knowledge units, which have a hierarchical internal structure. The purposes of this structure are: to avoid increased inference times when the number of rules increases, to make it easy to check the validity of the rules, and to mimic the reasoning structure of an expert. The system consists of 400 rules and was developed using ESHELL, a blackboard shell from Fujitsu.

The Refuse Incinerator Operation Guidance Expert System is a system that aids operation when the automatic combustion control system is difficult to continue. The knowledge is structured in hierarchical combinations of production rules and state transfer models. The system is divided into one preprocessing part that runs on a Fuji Electric L-300 and an inference part that runs on a Fuji Xerox 1121.

These described systems were developed 2 – 3 years ago. NKK's more recent development was not presented. However, the impression is that NKK is very active in applying expert system techniques to real problems. Furthermore, there seems to be a lot of interesting projects at the research centre, e.g., adaptive controllers (already now used in many applications), fuzzy controllers (partly used in the Blast Furnace Operation Control System), and neural networks (pattern matching for hand writing).

B.4.3 Toshiba

On Tuesday morning we visited Toshiba Fuchu Works were we met Dr. Kiyoshi Niki and some of his colleagues.

Toshiba develops expert systems for mainly three different areas: power transmission and distribution systems (planning, design, operation, diagnosis, maintenance, training, and simulation), power generation systems (mostly fault diagnosis), and paper production systems (production planning system). At Fuchu Works there are about 10 persons developing AI tools and about 150 persons

developing applications. The tools mostly used are TDES-3, IREX, and the language C. The computers are mostly engineering workstations. To increase the speed of the systems they are normally compiled into C even when developed in other environments. However, this makes the systems rather static.

Six systems were presented and demonstrated. Many of these system were delivered. They were mostly rule based fault diagnosis systems, often with simulation opportunities for training. The common way to verify the knowledge base is to test all possible combinations. Two of the systems will be presented here.

The Turbine-Generator On-line Monitoring and Diagnosis Expert System has been developed to detect vibration malfunctions and support the operators by recommending appropriate corrective actions. The system consists of a real-time diagnostic part and a detailed diagnostic part. The former can respond directly to the dynamic behavior of abnormal vibration and provide the operators with operation guidance in real time. The latter enables the operators to identify the cause of the malfunction through interaction with the computer. The real-time diagnosis was implemented with procedural programming in C, while the detailed diagnosis was implemented with frames, rules and backward reasoning in IREX, an integrated expert system building tool. The real-time diagnosis runs on a mini-computer and the detailed diagnosis runs on an engineering work station. The system has been in operation since 1986.

Another system was a production planning system for paper production. Based on three different knowledge-bases, one for product orders, one for product specific knowledge, and one for planning knowledge, the system presents a plan. The scheduling constraints are volume, sequence, delivery date, production interval, product combination, material, and energy. The plan is produced in three steps. First, a group product allocation is done, where products are put together with regard to volumes, sequences and delivery dates. Second, an individual product allocation is done according to production intervals and product combinations. Finally, adjustment is done with respect to energy and pulp constraints.

Our impression is that Toshiba are very active in the field of expert systems. Although many systems have been implemented, none of those we looked on was very impressive when it comes to advanced expert system techniques. The number of different fields of application, however, was impressive.

B.4.4 Life

On Tuesday afternoon we visited the Laboratory for International Fuzzy Engineering Research (LIFE), where we met Dr. Toshiro Terano, executive director of LIFE, and the directors of the three laboratories.

LIFE was founded in March 1989 by permission of the Minister for International Trade and Industry (MITI) according to the National Research and Development Program. The laboratory is associated with 48 companies for the purpose of R &

D in the application of fuzzy theory to engineering and also for the promotion of national and international exchange on the study of fuzzy theory. The laboratory has a budget of 6 MUSD over six years and today consists of 22 researchers taken from the different sponsoring companies.

During the first 6 months the three laboratories have determined what to concentrate their efforts on. These programs were presented to us.

The first research laboratory – Fuzzy control – involves the application of fuzzy theory to control of mechanical systems, process plants, etc. The three main headlines are: study of fuzzy control including fuzzy modelling, fuzzy reasoning, fuzzy adaptive and learning control, and stability evaluation; development and testing of support tools for design, simulation and stability evaluation; and testing and evaluation of application system, and by this establishing a development methodology for fuzzy control systems.

The second research laboratory - Fuzzy intellectual information processing – has five sub-topics: image recognition, databases, decision support system, natural language processing, and intelligent estimation.

The third research laboratory - Fuzzy computers – concentrates on fuzzy hardware and software which is able to represent and execute the concepts of qualitative expressions in the same way that humans do. These computers will be capable of high-speed processing of large amounts of fuzzy information and executing of fuzzy reasoning. They will be very user-friendly systems for fuzzy control and fuzzy intelligent information. The work proceeds on two fronts. In a top-down fashion, software and hardware that are anticipated are being developed on the basis of a total architecture. And in a bottom-up fashion, a system, which integrates an existing fuzzy technology and a computer technology, is being developed.

Fuzzy is a very hot subject in Japan today. The LIFE laboratory programs are very ambitious However, they had no results to show us yet.

B.4.5 Jaeri

General Information

Japan Atomic Energy Research Institute (JAERI) is supervised by the Science and technology agency. JAERI has 2.500 employees and mainly works with:

- Nuclear Energy production systems including high temperature gas-cooled reactors and fusion reactors.
- Nuclear safety research.
- Radiation applications.

- Nuclear ships.

JAERI is an active part of the OECD Halden Reactor Project in Norway. We visited two groups dealing with reactor safety research and reactor engineering.

Technical Information

The Department of Reactor safety research presented a project called: "A Computerized support system for the emergency technical advisory body in Japan". The main purpose of the system was to provide a national capability in emergency response to radiological accidents. Via this system, it should be possible to access all available information about, e.g., similar accidents in the past, in order to help the operators to make the right decisions. The system layout included access to central databases via modem communication.

The Department of Reactor Engineering's major R & D issues were:

- Reactor dynamics and Control.
- Telerobotics for Nuclear Applications.
- Intelligent Robotics

From an AI point of view the work is concentrated on intelligent control including expert Control and fuzzy control. Expert control is intended for reactor control and robotics while fuzzy control has so far been intended for robotics only. We received a paper describing a project using a conventional rule-based expert system approach to optimal reactor shutdown. Another paper described a project where a self tuning fuzzy controller for a mobile robot had been implemented. The control rules for the robot were acquired through an adaptive learning process. JAERI have purchased G2 for further work with the mobile robot.

B.4.6 Hitachi

General Information

Within Hitachi Ltd, there is about 78.000 employees, with 12.000 working with R & D. The corporate R & D is organized in nine laboratories with totally 5.000 persons. Totally there are 275.000 employees in the Hitachi group. We visited Hitachi Research Lab (HRL), located in Hitachi City which mainly deals with materials, electronics, and energy. The research on AI issues is handled by the Department for Electronics at HRL.

Technical Information

Hitachi has worked with AI since 1980, and neural networks since 1989. Hitachi have developed their own expert system tool called EUREKA which is sold as a commercial product.

They showed/presented the following AI-applications:

- Power generation scheduling.
- Restoration of distribution lines.
- CAE for power system analysis.
- BWR power maneuver planning.
- Load following demand allocation between nuclear power plants.

It would be going too far to describe all the systems above so we will only describe the KBS system for load following demand allocation between nuclear power plants. It is intended for engineers at a power company's head office. In the system knowledge base, procedures for planning the demand allocation are expressed as rules. The load following demand, the operation status of each plant, and constraints are expressed as frames. The functions for evaluating margins of the constraints are expressed as methods. As for the results, one week to three year operation plans are stored.

In common for most of the applications above seems to be that the knowledge base consists of rules and frames.

Neural network applications:

- Turbine Vibration Adaptive Prediction Control.
  This project was very interesting because they used time series information and the corresponding frequency spectrum from the generator to identify the different failure states. The neural network was "trained" to identify 15 failure states.

Fuzzy Control applications:

- Automatic operation method for control rods in BWR plants.
  The method has the following features. (1) judgement of control rod driving time using an event-driven method; and (2) tuning of control rod withdrawal length and the control parameters using fuzzy logic. (They admitted that they had some problems to get the authorities to accept fuzzy logic in nuclear power plants.)

During the meeting we got the feeling that they didn't show us their latest research. According to our Japanese guide from the Swedish Attache' office they had done research for several years on integrated knowledge based systems, similar to our ideas in the IT-4 project.

B.4.7 Tokyo Institute of Technology

On Thursday morning we visited Dr. Shigeyuki Tomita at the Research Laboratory of Resources Utilization, Tokyo Institute of Technology. Dr. Tomita began by presenting an AI-based system for synthesizing plant operating procedures for chemical processes. The system was based on a top-down approach where the functions in the plant were hierarchically classified and heuristic knowledge relative to operating procedures of fixed routine were generalized and represented in a script-like data structure. The system was mainly aimed at guidance during normal operating conditions.

The root in the function hierarchy was the function *normal-operation*. This function could be seen as made-up of a set of main functions such as *reaction*, *blending*, *storage*, and *separation*. Also, each main function could consist of a set of sub-functions.

The main representation form used are scopes. A scope is defined as a frame-like data structure that corresponds to a minimal portion of the plant that can realize one of the functions of the plant. There are two types of scopes. Primitive-scopes (P-scopes) represent the primitive elements of the plant, such as process components and pipes. These are interconnected in a directed graph. Functional-scopes (F-scopes) represent the process functions. Scopes are organized into a scope-library.

Scripts for plant operations embody generic knowledge for getting a scope to function: the overall purpose of the operation sequence, pre-conditions to be satisfied in advance, the order of realizing subordinate functions commonly accepted as a standard, etc. As with scopes, scripts are organized into a script-library.

Using the scripts and the scopes the system can automatically generate a scope-net for a plant and from this generate an appropriate sequence of operating procedures.

The system was implemented in Kyoto Common Lisp on a Vax 8600. It had no graphical interface. Several of the ideas in the system where similar to the MFM technique, of Morten Lind. Dr. Tomita was, however, not familiar with MFM. The paper "On the development of an AI-based system for synthesizing plant operating procedure" gives more details on the technique.

Dr. Tomita also gave us a paper where a bottom-up approach for synthesizing plant operating procedures was used. Here, heuristic knowledge was generalized and classified into a sets of various kinds of fragmentary knowledge. This

knowledge-base was used to get a guideline of the operation under abnormal operating conditions.

Dr. Tomita also briefly presented a project on fault diagnosis of chemical plants. The system was called FINDS (Fault Identification Using Natural Diagnostic Strategy). The system was able to automatically generate mass balance equations from the process and instrumentation diagram.

Finally, Dr. Tomita presented a system for planning of batch process operation where multiple products, different recipes, and different combinations of equipment must be taken into consideration. Originally, the system was based on finding an optimal solution using linear programming techniques. However, these techniques was found not to be flexible enough. The current system was therefore implemented using heuristic scheduling rules that did not necessarily generate an optimal solution, but instead provided the necessary flexibility with respect to, e.g., adding new constraints to the planning problem. The system was set up so the user could choose between four different sets of heuristic scheduling rules. The system was implemented in Smalltalk on a Tektronix AI workstation.

Several of the projects and ideas that Dr. Tomita presented were very interesting and fit well into the IT4 project.

B.4.8 Japanese Gas Company

On Friday morning we visited Japanese Gas Company in Yokohama. This visit was organized by Mr. Makoto Yokoyama, the Japanese representative for Gensym, who also accompanied us.

Japanese Gas Company is a plant engineering and construction company in the areas of petroleum refining, chemical production, nuclear energy, food processing, pharmaceutical manufacturing, pipeline construction, and environmental protection. The number of employees are 2.600 with 75% engineers, mainly chemical engineering.

At JGC we met Mr. Katsumi Tanaka, the manager of the AI Technology Team at the Systems Integration Division. The reason for our visit to JGC was their work with G2. For various reasons, we were not allowed to see this. Instead Mr. Tanaka presented their CATCH systems. CATCH (Computer Aided Operation & Trouble Checking System) is a series of expert systems developed within JGC.

- CATCH-ARIS (Advanced Refinery Instruction System) for diagnosis and operation support.
- CATCH-FOSTS (Fluid Catalyst Cracking Operation Support and Trouble checking System) for diagnosis, plant start-up support, and yield checking.
- CATCH-SR (Sulphur Recovery unit) for diagnosis and start-up support.

- CATCH-ARES for operation support of cooling water operation.
- CATCH-DE for diesel engine diagnosis.

The first three systems were implemented in Brains, a simple rule-based tool from Toyo Information Systems. CATCH-ARES was implemented in Guru and CATCH-DE directly in C. The systems were PC-based and all but one installed into plants with good results. From a technical point of view the systems were not impressive.

In their current systems, JGC is using G2. The plants they are working with are a phosphoric plant in Belgium and a liquid natural gas receiving plant in Japan. In both cases, G2 is intended for diagnosis and operator support. The plant in Belgium is developed together with Rhone-Poulenc and within this project JGC has developed an interface between G2 and Yokogawa CENTUM, which they will sell as a separate product.

JGC also has developed expert systems for flow meter selection and for regulatory information support. The latter is an interactive system that allows the formation of correct judgement on the application of related laws and regulations.

The main motivation for JGC to develop expert systems is the difficulties in finding new skilled process operators. This is a major problem in Japan, which leads JGC's customers to look into expert systems.

B.4.9 Petroleum Energy Centre

On Friday afternoon we visited the AI lab at the Petroleum Energy Centre in Tokyo were we met Dr. Yoshihiko Tamura. Dr. Tamura was from the Electrotechnical Lab at MITI and also a MITI representative in the European Communities Esprit project. Dr. Tamura presented the PRIOS project, a 4-year project started in June 1987, which goal was the development of technologies for advanced refinery operation systems including process operation support technology (faults inspection, prediction, and diagnosis, monitoring & measurement systems, consultation for operators' processing, start-up & shut-down, advanced controls) and production planning and scheduling support technology (daily plant operation scheduling for monthly demand, tanker loading/unloading planning, plan management and maintenance.)

The lab was equipped with Elis Lisp Machines, Symbolics, Suns, a plant simulator for an indirect desulphurization plant (NATRAS) together with Hitachi minicomputers, and Yokogawa Centum PLCs.

Some of their research issues for now were model-based approaches to fault diagnosis using influence digraphs and quantitative balance equations similar to the DMP method used in our project, feedforward control of systems with large

time delays, procedure planning for start-up/shut-down, intelligent monitoring and measurement, fuzzy control, and instrument maintenance support.

It was very difficult for us to get any detailed information about how far they had reached and they were not willing to show us anything. The general impression of this visit was that they had done a lot but were very reluctant to show any of it to us.

B.4.10 G2 Users meeting – C.Itoh Techno-Science Co.

On Monday Karl-Erik Årzén and Christer Gerding attended the first japanese G2 users group meeting at C. Itoh Techno-Science Co. in Tokyo. The meeting was attended by around 35 persons, mostly G2 users. Karl-Erik Årzén was invited as a guest speaker together with Dr. Don Gardner from Center for Integrated Systems, Stanford University. From Gensym Corporation, Robert Moore, Jim Allard, Greg Stanley, and Ray Haarstick attended.

The meeting was opened by Robert Moore and Roy Haarstick who gave an overview of Gensym. The company now has 42 employees. The sales are increasing with the result of the third quarter 1989 being 1.2 MUSD. The set of GSI interfaces to standard process control systems is increasing and now also includes Honeywell TDC 2000 and 3000, Bailey Net 90, Taylor Mod 300, Siemens S5, Allen Bradley, and the relational databases Oracle and DEC RDB. Combustion Engineering has developed a system specially for the paper industry based on G2.

Karl-Erik Årzén presented and demonstrated the Steritherm – G2 prototype After lunch Don Gardner showed a video tape where G2 was used for semiconductor manufacturing modelling simulation, and monitoring. Then, Mr Oka from Toyo Engineering Company gave a presentation (in Japanese) about a knowledge-based training simulator written in G2 and Mr. Tanaka from JGC presented the G2 - Yokogawa Centum interface also in Japanese.

Finally, the Gensym people presented and demonstrated some of the new G2 version 2.0 features. Apart of the things that we already have in our Alfa version (procedures, relations, parameters, transient objects, multi-region objects, user-defined menu choices, etc.), version 2.0 will also contain lists, improved alarm message features, kanji and swedish character support, and an icon editor. Telewindows that allows multiple G2 users running remotely will also be ready when version 2.0 is released in February 1990.

Unfortunately we had to leave before most of the demonstrations in order to catch our flight home.

B.4.11 Conclusions

The activities concerning real-time expert systems in Japan were impressive more with respect to the numbers of projects and the number of expert systems in industrial use than the technological level. We got the impression that in Japan expert systems are considered to be an accepted technique which is regularly applied in industrial applications. By expert system is usually meant a rule-base system. The two largest companies that we visited, Toshiba and Hitachi, had very large groups on AI applications. None of the systems that we saw were comparable to, e.g., G2 in power. One reason for this could be that we were not shown their latest developments. Sometimes we had a feeling that the systems they showed us were developed two or three years ago. The reason for this can be competition. However, it can also depend on the fact that in order not to loose their face the demonstrations that they show to other companies must be very well prepared. It is possible that they had shown more if our group only had consisted of persons from universities. Also, from Mr. Yokoyama, the Japanese G2 representative, we got information about places that perhaps would have been more interesting to visit than some of the places we actually visited.

Both Meidensha Electric Company, the fifth largest electric company in Japan, and Toyo Information systems have recently announced real-time expert system tools. According to Mr. Yokoyama both of these were modelled after G2. The system from Toyo was an extension to their tool Super-Brains. Some of the other actors on the Japanese arena were Mitsubishi Heavy Industries who had several installations of P-DIAS, a non-AI diagnostic systems; Hitachi who sold their Eureka-III system; Fuji Electric who used expert systems connected to their distributed process control systems for water treatment plants, hydraulic plants, and chemical plants; and Chiyoda Chemical Construction who distributed PICON in Japan and had a couple of installations of a real-time expert system called C-Rex.

The main reason for using industrial on-line expert systems is the difficulties that the process industries have in finding new operators. This was mentioned several times. Japanese youth are not willing to work as process operators. The companies see expert systems as a way to help them to better utilize the knowledge of skilled operators and to provide a better environment for new operators.

In spite of the technological level, it is interesting to note that several of the japanese control system developers had their own expert system tools which they could connect to their systems. Some examples are the Yokogawa XL/AI system, Toshiba's TDES3, and Hitachi's EUREKA-III. Even if the systems are not integrated in the IT4 sense, these companies have reached further than comparable companies in the US and in Europe.

The lasting impression of the trip is perhaps the enormous activity in fuzzy techniques and primarily fuzzy control. More than 100 industrial applications of fuzzy control have been reported. Fuzzy controllers are used for supervisory set-point

control of multivariable, nonlinear processes. Some examples are coordination control of elevator systems and train control. Apart from the LIFE project, an additional 80 MUSD are spent on a separate Fuzzy Systems Research Project led by Professor Sugeno from MITI.

B.4.12 Documentation

The following material was handed out from the different companies visited and is available from the project members.

Yokogawa

Kazuo Sueyoshi *et al*, (1989): Process Diagnostic Expert System for Cumene Plant (in japanese).

Nippon Kokan Steel

NKK Research & Development

NKK Blast Furnace Expert System

NKK Ohgishima: the ultramodern steelworks on the sea

NKK Refuse Incinerator Operation Guidance Expert System

H. Ase *et al* (1988): Refuse Incinerator Operation Guidance Expert System, Nippon Kokan Technical Report, Overseas No. 52

M. Shibata *et al*: Application of expert system for blast furnace operation control

Toshiba

Toshiba Fuchu Works

Toshiba Artificial Intelligence Technology

T. Fushimi *et al*: Turbine-generator on-line monitoring and diagnostic expert system

N. Inoue *et al* (1989): An expert system for intelligent alarm processing in EMS and SCADA systems, Second Symp. on Expert Systems Applications to Power Systems, Seattle

T. Kaneko *et al*: Development of FBR plant operational guidance system

S. Kawakita *et al* (1988): An integrated AI environment for industrial expert systems, Int. Workshop on AI for Industrial Applications, Hitachi-city

Y. Kojima *et al* (1989): The development of power system restoration method for a bulk power system by applying knowledge engineering techniques, IEEE Transactions on Power Systems, Vol. 4, No. 3

R. Megoro *et al*: Expert system for nuclear power plant feedwater system diagnosis

Y. Miyajima *et al* (1988): A knowledge-based water purification control system, IEEE Proc. of the International Workshop on AI for Industrial Applications, Hitachi-city

S. Moriguchi *et al* (1989): A large-scale SCADA system with real-time knowledge-based functions, Second Symp. on Expert Systems Applications to Power Systems, Seattle

S. Moriguchi *et al* (1989): An expert system for power system fault analysis and restoration, International Conference on Large High Voltage Electric Systems

H. Ogi *et al*: An expert system with cognitive model for power system outage scheduling

T. Sato (1989): An expert System for mill pacing in bar mills, 6th IFAC Workskop on Distributed Computer Control Systems, Tokyo

I. Takeyasu *et al* (1988): An expert system for fault analysis and restoration of trunk line power systems, Symp. on Expert System Applications to Power Systems

LIFE

LIFE Introduction to Laboratory for International Fuzzy Engineering Research

LIFE Research & Development Project Overview: The First Research Lab (Fuzzy Control)

LIFE Research & Development Project Overview: The Second Research Lab (Fuzzy Intellectual Information Processing)

LIFE Research & Development Project Overview: The Third Research Lab (Fuzzy Computer)

Colin Johnson (1989): New LIFE for fuzzy logic, Electronic Engineering Times

Proceedings of the 5th Fuzzy System Symposia, Meiji University, June 2–3 1989 (in Japanese with abstracts in English, only available at the Department of Automatic Control)

JAERI

JAERI Japan Atomic Energy Research Institute

Y. Fujii *et al* (1988): Design and operational experience of the man-machine interface of a fully computerized control system, Int. Conf. on Man-Machine Interface in the Nuclear Industry, Tokyo

K. Kobayashi *et al*: Development of a computerized support system for the emergency technical advisory body in Japan

Y. Shinohara: Application of an AI method to optimal reactor control problems

K. Suzuki *et al*: Self-tuning fuzzy control of a mobile robot

Hitachi

Hitachi, Hitachi Research Laboratory

Hitachi, Energy Research Laboratory

C. Fukui *et al* (1986): An expert system for fault section estimation using information from protective relays and circuit breakers, IEEE Transactions on Power Delivery, Vol 1, No. 4

T. Fukuzaki *et al* (1989): Knowledge-based system for load following demand allocation between nuclear power plants

T. Fukuzaki *et al* (1988): Knowledge-based system for core operation management of boiling water reactors, Int. Workshop on AI for Industrial Applications, Hitachi-city

T. Kasahara *et al* (1988): Maintenance work scheduling aid for nuclear power plants, Int. Workshop of AI in Industrial Applications, Hitachi-city

M. Kinoshita *et al* (1988): An automatic operation method for control rods in BWR plants using fuzzy logic

Y. Matsumoto *et al*: An expert system for restoration of distribution lines

T. Mitsuta *et al*: A knowledge-based approach to routing problems in industrial plant design

S. Osaka *et al* (1988): An expert system for power generation scheduling, Int. Workshop on AI for Industrial Applications, Hitachi-city

M. Suwa *et al*: A theory of frustration-based learning mechanism

N. Yamada *et al* (1984): A Plant diagnosis method based on the knowledge of system description, Jour. of Information Processing, Vol 7, No. 3

N. Yamada *et al* (1989): Knowledge-based operation guidance system for nuclear power plants based on generic task methodology, Jour. of Nuclear Science and Technology, Vol 26, No 7.

N. Yamada *et al*: A Theorem proving system for logic design verification

K. Yoshida *et al*: Knowledge-based layout design system for industrial plants

Tokyo Institute of Technology

S. Tomita *et al* (1989): Automatic synthesizer of operating procedures for chemical plant by use of fragmentary knowledge, Jour. of Chemical Engineering of Japan, Vol. 22, No. 4

S. Tomita *et al*: On the development of an AI-based system for synthesizing plant operating procedures

S. Tomita *et al* (1986): Development of batch operating system, World Congress III of Chemical Engineering, Tokyo

Japanese Gas Company

JGC, Facts about JGC

JGC, Outline of JGC

JGC, Plant operations support expert systems

M. Kitahara: Computer-aided operation & trouble checking system

Petroleum Energy Center

PEC Petroleum Energy Center

PEC Prios-project

G2 Users Meeting

Gensym/G2 User Society, Fall'89 Meeting Proceedings, Boston (only at the Department of Automatic Control)

D. Gardner (1989): Equipment modeling, simulation and monitoring using a knowledge base system

D. Rowan (1989): On-line expert systems in process industries, AI Expert (Du Pont's view on expert systems in the process industry)

Various G2 documentation in Japanese

B.4.13 Persons to contact

These are the persons that we met during our visits.

Yokogawa

Main Address: Yokogawa Electric Corporation, 2-9 32 Nakacho, Musashino-shi, Tokyo, 180 Japan. Fax: 0422 55-1728

Yasuro Hirata, Manager, AI Engineering Section, Application Engineering Dept., Sales Engineering Subdivision

Kiyokazu Konishi, Manager, Section 2, Development & Engineering Dept. II, Process Control Systems Div.

NKK

Main Address: NKK Corp., Ltd., NKK Keihin Bldg., 1-1, Minamiwatarida-cho, Kawasaki-ku, Kawasaki 210 Japan. Fax: (044) 322-6644

Hajime Ase, Senior Researcher, AI Research Project, Electronics Research Center

Masanori Itoh, Manager Electronics Research Center

Yasunori Ohnishi, Engineer, Systems & Control Engineering Section, Electrical & Control Engineering Dept., Engineering & Construction Div.

Motohiro Shibata, SE Manager, Engineering Dept., Technical Support Div., Knowledge Engineering Group

Toshiba

Main Address: Toshiba Corp., Fuchu Works, 1, Toshiba-cho, Fuchu-shi, Tokyo, 183 Japan. Fax: (0423)-33-0141

Shigeru Kawakita, Manager, Computer Control Development Section, Heavy Apparatus Engineering Laboratory

Kiyoshi Niki, Manager Nuclear Power Generation & Fusion Technology, Control Systems Development Section, Power Generation Computer Systems Department

Yoshihiro Takagi, Manager, Europe, International relations office

LIFE

Main Address: LIFE, Siber Hegner Building 4Fl., 89-1 Yamashita-cho Naka-ku, Yokohama-shi 231 Japan. Fax: +81 45-212-8255

Itsuko Fujimori, Chief Manager

Tomohiro Takagi, Division Director, 2nd Laboratory

Toshiro Terano, Executive Director

Takeshi Yamanaka, Division Director, 1st Laboratory

Seiji Yasunobo, Division Director, 3rd Laboratory

JAERI

Main Address: JAERI, Tokai-mura, Naka-gun, Ibaraki-ken 319-11, Japan, Fax: 0292-82-6147

Fumie Kawakubo, Research Engineer, Department of Reactor Safety Research

Kensuke Kobayashi, Principal Scientist, Department of Reactor Safety Research

Kiyoshi Matsumoto, Research Engineer, Human Factors Research Laboratory, Dept. of Reactor Safety Research

Yoshikuni Shinohara, Reactor Control Laboratory

S. Taniguchi, Deputy Director General, Tokai Research Establishment

Masao Yokobayashi, Senior Engineer, Human Factors Research Laboratory, Dept. of Reactor Safety Research

Hitachi

Main Address: Hitachi Research Laboratory, Hitachi Ltd., 4026 Kuji-cho, Hitachshi, Ibaraki-ken, 319-12 Japan, Fax: 0294-53-2810

Hiroyuki Kudo, Senior Researcher, The 10th Dept.

Hiroshi Motoda, Chief Researcher, Advanced Research Laboratory, Kokubunji

Munehiko Tonami, Manager, International Coordination Office

Tokyo Institute of Technology

Main Address: Tokyo Institute of Technology, 4259 Nagatsuta-cho, Midori-ku, Yokohama, 227 Japan. Tel: -45-922-1111

Shigeyuki Tomita, Research Laboratory of Resources Utilization

Japanese Gas Company

Main Address: JGC Corporation, 14-1, Bessho 1-chome Minami-ku, Yokohama, 232 Japan. Tel: 045-7121111

Katsumi Tanaka, Manager AI Technology Team, Industrial Systems Dept., Systems Integration Division

Petroleum Energy Center

Yoshihiko Tamura

G2 Users Meeting

Main Address: C.Itoh Techno-Science Co., LTD., 16-7, Komazawa 1-chome, Setagaya-ku, Tokyo, 154 Japan. Fax: 03-419-9209

Kiyoshi Azegami, AI Systems Support Group, FA Business Operations

Hirobumi Fukuoka, Special Consultant, AI/RT Dept, Systems Business Operations No. 2

Donald S. Gardner, Hitachi Research Laboratories

Yasuo Miyake, General Manager, AI/RT Dept, Systems Business Operations No. 2

Makoto Yokoyama, Manager AI Group AI/RT Dept, Systems Business Operations No. 2

C

Glossary

The glossary contains explanations for the terminology used in the report. It consists of words that are specific to the knowledge-based system area and words which are used, sometimes in a special, non-standard way, for describing the system concept.

Agenda — A prioritized list of waiting activities. Used in blackboard systems to schedule knowledge sources.

Antecedent — The IF-part of a production rule. Other names are premise and condition.

Application-specific system — Knowledge-based system framework aimed at a specific type of applications.

Artificial Intelligence — A subfield of computer science, which according to one definition is the study of how to make computers do things at which, at the moment, people are better.

Attribute — A property of an object. Also called slot.

Backward chaining — An inference method where the system starts with what it wants to prove and then tries to find the necessary facts in the database or as the conclusion of a rule. Also known as goal-directed search. Contrast with forward chaining.

Blackboard

A database used by several knowledge sources to exchange information about the problem solving and to store the problem solving state.

Blackboard architecture

An expert system architecture in which several independent knowledge sources each examine a common database, called a blackboard. An agenda-based control system continually examines all of the possible pending actions and chooses the one to try next.

Browser

The browser constitutes a graphical interface to the knowledge base that is common to all users of the KBCS.

Causal model

A model of a physical object that expresses the causal relations among the involved signals, or events.

Certainty factor

A number that measures the certainty, credibility or confidence a fact or rule has.

Class

A group of objects that share the same attributes and behaviour. Organized into an inheritance lattice.

Cognitive science

An interdisciplinary research area concerning the principles by which intelligent entities interact with their environment. Includes topics from psychology, computer science, physiology, philosophy, engineering etc.

Common knowledge base

The central part of the system concept. It contains all the knowledge that are of interest in an application. It consists of objects, rules, procedures, equations, documents, pictures, dynamic process data, models, etc. Can be separated into local databases and the main knowledge base.

Compiled knowledge

Knowledge that has been structured and compiled into a form that is efficient for executing. The heuristic knowledge of, e.g., a human process operator can, e.g., be compiled into a production rule format.

Composite object

An object that has an internal structure consisting interconnected objects representing subparts of the superior object.

Concept Used as a designation for real or abstract objects and terms, that are represented in a knowledge-based system.

Conflict resolution The technique of resolving the problem of multiple matches in a rule-based system.

Control (of a KBS) The method used by the inference engine to regulate the order in which reasoning occurs. Backward chaining, forward chaining, and blackboard agendas are all examples of control methods.

Data-driven An approach to problem solving that starts from current or initial information and employs forward chaining.

Declarative knowledge A description of *what is*. Contrast with procedural knowledge, which is a description of *how to*.

Declarative programming An organizational technique for computer programs. The wanted result and preconditions are stated instead of a step-by-step description of how to solve the task. Contrast with procedural programming.

Deep knowledge Knowledge of basic theories, first principles, axioms, and facts about a problem domain. Often in the form of a model of the behaviour of the problem domain. The model could be expressed as, e.g., a causal model.

Demon A procedure that is attached to a frame attribute. The procedure is executed when the attribute is changed or referred to. The programming style is called access-oriented.

Design tool A tool that is used by the designers to build up the knowledge base.

Diagnostic Model Processor – DMP A method for model-based diagnosis based on quantitative, governing equations.

Empirical knowledge Knowledge based on empirical or experiential observations of a process.

Experiential knowledge Knowledge based on empirical or experiential observations of a process.

Expert control

Seeks to extend the range of conventional control algorithms by encoding general control knowledge and heuristics in a supervisory expert system.

Expert system

A computer program that uses expert knowledge to attain high levels of performance in a narrow problem area. The results are compatible with those of a human expert. Knowledge-based system is sometimes used as a synonym.

Expert system framework

A computer environment that provides different tools for implementing expert systems. One or many knowledge representation techniques are supported. Also expert system shell.

Expert system shell

A computer environment that provides different tools for implementing expert systems. One or many knowledge representation techniques are supported. Also expert system framework.

Explorative programming

Programming without any given specifications. Requires powerful and flexible programming environments and programming languages that support rapid testing and prototyping.

First generation diagnosis system

Knowledge-based diagnosis system based on empirical knowledge, usually represented as rules, of how fault symptoms and causes relate.

Forward chaining

A problem solving technique where hypotheses are verifies by starting with known facts and trying to make deductions from these. The same as data driven. Contrasts with backward chaining.

Frame

A knowledge representation scheme based on the idea of a frame of reference. A frame consists of slots or attributes that describe the features of the frame. The slots are further described by facets.

Functional view

The functional view describes an object in terms of the goals that it should fulfill, the functions needed to fulfill these goals, and

	the process components that realize these functions.
Fuzzy controller	A controller based on rules of how the control variable be selected based linguistically quantized values of the measured variables. Uses fuzzy logic to describe the quantized values.
Fuzzy logic	A logical theory where the truth values 'true' and 'false' have been replaced with more approximate values like 'not very true', 'not likely' and 'very unlikely'.
Garbage collection	A background activity where free memory is reclaimed to the programming system. Important in languages with dynamic memory allocation, e.g., Lisp.
Geographical view	The geographical view describes a physical object in a geometrically and isometrically correct way.
Goal-directed system	An inference method where the system starts with what it wants to prove and then tries to find the necessary facts in the database or as the conclusion of a rule. Also known as backward chaining. Contrast with data-driven system.
Heuristic	A rule or some other piece of knowledge that is based on experience or observation: a rule of thumb.
Hybrid system	An expert system shell that allows a variety of different knowledge representation techniques.
Hypertext	A technique that extends the traditional notion of "flat text" files by allowing more complex organizations of the material. Mechanisms that allow direct machine-supported links from one textual chunk to another and new interactive interface techniques allow the user directly interact with these chunks and to establish new relationships between them.
Hypermedia	A hypertext system that also includes non-textual informations such as images, time series signals, audio recordings, etc. Also called multimedia.

Induction system — A system that can deduce rules from a material consisting of many examples from the problem area.

Inference — The process of drawing conclusions from premises.

Inheritance — A process where new objects in a hierarchical structure can get new attributes deduced from more general objects in the structure.

Inference engine — The part of a knowledge based system that contains the general problem-solving knowledge.

Instance — An object that describes a unique member of some object class.

Instantiation — The process where a new individual of a certain type is created.

Job shop scheduling — The job shop scheduling or factory scheduling problem concerns the allocation over time of a finite set of resources to specific manufacturing operations such that the orders for parts received by the factory are produced in a timely and cost-effective fashion.

KBMS — Knowledge Base Management System. The KBMS is the interface between the tools and the main knowledge base.

Knowledge — Information that is used to behave in an intelligent way. In the concept knowledge is used in a wide sense including all types of information, e.g., text book knowledge, heuristics, models, documents, control code, etc.

Knowledge acquisition — The process of acquiring, structuring and organizing the knowledge of a particular domain. Also knowledge elicitation.

Knowledge base — The part of a knowledge based system that contains the knowledge.

Knowledge-Based Control System — KBCS. A control system that integrates conventional programming techniques and knowledge-based techniques using a common data or knowledge base.

Knowledge-based system – KBS A computer system that contains knowledge and is able to reason with that knowledge to reach solutions. Sometimes a synonym for expert system. See expert system.

Knowledge elicitation The process of acquiring, structuring and organizing the knowledge of a particular domain. Also knowledge acquisition.

Knowledge engineer The person who designs and builds the expert system. This person should have experience of artificial intelligence methods.

Knowledge representation Formalisms used to represent knowledge. By using these formalisms, it is possible to handle and manipulate the knowledge. Typical formalisms are semantic networks, frames and predicate logic.

Knowledge source Knowledge module in a blackboard system.

Learning control A combination of AI techniques and control theory that utilizes various learning schemes for control purposes. Also intelligent control and self-organizing control.

Lisp machine Workstation with dedicated hardware for Lisp execution. Has very powerful programming environment.

Local databases Contains the dynamic process data of the common knowledge base. They are localized on the distributed processing units in the system.

Local fault model A model that relates faults in a physical component with possible causes internal to the component.

Main knowledge base Contains all the knowledge in the common knowledge base except for the dynamic process data. Can be distributed.

Mental model The human operator's apprehension of how the process behaves.

Message passing Communication method between objects in a object-oriented system. Supports data abstraction and generic algorithms.

Method Procedure associated with an object that responds to a certain message.

Modal logic

A logic system that includes the notions of necessity and possibility. A proposition is necessarily true if it could not be the case that it was false. A proposition is possible if it is not necessary that it be false

Modus ponens

An inferencing rule which says that whenever a fact **A** is known to be true and there is an implication $\mathbf{A} \Rightarrow \mathbf{B}$, it is permitted to conclude that **B** is true.

Monotonic reasoning

A logical system where axioms that have been stated and conclusions that have been drawn are not allowed to change during the reasoning process. The set of beliefs monotonically increases. The case for standard logic systems.

Multilevel flow models – MFM

A technique for describing physical systems that emphasizes functional relations among the involved components. Systems are described in terms of goals, functions, and components. Two main abstraction relations exist: the part–whole relation and the means–ends relation. Developed by M. Lind.

Multiple inheritance

An inheritance mechanism where a class may have more than one superclass. Contrasts with single inheritance.

Multiple perspectives

Used in some object-oriented systems for the case when a single object, at any one time, can be seen as the instance of one of a set of classes. It can be seen as a special kind of multiple inheritance where the behaviour and attributes from the inherited classes are kept separated in the object instead of being combined together.

Multiple worlds

Represents alternative states of knowledge in which different assumptions have been made. They allow the problem solver to set up hypothetical assumptions which are automatically withdrawn when worlds are deleted. Also multiple viewpoints and hypothetical worlds.

Multi-view object

An object in the main knowledge base that represents all the individual views that the object can be described from.

Neural network Connectionist system modelled on the neuron structure of the human brain.

Nonmonotonic reasoning A knowledge based system allowing new information that can make old deductions become false. This is very important when information changes during execution.

Object-attribute-value triplets A way of storing knowledge in an object-oriented system.

Object-oriented programming A style of programming based on directly representing physical objects and abstract concepts in the machine. The basic entity is the object which has a local state and a behaviour. Objects are asked to perform operations by sending messages to them. Examples of programming languages are SIMULA and SMALLTALK.

Ontological knowledge Knowledge based on theoretical knowledge which is analytic and derived from first principles.

Opportunistic reasoning system A reasoning system that changes inferencing strategy depending on the problem solving state.

Predicate logic A classical logic which is based on the use of predicates to express relations among objects. The formal basis for Prolog. Also predicate calculus or first order logic.

Premise The IF-part of a production rule. Sometimes it is called 'antecedent'. The THEN-part is called 'conclusion'.

Procedural A technique for organization of programs, by using a step by step description of how to solve a problem. The opposite word is declarative.

Procedural knowledge A description of *how to*. Contrast with declarative knowledge, which is a description of *what is*.

Process The controlled flow of matter, energy, or information from generation (source), via transport, storage, distribution, and change to consumption (sink). The flow may be discrete or continuous.

Process control — The different tasks that interact with a specified process and with the different users of the process.

Process control system — The system that controls and supervises the the operation of a process.

Production system — A rule-based system where rules or productions are matched against the contents of a working memory and executed by forward chaining inferencing.

Production rule — A type of rule in a knowledge based system, usually expressed as an IF-THEN statement.

Propositional logic — A classical logic which is based on propositions without any internal semantics.

Qualitative models — Models of a process in the terms that a human uses when describing and analyzing the process. Used for simulation and analysis. Based on the ideas in Naive Physics.

Qualitative knowledge — Knowledge about and based on matters that cannot be measured quantitatively.

Quantitative knowledge — Knowledge about and based on matters that can be numerically measured. An differential equation model is one example.

Realization tool — Realizes the different control functions in the system by extracting the relevant knowledge from the main knowledge, converting into executable code, and possibly, distributing it to local processing units.

Recognize-act cycle — The execution cycle of a forward chaining inference engine. Fulfilled rules are collected during the match phase. During the select phase one rule is chosen for execution. During the act phase the right hand side of the rule is executed.

Rule — A formal way to specify a fact, directive or strategy. The most common way to represent it is with the IF-THEN construction.

Rule based system — A program organized as a set of rules.

Script — A knowledge structure containing a stereotype sequence of actions.

Second generation diagnosis system Knowledge-based diagnosis system based on deep-level or first principles knowledge about the problem domain.

Semantic network A knowledge representation method based on a structure of nodes that represent objects and named arcs between the nodes that define attributes and relations.

Single inheritance An inheritance mechanism where a class may have only one superclass. Contrast with multiple inheritance.

Single-view object An object in the knowledge base with only one view. Is sometimes used to refer to one view of a multi-view object.

Superclass–subclass hierarchy A directed graph that describes the relations among object classes. A subclass inherits behaviour and attributes from its superclasses.

System A structuring primitive in the main knowledge base. A flow of material, energy, or information in the process.

Temporal logic A logic system that includes time intervals or time instants and truth relations over time.

Time constrained reasoning The situation where a reasoning system must be able to come up with the best solution before a certain deadline.

Tools Operates upon the main knowledge base. They build up the user interfaces and perform the different tasks in the process control system. Can be divided into design tools and realization tools.

Topological view The topological view described the internal structure of a physical object.

Truth maintenance system A system that revises sets of beliefs when new information is found to contradict old information. Inconsistencies in the set of beliefs are resolved by using dependency-directed backtracking to alter the minimal set of beliefs which is responsible for the contradiction.

User interface parameters The user interface parameters define the different user interfaces of the KBCS in terms

	of what parts of the knowledge base that should be presented and how they should be presented.
User view	The information from the common knowledge base that a specific user of the KBCS gets presented. May not necessarily equal a view in the main knowledge base.
Validity interval	A time interval that tells how long the associated fact is valid.
View	A structuring primitive in the main knowledge base. Describing an object from several views is a way of structuring the knowledge the object into "natural" parts. The most general views are the topological, the geographical, and the functional view.
Working memory	The fact database in a production system.

References

ALLEN, D.J. and M.S.M. RAO (1980): "New algorithm for the synthesis and analysis of fault trees," *Ind. Eng. Chem. Fundam.*, **19**, 1, 79–85.

ANDERSSON, M. (1989): "An Object-Oriented Language for Model Representation," *Preprints of the 1989 IEEE Control Systems Society Workshop on Computer-Aided Control Systems Design (CACSD)*, December 16, 1989, Tampa, Florida.

ARLABOSSE, F., B. JEAN-BART, N. PORTE, and B. DE RAVINEL (1988): "An efficient problem solving architecture using ATMS, tested on a non-toy case study," *AI Communications*, **1**, 4, 6 – 15.

ÅRZÉN, K-E. (1989): "Knowledge-based control systems – aspects on the unification of conventional control systems and knowledge-based systems," *Proc. ACC'89*, pp. 2233–2238.

ÅRZÉN, K-E. (1990): "Knowledge-based control systems," *Proc. ACC'90.*

ÅSTRÖM, K.J. (1974): "Ventiler och pumpar TFRT-3110, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden,".

ÅSTRÖM, K.J. (1989): "A dynamic model for the heat exchangers in Steritherm," Dept. of Automatic Control, Lund University, Sweden.

ATKINSON, M., F. BANCIHON, D. DEWITT, K. DITTRICH, D. MAIER, and S. ZDONIK (1989): "The object-oriented database system manifesto," *Proceedings of the First DOOD Conference*, Kyoto, Japan.

BERNARD, J.A. and T. WASHIO (1989): *Expert System Application Within the Nuclear Industry*, American Nuclear Society, La Grange Park, Illinois, USA.

BREUKER, J, C. DUURSMA, R. WINKELS, and M. SMITH (1989): "Knowledge representation in Eurohelp," *Proc. ESPRIT'89 Conference*, Bruxelles, pp. 258 – 270.

CHRISTIANSSON, M. and P. ERICSSON (1989): "Knowledge-based control and modeling with G2 TFRT-5411, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden,".

CLOCKSIN, W.F. and C.S. MELLISH (1981): *Programming in Prolog*, Springer Verlag.

DHALIWAL, D.S. (1985): "The use of AI in maintaining and operating complex engineering system,".

ELMQVIST, H. (1978): "A Structured Model Language for Large Continuous Systems TFRT-1015, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden,".

FORGY, C.L. (1982): "Rete: A fast algorithm for the many pattern/many object pattern match problem," *Artificial Intelligence*, **19**, 1, 17–37.

GREPA (1985): *Le Grafcet – de nouveaux concepts*, (GRoupe Equipement de Production Automatisée réuni à l'ADEPA) Cepadues - éditions, 111, rue Nicolas-Vauquelin, 31100 Toulouse, France.

HAYES, P.J. (1979): "The naive physics manifesto," in D. Michie (Ed.): *Expert Systems in the Micro-electronic Age*, Edinburgh University Press, Edinburgh.

HÖJERBACK, P. (1989): "Designing a process operator interface – a Hypermedia model TFRT-5414, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden,".

IEC (1975): "Chapter 351: Automatic control," *International Electrotechnical Vocabulary*, International Electrotechnical Commision, Geneève, Switzerland.

IT4 (1988): *Knowledge-Based Real-Time Control Systems: IT4 Feasibility Study.*

KUIPERS, B.J. (1986): "Qualitative simulation," *Artificial Intelligence*, **29**, 289–338.

LARSSON, J.E. and P. PERSSON (1987): "An expert system interface for Idpac TFRT-3184, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden,".

LEITCH, R. and K.D. HORNE (1989): "Prospects for intelligent training systems in industrial environments," *Proc. ESPRIT'89*, EC, Bruxelles.

LIND, M., *et al* (1987): "Systembeskrivelse og präsentation i proceskontrol," Rapport fra SIP-pilotprojekt, Servolaboratorium, Danmarks Tekniske Højskole, København.

LIND, M. (1987): "Multilevel Flow Modeling—Basic Concepts," The Servo Laboratory, Technical University of Denmark, Copenhagen.

MANNA, Z. and R. WALDINGER (1987): "How to clear a block: A theory of plans," *Journal of Automated Reasoning*, **3**, 343–377.

MARCHIONINI, G. and B. SCHNEIDERMAN (1988): "Finding facts vs. browsing knowledge in hypertext systems," *IEEE Computer*, **21**, 1.

MATTSSON, S. E. (1988): "On Model Structuring Concepts," *Preprints of the 4th IFAC Symposium on Computer-Aided Design in Control Systems (CADCS)*, August 23–25 1988, P.R. China, pp. 269–274.

MATTSSON, S. E. (ED) (1989): "New Tools for Model Development and Simulation TFRT-7438, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden,".

MATTSSON, S. E. (1989b): "On Modelling and Differential/Algebraic Systems," *Simulation*, **52**, No. 1, 24–32.

MOORE, R.L., L.B. HAWKINSON, M. LEVIN, A.G. HOFFMANN, B.L. MATTHEWS and M.H. DAVID (1987): "Expert system methodology for real-time process control," *Proc. 10th IFAC World Congress*, pp. 274–281.

NEUSCHAEFER, C.H., P.W. RZASA, E. FILSHTEIN, R.L. BURRINGTON, and R. DONAIS (1987): "Application of C-E's generic diagnostic system to power plant diagnostics," in S.M. Divakaruni, D. Cain, E. Baytch, and C. Saylor (Eds.): *Proc. EPRI Seminar Expert Systems Applications in Power Plants*, EPRI, Palo Alto, CA.

OYEN, R.A., M.A. KEYES, and M.P. LUKAS (1988): "An expert system shell embedded in the control system," *Preprints of the IFAC Workshop on AI in real-time control, September 21–23, Swansea, UK.*

PAN, J. Y-C., J.M. TENENBAUM, and J. GLICKSMAN (1989): "A framework for knowledge-based computer-integrated manufacturing," *IEEE Transactions on Semiconductor Manufacturing*, **SM(2)-2.**

PAVEK, P. (1990): "Personal Communication.,".

PETTI, T.F., J. KLEIN, and P.S. DHURJATI (1990): "Diagnostic model processor: using deep knowledge for process fault diagnosis," *accepted for AIChE J.*.

PLUS (1989): *Inside Plus*, Format Software GmbH, West Germany.

RASMUSSEN, J. and M. LIND (1981): "Coping with Complexity," Risø National Laboratory, DK–4000 Roskilde, Denmark.

ROWAN, D. (1989): "On-line expert systems in process industries," *AI Expert.*

SANDEWALL, E. (1988): "Non-monotonic entailment for reasoning about time and action, Part I: Sequential actions," Tech. Report LiTH-IDA-R-88-27, Dept. of Computer and Information Science, Linköping University, Sweden.

SCHANK, R.C. and R.P. ABELSON (1977): *Scripts, Plans, Goals and Understanding*, Lawrence Erlbaum Associates, Hillsdale, New Jersey.

TURNER, R. (1984): *Logics for Artificial Intelligence*, Ellis Horwood Limited, Chichester, England.