Knowledge-Based Methods for Control Systems

Larsson, Jan Eric

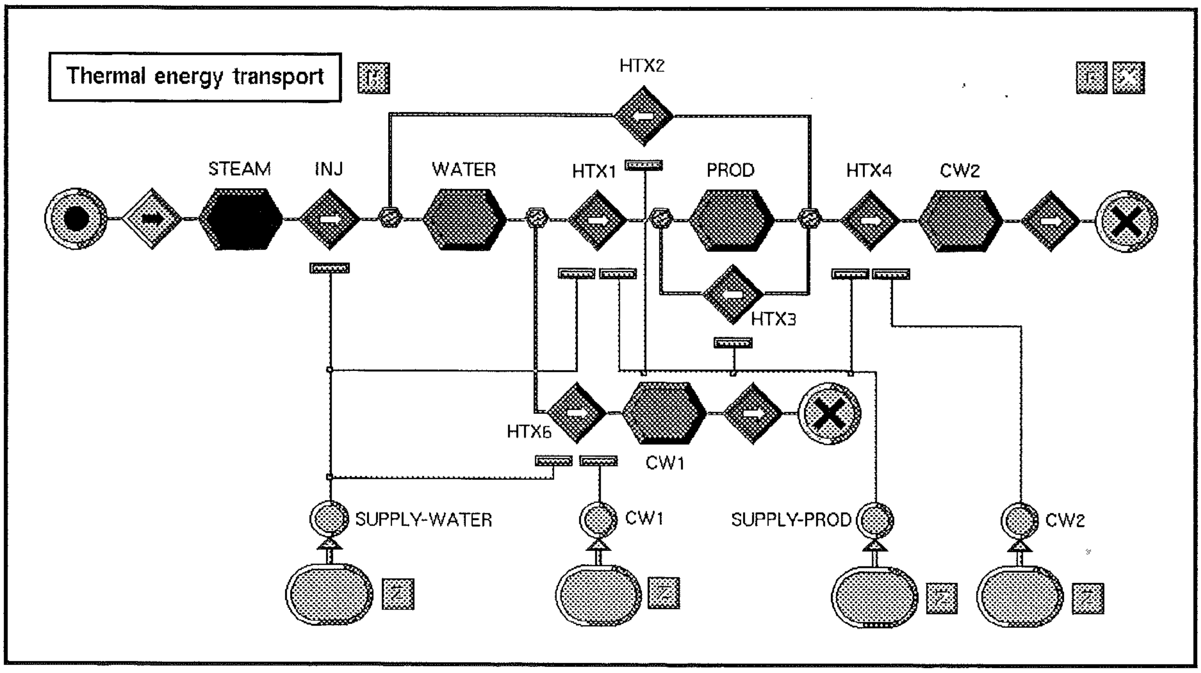
1992

LUND UNIVERSITY

PO Box 117
221 00 Lund
+46 46-222 00 00

Knowledge-Based Methods for Control Systems

Jan Eric Larsson



Department of Automatic Control, Lund Institute of Technology

Department of Automatic Control
Lund Institute of Technology
P.O. Box 118
S-221 00 Lund Sweden

Document name
Doctor's thesis

Date of issue
November 1992

Document Number
ISRN LUTFD2/TFRT--1040--SE

Author(s)
Jan Eric Larsson

Supervisor
Karl Johan Åström

Sponsoring organisation
STU, project no. 85–3042
IT4, project no. 3403
TFR, project no. 92–956

Title and subtitle
Knowledge-Based Methods for Control Systems

Abstract

This thesis consists of three projects which combine artificial intelligence and control.

The first part describes an expert system interface for system identification, using the interactive identification program Idpac. The interface works as an intelligent help system, using the command spy strategy. It contains a multitude of help system ideas. The concept of *scripts* is introduced as a data structure used to describe the procedural part of the knowledge in the interface. Production rules are used to represent diagnostic knowledge. A small knowledge database of scripts and rules has been developed and an example run is shown.

The second part describes an expert system for frequency response analysis. This is one of the oldest and most widely used methods to determine the dynamics of a stable linear system. Though quite simple, it requires knowledge and experience of the user, in order to produce reliable results. The expert system is designed to help the user in performing the analysis. It checks whether the system is linear, finds the frequency and amplitude ranges, verifies the results, and, if errors should occur, tries to give explanations and remedies for them.

The third part describes three diagnostic methods for use with industrial processes. They are *measurement validation,* i.e., consistency checking of sensor and measurement values using any redundancy of instrumentation; *alarm analysis,* i.e., analysis of multiple alarm situations to find which alarms are directly connected to primary faults and which alarms are consequential effects of the primary ones; and *fault diagnosis,* i.e., a search for the causes of and remedies for faults. The three methods use *multilevel flow models,* (MFM), to describe the target process. They have been implemented in the programming tool G2, and successfully tested on two small processes.

Key words
Alarm analysis, artificial intelligence, expert systems, fault diagnosis, frequency response analysis, help systems, system identification, intelligent front-ends, measurement validation, model-based reasoning, modeling, multilevel flow models, real-time, parameter estimation.

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
English

Number of pages
236

Recipient's notes

Security classification

The report may be ordered from the Department of Automatic Control or borrowed through the University Library 2, Box 1010, S-221 03 Lund, Sweden, Fax +46 46 110019, Telex: 33248 lubbis lund.

2 Tab.

Knowledge-Based Methods for Control Systems

Immanuel Kant pondering the nature of reality.

The illustration on the front page shows a screen dump of an alarm situation in the Steritherm energy network. Several MFM functions are alarmed, but a single failure, (shown in a darker red shade), can explain all the other alarms. The alarm analysis method is described in Chapter 5.

The quote in the preface translates to: "When the end is allowed, then the means are also allowed, excluding violence and injustice." See Pelle Holm, *Bevingade ord och andra talesätt,* Albert Bonniers Förlag, Stockholm, 1939.

Department of Automatic Control
Lund Institute of Technology
P.O. Box 118
S-221 00 Lund Sweden

Document name
Doctor's thesis

Date of issue
November 1992

Document Number
ISRN LUTFD2/TFRT--1040--SE

Author(s)
Jan Eric Larsson

Supervisor
Karl Johan Åström

Sponsoring organisation
STU, project no. 85–3042
IT4, project no. 3403
TFR, project no. 92–956

Title and subtitle
Knowledge-Based Methods for Control Systems

Abstract

This thesis consists of three projects which combine artificial intelligence and control.

The first part describes an expert system interface for system identification, using the interactive identification program Idpac. The interface works as an intelligent help system, using the command spy strategy. It contains a multitude of help system ideas. The concept of *scripts* is introduced as a data structure used to describe the procedural part of the knowledge in the interface. Production rules are used to represent diagnostic knowledge. A small knowledge database of scripts and rules has been developed and an example run is shown.

The second part describes an expert system for frequency response analysis. This is one of the oldest and most widely used methods to determine the dynamics of a stable linear system. Though quite simple, it requires knowledge and experience of the user, in order to produce reliable results. The expert system is designed to help the user in performing the analysis. It checks whether the system is linear, finds the frequency and amplitude ranges, verifies the results, and, if errors should occur, tries to give explanations and remedies for them.

The third part describes three diagnostic methods for use with industrial processes. They are *measurement validation,* i.e., consistency checking of sensor and measurement values using any redundancy of instrumentation; *alarm analysis,* i.e., analysis of multiple alarm situations to find which alarms are directly connected to primary faults and which alarms are consequential effects of the primary ones; and *fault diagnosis,* i.e., a search for the causes of and remedies for faults. The three methods use *multilevel flow models,* (MFM), to describe the target process. They have been implemented in the programming tool G2, and successfully tested on two small processes.

Key words
Alarm analysis, artificial intelligence, expert systems, fault diagnosis, frequency response analysis, help systems, system identification, intelligent front-ends, measurement validation, model-based reasoning, modeling, multilevel flow models, real-time, parameter estimation.

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280–5316

ISBN

Language
English

Number of pages
236

Recipient's notes

Security classification

The report may be ordered from the Department of Automatic Control or borrowed through the University Library 2, Box 1010, S-221 03 Lund, Sweden, Fax +46 46 110019, Telex: 33248 lubbis lund.

Knowledge-Based Methods for Control Systems

Jan Eric Larsson

Department of Automatic Control, Lund Institute of Technology
November 1992

5

Department of Automatic Control
Lund Institute of Technology
Box 118
S–221 00 LUND
Sweden

ISSN 0280–5316
ISRN LUTFD2/TFRT--1040--SE

Printed in Sweden by Lunds offset ab
Lund 1992

6

Contents

Preface

> Quia cum finis est licitus etiam media sunt licita, præcisa vi et injurio.
>
> HERMANN BUSENBAUM, *Medulla theologiæ moralis, 4, 3, dub. 7, 2, § 3, 1650.*

> Der Zweck heiligt die Mittel.
>
> J. G. BUHLE, *Lehrbuch der Geschichte der Philosophie 6, 471, 1800.*

> Nobody expects the Spanish Inquisition.
>
> MONTY PYTHON'S FLYING CIRCUS, *2, 15, 1970.*

Carelessly, we humans may think that we are in contact with the world itself, when we see things and people and feel the wind on our faces. And all the time it is a matter of models, generated to explain and foresee our own impressions and perceptions. What would you say, Prince Hamlet, about the very nature of existence? "Models, models, models."

It is rather trivial to realize that, for example, a mathematical model is not the real thing. However, it may take some insight to understand that making more correct models will not bring us closer to reality itself. Every time we think of a more "real" representation, it is still just another model in our thoughts. And even the distinction between the outer world and our mind rests on a model of reality.

The German philosopher Immanuel Kant thought that this was a problem. Thus, he came up with the idea of the thing as such. Of course, we cannot *know* anything about it, Kant admitted, but we can *believe* in its existence. Well, what Kant did was to provide us with yet another representation for the elusive reality, once again a new *model.*

In this thesis I will proudly follow in the footsteps of the good Kant, and present another type of model, MFM. This three letter acronym, (TLA), stands for *multilevel flow models,* or maybe *Morten Lind's funny models,* after the creator, (of MFM). As early as in the late 15th century, or indeed as late as in the early 16th century, the Spanish Inquisition is said to have proposed the thesis that a laudable purpose justifies intrinsically evil means, (this is probably not correct history, though).

MFM has the same general structure, but the relation between means and ends is not one of justification, but one of achieving, goes in the opposite direction, and is yet to be used for religious repression.

As an example of a means-end model: one of the author's goals has been to be awarded with a doctor's title, and a means whereby to achieve this would be the presenting of a doctor's thesis. It is the sincere hope of the author that this proposed *means-end model* will have some predictive power in the near future.

Talking about models, maybe it is a good idea to tell what the concept of "model" will mean in this thesis. It does not adhere to the common idea in control theory of assuming that a model must be mathematical, for example a set of linear differential equations on the form:

$$\dot{x} = Ax + Bu$$
$$y = Cx + Du.$$

This is *not* the only form for a model in this thesis. Instead, Minsky's definition, see Minsky (1965), comes closer to the truth:

> *A model (M) for a system (S) and an experiment (E) is anything to which E can be applied in order to answer questions about S.*

Well, there you are. This definition may seem a bit too general. For example, any system is a model of itself, if it is possible to investigate. But this generality is exactly what is needed, because MFM is entirely another beast than the differential equations. So be warned! From now on, a model is a chunk of knowledge that describes a system; a formal chunk of knowledge, but nothing more, nothing less.

And now for something completely different, a syllogism:

P_1	*No man is an island.*
P_2	*The author is a man.*
$C_{1,2}$	*The author is not an island.*

Thus, I have not been working in a mental vacuum, but owe my presumable success to many people.

First of all, I do want to give a great thanks to my supervisor, Professor Karl Johan Åström. He was the one who encouraged me to start on a research career in the first place, he had the first ideas of the expert system for Idpac, he brought me in contact with Morten Lind, and he suggested that I try to do something with MFM. He has been a formidable well of ideas, inspiration, and support, and without him this thesis would not have come into existence. He has also given me the opportunity of many useful contacts with people from all over the world.

I am very grateful to Professor Morten Lind, the originator of the multilevel flow modeling technique. His ideas and work are essential for all research in MFM, and my contributions rest firmly on that basis. He has been an invaluable source of ideas, information, and encouragement, and I have had many useful discussions with him.

I would also like to thank Doctor Karl-Erik Årzén, who has been a combined supervisor and colleague. Apart from giving invaluable advice in technical matters, he has been a good comrade, both when working, when traveling, and when drinking Duvel on late nights in different EEC countries.

A very special thanks goes to Doctor, *(sic!)*, Per Persson, with whom I spent many days in front of the DT80, looking at endless, (or rather: finite but very long), Franz Lisp traces, and with whom I hanged out in München and Swansea, as well as at Clemens. Those were the days, when hackers were real hackers, when expert systems still soared the heavens of science, and the `(ihs)` system was finally born.

I would also like to thank all the staff at the Department of Automatic Control in Lund for providing, or rather constituting, the "nice human environment" that is so important for all research. Especially, I would like to mention Professor Björn Wittenmark, whom I thank for never having to worry about my financial situation.

During my time at the department, I have had contacts with many researchers around the world; too many to list. However, I would like to mention Thomas Petti and Marcel Schoppers, who not only are nice guys, but who have helped me with many references, details, etc.

Furthermore, I would like to thank Jan Creutzfeldt, Morten Norby Larsen, and Ashraf Osman, all at the Institute of Automatic Control Systems at the Technical University of Denmark. They took good care of me on my visits in Lyngby.

During my work, I have been nicely influenced by the group of the Swedish IT4 project "Real-Time Knowledge-Based Control Systems," and I would like to thank Oddbjörn Evjen, Nick Hoggard, Börje Rosenberg, Claes Rytoft, Martin Uneram, and Anders Åberg for valuable inspiration and sometimes animated discussions.

Several other people have contributed in different ways. Mats Andersson, Bernt Nilsson, and Per Persson has given me valuable input, both concerning the MFM project and on topics of general interest.

Bernt also investigated how to take photos of the computer screen, and gracefully helped me to make dia slides for my presentations.

In addition to many other uses, a thesis can also be *read,* and I would like to thank Karl Johan Åström, Karl-Erik Årzén, and Bernt Nilsson, the actual readers of this thesis.

The more cooks, the worse the soup? Anyway, that was the recipe for the expert system for frequency response analysis, and I would like to mention the cooks that provided the main ingredients of the rather tasty knowledge acquisition soup: Jan Peter Axelsson, Ola Dahl, Kjell Gustafsson, Ulf Hagberg, Mats Lilja, Michael Lundh, Bernt Nilsson, Per Persson, Anders Wallenborg, Karl-Erik Årzén, and Ann-Britt Östberg, all of whom produced written recommendations on frequency response analysis in general and running the Solartron 1250 analyser in particular. Also, I would like to thank Sven Erik Mattsson for good advice during the project.

This thesis has been typeset on a digital (!) computer, (WYGIWYS, what you get is what you see), using the magic program TEX and some excellent local additions. For these, and for handling the computer facilities in general, I would like to thank Leif Andersson and Anders Blomdell. I would also like to mention Eva Dagnegård for her help with the design and printing of this thesis.

A typical property of human life is that a large part of it goes on in physically limited space, usually a set of rooms, each more or less important. I would like to mention my two roommates, Mats Lilja and Anders Nilsson, who not only have shared physical space with me, but mental and computational space as well, and in a very nice fashion.

Next to last but not next to least, I would like to mention my fellow graduate students at the Department of Automatic Control, for spreading good feelings, and making the local atmosphere a nice and friendly one here in Lund.

Last of all, my mother Berta, my brother Anders, and my wife Anu. They are my closest relatives, my support in everyday life, and my best friends. To them I dedicate this work.

So far about this topic. Now only the details remain.

J.E.L.

Introduction to the Thesis

This thesis uses ideas and methods from artificial intelligence, (AI), and computer science for solving control problems. The methods developed give useful solutions for the particular problems considered and they indicate the possibility and viability of such approaches in general.

AI and control theory are young sciences, and when they started they both belonged to the area of *cybernetics,* see Wiener (1948) and Ashby (1956). But since then cybernetics as a subject has broken up, and there has been academic boundaries between AI and control. On the other hand, it is clear that there is room for cross-fertilization between the fields.

A general development in control systems is to try and achieve automation on higher and higher levels, and the problems accountered here need more knowledge to be solved. Thus, in later years, there has been a renewed interest in combining AI and control. Åström *et al* (1986) gives some early ideas on the use of AI in low level control loops. Sandewall (1988) describes the need to join AI with robotics and conventional systems software, such as operating systems. Verbruggen and Åström (1989) states the usefulness of AI techniques in control. Årzén (1989, 1990, 1993) and Årzén *et al* (1990) describe the need for integration of conventional and knowledge-based techniques in control systems. Krijgsman *et al* (1990, 1991), Krijgsman and Jager (1992), Vina and Hayes-Roth (1991), and Crespo *et al* (1991, 1992) describe similar projects, all using blackboard architectures as a means to join conventional and AI techniques and to use them in real-time. The SCWERE project of Delft Technical University aims at using AI in real-time to support plant-wide supervision and diagnosis, see van den Ree *et al* (1991 a, b) and Terpstra *et al* (1991, 1992).

The IEEE and IFAC organizations have a joint project "Facing the Challenge of Computer Science in the Industrial Applications of Control," the results of which will be published at the 12th IFAC World Congress in Sydney 1993, and in a special issue of *Automatica,* and *IEEE Transactions of Automatic Control.* Some preliminary findings are given in Åström *et al* (1991), Benveniste (1991), Cohen (1991), and Åström *et al* (1993). These efforts attempt to analyze the possibilities to join control, signal processing, and computer science, and to determine what demands that must be met in the process.

Hamscher *et al* (1992) provides a good overview of current research areas, from an AI point of view. Forbus (1988) and Weld and de Kleer (1990) give overviews of the state of the art in qualitative physics. Isermann (1984) and Frank (1990) give overviews of fault detection and diagnosis in the more classical control area. Lees (1983) gives an overview of alarm and disturbance analysis from a chemical process point of view.

There are also several conferences and workshops which combine interests in AI and control problems, such as the IFAC International Workshops on Artificial Intelligence in Real-Time Control, see for example Verbruggen and Åström (1989), James and Herget (1991), MacLeod and Lun (1991), and Mõtus (1991); the IFAC Workshops on Computer Software Structures Integrating AI/KBS in Process Control, see Årzén (1991); the IFAC/IMACS Symposia on Fault Detection, Supervision, and Safety for Technical Processes; the IFAC Symposia on On-Line Fault Detection and Supervision in the Chemical Process Industries, see Frank (1992); and the International Workshops on Principles of Diagnosis, supported by ECCAI, see Struss (1991). The 8th National Conference on Artificial Intelligence, (AAAI–90), had a special part on diagnosis and control called "AI On-Line." Workshops on knowledge-based real-time control systems has also been arranged in conjunction with the AAAI conferences in 1990 and 1991. The research area of *intelligent control* investigates the possibilities of using AI methods in on-line control algorithms, see for example Saridis (1977), Antsaklis *et al* (1991), and Åström *et al* (1992). The American Control Conferences have had regular sessions on intelligent control and applications of AI to process control. The 12th IFAC World Congress to be held in Sydney 1993 will have a mini-symposium on Real-Time Computing for Control. Several of the information theory programs in Europe are also addressing problems in the area.

Real-time expert system shells such as COGSYS, G2, and RTworks have also started to appear as commercial products. Some of these are intended to be used together with conventional automation and control systems.

Intelligent Front-Ends

The user interface is an important part of automation systems and software for control system design. It strongly affects the usefulness of the system and the productivity of an engineer using a Computer-Aided Control Engineering, (CACE), program.

The first part of the thesis describes a help system based on expert system techniques. It works as an interface to Idpac, an interactive program for identification. The main features of the help system are that it is non-invasive, that it keeps track of what the user has done, and that it has procedural and diagnostic knowledge about identification.

There are several different kinds of knowledge involved in system identification using Idpac. A large part of this must be available in the expert system interface. The knowledge of the expert system interface can be divided into two groups:

- Knowledge of theory and practise of system identification. This involves knowledge about modeling, data validation, estimation methods, interpretation, validation of results, diagnosis of errors, etc. This knowledge is mostly interpretative and diagnostic in nature, and it suits well to be implemented with rules.
- Knowledge of Idpac. This involves knowledge about the methods supported by Idpac, the command language, the data representation, and several practical aspects of running Idpac. To a large part, this knowledge is concerned with command sequences, and does not fit very well for a rule-based implementation. The concept of *scripts,* see Schank and Abelson (1977) and Schank and Riesbeck (1981), was originally developed for natural language analysis. A simplified version of scripts was used to describe and interpret the command sequences of Idpac.

Several experiences were gained from the project. In order to preserve the target system's way of communication and to make the help system non-invasive, a knowledge-based help system should be designed as an *interface* to the target system. An expert system need not be implemented with rules only; in the project scripts were successfully used for describing command sequences. Some drawbacks were found in implementing a help system for an existing program. Idpac has no support for interfacing to other software, while CACE programs should be open and modular so that a help system can be successfully integrated. Ideally, they should include the intelligent help system in their design. This will also mean that some functions of current CACE programs, such as simple help systems, can be taken over by the front-end. The help system was tested in a course on process identification, and it proved to be a working and efficient solution; more efficient than the written guidelines normally used in the course.

Later efforts in developing knowledge-based help systems for identification are found in Nagy (1992), Nagy and Ljung (1989, 1991), and Szafnicki and Gentil (1991).

To interpret the command sequences of a program like Idpac is a much simpler and more well defined problem than natural language understanding, and the implemented system provides a successful solution of how to build an intelligent help system for Idpac. However, the idea of a help system using scripts and rules to track the user and give him knowledge-based help can be generally useful:

- It provides a solution for help systems for all kinds of CAD programs, and indeed for all programs that need at least reasonable amounts of knowledge to be used.
- It provides a general solution for how to use expert systems in man-machine communication, giving non-invasive help that is sensitive to what the user has done and suggestions based on what his plans may be.
- It provides a solution for supervising actions of operators of industrial processes, in order to help them in operation of the plant and warn them for erroneous and dangerous operating sequences. This task is called *plan recognition.* Here, the target system is not a CAD program with complex command sequences, but an process with complicated operating sequences. Still, exactly the same structure is useful for describing the knowledge domain.

Standard Diagnostic Expert Systems

The second part of the thesis describes a standard expert system, used as a guide for setting up, performing, and analyzing a frequency response experiment. For such a task, a rule-based system using backward chaining is very well fitted, and the resulting system provides a simple and efficient solution.

The experience gained from this project is that it is possible to increase the functionality of an instrument like a frequency analyser considerably with a moderate effort. The developed system runs stand-alone on a separate computer, while it would be an obvious advantage to integrate it in the frequency analyser itself. However, current hardware and software does not support AI tools, and the integration is problematic. Thus, future hardware and software tools such as frequency analysers should ideally accommodate for some type of knowledge-based tools also.

Once again, the results may be generalized. Rule-based expert system can be used with success whenever there is a clear diagnostic task to be performed off-line, and they have an extra strength in that they allow the use of empirical knowledge, such as experiences gathered by operators.

The approach naturally imposes a way of structuring the knowledge needed to operate complex instruments. It is likely that small, embedded expert systems will be extensively used in advanced instruments in the future.

Model-Based Diagnosis Using MFM

The third and largest part of the thesis describes a model-based approach to diagnosis. Explicit means-end models are used instead of a rule-based system. This solution clearly shows the strength of model-based methods. Instead of using a complex rule base, the implemented algorithms work on MFM graphs, which makes the database construction quicker and less error prone, and the algorithms more efficient. Several experiences were gained from the project:

- New model types are useful in control. Classical control theory is almost exclusively concerned with mathematical models, usually differential equations. New model types, such as MFM, can be successfully used to solve control systems problems; indeed they can be used to solve problems that sometimes cannot be efficiently handled with conventional techniques, such as alarm analysis, fault diagnosis, planning, and transfer of knowledge to operators.
- There is a considerable effort in introducing new model types, and much research is needed before they can be efficiently used.

- A mixture of different models can be used to solve problems which cannot be handled by one model type only. Once it is realized that different model types fit well to solve different types of problems, it is obvious that a control and supervisory system should use a whole set of different models, each for some different task or tasks that the system should be able to support.
- MFM clearly has a strong capability of describing diagnostic knowledge, which the three implemented methods show. But its use in different areas has also revealed some shortcomings. Thus, MFM provides a good basis for model-based diagnosis, but should be extended to handle, e.g., biochemical reactions and mechanical concepts such as momentum.
- Using MFM once again puts demands on the possibility of integration in the surrounding system's software and hardware.

The implemented methods all fit as building blocks in a modularized control and supervisory system. It is the intention that they should match an object-oriented system design, and this should ideally be the case for all algorithms developed for control systems.

The measurement validation method belongs on a rather low level. It takes its input data from filtered and estimated measurements, performs a consistency check, and sends validated values on to higher system levels, as well as to operators, and process engineers. The output information gives an immediate picture of the presence of sensor faults, and thus supports the operators in assessing the fault state of the process.

The alarm analysis operates on a higher level, on top of a measurement validation or alarm system, where it selects the primary alarms. The output data may be sent to other algorithms, but is primarily intended to help the operators in finding the important alarms and thus the most appropriate actions.

The fault diagnosis also operates on a high level in the control system. If run on-line, it uses data from measurement validation and alarm systems, and if run off-line also questions from the operators, and presents a description of faults and consequences. It also outputs explanations and remedies. The latter can be in the form of advice to the operator or automatic actions taken by the system.

The methods have been tested on two small examples, including one process of realistic size. They were successful in performing their tasks and quite efficient. Due to the hierarchical nature of MFM models, size is not a problem. The main limitation is that MFM can only describe mass and energy flows, and thus other effects cannot be handled. Further practical testing of the algorithms is also needed.

Knowledge-Based Methods Versus Search

The task of joining AI and control brings some difficulties with it. The academic borders between the two subjects have hampered progress. For example, control people sometimes fail to recognize that some of their problems are not handled well by classical control techniques, and the people of AI sometimes tend to pose complicated problems when simpler solutions are both more useful and easier to solve. But there are solutions that avoid both extremes.

As an example, consider the task of operations planning for a process plant. Control theory has been focussed on control loop behavior, and logic and sequencing has traditionally been handled by systems like programmable logic controllers, (PLCs), often giving *ad hoc* solutions to planning problems. Currently, new techniques are being utilized, e.g., Petri nets, see Peterson (1981), and Grafcet, see GREPA (1985) and David and Alla (1992). There is also a new interest in discrete event dynamic systems, (DEDS), see for example Ziegler (1990), Pollacia (1989), or the *IEEE Proceedings* special issue on Dynamics of Discrete Even Systems 1989, Ho (1989). Still, these attempts aim at representing plans, but not generation of new ones.

Turning to AI for help is not necessarily successful, however. AI planning usually involves defining the possible states, including initial and goal states, the possible operations, and designing a plan generation algorithm. This generally leads to an exponentially complex problem, and severe and unintuitive limitations must be put on the allowed states, operations, etc., in order to produce a viable algorithm. For an example of a formal method that often leads to very large complexity, see Ramadge and Wonham (1989).

However, there are solutions from AI that fulfill the demands of practical applicability, if the following points are satisfied:

- Existing knowledge can be used to battle the exponential complexity of general AI methods. Instead of search, knowledge-based solutions should be used. For the planning example, this implies that the knowledge that already exists should be used to produce a simpler and more efficient plan generator, for example using ready-made plans or plan fragments. The use of *scripts* in the expert system interface and the use of MFM as a database for the diagnostic methods are examples of such approaches.

- Deep knowledge should be preferred over shallow. Model-based approaches are better during development and use, because they represent the knowledge in a fashion that is closer to the designers' and operators' view of the process. In addition, the use of models often allow a crisper, more compact, and more consistent representation than a rule base. Once again, MFM provides a good example.
- The AI software must be *integrated* with the control system and the conventional control software, otherwise it will never be taken into practical use. Thus there is a need for a system architecture that can accommodate both conventional and AI techniques.

Programming Environments

One of the main results of AI over the years is the development of new programming methods, tools, and environments. Early examples are symbolic data processing, (Lisp), and time-sharing, while object-oriented programming, rule-based expert systems, and exploratory programming are more recent contributions. These tools are often powerful enough to be useful outside "pure" AI, e.g., in control.

The expert system interface was implemented in Lisp and used a standard expert system shell. This way of implementing the system was much more efficient than a conventional one would have been; the difference in effort was probably in the order of a magnitude.

The implementation of the MFM toolbox described in the third part of the thesis was done in G2. This is an expert system shell which provides hierarchical data structures, rules, sequential programming, simulation, and a graphical interface, all in a real-time environment. G2 is the state of the art for expert systems and rapid prototyping tools. The use of G2 in implementing MFM was a success. The reasoning on graph structures needed matched G2 very well, and the implementation effort has been modest.

Organization

The thesis consists of three parts, whereof two are previously published papers, and the third based on three conference papers.

- An Expert System Interface for an Identification Program
- An Expert System for Frequency Response Analysis
- Diagnostic Reasoning Strategies for Means-End Models

The first part is a paper from *Automatica,* Larsson and Persson (1991). It describes the design and implementation of an intelligent help system, `(ihs)`, based on expert system techniques. The system acts as an interface to Idpac, an interactive program for system identification. The main points of this work are knowledge-based operator help and support, plan recognition, and the general idea of a *command spy* used as an intelligent front-end. However, system identification, knowledge acquisition and knowledge engineering also form a large part of the effort. The main documentation of the project is a licentiate thesis, Larsson and Persson (1987 a). A full description of the implementation is found in Larsson and Persson (1987 b), while the complete knowledge database developed is described in Larsson and Persson (1987 c). Two other papers were also published, Larsson and Persson (1988 a, b). The beginning of the project was the author's master's thesis, Larsson (1984), see also Larsson and Åström (1986), while some earlier, intermediate efforts are described in Larsson and Persson (1986).

The second part is a paper from the 11th Triennial World Congress of IFAC, held in Tallinn 1990, see Larsson (1991 d). It treats an expert system that helps a user to perform a frequency response analysis experiment, taking him through the phases of knowledge gathering, experiment planning, range setting, performing the experiment, validation of the results, fault diagnosis, and finding remedies. The project involved knowledge acquisition and engineering by a team of graduate students, the construction of a small expert system shell with forward and backward chaining, and an implementation of a standard diagnostic expert system using backward chaining. The project has also been described in a technical report, Larsson (1990 c), while the expert system shell developed and used is described in Larsson (1988).

The third part is a monograph about three new diagnostic methods using the MFM framework. The use of MFM means a *model-based* approach to knowledge-based techniques. The main contributions are three invented and implemented methods: measurement validation, alarm analysis, and fault diagnosis. These have been described in three papers, Larsson (1991 c, 1992 a, b). The main documentation of the project is this thesis, while a more technical description is found in Larsson (1992 c).

The intelligent help system project has been supported by the National Swedish Board for Technical Development, (STU), project no. 85–3042. The MFM project has been supported by the IT4 project no. 3403 and the TFR project no. 92–956.

The gray area between computer science, AI, modeling, and control is a vast and largely untracked field. It is the intention of the author that these three projects together should form an interesting excursion into that field. Hopefully, one or two treasures may also have been brought home from the excursion, in the form of useful scientific results.

Part I

An Expert System Interface for an Identification Program

Automatica, Vol. 27, No. 6, pp. 919–930, 1991
Printed in Great Britain.

0005–1098/91 $3.00 + 0.00
Pergamon Press plc

An Expert System Interface for an Identification Program*

JAN ERIC LARSSON†‡ and PER PERSSON†

System identification requires skill and experience and the validity of the results strongly depends on the user's knowledge. An expert system used as an interface to a program for identification can give valuable help.

Key Words—Artificial intelligence; expert systems; parameter estimation; system identification; (help systems); (intelligent front-ends).

Abstract—This paper describes an expert system interface, named (ihs), for the interactive data analysis and system identification program Idpac. The interface works as an intelligent help system. The system is completely noninvasive and uses the previous command history to understand what the user is doing and gives help according to this. This way of monitoring the user's activities is called the command spy strategy. Scripts are used for representing procedural knowledge, and production rules for diagnostic knowledge. The system has been implemented and a knowledge database handling system identification with the maximum-likelihood method has been developed. An example run with the system is included.

INTRODUCTION

The work presented in this paper is a part of the Computer-Aided Control Engineering (CACE) project at the Department of Automatic Control in Lund. The CACE project aims at the development of a new generation of software tools for control engineering.

Today's CACE programs usually are quite complex and demand the user to know a lot, both about the program and the problem domain. For this reason, there is a need for help systems with knowledge about these areas. We believe that an expert system is well suited for the implementation of such a help system. It is our opinion that, in order to build a knowledge-based help system for a CACE program, the following goals should be satisfied:

- CACE programs usually have a flexible command dialogue. This way of communication should be retained when the expert system is added to the program.
- The expert system should be totally noninvasive, allowing the user to fall back on the plain CACE program in case it is not able to give the user any help.
- An inexperienced user often has a general idea of what he wants to do, but does not know exactly how to do it. The expert system should be able to guide the user from general ideas to specific commands, i.e. it should give goal-related help.
- An intelligent help system should have facilities to teach the user about the target program and to transfer knowledge from the knowledge database to the user.
- An expert system used as an intelligent help system interface must be able to trace the user's command sequences, and to give help with the interpretation of results. Thus it must be able to handle both knowledge about sequences and diagnostic knowledge.
- Existing software for control engineering contains much reliable knowledge. Therefore it is an obvious advantage to use an already developed program such as Idpac or Matlab, over producing new algorithms.

According to these design criteria an expert system interface has been developed for the command driven system identification package Idpac, Wieslander (1980). The system contains a command parser, a script-matching device with a database for scripts and rules, a query module, a file system, an on-line dictionary, and interfaces to the user and Idpac. A script is a data structure for representing command sequences.

The project has been focussed on a noninvasive, goal-related help system for Idpac, not on developing new forms of man–machine interaction. Therefore, we were satisfied with using the Idpac command language, instead of a window-and-mouse type of interaction.

* Received 14 June 1989; revised 18 July 1990; revised 2 January 1991; received in final form 4 March 1991. The original version of this paper was not presented at any IFAC meeting. This paper was recommended for publication in revised form by Editor H. Austin Spang III.

† Department of Automatic Control, Lund Institute of Technology, Box 118, S-221 00 Lund, Sweden.

‡ Author to whom all correspondence should be addressed.

This project was originally outlined in Larsson (1984). A previous system was described in Larsson and Persson (1986). Thorough descriptions of this project are given in Larsson and Persson (1987a,b,c,d).

OVERVIEW OF RELATED WORK

Other work has been done in this and related areas. System identification is described in Cox (1958), Eykhoff (1974; 1981), Fedorov (1972), Ljung (1987a), Ljung and Söderström (1983), Åström and Eykhoff (1971) and Åström (1980). Idpac is built with the interaction module Intrac, a framework that provides an interactive environment for numerical Fortran routines. Idpac and Intrac were developed at the Department of Automatic Control, Lund Institute of Technology, Wieslander (1979a,b; 1980). More about Idpac can be found in Gustavsson (1979) and Gustavsson and Nilsson (1979).

Other projects address the problem of using expert system techniques in system identification: the French project SEXI (Gentil and Conraux, 1985); another French project (Monsion *et al.,* 1988) and the Belgian ESPION (Haest *et al.,* 1988).

The program in the project SEXI is implemented in Lelisp, contains the rule-based expert system shell CRIQUET, and uses a numerical identification program. Instead of working as a help system, the expert system is run in a "batch" mode, taking over the complete responsibility of acquiring a reasonable model. In this way, the need for good communication with the user is avoided. On the other hand, the system can no longer "fall back" on the user if anything goes wrong, and the system's ability to teach identification to the user is probably impaired. When identifying, the SEXI system performs a heuristic search through different possible model structures, increasing and decreasing the order of nominator and denominator until the best model is found. The benefits of this search procedure are not altogether obvious. The project of Monsion *et al.* is in a similar vein.

The Belgian project ESPION is an expert system for identifying multiple input, single output models from industrial data. The system is implemented with the expert system shell OPS 83 and the identification package SYSID, developed at Université Catholique de Louvain. The system takes ASCII data sets as inputs and generates models. The system organizes an intelligent search through the model space and performs identifications of different model structures. The best model is picked according to a quality index, which consists of several standard model validation methods. Like SEXI, the system runs in batch mode and does not teach the user anything about identification. The system is reported to generate as good results as a human expert, but in much shorter time.

Another project which focuses on general help systems is the Esprit project EUROHELP (see Skogø Hansen *et al.,* 1988; Breuker *et al.,* 1989). The project is carried out by a consortium consisting of Computer Resources International (Denmark), ICL (UK), Courseware Europe (Netherlands), the University of Amsterdam (Netherlands) and the University of Leeds (UK). The size of the project is 120 man-years. The EUROHELP projects aims at constructing a shell which will be located between the user and the application program. By constantly monitoring the user's dialogue with the target program, the shell will be able to correct the user and to give him advice on the correct usage of the target program. The plans necessary for describing an Idpac session would be rather long, and many facts would have to be gathered and used during a session. It is not clear whether the EUROHELP shell is well fitted for representing this. If it is, EUROHELP might be an alternative tool for a future implementation. Furthermore, a set of tools enabling the development of an application model and allowing the shell to be refitted to certain hardware dependencies, and · a methodology describing the usage of these tools in an intelligent help system will be developed. Applications for the help system can be text editors, spreadsheets, database management systems, etc. A prototype of the system has been developed on a Xerox Lisp machine, but the final result of the project is intended for target machines running Unix.

Two further projects addressing similar problems are the Unix Consultant and the Knowledge-Based Emacs. The Unix Consultant is an intelligent help system for Unix. It reads questions in natural language and uses scripts to tell the user how to perform different tasks, Wilensky *et al.* (1986). Knowledge-Based Emacs is an AI system built into an editor. It allows the user to work on a higher level of programming than the ordinary Lisp or Ada level, Waters (1985a,b).

KNOWLEDGE REPRESENTATION

There are several different kinds of knowledge involved in system identification using Idpac. Ideally, most of this knowledge would be present in the knowledge database of the expert interface. The knowledge can be divided into the following groups.

X ej med

This project was originally outlined in Larsson (1984). A previous system was described in Larsson and Persson (1986). Thorough descriptions of this project are given in Larsson and Persson (1987a,b,c,d).

OVERVIEW OF RELATED WORK

Other work has been done in this and related areas. System identification is described in Cox (1958), Eykhoff (1974; 1981), Fedorov (1972), Ljung (1987a), Ljung and Söderström (1983), Åström and Eykhoff (1971) and Åström (1980). Idpac is built with the interaction module Intrac, a framework that provides an interactive environment for numerical Fortran routines. Idpac and Intrac were developed at the Department of Automatic Control, Lund Institute of Technology, Wieslander (1979a,b; 1980). More about Idpac can be found in Gustavsson (1979) and Gustavsson and Nilsson (1979).

Other projects address the problem of using expert system techniques in system identification: the French project SEXI (Gentil and Conraux, 1985); another French project (Monsion *et al.,* 1988) and the Belgian ESPION (Haest *et al.*, 1988).

The program in the project SEXI is implemented in Lelisp, contains the rule-based expert system shell CRIQUET, and uses a numerical identification program. Instead of working as a help system, the expert system is run in a "batch" mode, taking over the complete responsibility of acquiring a reasonable model. In this way, the need for good communication with the user is avoided. On the other hand, the system can no longer "fall back" on the user if anything goes wrong, and the system's ability to teach identification to the user is probably impaired. When identifying, the SEXI system performs a heuristic search through different possible model structures, increasing and decreasing the order of nominator and denominator until the best model is found. The benefits of this search procedure are not altogether obvious. The project of Monsion *et al.* is in a similar vein.

The Belgian project ESPION is an expert system for identifying multiple input, single output models from industrial data. The system is implemented with the expert system shell OPS 83 and the identification package SYSID, developed at Université Catholique de Louvain. The system takes ASCII data sets as inputs and generates models. The system organizes an intelligent search through the model space and performs identifications of different model structures. The best model is picked according to a quality index, which consists of several standard model validation methods. Like SEXI, the system runs in batch mode and does not teach the user anything about identification. The system is reported to generate as good results as a human expert, but in much shorter time.

Another project which focuses on general help systems is the Esprit project EUROHELP (see Skogø Hansen *et al.*, 1988; Breuker *et al.*, 1989). The project is carried out by a consortium consisting of Computer Resources International (Denmark), ICL (UK), Courseware Europe (Netherlands), the University of Amsterdam (Netherlands) and the University of Leeds (UK). The size of the project is 120 man-years. The EUROHELP projects aims at constructing a shell which will be located between the user and the application program. By constantly monitoring the user's dialogue with the target program, the shell will be able to correct the user and to give him advice on the correct usage of the target program. The plans necessary for describing an Idpac session would be rather long, and many facts would have to be gathered and used during a session. It is not clear whether the EUROHELP shell is well fitted for representing this. If it is, EUROHELP might be an alternative tool for a future implementation. Furthermore, a set of tools enabling the development of an application model and allowing the shell to be refitted to certain hardware dependencies, and a methodology describing the usage of these tools in an intelligent help system will be developed. Applications for the help system can be text editors, spreadsheets, database management systems, etc. A prototype of the system has been developed on a Xerox Lisp machine, but the final result of the project is intended for target machines running Unix.

Two further projects addressing similar problems are the Unix Consultant and the Knowledge-Based Emacs. The Unix Consultant is an intelligent help system for Unix. It reads questions in natural language and uses scripts to tell the user how to perform different tasks, Wilensky *et al.* (1986). Knowledge-Based Emacs is an AI system built into an editor. It allows the user to work on a higher level of programming than the ordinary Lisp or Ada level, Waters (1985a,b).

KNOWLEDGE REPRESENTATION

There are several different kinds of knowledge involved in system identification using Idpac. Ideally, most of this knowledge would be present in the knowledge database of the expert interface. The knowledge can be divided into the following groups.

- Knowledge of the theory and methods of system identification, i.e. knowledge about modeling, data validation, estimation methods, interpretation and validation of results, etc.
- Knowledge about Idpac, i.e. knowledge about the methods supported by Idpac, the Idpac command language, Idpac's data representation, and a lot of practical aspects of running Idpac, such as naming conventions for data files, setting internal parameters right, and so on.
- Knowledge of the process to be identified, i.e., knowledge about the real world process and the experiment design. For a certain process it might be known whether it is fairly linear or very nonlinear, its time-scales, reasonable limits of the model order, and so on. There may also be *a priori* demands on the results, e.g. a limit of the complexity of the model, that the best second-order model is desired, etc.

The implemented system tries to work as an expert in system identification and as an Idpac expert. The user's *a priori* knowledge of the process is taken care of by a group of rules in the beginning of the session. If needed, these rules could be extended for new types of processes.

Running Idpac requires a lot of procedural knowledge, i.e. knowledge about sequences of commands, and, on a higher level, knowledge of sequences of subtasks of Idpac sessions. In order to represent sequences, the concept of scripts was introduced. We were inspired by a data structure used in natural language understanding (Shank and Abelson, 1977; Shank and Riesbeck, 1981). It should be noticed that our script concept is different from Shank's.

Of course, it would be possible to handle the knowledge of command sequences with production rules. One way of doing this would be to implement a finite state machine with rules. The current state would be represented as a fact in the database, and each transition described by a rule, where a transition would correspond to a command given to Idpac. Very soon this approach becomes quite complex. It is very hard to add new commands to the command sequences, and it is also very hard to read and understand existing rules.

The scripts are implemented as Lisp lists. The system matches the incoming commands with the scripts in its database, and updates them incrementally. The commands must have certain attributes, e.g. parameters, in order to match a script. Implementing scripts with lists makes them easy to understand, write, and modify. Here is an example of a script.

```
((command plot
  (infile INSI) (infile OUTSI))
 (repeat
  ((command mlid (outfile SYST) (infile INSI)
                 (infile OUTSI) (number N))
   (kscall
    (Estimation of order (parameter N) performed))
   (or
    ((command residu (outfile RES) (infile SYST)
                     (infile INSI) (infile OUTSI)))
    ((command sptrf (outfile FREQ) (infile SYST))
     (kscall
      (Recommend Bode plot with (parameter FREQ)))
     (command bode (infile FREQ))))))
(kscall (Give advice on most probable order))).
```

This script describes a way of performing a parameter estimation with increasing order of the fitted model. First the user should inspect the signals with the PLOT command. The input files corresponding to INSI and OUTSI should already exist. A model is produced with MLID. The output file SYST is created and N, the order of the model, is associated with the actual argument used. Next, a fact stating that a parameter estimation has taken place is sent to the production rule database. This is done with the KSCALL clause. Kscall stands for Knowledge Source Call and puts a fact in the fact database of the production rule system, which is associated with each script, and activates the inference engine. After this, the user may either look at the residuals with the RESIDU command, or produce data for a Bode plot of the transfer function, with SPTRF and plot it with BODE. This is expressed with the OR clause. The REPEAT clause means that this whole procedure may be repeated, and every time the rule system gives advice on whether the order is sufficiently high. During the process it may use facts put into the database by previous kscalls. This script is of course far too small to be realistic, but it shows what a script may look like. For a script of reasonable size, see Larsson and Persson (1987c).

Other clauses in the script language are the ALL clause, which expresses that all the following commands must match, but the order of them is not essential, and the SCRIPTMACRO clause, which is a subroutine facility.

OVERVIEW OF THE SYSTEM

How does one combine a CACE program with an expert system while keeping the good features of both? The solution proposed in this paper is the command spy strategy, see Fig. 1.

The expert system is used as an interface to

Idpac. In our solution it is placed before the command decoder of Idpac, but an alternative would be to build it into the outermost level of Idpac.

We keep the command language of Idpac. In this way a question and answer (Q/A) dialog is avoided. The expert system traces the user without asking any questions, and gives help only on demand. Thus the expert system never forces the user to do anything. A user that does not need or want any help is not bothered and one can still use Idpac in case the help system does not have enough knowledge to work properly. The expert system may also use a Q/A dialog to find out facts about the experiment and expected results, but only if the user initiates it.

The command spy uses scripts in order to understand what the user is doing. By matching scripts against the actual command history, the expert interface is able to guess what the user wants to do. The scripts also provide information on the next sensible step for reaching a desired goal (Larsson and Persson, 1987a). The knowledge database may contain several scripts, each designed to take care of a special task. There may be scripts for maximum-likelihood estimation, frequency analysis, correlation analysis, etc. Several scripts may be active at the same time, as long as they match the commands typed by the user. This is typically the case when the system is started.

Diagnostic reasoning is needed at certain points in an identification, notably when a result is difficult to interpret, a method did not give an appropriate result, or when there is a choice of what to do next. Based on what the user has done earlier, the results may have to be interpreted in different ways. This taken care of by production rules associated with each script. The rules also allow for automatic documentation by writing script based information to a text file.

The user can get different kinds of help. Typing "?" asks for advice on what to do next. The system will give comments on what has happened so far and suggest what to do next, usually together with some motivation, especially when alternatives are available. It will also give advice on how to interpret the results of commands, e.g. Bode plots. When the system is allowed to ask questions, the user may be asked to pick out information from results, e.g. a frequency range in a Bode plot. The user may enter and exit the "questions allowed" mode at will. Via the query mode, a context-sensitive defaulting facility, the user will be helped with remembering trivial things, such as file names, previous parameter and result values, lengths of data series, etc. This has proved very useful and we call it the "on-line note book".

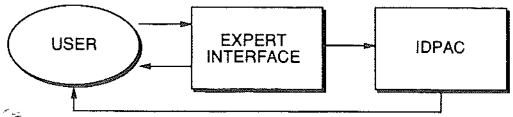


FIG. 1. The expert system is used as an interface to Idpac.

At any time the user may ask for the next sensible command. The command spy then looks at the next possible commands in all the active scripts, and gives these commands as the answer. If, however, the user does not follow this advice, some error recovery actions are taken. One possibility is to assume that the user wants to start all over again, so the initial scripts are tried once more. If one of them should match, the current script gets suspended and the new script becomes the current script. If the user's command does not match any of the initial scripts, the command spy checks whether the command matches any previously suspended scripts. If so, the current script gets suspended and the script which matched the command becomes the current script. If none of this is the case, the command spy stays in its current script, sends the (for the command spy unintelligible) command to Idpac, and from the next command on it tries to restart. In this way the user can issue a few "off-the-track" commands for his own purposes, and then return to the original script.

When several scripts are active and the user demands help, information from all active scripts is displayed. The help is tagged so that the user can see to which script a certain help message belongs.

System identification theory and the intelligent help system use many technical terms. The user may ask about such terms via the on-line dictionary. This is a simple "standard" help system, which reads a word and writes a small explanation of it. Ideally, the help texts produced by the on-line dictionary should vary according to the script context. It would also be useful if more complex questions could be handled. No such things have been implemented, however.

Architecture of the system

The command grammar describes the command language of Idpac. The knowledge database is composed of scripts and rules, and describes the domain knowledge of the Idpac

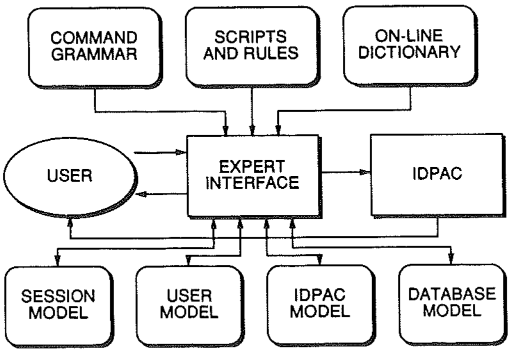


FIG. 2. The architecture of the intelligent help system.

experts. The on-line dictionary contains explanations of the vocabularies of system identification and Idpac. These databases are the sources of the help system's knowledge.

The intelligent help system also uses several models by which it tracks the user and Idpac (Fig. 2). The session model keeps track of the session, i.e. what commands have been performed, what method is currently utilized, what is known about the process, etc. This model is implemented with scripts and rules.

This system also has a model of the user. This model is currently very naive and only characterizes the user as being in one of two states, expert or beginner. In expert mode the system is totally passive and gives help on demand only. In beginner mode the system always gives as much help as possible, and it is allowed to ask the user questions, to be able to give better help. The user state is not changed automatically. The user himself decides in what state he is, i.e. how good he is.

Idpac has many internal state variables, and thus the expert interface must have a model of Idpac. Almost all of Idpac's states change due to certain commands. The help system checks whether a command affects any internal Idpac state, and, if so, updates the Idpac model accordingly. This is done at the end of the query phase.

The expert interface also has a model of the database of files used by Idpac. Whenever a file is created or deleted, this model is updated. This file model also tracks the dependencies between files. This information would otherwise be lost, as Idpac does not take care of it.

IMPLEMENTATION OF THE SYSTEM

The expert interface is made up from several independent parts (see Fig. 3). Most of the parts work on a common database.

The user interface module reads a command from the user and transforms it into a Lisp list. It provides all input and output functions used in other parts of the interface. In this way, all of the system's dependence on terminal types, graphics, etc. is collected in one place.

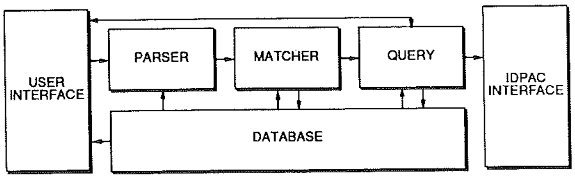


FIG. 3. Layout of the system.

The command parser module checks the command for syntax and supplies defaults in the same way that the parser of Idpac does. Of course this means that parsing of a command will take place twice, once in the interface and once in Idpac. This is inefficient, but there is not much we can do since Idpac is a closed program. In this process, the parser transforms the command into a more convenient form. The parser accepts commands with the arguments left out, as the other routines will fill information in, by defaulting from scripts and asking the user.

The script matcher module keeps track of the scripts incrementally and updates them as it gets commands from the parser. The commands once again are transformed, and files may be defaulted using knowledge from the scripts, i.e. filenames given earlier in the session are filled in. This is the only way that filenames are defaulted. New filenames are not created automatically.

The query module goes through the command description and tries to fill in the remaining unknown entries by asking the user about them. In this way the user may give only the command name. Some parameters will be filled in from the scripts and, if necessary, the user will then be prompted for any parameters still unknown. The query module also sends messages to the file system about created files.

The database contains the command grammar used by the parser, the scripts and rules used by the script matcher, the file tree of the file system, and state variables for keeping track of the user state, setting a debug mode, etc.

The file system keeps track of all the data files created and used during an Idpac session. It does this by storing data about the files in a directed graph structure. This enables the file system to show, e.g. the "ancestors" or "descendants" of a file.

Software tools

The expert interface and Idpac reside in two different VMS processes. The Idpac interface sends the processed command to Idpac via a VMS mailbox. In this way no changes had to be done to the Idpac program itself. The interprocess communication routines are written in C.

The system is written in Franz Lisp (Foderaro and Sklower, 1981), extended with Flavors (Allen *et al.*, 1984), and YAPS (Allen, 1983). Flavors is a system for object oriented programming added to Franz Lisp. YAPS is an expert system shell, written in Flavors. The expert system interface consists of about 8100 lines of code and runs under VMS on a VAX 11/780. The system is documented and the complete source code given in Larsson and Persson (1987b). The time of handling one command in `(ihs)` is typically 1–5 seconds, which is about the same as for most of Idpac's commands.

The choice of Idpac as the target program was made in 1985. More recently, Matlab has become very popular, and an identification toolbox has been developed for it (Moler *et al.*, 1987; Ljung, 1987b). Had this been available when the project started, we would probably have chosen it instead of the more old-fashioned Idpac. Also, a workstation environment would have been preferred over the VAX implementation. Matlab and the identification toolbox have been used in another tool for intelligent system identification (see Nagy and Ljung, 1989).

THE KNOWLEDGE DATABASE

One major knowledge database has been constructed during the project. It deals with parameter estimation using the maximum-likelihood method, and thus only covers a part of everything that Idpac may be used for. Still, we believe that this database shows that our solution will indeed work. The system is used for demonstrations and has also been used in an undergraduate level course on system identification. The database is thoroughly described in Larsson and Persson (1987c). The full script is about 330 lines long, and the rule base contains some 135 rules. This makes up somewhat more than 28 pages of code. The database handles the interactive session reasonably well. It was written by the authors during three months in the summer of 1987.

We have learned that it takes quite some time to build even a relatively small database. Knowledge acquisition, converting knowledge into scripts and rules, and testing is a complicated and lengthy process. Thus, the construction of a complete knowledge databse for Idpac is certainly a major effort.

Idpac uses sampled input and output signals to find the parameters of the A, B and C shift operator polynomials of a difference equation of the following form.

$$Ay = Bu + Ce.$$

Here, u and y are the input and output signals, and e a white noise disturbance. All of this is well described in Eykhoff (1974), Ljung and Söderström (1983), Åström (1980), and others.

The script

The method for identifying a system, as described in this section, has been compiled into the script and rules database, which is used in the expert system interface.

It is assumed that the measured data is available in two separate ASCII text files. These files are first converted to Idpac's internal binary format, and the binary files are examined with the command `STAT`. This command displays some statistics of a data file, e.g. the mean value, the variance and the length of the file. The files are then plotted to see if the signals are reasonable. Now it is possible for the user to change data points which are obviously wrong with the command `PLMAG`. The script guides the user through the subcommands of `PLMAG`.

When this is done, both files are plotted in the same diagram to get a feeling for the interaction between the signals. If the data files have been sampled with too high a frequency there is now a possibility to resample the signals internally in Idpac with the command `PICK`.

If the files are long enough they may be cut in two parts; one for identification and the other for cross-validation. If the user intends to make a cross-validation, it is assumed that he will use the first half of the signals for the first part of the cross-validation and the second half for the second part. A natural choice is to cut the signals in two equal halves, but if the system is affected by some unknown disturbance it may be wise to cut out pieces of the signals which are not affected by it. After the files are cut, their trends are removed with the command `TREND`, and after this the files are plotted again to see that the new signals look reasonable.

Before the actual parameter estimation, the user has the option to carry out some tests on the signals. He may compute the coherence between the input and output signal, and the autospectrum of the input signal, and display the result in Bode plots. The autospectrum is computed to see if the input signal contains sufficient energy for excitation of the system. These computations are done in order to get an estimation of a frequency range in which the model will be reliable. It is recommended that the coherence between input and output signal should not be less than 0.7 in the frequency range in which a purely deterministic and linear model should be reliable. The user can also prewhiten the input and output signals and compute the cross-correlation of these to get an estimation of the impulse response, from which it may be possible to detect a time delay. If a time delay is detected, it is possible to slide the signals relative to each other with the macro SLID, and then once again compute the prewhitened signals and the cross correlation.

After this, parameters of models of different orders are estimated and the residuals are examined. The expert system keeps information of the models' AIC values (Akaike, 1972), continuously computes the minimum value and keeps track of the corresponding model.

After a successful identification we should get "white" residuals, i.e. residuals with zero autocorrelation except for $\tau = 0$. It can also be useful to calculate the cross-correlation between the residuals and the input. A successful identification should give zero cross-correlation for positive lags. The presence of feedback in the experiment is seen from correlation at negative lags. The actual residuals are also plotted in order to detect outliers.

It is also recommended to compute the frequency responses of the estimated models and display them in a Bode plot. When the curves coincide well in regions with high coherence, this is a sign that the order of the transfer function is sufficiently high. The script handles computation and plotting of frequency responses of models relating input signal to output signal and from noise to output signal.

When a sufficiently high model order, according to the AIC test and the Bode plots, has been found, it is time for the cross-validation. The cross-validation consists of two parts. The first part starts by removing the trends of the complete input and output signal. After that, the residuals of these signals are computed using the models estimated earlier and the loss function of the residual files is computed using parts of the residual files not used in the parameter estimation of the models. In the second part of the cross-validation the second half of the signals are cut out, and trends are removed from these signals. The rule system suggests the use of the same order of trend polynomial as used on the first half. Then models of the same orders as in the first part of the cross-validation are estimated on the second half. Residuals and loss functions are computed. The loss functions are computed using the last 80% of the residual files in order to avoid initial transients. The models generated from the different parts of the data sets should agree in order for the cross-validation to be successful.

During the cross-validation, the values of the loss functions and their corresponding models are stored as facts. Actually, the rule system also computes a set, containing the model with minimal loss function and all models with loss functions less than

$$minimum\_value + 0.05 * |minimum\_value|.$$

Using these sets, the rule system chooses the best model. For details, see Larsson and Persson (1987c).

After the parameter estimation, some tests are carried out on the model chosen. If a covariance matrix has been stored during the parameter estimation, it is possible to compute a random distribution set of models based on these statistical measures, and compute step responses from the models. A plot with a number of step responses is shown as a result of the macro RANDSTEP, and the user is asked if it looks reasonable.

Another possibility is to compute a random distribution of the frequency response of the different models, and plot them in a Bode diagram to see if they coincide. This is done with the macro RANDTF.

The third, and perhaps best, test is to compute the output signal from the model using the input signal and then plot it together with the real output signal, and see if they look reasonably similar.

At last the chosen model is listed and if the leading B-coefficients are very small the user can now fix them to zero and make a new parameter estimation. This part of the script could have been more elaborated.

Most of the motivations presented in this section is available in rule form, and these rules are triggered from the script during the session.

The rule database

Some rules are useful in all scripts. They might, e.g. be rules for generating output, automatic documentation, etc. Therefore a list of global rules was introduced. These rules are added to all scripts at startup time.

Throughout the session with the system the user gets a lot of information from the expert system, and is also occasionally asked questions. This happens when the expert system needs information which is displayed on the screen and not written to any file, or when it needs information, which is a result of the user's judgement, e.g. "In which region does the plot look reasonable?"

Currently the rule base contains only a small part of all knowledge needed in identification. A lot of rules must be added to take care of all special cases that may arise. For example, the system assumes that the user wants do to a cross-validation if he cuts the signals. However, it may be the case that he cuts them in such a way that it is impossible to carry out a parameter estimation on the remaining parts, after the first `CUT`. Much more of this safety net remains to be implemented in the rules.

AN EXAMPLE RUN

As an example of a session with the system, the slightly edited output from a terminal is given in Fig. 4. The example shows the identification of the dynamics of a ship, using the script presented in Section 7. The data was gathered during an experiment with the freighter "Atlantic Song" on Sunday, 21 December 1969, off the west coast of Denmark. The complete session can be found in Larsson and Persson (1987c). In this example the system is run in beginner mode, which automatically outputs the help texts, without the user having to ask for them. In beginner mode, the system is allowed to ask the user questions.

The input provided by the user is shown in *italics,* and the output from the system is shown in `typewriter font`. All information starting with `ML:` comes from `(ihs)`, and indicates that we are following the maximum likelihood script. The `IHS⟩` prompt comes from the user-interface module, while the rest of the output is from Idpac.

INTEGRATING THE INTERFACE WITH A CACE PROGRAM

During the entire project, there has been a focus on ideas and general facilities for help systems and we have tried to avoid Idpac specific things as far as possible. This means, among other things, that most of the ideas in the system would, hopefully, work with other programs also, e.g. Matlab and the identification toolbox.

The Idpac language is somewhat old-fashioned and has several shortcomings. Some commands have a syntax different from the usual one and there is quite a lot of special cases, some of which have the typical look of "programming tricks." Part of an explanation is that Idpac has been rebuilt and added to in several steps. It would be much easier to build an expert interface for Idpac if some changes were made in Idpac itself.

Idpac is designed as a stand-alone program with a top level that is only useful for communication with a user at a terminal. There are no facilities that enable another program to get values of parameters, results or error messages from Idpac. The only way that the expert interface can get such data is either to analyse the resulting data files or to ask the user. Neither of these alternatives is really satisfying. Currently, the expert interface cannot even find out if Idpac has had an error without asking the user. If a user specifies a column number in a file as being say 10 000, the interface just sees an integer and accepts it, but Idpac will not. It is very difficult to make sure that the interface will never send anything that would cause an error in Idpac. Therefore a help system must have the possibility to check for errors in the target program. This could be done by supplying functions in Idpac whereby values could be sent to another program, making it easy to transfer results to the expert interface. This would have the beneficial effect of heavily reducing the number of questions asked by the help system. Most questions concern results of different operations, and these could instead be read into the expert interface automatically.

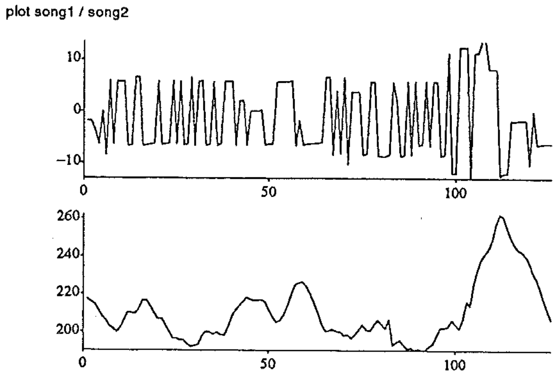
When the expert interface is to be built into a CACE program, there may be reasons for implementing it in some other language than Lisp, e.g. Fortran, C or Pascal. We believe that once it is known what features should be available, implementing an expert interface in C, say, is no problem. It will not be very easy to make large changes in the code, and the size of it will grow considerably. However, if the CACE program would use Lisp as its interactive user interface, the incorporation of an expert interface would be very straightforward. At the present stage it would be a big mistake to leave the Lisp environment.

CONCLUSIONS

The main conclusion of the project is that if one wants to combine an expert system with a CACE program, a good possibility is to use the expert system as an intelligent and noninvasive help system. This retains the advantages of both the CACE program and the expert system. It may be accomplished by implementing a command spy, as outlined in this paper. Secondly, not all knowledge need be implemented with production rules. Scripts may be a better way of representing sequences, especially in problems where both methods and goals are well known. Scripts are much easier to read and understand than a rule representation of sequences. A good rule is to use as much as

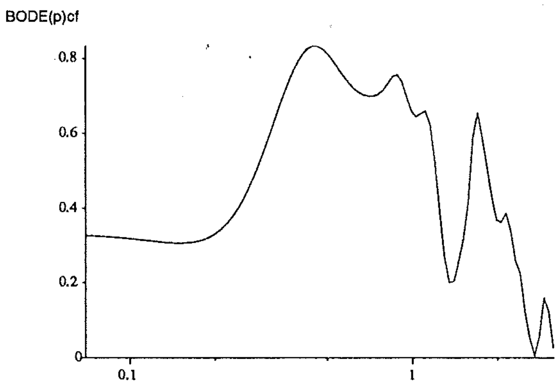
```
ML: This script assumes that the input signal is
available in ASCII format, use CONV to convert it
to binary format.
ML: conv
IHS> conv song1 < song1 1
ML: stat
IHS> stat

 MEAN     =  -1.14047611
 VARIANCE =   49.3786011
 LENGTH   =    126

What is the the length of the file song1? > 126
ML: Set appropriate plot width with LET NPLX. = 126.
IHS> let nplx. = 126
ML:  This script assumes that the output signal is
available in ASCII format, use CONV to convert it
to binary format.
ML: conv
IHS> conv song2 < song2 1
ML: The length of the file song2 is unknown. This
may lead to trouble later on.
ML: stat
IHS> stat

 MEAN     =   210.736511
 VARIANCE =   266.270721
 LENGTH   =    126

What is the the length of the file song2? > 126
ML: Plot the files song1 and song2 with one PLOT
command. GIVE THE COMMAND PLOT song1 / song2.
ML: plot
IHS> plot song1 / song2
```

plot song1 / song2



```
ML: Select two regions. If you can't find any
good choice, just take the first and second half
of the data files. You must CUT out two regions
if you want to do cross validation later.
   Use CUT to cut out the first half of the input
signal in the file song1. Use first record = 1
and number of records = 63.
ML: cut
IHS> cut 1 90
cut ... < song1 1 90
resultfile? > song1c
ML: The last file you CUT was song1 , the starting
column 1 and the record length 90. Use CUT to cut
out the first half of the output signal in the
file song2.
ML: cut
IHS> cut song2c 1 90
ML: Use the TREND command to remove any trends in
song1c.
ML: trend
IHS> trend song1ct < song1c 0
ML: The last trend polynomial you removed was of
order 0. Use the TREND command to remove any
trends in song2c.
ML: trend
IHS> trend song2ct < song2c 0
ML: Check that it is possible to get reasonable
results by making a coherence test. The coherence
must be larger than ~ 0.7 if you are going to use
the results in a parameter estimation and you use
one input signal and expect a deterministic
linear model. Use number of lags = 20 - 25% of
the number of data points.
   Parameter estimation may be started. Use the
MLID command to fit models of INCREASING order.
Look at the parameters and the estimated
uncertainties. A minimal AIC is a sign that the
model order is high enough.
ML: coh, mlid
IHS> stat song1ct (The user diverts from the script)

 MEAN     = -5.671668798E-02
 VARIANCE =   35.8573837
 LENGTH   =      90

ML: coh, mlid
IHS> coh (The user resumes the script)
coh ... < song1ct song2ct ...
coherence outfile? > cf
coh cf < song1ct song2ct ...
number of lags? > 25
```

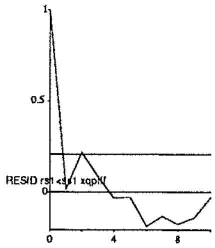
BODE(p)cf



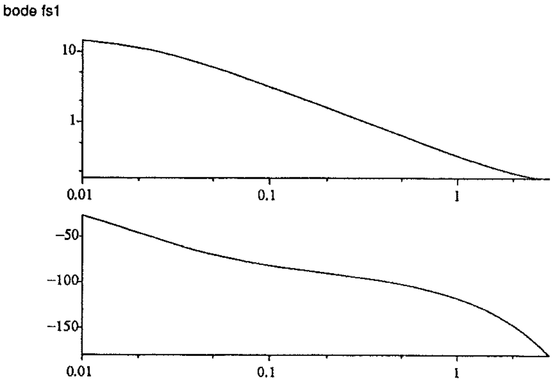
```
Lower limit of interval where the coherence is
greater than ~0.7 ? > 0.5
Upper limit of interval where the coherence is
greater than ~0.7 ? > 1.0
   Parameter estimation may be started. Use the
MLID command to fit models of INCREASING order.
Look at the parameters and the estimated
uncertainties. A minimal AIC is a sign that the
model order is high enough.
ML: mlid
IHS> mlid ss1 1

 A1            -0.980296     +-   3.511994E-02
 B1             0.316539     +-   5.112291E-02
 C1             0.107419     +-   0.101487

 LOSS FUNCTION   292.482
 AIC             429.866
```

FIG. 4. An example session.

```
ML: You have identified the model ss1 with order 1
aic = 429.87 and loss function = 292.48.
ML: residu
IHS> residu
residu ... < ss1 song1ct song2ct
residuals outfile? > rs1
```

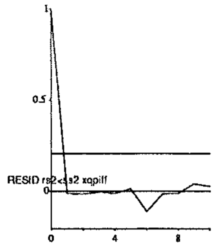


```
ML: If the order is high enough, the residuals
will be white, i.e., their autocorrelation close
to zero except at tau = zero. Note that small
residuals due to quantization will not be white.
   Use SPTRF ... < ss1 B / A to produce a
frequency response file from ss1. You have
identified the model ss1 with order 1 aic =
429.87 and loss function = 292.48.
ML: mlid, sptrf
IHS> sptrf fs1 < ss1 b / a
ML: Plot the frequency responses for all systems
in one diagram. Use the command BODE fs1. The
0.7 limits of the insignal and outsignal which
you have identified from (song1ct song2ct) are
0.50 to 1.00.  This is the interesting region in
the Bode diagram.
ML: bode
IHS> bode fs1
```

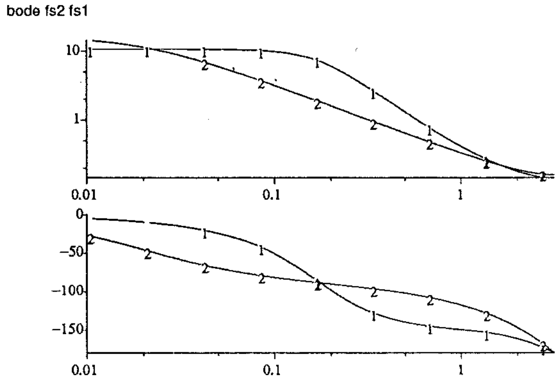


```
ML: You have identified the model ss1 with order
1 aic = 429.87 and loss function = 292.48. You
may perform another estimation with a higher
model order.
ML: mlid, sptrf
IHS> mlid ss2 2

 A1              -1.75515     +-   5.400475E-02
 A2               0.780311    +-   5.314677E-02
 B1               0.386883    +-   3.848762E-02
 B2              -0.119482    +-   5.575246E-02
 C1              -1.03639     +-   0.123959
 C2               0.180254    +-   0.111139

 LOSS FUNCTION    182.110
 AIC              393.224
```

```
ML: residu
IHS> residu rs2
```



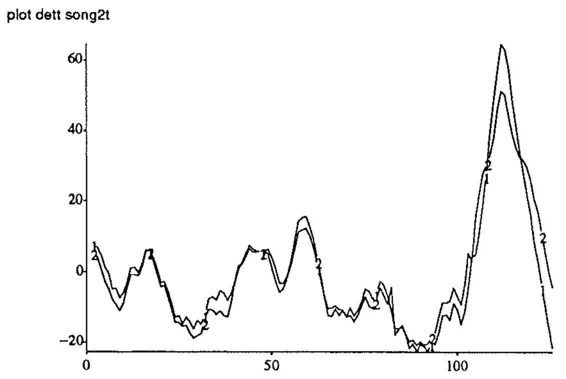
```
ML: If the order is high enough, the residuals
will be white, i.e., their autocorrelation close
to zero except at tau = zero. Note that small
residuals due to quantization will not be white.
Use SPTRF ... < ss2 B / A to produce a frequency
response file from ss2.
   You have identified the model ss2 with order 2
aic = 393.22 and loss function = 182.11 and the
model ss1 with order 1 aic = 429.87 and loss
function = 292.48.
ML: mlid, sptrf
IHS> sptrf fs2 < ss2 b / a
ML: Plot the frequency responses for all systems
in one diagram. Use the command BODE fs2 fs1.
The 0.7 limits of the insignal and outsignal
which you have identified from (song1ct song2ct)
are 0.50 to 1.00. This is the interesting region
in the Bode diagram.
ML: bode
IHS> bode fs2 fs1
```



```
ML: You have identified the model ss2 with order
2 aic = 393.22 and loss function = 182.11. You
have identified the model ss1 with order 1 aic =
429.87 and loss function = 292.48. You may
perform another estimation with a higher model
order. Probably the best model is that in the
file ss2, according to the AIC.
The length of song1 is 126. You must simulate 126
points in DETER.
ML: mlid, sptrf, deter
IHS> deter 126
deter ... < ss2 song1 126
outfile? > det
ML: To remove irrelevant biases in the plots,
remove trends of order 0 from det and song2.
ML: trend
IHS> trend dett < det 0
ML: trend
```

FIG. 4. (*continued*)

```
IHS> trend song2t < song2 0
ML: Plot the simulated and real output signal
in the same diagram. Give the command PLOT dett
song2t.
ML: plot
IHS> plot dett song2t
```



```
ML: The simulated signal, dett should agree well
with the real output from the system, song2t. If
not, try another model. Probably the best model
is that in the file ss2 , according to the AIC.
ML: stop
IHS> stop
```

FIG. 4. (*continued*)

possible of the structure of the problem in its solution. The use of scripts will also reduce the size of the knowledge databases considerably.

We believe that the knowledge database built during our project clearly shows that it is possible to construct a useful help system for Idpac, containing knowledge about system identification theory. The database we built can handle maximum-likelihood parameter estimation. Another conclusion is that knowledge engineering demands a large effort, and thus a full knowledge database for system identification will not be constructed easily.

Acknowledgements—We would like to thank our supervisor Karl Johan Åström for initiating the project and our colleagues Sven Erik Mattsson and Karl-Erik Årzén for supporting us in our work.

This study has been part of the Computer-Aided Control Engineering (CACE) project at the Department of Automatic Control, Lund Institute of Technology, supported by STU, the National Swedish Board for Technical Development, under contract no. 85-3042.

REFERENCES

Akaike, H. (1972) Use of an information theoretic quantity for statistical model identification. Proceedings of the 5th Hawaii International Conference on System Science, Honolulu, Hawaii.

Allen, E. M. (1983). YAPS: yet another production system. Technical report, TR-1146, Department of Computer Science, University of Maryland, Baltimore, Maryland.

Allen, E. M., R. H. Trigg and R. J. Wood (1984). The Maryland Artificial Intelligence Group Franz Lisp environment. Technical report, TR-1226, Department of Computer Science, University of Maryland, Baltimore, Maryland.

Breuker, J., C. Duursma, R. Winkels and M. Smith (1989). Knowledge representation in EUROHELP. *Proc. ESPRIT '89.* Bruxelles, pp. 258–270.

Cox, D. R. (1958). *Planning of Experiments.* New York.

Eykhoff, P. (1974). *System Identification, Parameter and State Estimation.* Wiley, London.

Eykhoff, P. (1981). *Trends and Progress in System Identification.* Pergamon Press, Oxford.

Fedorov, V. V. (1972). *Theory of Optimal Experiments.* Academic Press, New York.

Foderaro, J. K. and K. L. Sklower (1981). *The Franz Lisp Manual.* University of California, Berkeley, California.

Gentil, S. and Ph. Conraux (1985). SEXI: Système Expert en Identification. DEA Report, Laboratoire d'Automatique de Grenoble, Institut National Polytechnique de Grenoble, Grenoble.

Gustavsson, I. (1979). Några macros för Idpac (Some macros for Idpac). Technical report, TFRT-7170, Department of Automatic Control, Lund Institute of Technology, Lund.

Gustavsson, I. and A. B. Nilsson (1979). Övningar för Idpac (Exercises for Idpac). Technical report, TFRT-7169, Department of Automatic Control, Lund Institute of Technology, Lund.

Haest, M., G. Bastin, M. Gevers and V. Wertz (1988). An expert system for system identification. *Proc. 1st IFAC Workshop on Artificial Intelligence in Real-Time Control.* Swansea, Wales, pp. 101–106.

Larsson, J. E. (1984). An expert system interface for Idpac. Masters thesis, TFRT-5310, Department of Automatic Control, Lund Institute of Technology, Lund.

Larsson, J. E. and P. Persson (1986). Knowledge representation by scripts in an expert interface. Proceedings of the 1986 American Control Conference, Seattle, Washington.

Larsson, J. E. and P. Persson (1987a). An expert interface for Idpac. Licentiate thesis, TFRT-3184, Department of Automatic Control, Lund Institute of Technology, Lund.

Larsson, J. E. and P. Persson (1987b). The (ihs) reference manual. Technical report, TFRT-7341, Department of Automatic Control, Lund Institute of Technology, Lund.

Larsson, J. E. and P. Persson (1987c). A knowledge database for system identification. Technical report, TFRT-7342, Department of Automatic Control, Lund Institute of Technology, Lund.

Larsson, J. E. and P. Persson (1987d). Experiments with an expert system interface. Final report, TFRT-3196, Department of Automatic Control, Lund Institute of Technology, Lund.

Ljung, L. (1987a). *System Identification: Theory for the User.* Prentice-Hall, Englewood Cliffs, New Jersey.

Ljung, L. (1987b). *The System Identification Toolbox for Use With Matlab, User's Guide.* Math Works, Sherborn, Massachusetts.

Ljung, L. and T. Söderström (1983). *Theory and Practice of Recursive Identification.* MIT Press, Cambridge, Massachusetts.

Moler, C., J. Little and S. Bangert (1987). *PC-Matlab for MS-DOS Personal Computers.* Math Works, Sherborn, Massachusetts.

Monsion, M., B. Bergeon, A. Khaddad and M. Bansard (1988). An expert system for industrial process identification. *Proc. 1st IFAC Workshop on Artificial Intelligence in Real-Time Control.* Swansea, Wales, pp. 95–99.

Nagy, P. A. J. and L. Ljung (1989). An intelligent tool for system identification. Proceedings of the 1989 IEEE Control Systems Society Workshop on Computer-Aided Control System Design, Tampa, Florida.

Schank, R. C. and R. P. Abelson (1977). *Scripts, Plans, Goals and Understanding.* Lawrence Erlbaum Associates, Hillsdale, New Jersey.

Schank, R. C. and C. K. Riesbeck (1981). *Inside Computer Understanding.* Lawrence Erlbaum Associates, Hillsdale, New Jersey.

Skogø Hansen, S., L. Holgaard and M. Smith (1988). *EUROHELP: Intelligent Help Systems for Information Processing System*. Computer Resources International, Birkerød, Denmark.

Waters, R. C. (1985a). KBEmacs: A step toward the programmer's apprentice. Technical Report 753, MIT Artificial Intelligence Laboratory, Cambridge, Massachusetts.

Waters, R. C. (1985b). The programmer's apprentice: A session with KBEmacs. *IEEE Trans. Software Engng,* **SE-11,** 1296–1320.

Wieslander, J. (1979*a*). Interaction in computer-aided analysis and design of control systems. Doctorial dissertation, TFRT-1019, Department of Automatic Control, Lund Institute of Technology, Lund.

Wieslander, J. (1979b). Design principles for computer-aided design software. Preprints of the IFAC Symposium on CAD of Control Systems, Zürich.

Wieslander, J. (1980). Idpac commands—user's guide. TFRT-3157, Department of Automatic Control, Lund Institute of Technology, Lund.

Wilensky, R., J. Mayfield and A. Albert (1986). UC—A progress report. Report no. UCB/CSD 87/303, Computer Science Division (EECS), University of California, Berkeley, California.

Åström, K. J. and P. Eykhoff (1971). System identification—A survey. *Automatica,* **7,** 123–162.

Åström, K. J. (1980). Maximum likelihood and prediction error methods. *Automatica,* **16,** 551–574.

Part II

An Expert System for Frequency Response Analysis

AN EXPERT SYSTEM FOR FREQUENCY RESPONSE ANALYSIS

J. E. Larsson

Department of Automatic Control, Lund Institute of Technology, Box 118, S-221 00 Lund, Sweden

Abstract

Frequency response analysis is one of the oldest and most widely used methods to determine the dynamics of a stable linear system. Though quite simple, it requires knowledge and experience of the user, in order to produce reliable results. Available equipment will perform an experiment automatically, but does not support the user in designing the experiment nor in validating the results. The expert system FREX is designed to help the user in performing the analysis. It checks whether the system is linear, finds the frequency and amplitude ranges, verifies the results, and, if errors should occur, tries to give explanations and remedies for them.

Introduction

FREX is a small expert system for frequency response analysis. It is intended to be used as an advisory system together with a frequency response analyser. The system is completely stand-alone, i.e., it runs separately from the frequency analyser, and the user handles all communication between the two. The idea is that of an expert advisor guiding the user through an experiment, with a Q/A dialog.

The system is supposed to be useful for all non-expert users, e.g., students in undergraduate identification courses, and engineers in industry.

The idea of building an expert system for running a frequency analyser was kindled during a graduate course in identification, in the fall of 1985. During this course, all participants performed a frequency response analysis experiment, and gave a small recipe for the method. The knowledge database of FREX originates partly from these recipes. Thus, the knowledge acquisition was performed in part by more than 10 persons.

The knowledge database is quite small, consisting of 65 rules. These are run using backward chaining and a Q/A dialog. The expert system was built with MESS, a very small and simple shell, Larsson (1988). Some minor changes has been made in the MESS source code.

This project has also been described in Larsson (1989).

Frequency Response Analysis

Frequency response analysis is used to find a mathematical model of a real world process with one input and one output signal. A linear model describing the gain and phase shift as functions of the input signal's frequency will be obtained. In the experiment, the process to be identified is driven by a sinusoid input signal,

$$u(t) = u_0 \sin(\omega t).$$

As the method demands the process to be linear, time invariant, and asymptotically stable, the output will become sinusoid when all transients have decayed. The output will be

$$y(t) = |G(i\omega)| u_0 \sin(\omega t + \arg G(i\omega)),$$

where G is the transfer function of the system and ω the input frequency. Normally, the phase difference, $\arg G(i\omega)$, is negative. By measuring the amplitude gain and the phase difference at several different frequencies, ω_i, it is possible to obtain, e.g., a Bode diagram.

The procedure outlined is very sensitive to noise, and it is seldom possible to use it in this simple form. However, the method can be improved by a correlation technique. The output is multiplied with $\sin(\omega t)$ and $\cos(\omega t)$, and integrated.

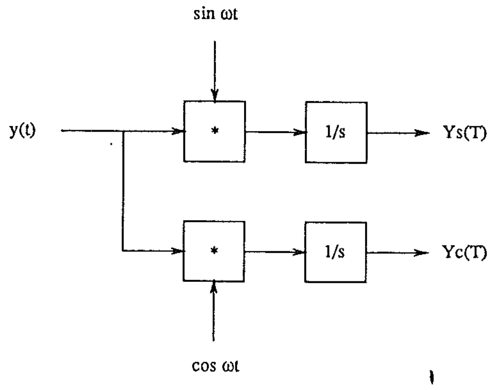


Figure 1. The correlation method setup.

This experimental setup will result in the computation of the following integral values:

$$Y_s(T) = \int_0^T y(t)\sin(\omega t)dt$$

and

$$Y_c(T) = \int_0^T y(t)\cos(\omega t)dt.$$

The integration time, T, should be a whole number of periods, i.e.,

$$T = \frac{2\pi n}{\omega}.$$

This gives

$$Y_s(T) = \frac{T}{2}u_0\text{Re}G(i\omega)$$

and

$$Y_c(T) = \frac{T}{2}u_0\text{Im}G(i\omega).$$

From Y_s and Y_c the amplitude gain and phase difference may easily be computed:

$$|G(i\omega)| = \frac{2}{Tu_0}\sqrt{Y_s^2(T) + Y_c^2(T)}$$

and

$$\arg G(i\omega) = \arctan \frac{Y_c(T)}{Y_s(T)}.$$

An intuitive way of looking at this method is to view it as a filtering of the output signal, using a band pass filter at the frequency ω, with the filter width proportional to $1/T$. The problem with the method is that it often requires long experiments. Of course, it can only be used if it is possible to disturb the process with a sinusoid input. For readings on frequency response analysis in general and this method in particular, see for example Åström (1975) or Söderström (1984). Frequency response analysis was used by the physicist Ångström as early as 1861. It enabled him to make a significant improvement in the determination of thermal diffusivity, Ångström (1861).

Commercial equipment exists for performing frequency response analysis using the method described above. In this project, a Solartron 1250 frequency analyser, Solartron Instrumentation Group (1983 a, b), was used. Before an experiment is performed, one must decide a frequency interval, decay and integration times, and a suitable input signal amplitude. Then the analyser runs experiments at a number of frequencies, covering the frequency interval, computes the integrals, the amplitude gains and the phase differences, and plots a Bode diagram. Thus, the user has to make several experiment design decisions. The expert system is mainly concerned with these decisions, and leaves the numerics to the frequency analyser.

The Knowledge Database

The knowledge database consists of rules, used by the expert system to monitor the frequency response analysis and to diagnose the results. The system performs the analysis in several stages. First, the expert system checks whether the process is stable or must be stabilized. Then it tries to establish that the process is fairly linear, at least in a certain range of amplitudes. After this, the expert system proposes tests that will find the limits of the frequency and amplitude ranges. Then the experiment is performed and several validation tests are made. The validation includes checking with a priori knowledge, verification of frequency limits, and a comparison between the process and a simulation of the model obtained. If, somewhere in this process, anything goes wrong, there are rules for explaining probable causes of the errors. The error diagnostic rules make up a large part of the database.

- The stabilization is given a simple treatment. If the process is not stable from the beginning, the system suggests a proportional regulator to stabilize it. If, when the results have been obtained, the model does not agree with the a priori knowledge, the system may guess that the experiment has led to the identification of the inverse of the controller. In this case it recommends a new experiment with better noise suppression, a different controller or a different input connection place.
- In order to assess the linearity properties of the process, the system tries several things. A visual inspection of the signals is used if possible. A dual step test is proposed, i.e., the results of two step inputs with the same amplitude, one positive and one negative, are examined. Also, several small step inputs are tried at different signal levels. All the outputs should look essentially the same in the linearity range. If there is friction in the process, this can usually be seen in the output signal. Thus, if the user knows that there is friction in the process, or the output signal is deformed, the system recommends the use of a bias in the experiment, in order to avoid friction.
- If the user knows or has an educated guess on the frequency range, the system uses

this in the experiment. If the result is not satisfactory, the error diagnosis tries to adjust the frequency limits. In cases where no guess is available, the system uses a step response experiment to get a suggestion for the cut-off frequency. The frequency limits are centered around this frequency. The system may also be designed for a certain frequency. If so, an interval around this frequency is included.

- The amplitudes of the signals used in the experiment must be larger than the noise level of the process, and, in case the process is digital, larger than the quantization level. On the other hand, the signals must be small enough not to reach linearity or other boundaries. In order to check this, the static gain of the process transfer function is measured. Should the model obtained not be satisfactory, the system checks whether the signals reaches some kind of amplitude limitation. A smaller amplitude is then recommended.
- The frequency response analysis experiment is executed using the gathered information. First a quick experiment with frequencies covering the whole frequency range is performed. A rather low amplitude is recommended. If there are points in the Bode diagram where the curves change rapidly, new experiments with a frequency range fitted rather closely around these points are suggested.
- When a result has been obtained, it must be tested and validated. For this reason the system compares the model against any a priori knowledge available. If the frequency range has only been guessed, that guess must also be checked afterwards. In this case the system suggests a new experiment with a wider frequency range. Finally, the system recommends that the model obtained is used in a simulation. In this way it is possible to make sure that the process and the model has the same behavior for some selected input signals.
- If the analysis should fail, or the results be incompatible with a priori knowledge, several errors may be diagnosed. If the process has been used in a closed loop, the inverse controller may have been identified. When the signal to noise ratio is low in the higher frequencies, the method will only produce reliable results at lower frequencies, and the upper frequency limit must be lowered. If the signals reach amplitude limitations the experiment should be rerun with a smaller amplitude; and vice versa: if the amplitude is close to the noise or quantization level, the experiment should be rerun with a larger amplitude or longer integration times. If there is backlash or hysteresis in the process, making it strongly non-linear, frequency response analysis may be unable to provide a good result. If there is friction in the process, the linearity properties may have been destroyed. In this case the experiment should be rerun with a bias if possible. Another explanation may be that the a priori knowledge is wrong. Finally, if the cut-off frequency differs quite a lot from the desired working frequency, very tight control will be needed, and it will probably be a good idea to run the experiment with a feedback to speed up the process.

Structure of the Rule Base

The knowledge database currently consists of 65 rules, running in backward chaining. The rules are split into seven groups, each of which is concerned with a different task.

- The `phase control` rule is the only one in its group, and it executes the other rule groups, one after the other.
- The `linearity` rules are concerned with checking whether the system is linear or not. If it is, they try to establish a linearity range.
- The `frequency range` rules try to find the frequency range, i.e., the lower and upper frequency limits to be used in the experiment.
- The `amplitude range` rules try to find lower and upper limits for the amplitude of the input signal. They then pick an amplitude close to the lower limit.
- The `perform experiment` rules take care of telling the user to actually run the experiment. First, an experiment covering the whole frequency range is performed, then, if there are any points of special interest, e.g., where the transfer function changes very rapidly, experiments are made, which cover these points more closely.
- The `verify` rules propose some tests to verify the results. Any a priori knowledge is checked, if the frequency range was guessed, it is increased, and a simulation is suggested.
- The `explain errors` rules are invoked if the analysis does not end with success, and certain error conditions are present. These

rules try to guess the source of the error and, if possible, suggest a remedy.

The rules are written in the MESS rule format, which should make them rather easily understood. Three examples of actual rules are shown below.

```
(rule phase-control
  (if
    (the system is stabilized)
    (the linearity range is known)
    (the frequency range is known)
    (the amplitude range is known)
    (the experiment has been performed)
    (the result is verified))
  (then
    (the frequency analysis has been performed)))
```

The `phase control` rule schedules the other rules, so that the different phases of a frequency response analysis are performed, one after the other. Next, here is an experiment design rule.

```
(rule stabilize-2
  (if
    (not (the system is stable))
    (* the system can be stabilized with a
       feedback loop))
  (then
    (action (use a proportional regulator
             to stabilize the system))
    (action (connect the input to the reference
             signal of the controller))
    (the system is stabilized)))
```

This rule takes care of stabilizing the system before the experiments take place. This is necessary if the process is unstable. The next rule is concerned with error diagnosis.

```
(rule explain-errors-11
  (if
    (not (the resulting transfer function is
          confirmed by the a priori knowledge))
    (the system is digital)
    (a priori knowledge about the transfer
     function is available)
    (* the transfer function is wrong close to
       the sampling frequency))
  (then
    (it is not possible to get good results close
     to the sampling frequency)
    (action (lower the upper frequency limit and
             rerun the experiment))
    (an error has been diagnosed)))
```

This last example shows a rule that tries to explain erroneous results due to sampling. The error diagnosis rules are tried in order from more specific to more general error, until a likely one is found. The expert system then reports this diagnosis as the probable cause of error.

Some Example Runs

The expert system communicates with the user through a Q/A dialog. The questions will concern the user's knowledge about the process and the results of the experiments. Here is an example of part of a typical dialog.

```
Is it true that the frequency range is known a
priori? (y/n)
n
Is it true that you have a good guess about the
limits of the frequency range? (y/n)
n
Is it true that a step response frequency range
experiment has been performed? (y/n)
n
PERFORM:             perform a step response
                     frequency range experiment
Deduction/advice:    the results of a step response
                     experiment are available
Deduction/advice:    the cutoff frequency is appr.
                     omega = 1 / rise time
Deduction/advice:    center the frequency range
                     around the cutoff frequency
Deduction/advice:    checking of the frequency
                     range is needed afterwards
Deduction/advice:    the frequency range is known
```

This example shows how the system tries to establish a frequency range for the experiment. First, it checks whether the user already knows or has an educated guess about this range. After negative answers, the system proposes a step test experiment, and uses the result to guess a reasonable frequency range. This range may be changed later on, though. The fact `(checking of the frequency range is needed afterwards)` actually forces the system to consider changing the range after the experiment has been performed.

The next dialog part shows the error diagnosis of a session.

```
Is it true that the resulting transfer function
is confirmed by the a priori knowledge? (y/n)
n
Is it true that the amplitude is as small as the
noise or quantization level? (y/n)
n
Is it true that the resulting transfer function
is wrong in the lower frequencies? (y/n)
n
Is it true that the resulting transfer function
is wrong in the higher frequencies? (y/n)
n
Is it true that the signal to noise ratio is low
in the higher frequencies? (y/n)
n
Deduction/advice:    the friction probably destroys
                     the experiment
PERFORM:             rerun the experiment with a
                     bias, to avoid the friction
Deduction/advice:    the signals are not linear
Deduction/advice:    an error has been diagnosed
```

Here, the experiment has already been performed, and at a certain stage the validation goes wrong. Therefore the system tries several guesses of what might be the problem. Earlier it has been established that some unknown friction is present. As no specific explanation works, the system shows good expertise, and blames the unknown friction.

The Inference Engine

The expert system shell used is the MESS system, Larsson (1988). It is a small, (192 lines of source code), Scheme system that provides the user with forward and backward chaining. The backward chaining strategy is used in FREX. Some changes in the source code of MESS have been made. All fact deductions are traced, and there are some new clauses allowed in the then parts of the rules. The input and output has been changed so that the Q/A dialog gives an impression of semi-natural language. This demands the facts to be written in appropriate syntactic versions, i.e., all facts to be asked must fit after the phrase "Is it true that...".

The MESS system has obvious strengths in that its source code is short, simple and easy to change. Its main drawback is that it does not use an effective matching strategy, which makes its execution slow for larger rule bases. As the present rule base is rather small, the speed has not been a problem, though.

The FREX system was originally implemented in Chez Scheme on a SUN 3/50, but it has also been moved to PC Scheme on an IBM PC/AT, and to MIT Scheme on a Macintosh+. The response time on the SUN is immediate, and on the AT it is a matter of a few seconds. On the Macintosh+, however, FREX is quite slow, often using close to a minute per query.

Conclusions and Further Developments

The chosen target domain, frequency response analysis using a frequency analyser, is a good, clean area for an expert system project. There are very few interactions with other areas; most of the tasks can be solved at the site of the frequency analyser. This has made it possible to produce a rather complete knowledge database, and a system that is useful for most non-experts. There exists only a few written descriptions of both the theoretical and *practical* problems of frequency response analysis, Åström (1975) is an exception, but this knowledge may now be found in this system. Thus, the project has also resulted in valuable knowledge refinement.

The system probably needs more testing before it will be robust enough for production use or inclusion in a frequency analyser. Currently, it runs completely stand-alone. Work has been done to improve the front-ends of frequency analysers, see for example Wichtel (1989), and a better way would be to embed the system in the analyser, making it an integral part of an intelligent instrument.

Acknowledgements

I am indebted to Karl Johan Åström, who gave the identification course and forced his students to try a little knowledge acquisition, and to Jan Peter Axelsson, Ola Dahl, Kjell Gustafsson, Ulf Hagberg, Mats Lilja, Michael Lundh, Bernt Nilsson, Per Persson, Anders Wallenborg, Karl-Erik Årzén, and Ann-Britt Östberg, all of whom produced written recommendations on frequency response analysis in general and running the Solartron 1250 analyser in particular. Also, I would like to thank Sven Erik Mattsson for good advice during the project.

References

Larsson, J. E., (1988): "MESS—A Minimal Expert System Shell," Technical report, TFRT–7380, Department of Automatic Control, Lund Institute of Technology, Lund.

Larsson, J. E., (1989): "Knowledge-Based Frequency Response Analysis," in Larsson, J. E., (Ed.) *Proceedings of the SAIS '89 Workshop,* Department of Automatic Control, Lund Institute of Technology, Lund.

Solartron Instrumentation Group, (1983 a): "1250 Frequency Response Analyser, Operating Manual," Solartron Instrumentation Group, Schlumberger Electronics (UK) Ltd., Farnborough, England.

Solartron Instrumentation Group, (1983 b): "1096 Data Management System," Solartron Instrumentation Group, Schlumberger Electronics (UK) Ltd., Farnborough, England.

Söderström, T., (1984): *Lecture Notes in Identification,* Automatic Control and Systems Analysis Group, Department of Technology, Uppsala University, Uppsala.

Wichtel, E., (1989): "A Control Program for a Frequency Analyser," Master thesis, TFRT–5400, Department of Automatic Control, Lund Institute of Technology, Lund.

Ångström, A. I., (1861): "Neue Methode, das Värmeleitungsvermögen der Körper zu bestimmen," Annalen der Physik und Schemie, **114**, pp. 513–530.

Åström, K. J., (1975): "Lectures on System Identification, Chapter 3, Frequency Response Analysis," Internal report, TFRT–7504, Department of Automatic Control, Lund Institute of Technology, Lund.

Part III

Diagnostic Reasoning Strategies for Means-End Models

CHAPTER 1

INTRODUCTION

Industrial processes can be described and modeled in several ways, and the models obtained are used for many different tasks. Designers, operators, and process engineers often reason about the goals of a process and the means available for achieving these goals. However, most model types contain little or no means-end information, and thus provide no good support in these tasks.

This work utilizes one type of explicit *means-end* models, *multilevel flow models,* (MFM), which were proposed by Lind (1990 a). Lind has suggested a syntax for a formal language and given general ideas on how to use the MFM representation. The contributions of this work are descriptions of three methods or strategies for diagnostic reasoning using MFM, and a set of examples of presentation of means-end information; altogether four parts:

- Measurement validation
- Alarm analysis
- Fault diagnosis
- Presentation of means-end information

The measurement validation algorithm takes a set of measured values and uses any available redundancy to check consistency. A single erroneous flow measurement will be marked and corrected; if there are several conflicting values, the consistent subgroups of measurements will be marked but no flow value corrected.

The alarm analysis algorithm takes as input a set of alarm states such as *normal, low flow, high flow, low volume,* and *high volume.* Each alarm is associated with a part of an MFM model, and the method recognizes some of the alarms as primary, while the others are either primary or consequences of the primary alarms.

The fault diagnosis algorithm uses an MFM model to produce a "backward chaining" style of diagnosis. The input can come from questions answered by the user, from measured signals, or triggering of rules. The system will trace faults, provide explanations, and give remedies.

The MFM models have a graphical representation themselves, but it is also possible to use other types of pictures and diagrams to present the means-end information to the user. This includes showing goal hierarchies, using special flow diagrams to show the thermal flows through the process, highlighting mass and energy flows in flow sheets, and using block diagrams to show the control systems.

1.1 Means-End Models

Today's control and supervisory systems almost exclusively use models based on physical topology and mathematical and logical behavior. However, the problems handled and the questions asked by designers, operators, and process engineers often involve other categories of information, equally important for the different tasks performed. In particular, a large part of the reasoning performed concerns means and ends, i.e., the goals of the process in question, and the functions available to achieve these goals. This reasoning is needed in planning of operation, supervision, and fault diagnosis. It should be noted that the same kind of reasoning is extensively used in knowledge-based systems.

The models available today give limited support only for diagnostic tasks. The continuing use of topological and behavioral models for diagnostic reasoning tasks has sometimes led to an unawareness of the important and fundamental difference between different types of models.

`Which components are present?`
`How are they connected?`
	`Where is a component located?`
	`How large is the component?`

`What do the signals look like?`
`What is the value of K?`
	`How is this task performed?`
	`What does the component do?`

Figure 1.1 Questions of different information categories. The examples concern physical topology, (upper left), geographical properties, (upper right), behavior, (lower left), and means-end information, (lower right). All these questions may be important for design, operation, and diagnosis, but they belong to different model categories.

Examples of a number of questions concerning different information categories have been gathered in Figure 1.1. There are questions about physical structure, geographical properties, behavior, and means and ends.

current state will be viewed as a fault. This approach is quite common in model-based diagnosis.

Of course, all model types are more or less normative, as they describe only a subset of all possible properties and behaviors of the target system, but the degree of explicitness may vary. Thus, in some modeling descriptions, such as, e.g., simulation equations, the normative nature is not always obvious, while in MFM, it is both essential and explicit.

1.3 An Example of an MFM Model

The following example will be used to explain the basic concepts of MFM. The target process consists of a plate heat exchanger, which is used to heat a product medium to a certain temperature, see Figure 1.2. Water is heated through injection of steam and then pumped through a heat exchanger. The product medium is also pumped through the heat exchanger where it is heated by the water.

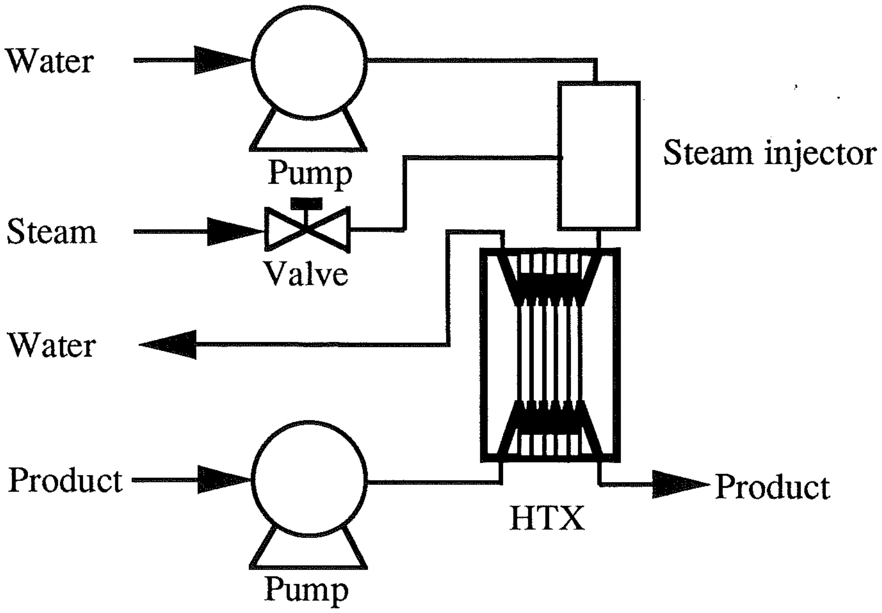


Figure 1.2 A heat exchanger system. The flowsheet shows how water is pumped through a steam injector, where it is heated with steam, and through a plate heat exchanger. The product is pumped through at the other side of the heat exchanger.

The primary goal is to heat the product to a certain temperature. But there are also two subgoals: having water and product available, i.e., bringing the media to the heat exchanger:

- `G1`: Heat product to a certain temperature

Once it is understood that diagnostic reasoning tasks are concerned as much with means-end information as with physical topology and behavior, the benefit of explicit means-end models becomes apparent. This is the rationale for MFM. It provides a way of explicitly capturing the goals of a process, the functions available, and the relations between goals and functions: a goal can be achieved by a set of functions, and a function may be conditioned by a subgoal.

1.2 Basic Ideas of MFM

In multilevel flow modeling, a system is modeled as an artifact, i.e., a man-made system constructed with some specific *purpose* in mind. Thus, MFM contains some concepts from the natural sciences, (mathematics and physics), and some from the human sciences, (cognitive science and psychology), and in a way forms a bridge between these sciences.

The purposes of a system are modeled with the MFM concept of *goals,* i.e., objectives of running a system or using a process. The physical components of a system are used to provide one or several *functions.* These functions are the means with which the goals are achieved. The concepts of goals, functions, and components are explicitly represented in the MFM models.

Just as important are the different *relations* between goals, functions, and components. In MFM, these relations are explicitly described. A set of functions used to fulfill a goal are grouped together and connected to that goal via an *achieve* relation. If a subgoal is a necessary condition for a function to be working, it will be connected to the function via a *condition* relation. If a physical component is used for a certain function, the component object is connected to the function object via a *realize* relation. These relations connect the objects into a graph, i.e., the MFM model proper. Algorithms can then traverse this graph in order to perform different reasoning tasks. The graph will contain multiple levels of goals, functions, subgoals, and subfunctions. It should be noted that all the relations are many-to-many; thus the graph is not a tree, as described in Chapter 3.

It is important to observe that MFM models are *normative,* i.e., they describe how the system is supposed to work, not how it is actually working in the present state. Each difference between the model and the

- `G2:` Bring product to heat exchanger
- `G3:` Bring water to heat exchanger

It would be possible to define more goals, as the modeling depends on the designer's *intent,* but in this example, three goals are chosen for simplicity.

The given example process is rather small, but there are many functions present. Most of the physical components have well known tasks to perform, which makes it straightforward to produce a list of functions. Keeping the goals in mind can also help discovering functions:

- `F1:` Provide product
- `F2:` Transport product
- `F3:` Transfer thermal energy between media
- `F4:` Provide thermal energy
- `F5:` Transport thermal energy
- `F6:` Transport water
- `F7:` Provide water
- `F8:` Provide stream
- `F9:` Transport steam
- `F10:` Mix water and steam

As with the goals, more functions could be defined, e.g., keeping the water and product separated, but the set presented will suffice for the example.

It is quite uncomplicated to produce the list of components. Note that the product and water tanks do not actually appear in Figure 1.2. As with goals and functions, a selection has been done here too, mainly on the basis of detail. Thus single pipes, bolts, machine parts, etc., has been left out:

- `C1:` Product tank
- `C2:` Product pump
- `C3:` Heat exchanger
- `C4:` Water tank
- `C5:` Water pump
- `C6:` Steam system
- `C7:` Steam valve

- `C8`: Steam injector

These are the sets of goals, functions, and components. However, the relations between these objects are as important as the objects themselves. First, the goal `G1` is superior to `G2` and `G3`, i.e., the latter are subgoals of `G1`. Thus, there is a *goal hierarchy,* formed by goal-subgoal relations. There are also relations between goals, functions, and components. For example, the heat exchanger component is used to realize the function of transferring thermal energy from water to product, and this function is used to achieve the goal of heating the product. Thus, there is an *abstraction hierarchy* in the means-end dimension. In Figure 1.3, both the goal hierarchy and the means-end relations are shown in a graph.

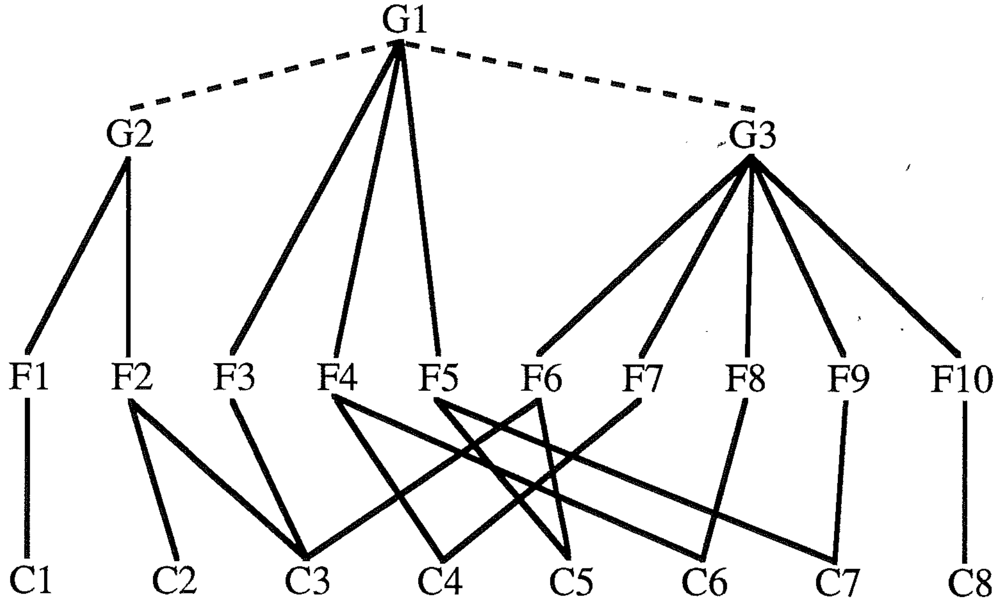


Figure 1.3 Goals, functions, components, and relations of the heat exchanger system. The goals are achieved by several different functions, while the functions in turn are realized by several different components. In addition, the three goals also form a goal-subgoal hierarchy, (dashed lines).

As can be seen, the graph of objects and relations is quite complex, even for a small process as the one in the example. In an MFM model, the goals, functions, and relations are represented in a graphical language. A model of the example process is found in Figure 1.4. Note that the symbols used will be described in the following chapters.

The top level goal is achieved by a network of flow functions, describing the flow of thermal energy through the system. The energy initially contained in the water and steam is transferred to the product. In order to transfer the thermal energy, both product and water must be available, thus the function for transporting energy is conditioned by two subgoals. These goals in turn are achieved by networks of functions for transporting product and water through the system.

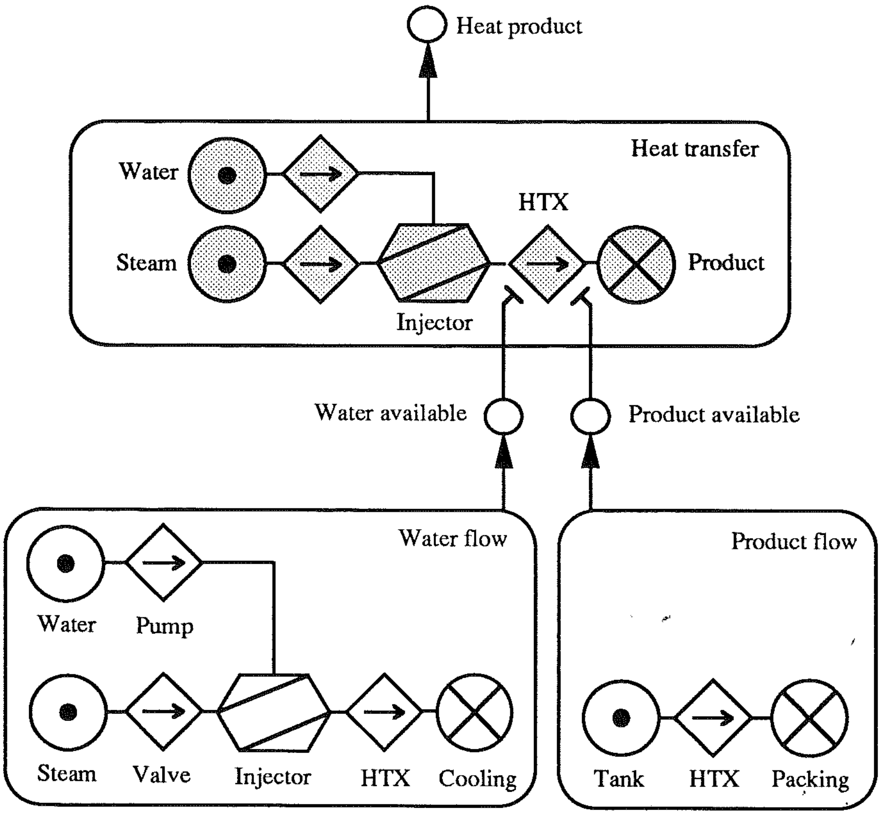


Figure 1.4 An MFM model of the heat exchanger system. The goals and goal hierarchy is shown in the tree structure of the graph, while the functions are connected into flow paths in three networks. The topmost goal, to heat the product, is achieved by the energy flow path, (upper network), while the subgoals, to bring water and product to the heat exchanger, are achieved by the water flow path, (lower left), and the product flow path, (lower right). The components and realize relations are not shown.

1.4 *Multiple Views*

It is beneficial to model a process in several ways, and the different models should be used for the tasks that they are best suited for. This means that the process must be represented and presented in several ways at the same time; there should be multiple *views* or *perspectives* of it. The different views would then contain models belonging to different model categories or abstraction levels.

Multiple views have been used previously in knowledge representation. In qualitative physics it is necessary to separate physical topology from function, as is pointed out in de Kleer and Brown (1984). Several diagnostic expert systems use a multiple view representation, see for example Struss (1987, 1991, 1992) and Mariño *et al* (1990). The project described in Larsson (1990 a), Årzén (1989, 1990, 1993) and Årzén *et al* (1990), has proposed an integrated multiple view system for control, supervision, and diagnosis.

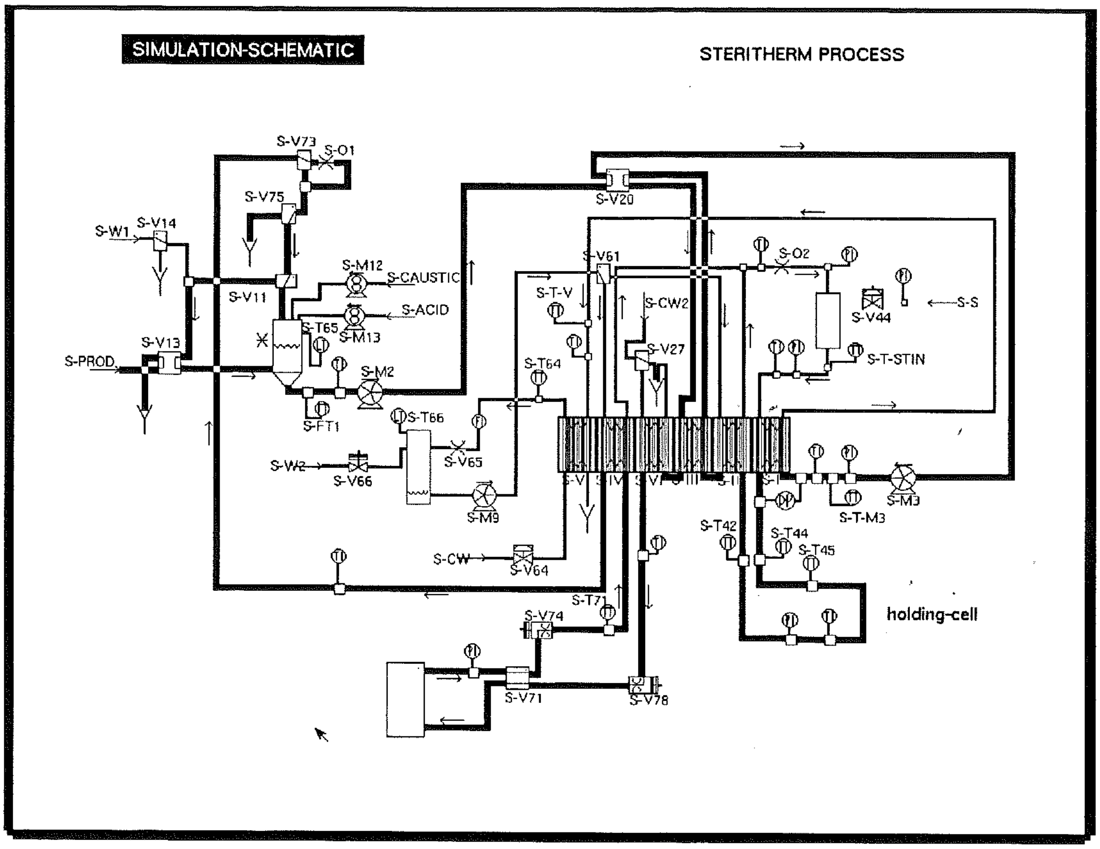


Figure 1.5 The Steritherm flow sheet shows a typical example of a small industrial process. The diagram contains information about the physical topology of the process, showing what components it consists of and how they are connected.

The most common way of presenting a process is probably with a flow sheet; see Figure 1.5 for an example. This is a picture which contains objects and connections, roughly corresponding to the physical components of the process and the connections between them. A flow sheet may contain other types of information as well, but this is seldom clearly stated nor obvious to the user. For example, it is usually unclear whether there is any geographical information in the layout of the flow sheet, e.g., if the components in the physical world are arranged in the same way as the flow sheet objects. The lack of clear knowledge of what is shown or not is an obvious drawback, and it can be potentially dangerous, if, e.g., an operator misreads a diagram for information which really is not there.

It is possible to resolve this problem with the help of a multiple view representation, in which it is clearly stated exactly what kind of information there is in each type of picture. Thus, several types of views can be recognized.

The Geographical View

In this view, the representation contains information about which physical components that are present and how they are connected. In addition, there is some kind of geographical information, i.e., the representation may describe the physical location of the components and the connections, how they look, their relative size, etc. Thus, the geographical view has some geometrical similarity to the physical world. Typical examples of models that could be used in a geographical view are 2D and 3D pictures, technical drawings, photographs, etc. A simple example of a lamp circuit is shown in Figure 1.6. The geographical property here is mainly that the icons look more or less like the physical components, which could be the case in some flow sheet presentations.

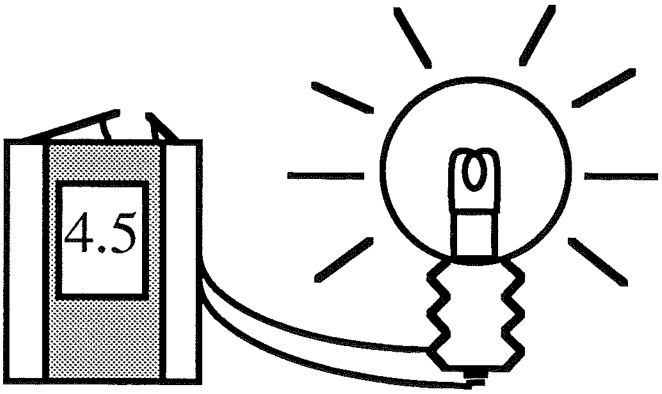


Figure 1.6 A geographical view of a lamp circuit. The diagram shows the physical components and how they are connected, but in addition it gives some hints about the physical outlook of the components. Other versions of the geographical view would be photos, technical drawings, or 3D graphics. In this example, the geographical information is simply that the icons look similar to the real world objects.

The Topological View

In this view, the representation contains information about the physical components and how they are connected, but nothing else. Specifically, there is no geometrical information in the pictures. Thus, the icons used may or may not resemble the physical components, and components and connections may be arranged in a completely different way than in the real world. The most typical example of a model used in a topological view is probably the standard flow sheet, but circuit diagrams, block diagrams, etc., are also of a topological nature. A topological representation of the lamp circuit can be seen in Figure 1.7; in this case a standard circuit diagram.

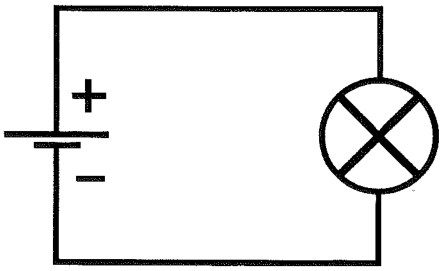


Figure 1.7 A topological view of the lamp circuit. The components and connections are shown, but there is no information about size, placement, or physical outlook of the components.

The Behavioral View

So far, the different views have been of a graphical nature. However, it is also useful to have *behavioral* descriptions, which contain information about the temporal development of states of the process. The most common forms are mathematical and logical models, such as algebraic and differential equations, transfer functions, state transition graphs, qualitative models, Petri and Grafcet nets, etc. A set of equations for the lamp circuit is shown in Figure 1.8.

$$\begin{cases} U = R \times I \\ U = 4.5V \\ R = 0.5\Omega \end{cases}$$

Figure 1.8 A behavioral view of the lamp circuit. The equations may'be used to predict the behavior of the system's state. Other versions of the behavioral view would be Grafcet nets, alarm logic, etc.

The Means-End View

The information about means and ends can also be represented in a view of the process, the *functional* or *means-end* view. The obvious choice of model type is of course MFM. Other types of representations capture some of the information present in an MFM model, e.g., function block diagrams, and fault trees. However, none of these provide all the information of MFM, nor do they always present it in a clear and logical fashion. An MFM model of the lamp circuit is found in Figure 1.9.

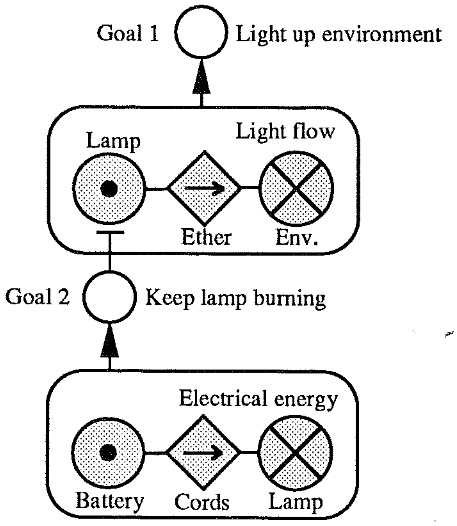


Figure 1.9 A means-end view of the lamp circuit. The purposes of the process are described, together with the functions that must be available to obtain the desired goals, while the physical components and topology are unimportant. The top level goal, to light up the environment, is achieved by an energy flow, (of light), from the lamp to the environment. The lamp's function as a source of light depends on a subgoal, to keep the lamp burning, which in turn is achieved by another energy flow, of electrical energy from the battery to the lamp.

There are many other possible views of a process, such as function block diagrams, sequential logic charts, etc. In a multiple view system, the different representations may be given their own views. Such a system is needed for the integration of MFM with other models.

Thus, the ideas presented in this work are not supposed to replace the standard techniques. Instead, several of the standard techniques should be joined into one system, and *extended* with means-end models, in order to obtain a system with greater capabilities than what is available today.

1.5 Three Methods for Diagnostic Reasoning

So far, MFM has been presented as a modeling and representation language, with a defined syntax and graphical representation. A general semantics, i.e., some advice on how to interpret the different MFM symbols is also available, see Lind (1990 a). With some minor modifications, these suggestions will be used in this work.

However, the usefulness of MFM is still an open question. What tasks can it be used for, what reasoning methods can be successfully implemented with MFM, and what specific semantics is needed to accomplish it? The main contributions of this work are three diagnostic reasoning methods, which are easily and efficiently implemented with the help of MFM: measurement validation, alarm analysis, and fault diagnosis.

Measurement Validation

Industrial processes usually have many sensors which directly or indirectly measure the same variables. When mass and energy balance equations are taken into account, the set of measurements gives rise to redundancy. The proposed method uses some of this redundancy to check the measurements of a mass or energy flow, and is thus called *measurement validation* or *data reconciliation.* A typical situation is illustrated in Figure 1.10.

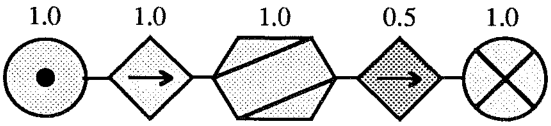


Figure 1.10 A flow path with flow values. The numerical values of the flows are shown above the corresponding MFM symbols. One flow deviates from the rest. Thus, at least one of the measurements must be faulty; either the single one, the four, or all five are wrong. The method developed presents all these three hypotheses.

There are five flow measurements that should agree, i.e., have more or less the same value. However, one of them clearly deviates from the rest. Three hypotheses could explain the situation:

- The single measurement is wrong, and the rest are correct.
- The single measurement is correct, and the rest are wrong.
- All measurements are wrong.

A failure to consider any of these possible explanations could be potentially dangerous, thus the method uses the sensor values to assign the different measurements to *consistent subgroups.* For each value there is a *validated value,* which can be set to a value different from the one actually measured. The method outputs the following information:

- Each consistent subgroup is presented. Thus the user can get an impression of the general agreement or disagreement of the measurements.
- A single deviating measurement is highlighted. Thus a probable fault can be detected and isolated quickly.
- If a single deviating measurement is surrounded by several consistent ones, the validated flow value corresponding to the deviating one will be set to that of the surrounding group.

Several other decisions could be made; the main aim of presenting the method is not to give the best solution of what decisions to make based on a redundancy analysis, but to show the possibilities of using MFM as a tool for automated measurement validation. The presented method has more features; for example it uses a *flow propagation* algorithm to handle the case when no sensor is connected to a part of the model. A thorough description of the algorithm is given in Chapter 4.

Alarm Analysis

Most processes are equipped with a large number of alarms, and in a failure state many of these will trigger. Some of them will be directly connected to the primary sources of faults, but other may be secondary, i.e., due only to consequential effects of the primary failures. The method helps to separate primary alarms from those that might be secondary, a task which can be vital in a fault situation.

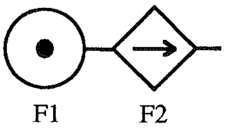


Figure 1.11 A source connected to a transport function. If the capacity of the source goes down, the transport flow will be forced out of its working interval, while if the desired flow of the transport increases, the source may or may not be able to supply it. If the desired flow through the transport decreases, however, the source is not affected. Thus, a low capacity in the source *will* cause a low flow through the transport; a high flow through the transport *may* cause a low capacity in the source; but a low flow through the transport will not cause any fault in the source.

Each flow function is associated with a working condition, which will give rise to an alarm when violated. Due to the semantic interpretations of the flow functions, certain faults may or will cause consequential faults in connected flow functions. An example is given in Figure 1.11.

The function **F1** is a source that provides mass or energy for **F2**, a transport function. If **F1** would loose its capacity of delivering the required amount of mass or energy, the transport would not be able to keep a sufficiently large flow. If, on the other hand, **F2** was to require too large a flow, **F1** might not be able to provide it. Thus, two causation rules can be formulated for the connection of a source and a transport:

- A low capacity alarm in a source *will* cause a low flow alarm in a connected transport function.
- A high flow alarm in a transport function *may* cause a low capacity alarm in a connected source.

The method uses a set of causation rules like the ones above to analyze alarm situations and separate out those alarms that must be primary from those that might be secondary. However, no alarm is hidden from the operator, as there is always the possibility of multiple faults, i.e., a primary fault could look as if it was caused by another one.

The method uses a *consequence propagation* algorithm in order to handle unknown alarm states when parts of the process are not equipped with alarms. It is further described in Chapter 5.

Fault Diagnosis

This method uses the MFM model of a process to search for faults and give explanations and remedies, much like a standard rule-based expert system would do. The MFM model contains information about the goals of a process, how these goals are achieved by networks of functions, how the functions depend on subgoals, and how they are realized by physical components. In a standard rule-based expert system, this information structure is implemented in rules, but in MFM it is explicitly described. Thus, a fault diagnosis can be implemented as a search in the model graph. The strategy used is as follows:

- The user chooses a goal for diagnosis. If this is a top-level goal, the whole model, (and thus the entire process), will be investigated. However, the goal chosen could also be a subgoal, in which case only part of the process will be diagnosed.

- A search propagates downward from the goal, via the achieve relations, into the networks of flow functions, each of which is investigated.
- As in the alarm analysis, the flow functions are associated with working conditions, and each function may have a diagnostic question, which is asked in order to find out whether it is working or not. Alternatively, there can be a rule or a relation to a physical component to find out whether the function is in order.
- If a flow function conditioned by a subgoal is found to be at fault, or has no means of being checked, the connected subgoal is recursively investigated. If, however, a function is working, that part of the subtree can be skipped.

The method uses a search that propagates along static connections. Thus neither global search, pattern matching, nor conflict resolution is needed. It works together with the alarm analysis method and is described in Chapter 6.

1.6 The Architecture of a Control and Supervisory System

The three methods for diagnostic reasoning fit into different parts of a control and supervisory system. The measurement validation algorithm typically belongs on a rather low level, where it can be used to feed validated signal values to higher-level algorithms, such as the alarm analysis, the fault diagnosis, and other tasks dealing with supervisory diagnosis and control, see Figure 1.12.

The inputs needed can be obtained in several different ways. They can be direct or filtered signals from sensors. It would be more probable, though, that they were the outputs of some low level data filtration on the direct signals, e.g., outputs from a Kalman filter or from some statistical algorithm.

The measured flow values are assigned to the attributes of the appropriate flow functions. It is these flow values that the measurement validation algorithm operates on. The validated output values could then be used by algorithms on higher levels.

The alarm analysis would be found in the higher levels of the system, together with supervisory control, user presentation, etc. The fault diagnosis method would also be placed here.

A more thorough discussion about the architecture of a control and supervisory system is given in Chapter 7.

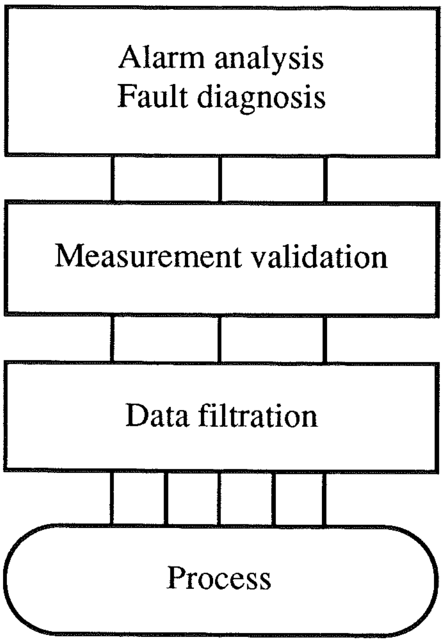


Figure 1.12 The places for the diagnostic methods in a control and supervisory system. Raw data from the process is treated by filters and low level numerical routines before it is sent on to the other algorithms. The measurement validation algorithm works on an intermediate level, while the alarm analysis and fault diagnosis algorithms belong in the topmost, supervisory level of the system.

1.7 A Guide for the Reader

This chapter gives an introduction to multilevel flow models and multiple views, and then presents three newly developed diagnostic methods. The chapter is suggested as an overview of the work. Chapter 2 contains an overview of literature and other projects.

Chapter 3 gives a description of the MFM concepts, the graphical language, syntax, and semantics. Examples of how to use MFM are also provided. This is intended as a quick course in multilevel flow modeling. The second half of the chapter describes some new developments of this work.

Chapters 4, 5, and 6 describe three new diagnostic methods, providing definitions, explanations of the algorithms, and examples. Chapter 4 treats measurement validation, while Chapter 5 handles alarm analysis and Chapter 6 fault diagnosis. These chapters are the main contributions of the work. Each may be read separately, but all need the introduction to MFM given in the first half of Chapter 3.

In Chapter 7, the implementation of an MFM toolbox is described. Here, information about the G2 implementation is found, together with a short overview of the G2 system itself.

Chapter 8 treats the problem of how to present means-end information in general and MFM in particular to users and operators. Some ideas and suggestions are given, together with several examples.

Chapter 9 contains the conclusions of the project and Chapter 10 references.

CHAPTER 2

RELATED WORK

MFM is a young and largely unexplored research area. This work is related to several other areas of research, such as (model-based) diagnosis, model-based reasoning, qualitative physics, and general modeling. This chapter will give an overview of previous work and related projects, and some overview papers and books are also mentioned.

	MFM	*Functional*	*Qualitative*	*Quantitative*	*Integrated*
General	Lind		de Kleer Forbus Kuipers Woods	Isermann Frank Woods	Årzén Krijgsman Crespo
Validation	Larsson Creutzfeldt			Mah	
Alarms	Larsson	Modarres Padalkar	Kramer	Lees	
Diagnosis	Lind Creutzfeldt Sassen Larsson Walseth	Modarres Padalkar Allen	Dvorak Ng	Petti	Vina Struss Mariño
Planning	Norby Larsen	Tomita			
Presentation	Lind Duncan Larsson	Rasmussen Modarres			
Simulation		Chéruy Bond graphs	Kuipers		

Figure 2.1 An overview of some different projects, according to model type used, (columns), and the type of problem solved, (rows).

In the following overview, the different projects has been sorted according to to the *type of models* they use, and in each group they are separated into different *problem areas,* such as fault diagnosis, simulation, presentation, etc. This has been summarized in Figure 2.1.

2.1 Projects Using MFM

LIND. Morten Lind is the creator of MFM. The most important document about MFM itself is Lind (1990 a), where the basic ideas, the syntax, and semantics are defined. The report gives a short introduction to the background of MFM, and works as the definition of Lind's current version. Some examples are also given. Chapter 3 of this work relies heavily on this publication. Lind (1987) is an earlier version of the report, while Lind (1979) is the original paper of MFM.

Lind (1990 b) describes Lind's solution for the data structures and diagnostic algorithms to be used under the MFM top layer. The implementation is done in Smalltalk 80. The use of Smalltalk gives the freedom to design the system precisely as wanted, but the programming effort needed is large compared to using a higher level tool such as G2, see Moore *et al* (1987, 1991). The basic data structures have already been implemented, while inference engines for diagnostic algorithms are under development. Together with a graphical interface for building MFM graphs, also written in Smalltalk, see Osman (1990), this forms the main effort of Lind's group.

Measurement Validation

LARSSON. The only MFM method that treats measurement validation as a separate problem is the one described in Chapter 4 of this work.

CREUTZFELDT. The project described in Creutzfeldt (1990) is partly concerned with measurement validation. The project treats sensor validation together with alarm analysis and fault diagnosis. It is a real-time diagnostic system, with low level data collection routines written in Fortran. MFM models are manually translated into Nexpert Object rules, which handle the high level processing. The system generates hypotheses and tests them by propagating values through the MFM graphs, thus performing a mixture of measurement validation, alarm analysis, and fault diagnosis. A working demonstration of the system exists, with a small heat distribution plant as target process, but the thesis is yet to be published.

Alarm Analysis

LARSSON. A method for alarm analysis with MFM is presented in Chapter 5 of this work, and it seems to be the only one separately concerned with alarm analysis. However, the project of Creutzfeldt, (see above), partly treats this problem.

Fault Diagnosis

Several projects use MFM for fault diagnosis.

LIND. Lind has described his approach to real-time diagnosis in Lind (1990 c), where it is argued that the structure of MFM models is well suited for fault diagnosis under time constraints. By searching top down in the MFM graphs, it is possible to obtain algorithms that produce an answer with low resolution quickly and then can use any additional time to increase the resolution of the diagnosis. Each goal corresponds to a part of the diagnostic task, and as lower level subgoals are met, the diagnosis becomes finer and finer, thus giving a behavior similar to that of *any-time algorithms,* see Dean and Boddy (1988) and Boddy and Dean (1989). No implementation has been suggested, though.

CREUTZFELDT. Fault diagnosis is also the main aim of the Creutzfeldt project, (see above).

SASSEN ET AL. The Dutch project PERFECT uses a preprocessor to translate MFM models into COGSYS programs for diagnosis, see Sassen *et al* (1991, 1992) and Sassen and Jaspers (1992). The implemented method uses external measurement values to check the working condition of every leaf node in the MFM graph, and then propagates the fault information upwards, thus enabling the system to quickly find low level faults, and then to use any additional time to give descriptions of the consequences on higher levels. An advantage with checking all leaf nodes is that they may be ordered according to failure likelihood, but a drawback is that all leaf nodes must be continually checked. The project has several points in common with the method described in Chapter 6 of this work, and further comparisons will be made there. COGSYS is a real-time expert system shell developed as a cooperation between 35 British and European companies. It is written in POP-11 and C, uses the text-based language KRL, (Knowledge Representation Language), to describe the knowledge database, uses a blackboard architecture, and is designed to be quite efficient, see COGSYS (1990). The PERFECT system works as a compiler and translates MFM models into code for the COGSYS system.

Sassen is part of the SCWERE project, (Supervisory Control With Embedded Real-time Expert systems). This is a joint project between the faculties of Informatics, Electrical, Mechanical, and Chemical Engineering of Delft Technical University. The aim of the project is to support plant-wide control systems on a supervisory level using AI techniques in real-time, see for example van den Ree *et al* (1991 a, b), and Terpstra *et al* (1991, 1992).

LARSSON. Chapter 6 of this work presents a method for fault diagnosis using downward search in MFM graphs.

WALSETH. The Norwegian project of Walseth *et al* uses MFM for diagnosis of a water/ammonia separation unit, see Walseth *et al* (1992). The main contribution so far is the idea of connecting MFM goal satisfaction with tests of quantitative state variables in the functions. This implies a change in the semantic rule for goal satisfaction in MFM. The project is still in the starting phase.

Planning

NORBY LARSEN. One project of Lind's group uses MFM to control the production of STRIPS plans for startup of plants, see Norby Larsen (1990). The standard way of using STRIPS for planning is to perform a search among applicable operators, in order to construct a viable plan, i.e., a sequence of actions that takes the process from initial to goal state. MFM contains information about which functions that must be available for a goal to be achieved, and the implemented system uses this information to control the, (otherwise blind), search through operators. Sometimes this needs guessing and backtracking, and truth maintenance techniques are used. For readings on STRIPS, see Fikes and Nilsson (1971).

Presentation

LIND. Lind has also treated presentation of means-end information and the design of operator interfaces, see Lind (1989). The main idea is to use the graphical representation of the MFM models, combined with flow sheets and highlighting of functions and corresponding physical components. Another set of symbols and a graphical environment were developed in the SIP project, see Lind *et al* (1987), but the new symbols have not been put to further use; this thesis uses Lind's older versions. A part of the effort of Lind's group is the development of a general graphics environment for MFM, see Osman (1990).

DUNCAN AND PRÆTORIUS. Duncan and Prætorius (1989) use MFM as an alternative way of presenting diagnostic information to operators, and have made very interesting comparative experiments with students acting as unexperienced operators. The students received a few hours of training to find faults in an example process using either a standard flow sheet or an MFM model, and in this test, MFM proved to be more efficient than a flow sheet as a fault diagnosis aid. It should be noted, however, that the test did not involve trained operators, and that the test series was small. In spite of this, the result is very interesting, and clearly provides a good reason for further work with MFM as a means of presentation. To perform the test, Duncan and Prætorius developed a graphical presentation system for MFM.

LARSSON. Some ideas about and examples of presentation of means-end information are shown in Chapter 8 of this work. The conclusion is that MFM may be suitable for presentation, when integrated in a multiple view system, allowing several ways of presenting the same and related information.

2.2 Projects Using Other Functional Models

Several projects have used means-end and functional models, which are not pure MFM, but closely related. This means that some representation of goals, functions, or both is used; usually a tree or graph describing a hierarchy of goals or functions.

Alarm Analysis

MODARRES ET AL. The Goal Tree Expert System, (GOTRES), project uses a database of goal trees and success trees to perform diagnostic tasks. The representation consists of tree structures containing goals and subgoals on the higher levels, and hardware requirements, i.e., what components that must be working, on the lower ones. Thus, it clearly resembles MFM. This data structure has been used to implement the UMPIRE-I program, that helps to evaluate alarm systems, see Modarres and Cadman (1986). The system is written in CommonLisp and runs on an IBM-PC/AT. It has been successfully tested on real processes.

PADALKAR ET AL. The Intelligent Process Control System, (IPCS), uses hierarchical models of structure and function to perform fault diagnosis, see Padalkar *et al* (1991). The system models fault propagation in graphs, and thus in fact performs an alarm analysis. The target process is described in hierarchical tree structures with a functional representation of functions and subfunctions, and a structural representation of systems and subsystems. Constraints are used to find faulty components, and this information is then propagated downwards in the hierarchies, to find the lower level causes. In this way, a diagnosis with low resolution is quickly available, and then any extra time is used to improve the granularity of the diagnosis. The system has been tested successfully on a small power plant producing electricity and steam at the Senboku Works of Osaka Gas Company in Osaka, Japan.

Fault Diagnosis

MODARRES ET AL. The GOTRES system has also been used to perform fault diagnosis. The implemented program uses a depth-first search downwards in goal trees to find failures in equipment found in the leaf nodes, see Chung and Modarres (1989). The system has been used to construct an on-line fault diagnosis expert system for an experimental nuclear reactor facility, and the results seem to have been satisfactory.

PADALKAR ET AL. The IPCS system of Padalkar *et al* (1991), deserves to be mentioned under fault diagnosis too, as it performs a mixture of alarm analysis and fault diagnosis.

ALLEN AND RAO. Fault trees describe a process and its possible faults in a tree structure, where the top levels of the trees contain alarms, while the lower levels contain components and subcomponents. A diagnosis consists of a search path through the tree from the root to one or several leaves, where the primary faults are found. See for example Allen and Rao (1980).

Planning

TOMITA ET AL. The project of Tomita *et al* describes an automatic synthesizer of operating procedures based on functional models of plants. The system uses a heuristic search through automatically generated subgoals to construct operating sequences for chemical plants. The goal is to give operator support. The database contains directed graphs to give a qualitative description of the plant behavior, fragments of operating sequences, called *scopes,* which are used to construct plans in a bottom-up fashion, and *scripts* to help build plans top-down. Working plants are described as networks of scopes, where each scope corresponds to an abstract function or subfunction of the plant. Scripts describe the conditions for scopes to work, and are used as guidelines for constructing plans, consisting of sequences of scopes. The system has been applied to the startup of a practical chemical plant: a subprocess of an existing ethylene plant, and this application seems to have been successful, see Tomita *et al* (1989).

Another system is specifically aimed at batch processes, see Tomita *et al* (1986). Here the data structure used is a tree of tasks and subtasks, which must be performed to ensure successful operation. The process itself is described as a directed graph, showing the physical topology of the plant. The operational knowledge is contained in a table of recipes, i.e., description of how to perform each possible task. The system uses the tree of needed subtasks to schedule a set of operations, using a linear programming technique, and then performs a quick simulation to check the result.

Presentation

RASMUSSEN. Rasmussen has thoroughly discussed the design of man-machine interfaces and describes the importance of presenting means-end information in order to accomplish this, see for example Rasmussen (1986) and Rasmussen and Goodstein (1988). Rasmussen has done a greatly original work of structuring the tasks of operators and other users of man-machine systems, and of the behavior used for the tasks. Some tasks are usually performed on a *skill-based* level, which means a more or less automated or reflexive behavior. Other tasks, notably some kinds of diagnosis, are performed on a *rule-based* level, where reasoning is needed but the knowledge is expressed on a rather simple, symptom-action form. Yet other tasks, and especially the most complex and difficult ones, are performed on a *knowledge-based* level, which implies complex reasoning with the use of models. This structure is of paramount importance, since the different task levels fit differently well to be performed by operators and for being automated. The demands on an implementation are also very different depending on the task type; thus a skill-based task may be solved by conventional control techniques, while a rule-based task may be better solved by an expert system, and a knowledge-based task by a more advanced system for automated reasoning. Rasmussen's contributions are very rich; the reader is referred to, e.g., Rasmussen (1986) for an extensive overview. Rasmussen and Lind (1981) is a starting point for the ideas behind MFM.

MODARRES ET AL. The GOTRES representation has also been utilized as a database for an operator advisory system for operation of nuclear power plants, see Kim and Modarres (1987). In this project, a Prolog implementation was used for building a small expert system.

Simulation

Several projects use functional models for generation of equations and simulation. The general idea here is to use a more abstract representation of a system's behavior, and to move away from physical detail which is not needed, in order to produce equations, which then may be used in modeling or simulation.

CHÉRUY ET AL. The system CAMBIO uses graphical diagrams to give a functional description of biochemical reactions. The different types of reactions and media affected by the reactions are represented by graphical symbols, and the reaction structures are described by connections. Thus, the system lets a user design a compound reaction by building a graph on a computer screen. The system reads this graph and can produce equations semi-automatically and then perform a simulation. Some extra information must be given manually, e.g., about the order of the different reactions. The CAMBIO representation may provide an interesting connection to MFM, as described in the latter half of Chapter 3. For readings on CAMBIO, see Chéruy, *et al* (1989), Montellano *et al* (1990), and Farza and Chéruy (1991). The system has been implemented in Pascal, but it is unclear how successful it has been.

BOND GRAPHS. From the study and systematic use of block diagrams as process representation comes the concept of *bond graphs,* see Paynter (1961), Karnopp and Rosenberg (1975), or Thoma (1975). They display both energy and signal exchanges between components in processes. Each connection is described as a product of two variables, the *effort* and the *flow.* The bond graphs serve as a graphical representation in several disciplines, often corresponding to equations. In electronics, the effort corresponds to the voltage, while the flow corresponds to the current; thus, the product is the effect. In mechanics, the effort is the force or torque and the flow the velocity of rotation frequency, while in thermodynamics the effort is the absolute temperature and the flow the entropy, see Thoma (1975).

The elements connected are *functional* components of several kinds. They may be *simple bonds* corresponding to wires, shafts, or rods, *resistance elements* describing for example resistors, *inertia elements* corresponding to inductors, flywheels, or masses, *capacity elements* which match condensors or springs, *effort sources, flow sources, transformers,* etc. There are also two kinds of junctions, *p* and *s* junctions, for parallel and series connection. The causality between different components are shown with a system of arrows and bars.

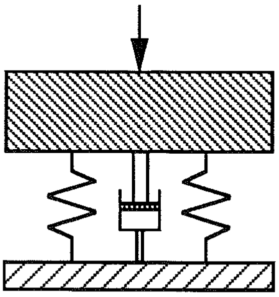


Figure 2.2 A simple mechanical system with a mass, a spring, and friction. The mass is seen as a point mass, and the excitation is a force independent of velocity. From Thoma (1975).

With these building blocks, it is possible to describe electrical, mechanical, thermodynamical, and other systems. A mechanical system is shown in Figure 2.2.

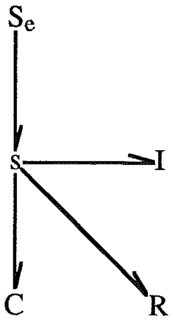


Figure 2.3 A bond graph of the system in Figure 2.2. The excitation is S_e. The spring is modeled as a capacity element, the friction as a resistance, and the mass as an inertia element. The connection is a series junction, as the velocities are equal and the different forces added. From Thoma (1975).

This system is a simple example from mechanics, and has a point mass, a linear spring, and some friction. It may be described by a simple bond graph, see Figure 2.3.

An augmented bond graph of the system is shown in Figure 2.4. Here the causality in the system is also described. The forces and velocities are shown in the graph.

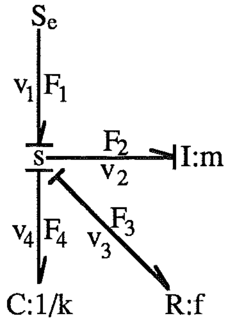


Figure 2.4 An augmented bond graph of the system. The force of the effort source acts on the mass which responds with the velocity v_2, but after deducting the spring and friction forces, which are functions of the velocity. From Thoma (1975).

From bond graphs it is possible to automatically generate differential equations or transfer functions. Bond graphs have some superficial properties in common with MFM, thus they are both built from abstract *functions* connected into flows, and may be used to generate balance equations. There, however, the similarities end. The major differences between bond graphs and MFM are as follows:

- Bond graphs have no achieve or condition relations, and thus cannot represent the means-end dimension, which is the main point of MFM.
- The functions are not the same. Thus, bond graphs have no barriers, while MFM have no capacity elements, etc.
- The flow paths of bond graphs are described by efforts and flows, and the actual values are usually not decided beforehand but solved for, as equations are solved. MFM flows are built from pure flow variables, and the flow values must be known beforehand; the *normative* nature of MFM.
- The intended use is vastly different. Bond graphs are often used as a graphical representation of equations, while MFM is mainly concerned with diagnostic reasoning.

This leads to the clear conclusion that bond graphs and MFM has very little to do with each other. As bond graphs do not encompass the most important aspect of MFM, it would be misleading and potentially dangerous to use them in trying to understand it.

One connection is possible, though. If good bond graph descriptions of a process are available, they represent energy balances. These flows may be used when building the MFM models, provided that the constructor is aware of the danger of such an approach, as will be described in Section 3.8, on building MFM models.

2.3 Projects Using Qualitative Behavioral Models

There are many model-based diagnostic methods based on combinations of AI techniques and behavioral models. As the automatic reasoning usually does not need the quantified detail of a mathematical model, and because sometimes only knowledge about qualitative behavior is available, several qualitative representations have been suggested to replace mathematical models in physics. The common approach is to use some representation based on, e.g., logics, constraints, or directed graphs. This area of *qualitative physics* is nicely described in Forbus (1988) and Weld and de Kleer (1990).

The main approaches to qualitative reasoning have been taken by de Kleer and Brown (1984), which points out the need for both topological and functional models and formulates a theory based on confluences, by Forbus (1984), which describes a modeling method starting with a physical description of a system and giving a set of constraints, and by Kuipers (1984, 1986, 1989), which describes a qualitative simulation method, starting with a set of qualitative constraints and an initial state, and which can predict the set of possible futures for the system.

In the Hybrid Phenomena Theory, (HPT), Woods describes a combination of qualitative and quantitative models, see Woods (1991) and Woods and Balchen (1991). HPT is an attempt to combine state space models with a symbolic framework derived from the Qualitative Process Theory, (QPT), see Forbus (1984).

Alarm Analysis

KRAMER ET AL. The Model Integrated Diagnosis Analysis System, MIDAS, basically performs alarm analysis, with extensions towards fault diagnosis. It uses qualitative information about process variables described in graphs. For readings on MIDAS, see Oyeleye (1989) and Finch (1989). An alternative implementation in G2 of part of the MIDAS system is described in Nilsson (1991), from where the example below is taken.

MIDAS is a qualitative method for finding deviations from a nominal steady state. The incoming measurements are turned into alarms, which are grouped into clusters that belong to the same primary fault. In order to do this, MIDAS uses a chain of different models, where each is translated into the next one more or less automatically.

The first type of model used is the Signed Directed Graph, (SDG), which is derived from the physical equations of the process. State variables are represented by nodes, and qualitative relationships by arcs. SDG also contain the total set of *root causes,* the possible primary faults. The SDG graphs are transformed to Extended SDGs to handle global feedback loops. The ESDGs are used to generate Event Graphs. In these, a set of events, i.e., qualitative state changes, are linked together with a root cause.

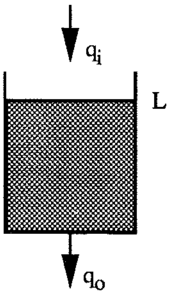


Figure 2.5 A small gravity tank. The level, L, of the tank is controlled by the inflow, q_i, and outflow, q_o, and there is a flow resistance, R, in the outflow pipe.

Consider the gravity tank in Figure 2.5. The level, L, of the tank is controlled by the inflow, q_i, and the outflow, q_o. In the outflow pipe, there is a flow resistance, R, that may vary. The balance equations of the tank are

$$\dot{L} = c_1 q_i - c_2 q_o$$

and

$$q_o = f^{-}(R)\sqrt{L}.$$

The SDG produced from these equations is found in Figure 2.6.

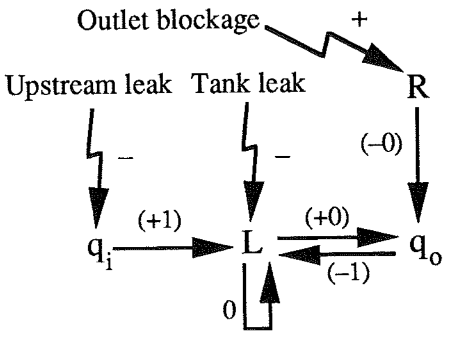


Figure 2.6 A Signed Directed Graph describing the gravity tank. The variables q_i, L, q_o, and R are shown, together with arcs that describe how a change in one of the variables will affect the others. From Nilsson (1991).

The SDG model is translated into an Extended SDG to handle global feedback loops and then used to produce an event graph of the gravity tank, see Figure 2.7.

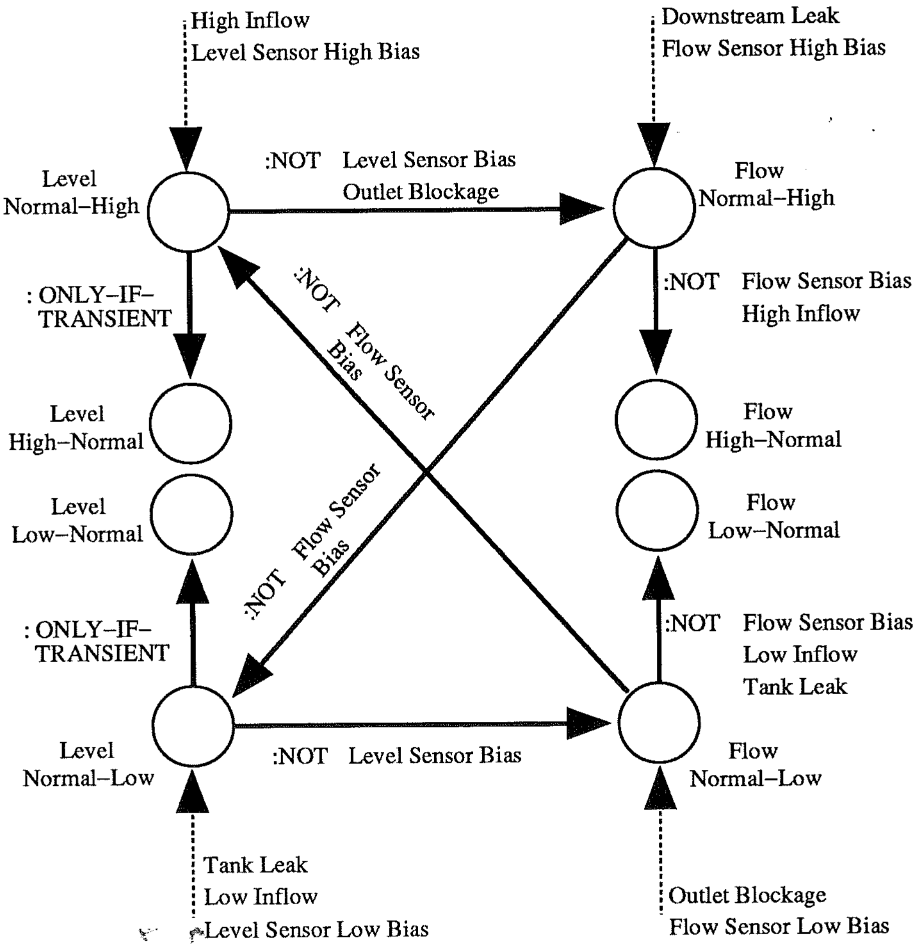


Figure 2.7 An event graph describing the gravity tank. There are eight events and four root causes. This is the data structure that MIDAS uses on-line. From Nilsson (1991).

MIDAS supervises all measured variables with a set of monitor procedures. These send qualitative messages to an event interpreter, which uses the event graph representation to construct an on-line graph of the actual events, their links, and root causes. Thus, the alarms are analyzed and connected. The architecture of the MIDAS on-line system is shown in Figure 2.8.

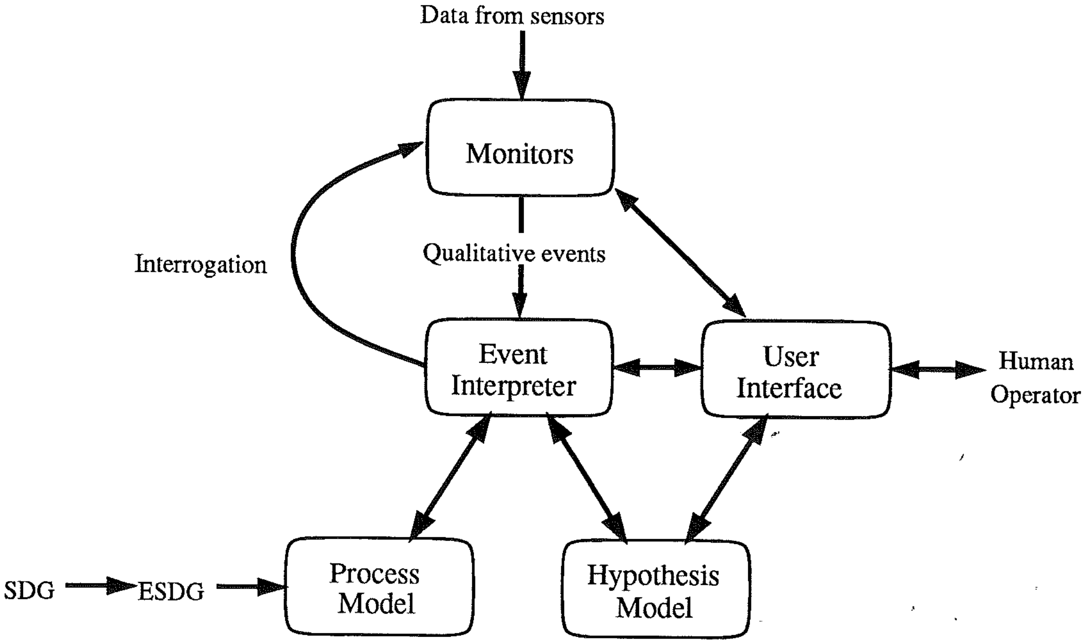


Figure 2.8 The architecture of the MIDAS on-line system. The input signals are sent to monitors, which turns them into qualitative events. The event interpreter receives all events and uses them to construct an on-line event graph, which represents the current situation in the process. From Nilsson (1991).

The MIDAS system has some similarities to the alarm analysis algorithm of Chapter 5, and the two algorithms will be compared in that chapter.

Fault Diagnosis

Several projects within AI have used model-based reasoning to perform fault diagnosis. Davis and Hamscher (1988) gives a good overview, focusing on diagnosis of electronic circuits. Torasso and Console (1989) gives a more theoretical overview and relates to standard expert system techniques.

Dvorak and Kuipers. The MIMIC system, see Dvorak and Kuipers (1991), Dvorak (1992), is based on the QSIM modeling language. It uses

QSIM models to perform a semi-quantitative simulation of a system, i.e., a qualitative simulation which uses quantitative information about some values and relationships. The system compares the simulation and the real process, and can track one or several models, each corresponding to a working or fault state. The system introduces several new ways of using alarms, basing them on the model instead of the process. It has been tested on small example processes and seems to be working well on these.

NG. The Inc-Diagnose project described in Ng (1991) uses a set of QSIM qualitative states and a new version of the diagnostic algorithm of Reiter, see Reiter (1987), to perform fault diagnosis of simple physical systems. Reiter's theory describes a diagnosis problem as a system description, *SD,* a set of components, *Comp,* and a set of observations, *Obs.* In Reiter's formulation, *SD* is usually a set of first-order logical formulas, but Ng instead uses QSIM constraints. Viewed as a formal problem, diagnosis is a NP-complete problem, but Ng gives an algorithm that he claims is reasonably efficient. The method has been tested on a temperature controller, a pressure regulator, and a toaster; small processes, where the execution times were a few minutes on an Explorer Lisp machine.

Simulation

KUIPERS. Kuipers has developed the language QSIM for qualitative simulation, see Kuipers (1984, 1986, 1989). A system is described as a set of qualitative constraints with a given initial state, and QSIM can then compute the possible future behaviors of the system, by producing a tree of all possible, qualitative states. The expressive power of QSIM supports such relationships as addition, subtraction, and derivation, while others only state a monotonical functional relationship. This idea can be seen as an abstraction of differential equations. The QSIM representation has been used to build the MIMIC diagnostic system, (see above).

2.4 Projects Using Quantitative Behavioral Models

The classical models of control theory are quantitative; usually differential equations. Process fault detection using mathematical models and parameter estimation, extended with knowledge-based reasoning, so called observer-based methods, is overviewed in Isermann (1984). Fault detection with classical methods is given excellent overviews in Frank (1990, 1991, 1992). The HPT theory of Woods deserves to be mentioned here too, as it is an attempt to join qualitative and quantitative models. In this group there are many results and only a few examples will be described.

Measurement validation

MAH ET AL. The classical form of data reconciliation uses statistical methods in order to find the most probable values of a set of interdependent sensors, see for example Mah (1990) for an instructive description. There are essentially two kinds of methods, those that handle small, random errors and those concerned with finding gross errors.

An example of the first type of method is found in Mah (1990). The assumed model is

$$y = x + \varepsilon,$$

where y is a vector of measurements, x is a vector of true flow rates, and ε is a vector of random errors. The system is described by an *incidence matrix, A*, which describes the process as a set of nodes and directed arrows. The nodes correspond to the rows and the arrows to the columns of the matrix, and a -1 marks an inflow to and a 1 an outflow from a node. The errors, ε_i, are described by a covariance matrix, Q, which should be positive definite and known. The data reconciliation problem can then be formulated as a constrained weighted least-squares estimation problem:

$$\min[(y-x)^T Q^{-1}(y-x)]$$

subject to the constraint

$$Ax = 0.$$

An approximation, $\hat{x}$, of the true flow values, x, is given by

$$\hat{x} = y - QA^T(AQA^T)^{-1}Ay.$$

The use of linear programming techniques, interval arithmetic, see Himmelblau (1987), and fuzzy logic has also been suggested.

The most common techniques for treating gross errors are based on statistical hypothesis testing, for example *chi tests*. There are global tests, see Almasy and Sztano (1975), and nodal tests, see Mah *et al* (1976), which finds out whether a gross error is present, but which require some other method to find out where the fault is, and direct tests on each measurement, see Mah and Tamhane (1982), which instead require a previous data reconciliation. These methods are usually combined with a search for the erroneous measurements using some kind of elimination of suspicious values and retesting. Common to all methods are that they need a preset significance level, a covariance matrix must be known, they are probabilistic and thus may fail, and they usually find the most probable fault hypothesis only, instead of giving all possibilities. In Chapter 4 these methods will be compared to the MFM-based method developed in this work.

Alarm Analysis

There are many methods for alarm analysis based on quantitative techniques. See for example Lees (1983) for an overview.

Fault Diagnosis

PETTI. The Diagnostic Model Processor method has been developed by Tom Petti at the University of Delaware, see Petti *et al* (1990), Petti and Dhurjati (1991), and Petti (1992). This system uses quantitative equations to find a set of violated working assumptions, i.e., a set of faults, and its internal structure in much resembles a neural network. Petti *et al* (1990) describes the DMP methodology, while Petti and Dhurjati (1991) shows an implementation and some examples of the use of the method.

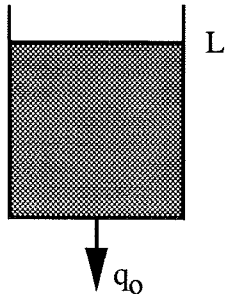


Figure 2.9 A tank with a gravity outflow. The outflow depends on the level as $q_{out} = \alpha\sqrt{2gh}$. Both the level and the outflow are measured, and can thus be used in a model equation.

DMP uses model equations and available measurements to arrive at the most likely fault conditions. It assumes that during fault free operation, all model equations agree with the real measurements. By analyzing in what direction and to what extent each model equation is violated, the most likely failed assumptions can be deduced, and each case of redundancy helps to make the diagnosis more certain. The process model consists of a set of equations written on residual form, i.e., so that they ideally equal zero. Each equation also has tolerance limits, representing the upper and lower limits for which the equation is satisfied. With the use of the tolerance limits and a sigmoidal function, the residual of each equation is turned into a number between −1 and 1, telling how much the equation deviates from the ideal value. An example of a tank with a gravity outflow is given in Figure 2.9.

The outflow of the tank is given by:

$$\varepsilon = q_{out} - \alpha\sqrt{2gh},$$

which is written in residual form. Each equation depends on a number of *assumptions,* i.e., conditions which must be fulfilled in order for the equation to be satisfied. The assumptions may be explicit, such as correct sensor readings for values immediately visible in the equations, or implicit, e.g., that there are no leaks or blocks in the piping, etc. The assumptions for the tank equation are:

- The level sensor is working
- The outflow sensor is working
- No leaks in the tank or pipe

Each equation is related to assumptions via connections, which state the sensitivity of the equation for a fault in this assumption.

Finally, failure likelihoods, F_i, for each assumption may be computed, by a combination of the satisfaction values of the connected equations. The deviation from zero indicates both how much and in what direction the assumption fails, i.e., whether a fault has been found. A DMP model of Steritherm is shown in Figure 2.10.

DMP has been shown to be a simple and useful diagnostic method. An advantage is that it is relatively easy to change the DMP data structure when a process is rebuilt. Due to the summing and weighting methodology, it is rather insensitive to the setting of tolerance levels, and there are no problems of turning quantitative measurements into boolean values. DMP can handle multiple faults, but it will find only one possible hypothesis, which may be potentially problematic.

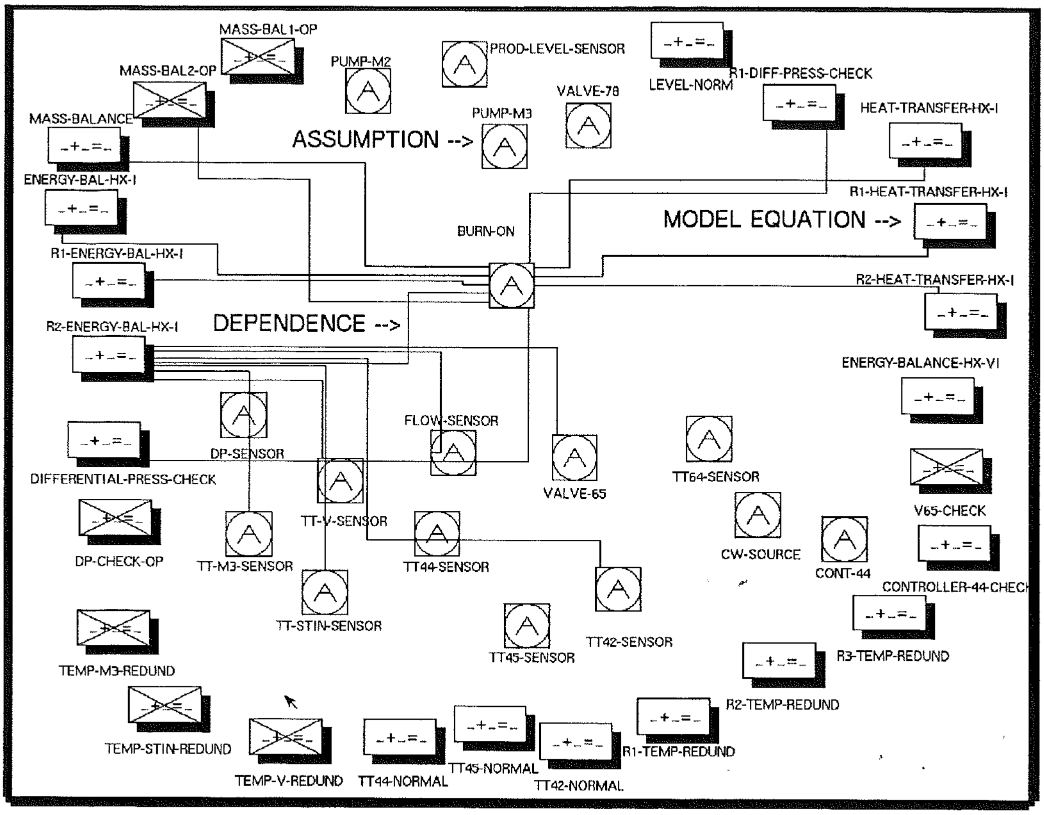


Figure 2.10 A DMP model of Steritherm. The picture contains the model equations and the assumptions, and some of the connections are also shown.

2.5 *Projects Using Integration of Different Models*

As different model types are more or less well suited for different tasks, it would be advantageous to use several model types in a mixed representation. Some projects have focussed on the integration of different methods and model types.

ÅRZÉN ET AL. The Knowledge-Based Control System project described in Asea Brown Boveri *et al* (1988, 1990, 1991), Larsson (1990 a), Årzén (1989, 1990, 1993), and Årzén *et al* (1990), suggests a general architecture for a knowledge-based system accommodating all the tasks needed in a full control and supervision system. The tasks currently handled are low level loop control, sequence control, alarm and control logic, and quantitative simulation, as well as supervision and diagnosis based on components for realizing the functions. This use of abstraction is as common in human understanding and decision making, but has not yet come to much use in modeling or programming.

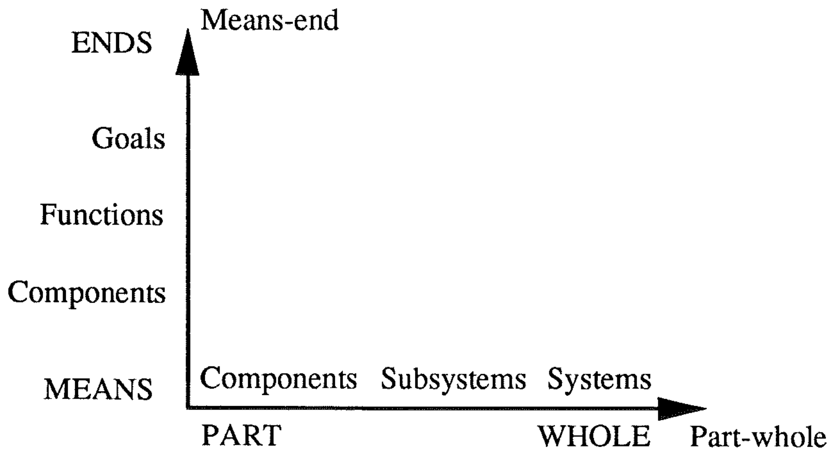


Figure 3.1 The means-end and part-whole dimensions. Abstraction in the part-whole dimension allows for composition of smaller components into larger systems, and decomposition of the systems into subsystems. In the means-end dimension, goals are related via achieve relations to functions, and functions via realize relations to components.

It is important to observe that the two dimensions actually are separate, thus a goal could be broken down into subgoals, a function into subfunctions, and a component into subcomponents.

A Motivation for MFM

The use of multilevel flow models can be motivated in several ways, although the basic one is that they provide a basis for design of intelligent control systems. In this, four tasks can be recognized:

- Design of real-time knowledge-based systems
- Design of man-machine interfaces
- Design of control systems
- Models for control systems development

The design of man-machine interfaces to processes and control systems is leaving the *one sensor-one indicator* philosophy, with more or less success. A natural objective here is to use new techniques for presenting information about the process state to the operator, and also to present new types of information. As an operator often reasons about the goals and functions of a process, MFM can provide a basis for *explicitly representing* this kind of information, and maybe also for the *presentation* of the information to the operator, see Lind (1989).

MFM is a powerful tool for constructing knowledge databases, and it enables the writing of algorithms that can safely and efficiently handle real-time demands, both when concerning concurrent execution of diagnostic tasks and when it comes to hard real-time, i.e., when a task must be finished inside a specified time interval. The methods described in this work are all good examples of knowledge-based real-time algorithms. Hard real-time problems and solutions, i.e., the problem of making sure that a diagnostic system will have a satisfactory solution ready in a certain time, have been described in Lind (1990 c).

MFM can also be useful as a design tool, both for process design and for control system design. In the latter, the following phases can be more or less clearly observed:

- Analysis of demands, goals, functions, and components
- Design and choice of algorithms
- Hardware and software implementation

The first phase is usually performed in an informal way, while the other two will have to be formal. However, MFM could be used for a formal description of all the three phases, thus turning the informal design engineering into a well-defined task and making the knowledge involved become explicit.

The design of processes and control systems rests firmly on the use of laws of physics, chemistry, etc. These models are well aimed at describing the behavior of a system, but they are not so good for some problems concerning diagnosis and supervision, that demands strategical and tactical decisions on management of resources. Here MFM could also provide the corresponding modeling language.

3.2 The Basic Concepts

An MFM model is a normative description of a system, a representation of what it has been designed to do, how it should do it, and with what it should do it. Thus, the three basic types of objects in MFM are:

- Goals
- Functions
- Physical components

The *goals* are the objectives or purposes of the system, i.e., the ends that the constructors and operators want the system to reach. The *functions* are the means by which the goals are obtained, i.e., the powers or capabilities of the system. The physical *components* are what the system is constructed from, the equipment of which it consists.

The goals, functions, and components depend on each other in specific ways. Thus, in MFM there are different types of relations, that can be used to connect the objects:

- Achieve relations
- Condition relations
- Realize relations

An *achieve* relation connects a set of functions to a goal, and it signifies that these functions are used to obtain that goal. For example, the function of transporting water into a tank could be used to achieve the goal of keeping the volume of water in the tank at a certain value.

A *condition* relation connects a goal to a function, and signifies that the goal must be fulfilled in order for the function to be available. For example, this would be the case for a mass transport function realized as a pump, where the subgoal of transporting electrical power to the pump must be fulfilled in order for the pump to drive the mass flow.

A *realize* relation connects a physical component to a function, and signifies that the component is used to realize or implement the function. For example, a pump could be used to realize a mass transport function.

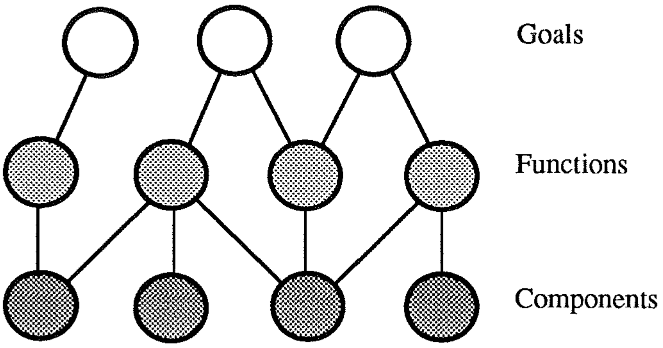


Figure 3.2 Goals, functions, components, and relations. The functions are connected to goals via achieve relations, while the components are connected to functions via realize relations. Subgoals may be connected to functions via condition relations, but this is not shown in the figure. All three kinds of relations are *many-to-many.*

It is important to observe that all the relations can be many-to-many. Several functions can be used to achieve one goal, one function may satisfy several goals, one goal can be a condition to several functions, one function may be conditioned by several goals, one function can be realized with many different components, and one component can implement several different functions, see Figure 3.2.

3.3 Goals

The concept of a goal is central to MFM. It is important to be able to recognize and describe goals, as they play an important part in every activity using means-end information. Without knowing the goals, it is more or less impossible to know the available functions. A broad definition of a goal is as follows:

> *A goal is the outcome towards which certain activities of a system are directed.*

However, this definition is very general, and it will be useful to try and narrow it down to more specific descriptions. Thus, three different types of goals will be recognized:

- Production goals
- Safety goals
- Economy goals

Production Goals

A production goal is used to express that to enable production, some specific process variable should be kept within a specified interval, i.e., that an inequality of the following general form should be satisfied:

$$x_0 \le x \le x_1.$$

Of course, one of the limits could be infinitely small or large. This means that the system will be kept in a certain state, where production is indeed possible.

Safety Goals

A safety goal is used to express that for reasons of safe operation, some specific process variable should be kept above or below some value, or inside or outside an interval, i.e., essentially the same test as for a production goal. However, a more common case is probably a one-sided interval. In practise, this means that some process variables should be kept within safe regions, far enough from dangerous values.

fault trees, symptom-based diagnosis, MIDAS, MFM, and DMP. All this is contained in a multiple view data structure. A demonstration system has been implemented in G2.

KRIJGSMAN ET AL. The DICE system, Jager (1990), Krijgsman *et al* (1990, 1991), and Krijgsman and Jager (1992), is a real-time expert system for control systems. It is based on a blackboard architecture and represents its knowledge with production rules. Boolean and multivalued logic based on fuzzy logic ideas are available, together with a truth-maintenance system. The system is written in C and runs on VAX/VMS platforms.

CRESPO ET AL. The RIGAS system, see Crespo *et al* (1991, 1992), is a real-time expert system based on a blackboard architecture. Some knowledge sources are assigned to solve control subproblems, while others, called processes, handle the activities needed, such as responding to external and internal stimuli. They may be either periodic or sporadic, and react to, e.g., requests from the operators.

Fault Diagnosis

VINA AND HAYES-ROTH. Vina and Hayes-Roth (1991) use a set of different models to build a real-time knowledge-based system using a blackboard architecture. The *prime* models, (each describing a component), are organized in five levels of information: a structural level, which describes the physical structure of the component; a functional level where the processes performed by the component are represented; a parameter level, where all parametrization of the component takes place; a sign level, which is a qualitative description of the system and its parameters; and a fault level, which defines all causes of faults, i.e., erroneous signs. With the prime models, domain models are constructed. These have three levels of information: physical connectivity, functional connectivity, and sensor location. The system uses these models off-line to precompute a hierarchy of abstract models which are analyzed to find a worst-case timing estimate. This information is then used on-line, to choose the appropriate model for real-time simulation and diagnosis.

Vina and Hayes-Roth test this system on two control systems, RCIC, an auxiliary subsystem for a power plant cooling system, see Hewett (1990), and GUARDIAN, a system for patient monitoring in a surgical intensive care unit, see Hayes-Roth *et al* (1989). The precomputed models enable the system to perform simulation and model-based diagnosis in real-time. The system has been implemented in a blackboard architecture developed for control problems, see Hayes-Roth (1985, 1990). This blackboard architecture is specifically designed to handle reasoning with hard and soft deadlines, i.e., when the result of some AI-based algorithm will be useless or less valuable after a certain time. The system has been used in real-time control applications in semiconductor manufacturing, see Murdock and Hayes-Roth (1991).

STRUSS. Struss (1987, 1991, 1992) describes a fault diagnosis system using multiple representation of physical structure and function. The system is concerned with diagnosis of electronic circuits, and the importance of using multiple views is pointed out. An assumption-based truth maintenance system, (ATMS), see de Kleer (1986 a, b), is used to keep track of constraints between different levels in the structural and functional hierarchies, and it seems to work well in the domain of analyzing simple electronic circuits. The long term goal of the work is to arrive at a general theory of diagnosis.

MARIÑO ET AL. Mariño *et al* (1990) describes a fault diagnosis expert system with a general multiple-view representation. The system is object-oriented and used for classification problems. Here, the inference engine is designed to switch between different views during the diagnostic search, and the contact points between the views are described by *bridge* objects.

2.6 Other References

The main documentation of the current project is this work, while some more detailed information about the implemented toolbox is found in Larsson (1992 c). The three main chapters each correspond to a conference paper. Thus, Chapter 4, measurement validation, has been presented as Larsson (1992 a), Chapter 5, alarm analysis, as Larsson (1991 c), and Chapter 6, fault diagnosis, as Larsson (1992 b). The alarm analysis has also been presented in Larsson (1991 a), and the fault diagnosis in Larsson (1991 b). The early plans and ideas was presented in Larsson (1990 b, d, e, f).

For readings about expert system techniques in general, see Brownston *et al* (1985), Harmon and King (1985), Hayes-Roth *et al* (1983), Stefik *et al* (1982), and Waterman (1986). Torasso and Console (1989) also contains information about standard expert system techniques, while Shortliffe (1976) describes MYCIN, the premier example of rule-based diagnosis using backward chaining.

A full overview of model-based diagnosis, mainly form the AI point of view, is given by Hamscher *et al* (1992).

The ideas for modeling information flows used in this work are taken from Lind (1988).

The basic idea of MFM, to give an formal and explicit description of means and ends, is interesting in other contents too, for example in the philosophy of mind. Dennett (1987) contends that while all systems may be described in physical terms, some systems, (for example human beings), can also be described in terms of *intentionality,* i.e., as having a goal-directed behavior. The latter is an essential property of intelligent life. Dennett have no formal way of describing intentionality, however, but it is possible that MFM may provide some useful ideas. The philosophy of mind will not be further pursued in this work, however.

CHAPTER 3

MULTILEVEL FLOW MODELS

Multilevel flow modeling, (MFM), is a systematic and reasonably well-developed technique for building means-end models. It consists of a set of concepts, a graphical representation of these, a specified syntax, and some suggestions for semantics. This is the base for the methods developed in this work. Thus, before the methods can be described, the basic facts about MFM will be presented.

Fuller descriptions and discussions can be found in Lind (1990 a, b). The basic syntax and semantics are thoroughly described in Lind (1990 a), while Lind's suggestions for semantics and data structures for diagnostic reasoning are found in Lind (1990 b). These two works are the basis of MFM, and the present chapter is a condensed version of Lind (1990 a). The version of MFM described here differs from the original formulation of Lind in a few minor details. These will be noted when they occur.

3.1 Means-End Modeling

The basic idea of MFM is to model a system as an artifact, i.e., a man-made system designed and used with certain purposes in mind. It provides hierarchical abstraction in two different dimensions, the *means-end* and *part-whole* dimensions. The part-whole decomposition of a system is well known and used in many hierarchical representation systems, while the means-end decomposition is typical of MFM, see Figure 3.1.

In the part-whole dimension, the abstraction means that a system can be viewed as a whole or decomposed into subsystems. The subsystems can be further decomposed, until the leaf-components are reached. This use of abstraction is a well-known way of understanding and operating complex system; to avoid being drowned in complexity.

In the means-end dimension the global goals of a system are related to functions for achieving the goals, and the functions are related to

Economy Goals

An economy goal is used to express considerations of overall process optimization. Thus, it is typically connected to a rather complex function $G(x_1, x_2, x_3, ...)$, depending on operational constraints and economical efficiency. The satisfaction could be expressed as satisfaction of the following inequality:

$$G_0 \leq G(x_1, x_2, x_3, ...) \leq G_1.$$

It would also be possible to define an economy goal as an optimization, i.e., an inequality of the following form:

$$|G - G_{max}| \leq \varepsilon_1.$$

Often, however, it is impossible to use this form directly. Instead, the test must be translated into one of the earlier forms in order to be useful.

Representation of Goals

In MFM, a goal has a graphical representation of the following form, see Figure 3.3.

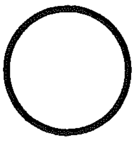


Figure 3.3 The MFM goal symbol.

The goal object may have attributes that describe the goal expression, whether it is failed or not, a textual description, etc.

In the current work, goals have no immediate use in the implemented algorithms, except as starting points for diagnostic searches and as connection points between levels of functions. Thus, the internal data structure is quite simple, see Table 3.1.

`failure state`	*working or failed*
`diagnose`	*true or false*
`description`	*a textual description*

Table 3.1 The attributes of a goal.

failed, (due to faults in the achieving functions). The `diagnose` parameter is used by the fault diagnosis and indicates whether this goal and the parts of the MFM model connected to it should be investigated or not. In addition, there may be a text giving a short verbal description of the goal.

3.4 *Functions*

The second important concept of MFM is that of a *function.* In the context of processes it is possible to find three interpretations of the function concept:

- The function of a part is the rôle it plays in fulfilling the goals of a system, i.e., what it is doing in order to further the purpose of the system. An example of this could be the heating of water, with the explicit goal of cooking an egg.
- A function is a capability of parts of the system, which will enable the system to operate safely, efficiently, or at all, e.g., lubrication of moving parts, power supply, and cooling systems. An example of this would be the heating of the kettle when cooking an egg. This heating is not needed to fulfill the goal, but it cannot be avoided when heating the water.
- A function is a process going on in the system. An example of this could be the bubbling in the water when cooking an egg.

In MFM the first definition is used. Thus, a function is always associated with a goal, and correspondingly, goals are always associated with functions.

The second definition is a common one when concerning biological systems. However, in artifact systems, where the capabilities are present for a purpose, the second definition can be reduced into the first one.

The third definition disagree with the function concept of MFM. It will be sorted under the concept *behavior* instead. Sometimes, one might observe “functions without goals,” in a system, but these are not treated in MFM.

In general, a function of a system could be given the following definition:

A function is a rôle that a system has in the achievement of a goal.

MFM describes the functional structure of a system as a set of interrelated flow structures on different abstraction levels. The levels are connected via achieve and condition relations, and the flow structures consist of connected functions. The types of flow structures currently treated in MFM are:

- Mass flows
- Energy flows
- Information flows

These flows are of completely different types, but they have many properties in common. Most flow functions can appear in each type of flow structure, thus, there are three *flow types* of flow functions.

There are also several *function types,* which are treated in MFM. First, there are the following mass and energy flow functions:

- Source
- Transport
- Barrier
- Storage
- Balance
- Sink

These functions can be used for describing information flows also. There are also some specific information flow functions:

- Observer
- Decision maker
- Actor

In addition to the flow functions proper, some organizational functions are also used. They are concerned with expressing support and control:

- Network
- Manager

The network function is used to group a flow structure and connect it to a goal, while the manager function describes control and supervisory systems, including human operators.

The Choice of Flow Functions

The flow functions chosen in MFM are specifically aimed at describing certain aspects of physical systems, primarily mass and energy flows. This choice is a limitation, but it is not arbitrary. The motives for choosing *flow* functions are as follows:

- Interchange of mass and energy is responsible for the causal interaction between physical components.
- Controlling the flows of mass and energy is important for safety.
- Controlling the flows of mass and energy is important for production efficiency.
- Flow concepts applies to a wide range of physical processes.

Thus, while the choice of flow functions is a limitation, it will be general and useful enough to be interesting. MFM could be extented with other types of functions, see Sections 3.10 and 3.11 of this chapter.

Source Functions

A source function represents the capability of a physical system to act as an infinite reservoir of mass, energy, or information. Of course, this is an idealization of the behavior of a physical system, but it is often useful. Typical examples of source functions are provided by tanks, storages, power supplies, information transmitters, etc.

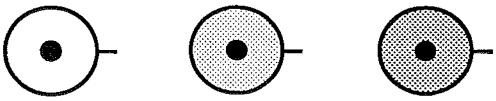


Figure 3.4 The symbols for mass, energy, and information source functions.

A source function is characterized by an output flow F, and it can have capability limitations which maximize F. It has one output port where it can be connected to other flow functions, and it can also be connected to subgoals via condition relations. The symbols for source functions are shown in Figure 3.4.

Transport Functions

A transport function represents the capability of a system to transfer mass, energy, or information from one part of the system to another or from one medium to another. Typical examples of transport functions are provided by pumps, pipes where liquids are transported by gravity, heat exchangers, information channels, etc.

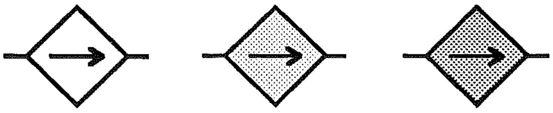


Figure 3.5 The symbols for mass, energy, and information transport functions.

A transport function is characterized by a throughput flow F, and it can have capability limitations, e.g., an interval in which F must lie. It has one input and one output port where it can be connected to other flow functions, and it can be connected to subgoals via condition relations. The symbols for transport functions are shown in Figure 3.5.

Barrier Functions

A barrier function represents the capability of a system to prevent the transfer of mass, energy or information from one part of the system to another or from one medium to another. Typical examples of barrier functions are provided by traps in water systems, heat isolating materials, the safety encasing of nuclear reactors, encryption devices, etc.

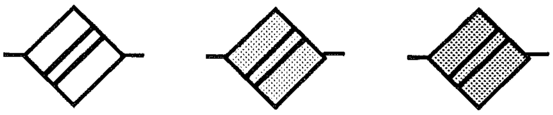


Figure 3.6 The symbols for mass, energy, and information barrier functions.

A barrier function is characterized by a throughput flow F, that should be close or equal to zero. It has one input and one output port where it can be connected to other flow functions, and it can be connected to subgoals via condition relations. The symbols for barrier functions are shown in Figure 3.6.

Storage Functions

A storage function represents the capability of a system to accumulate mass, energy, or information. Typical examples of storage functions are provided by tanks where liquids are stored, the water in a central heating system where energy is stored, memory where information is stored, etc.

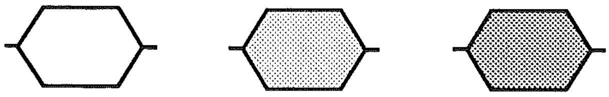


Figure 3.7 The symbols for mass, energy, and information storage functions.

A storage function is characterized by a state variable V, representing the amount of mass or energy accumulated, and one input flow F_i and one output flow F_o. These attributes should fulfill the inequality:

$$\left|\frac{dV}{dt} - F_i + F_o\right| \leq \varepsilon_1,$$

if it can be defined. A storage function has one input and one output port where it can be connected to other flow functions, and it can be connected to subgoals via condition relations. In the original formulation of Lind, see Lind (1990 a), a storage function may have several inputs and outputs. For simplicity, this has been disallowed in the current work. The symbols for storage functions are shown in Figure 3.7.

Balance Functions

A balance function represents the capability of a system to provide a balance between the total rates of incoming and outgoing flows. Typical examples of balance functions are provided by forks in pipes, injection of steam in heated water, channel selectors, etc.

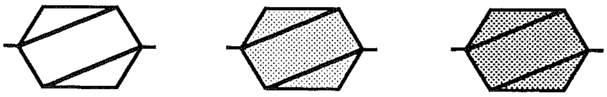


Figure 3.8 The symbols for mass, energy, and information balance functions.

A balance function is characterized by a set of input and output flows. These flows should fulfill the inequality:

$$|F_1 + F_2 + \cdots + F_n| \leq \varepsilon_1.$$

A balance function has several input and output ports where it can be connected to other flow functions, and it can be connected to subgoals via condition relations. The symbols for balance functions are shown in Figure 3.8.

Sink Functions

A sink function represents the capability of a system to act as an infinite drain of mass, energy or information. As with sources, this is an idealization of the behavior of a physical system, but it is often useful. Typical examples of sink functions are provided by tanks, storages, energy dissipation, information receivers, etc.

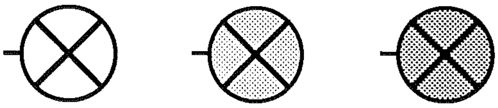


Figure 3.9 The symbols for mass, energy, and information sink functions.

A sink function is characterized by an input flow F, and it can have capability limits which maximizes F. It has one input port where it can be connected to other flow functions, and it can also be connected to subgoals via condition relations. The symbols for sink functions are shown in Figure 3.9.

Observer Functions

An observer function represents the capability of a system to translate physical observations to information. Typical examples of observer functions are provided by measurement devices, but they can also be performed by operators.

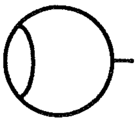


Figure 3.10 The symbol for an observer function.

An observer function has one output port where it can be connected to other flow functions, and it can also be connected to subgoals via condition relations. The symbol for an observer function is shown in Figure 3.10.

Decision Functions

A decision function represents the decision making capabilities of a system. Decision functions are performed by both control systems and operators. In this work, even low level control algorithms will be modeled with decision functions, which differs from the intentions of Lind (1990 a). Lind does not view simple control algorithms as decision making.

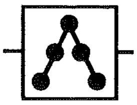


Figure 3.11 The symbol for a decision function.

A decision function has one input and one output port where it can be connected to other flow functions, and it can also be connected to subgoals via condition relations. The symbol for a decision function is shown in Figure 3.11.

Actor Functions

An actor function represents the capability of a system to turn information into physical consequences. Typical examples of actor functions are provided by valves and motors, but they can also be performed by operators.

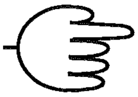


Figure 3.12 The symbol for an actor function.

An actor function has one input port where it can be connected to other flow functions, and it can also be connected to subgoals via condition relations. The symbol for an actor function is shown in Figure 3.12.

Network Functions

A network function represents the property of a system to provide the conditions necessary to allow another system to perform its function. It is used as a way of grouping several connected flow functions into a flow structure.

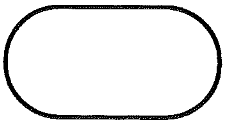


Figure 3.13 The symbol for a network function.

A network function can be connected to goals via achieve relations. The symbol for a network function is shown in Figure 3.13.

Manager Functions

A manager function is used to represent resource management and control. A typical example would be the description of a control system, not as an information flow, but as a system intended to manage a certain task. In the current work, manager functions may be connected to an information flow path, see Section 3.9 of this chapter.

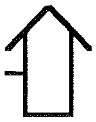


Figure 3.14 The symbol for a manager function.

A manager function has one port where it can be connected to an achieve-by-control relation. It can contain a network of flow functions describing an information flow. The symbol for an manager function is shown in Figure 3.14.

All the flow functions have several attributes associated to them, but as these depend on the algorithms that use them, they will be described together with the reasoning methods, in Chapters 4 to 6.

3.5 Flow Structures

Flow functions may be connected to each other into flow paths or *flow structures*. These structures are used to model how mass, energy, or information flows from function to function. In fact, flow functions always belong to a flow path and may never be used in isolation.

A flow structure is a graph of connected flow functions. The functions can be connected via three different types of relations, called *links* in the terminology of Lind:

- Mass flow connections
- Energy flow connections
- Information flow connections

The Flow Function Connection Syntax

Flow functions may only be connected according to specific syntax rules. Most of these are simple and involve only direct connections. The last one is more complex, however, and may require an analysis of a larger part of the flow structure:

- A flow function can only be connected at its specific connection points.
- A flow function may only be connected to functions of the same flow type, i.e., mass, energy, or information.
- Sources may only be connected to outgoing transports.
- Transports may be connected to sources, storages, balances, sinks, observers, decision functions, and actors. They must be outgoing from sources and observers, and incoming to sinks and actors, but may have any direction when connected to storages, balances, and decision functions.
- Barriers may only be connected to balances.
- Storages may only be connected to transports.
- Balances may only be connected to transports and barriers.
- Sinks may only be connected to incoming transports.
- Observers may be connected to outgoing transports and decision functions.
- Decision functions may be connected to transports, observers, and actors.
- Actors may be connected to incoming transports and decision functions.
- Flow functions may not be connected so that any node is filled or emptied only.

The original formulation of Lind allows storages and barriers to be connected, and storages may have multiple inports and outports, but for simplicity this has been disallowed here. A storage with multiple connections can always be replaced by a storage connected via transports to one or two balances with many ports.

Lind also allows a *storage* node to be filled or emptied only. This seems to be an unusual and somewhat unsound case, so it has been disallowed in the current work. The syntax checking rules implemented in the toolbox described in Chapter 7 do not check this case, however, as it requires a more complex analysis of the connections around every storage node.

The rationale for these connection rules may be formulated in the following points:

- Some rules are needed in order to ensure intelligible models, e.g., the demand that only flow functions of the same type be connected.
- Some rules are motivated by consistency, e.g., that sources may not be connected to incoming transports, or that transports may not be directly connected to each other.
- Some rules derives from the assumption that transports are active and other functions passive, i.e., the transports drive the flows. Thus sources, storages, balances, and sinks must be connected to transports, to produce flows. This is also the reason for why transports may not be connected directly to barriers.

EXAMPLE 3.1
Let us illustrate the construction of flow paths with an example. The process used consists of a storage tank, a pump, and two cylindrical tanks. Water is pumped from the storage tank into the upper tank. From there it flows through a hole in that tank's bottom to the lower tank, and back to the storage tank, see Figure 3.15. The level of one of the cylindrical tanks can be measured, and the pump flow can be controlled. A controller is used to keep a specified level.

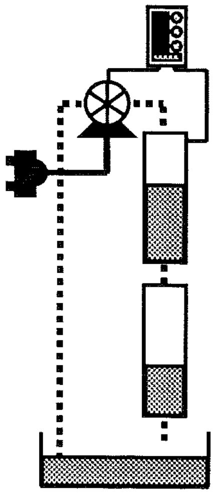


Figure 3.15 The tanks process. It consists of a storage tank, a pump, and two cylindrical tanks. Water is pumped into the upper tank, from where it flows through a hole in the bottom to the lower tank, and then back to the storage tank. The water level of the cylindrical tanks can be measured, and a controller is used to control the flow through the pump. This is a standard experimental process used for teaching and laboratory exercises at the Department of Automatic Control in Lund.

The primary flow of this process is the water that flows from the storage tank, via the pump, the upper and lower tanks, and back to the storage again. The type of this flow is “mass.” A flow structure describing this flow can be seen in Figure 3.16.

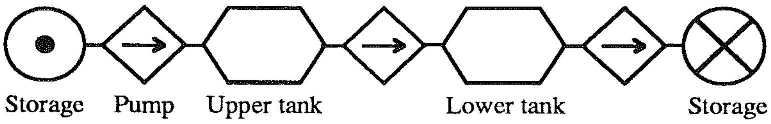


Figure 3.16 The primary water flow of the tanks process. The storage tank has both a source and a sink function. The pump and the two gravity outflows are modeled by transport functions, while the cylindrical tanks are modeled by storages.

The storage tank has been modeled as both a source and a sink. If the volume would have been interesting, it could just as well have been described with a single storage function instead. This exemplifies that the same system can be described with different MFM models. The pump is modeled by a transport function and the upper and lower tanks as storages. The outflows of water through the bottom holes, (driven by gravitation), also appear in the flow path, as transport functions. These transport functions could be easily overseen when using a topological or geographical model. □

EXAMPLE 3.2

The power supply system of the pump is quite complicated but has been modeled simply as a source, a transport, and a sink. The source could correspond to the power supplying part of the system, from power plant and all the way up to, say, the power switch, which is described with a transport function. It should also be noted that in the electrical system the pump motor acts as a power sink. The type of this flow is “energy,” see Figure 3.17. □

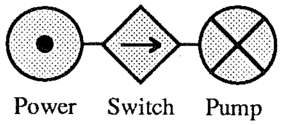


Figure 3.17 The electrical power flow of the tanks process. The flow has been greatly simplified; the whole power network up to the switch is modeled with a single source. Also, the pump shows up again, now as an energy sink.

EXAMPLE 3.3
The water level of the upper tank is measured and a PID controller is used to control the flow of the pump. Thus information flows from the level sensor to the controller, and from the controller to the pump motor. The type of this flow is "information," and the MFM model can be seen in Figure 3.18. □

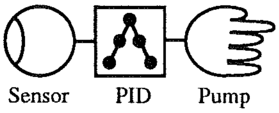


Figure 3.18 The information flow of the tanks process. The level sensor is modeled by an observer function, while the PID controller is modeled by a decision function and the pump flow control by an actor function.

Composition of New Flow Functions

The construction of flow structures from single flow functions could be interpreted as the creation of larger, more complex flow functions. Some general constructs could be common enough that they should be viewed as standardized building blocks, *macro flow functions*. They could be given names and put into model libraries. There are two different types of useful macro flow functions. First, there are some very general concepts as distribution, collection, etc., see Figure 3.19.

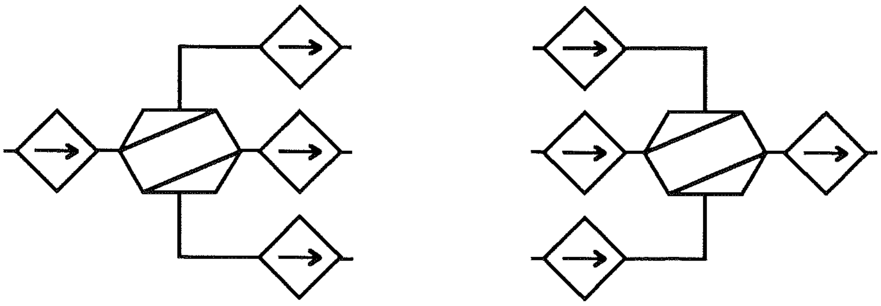


Figure 3.19 Macro flow functions for distribution and collection. Many such macro functions could be defined and put in a macro function library.

Secondly, some flow structures might be useful because they describe some common physical component. In Figure 3.20 the mass'flows of a heat exchanger can be seen. Flow structures like this could be put in a special *component* library.

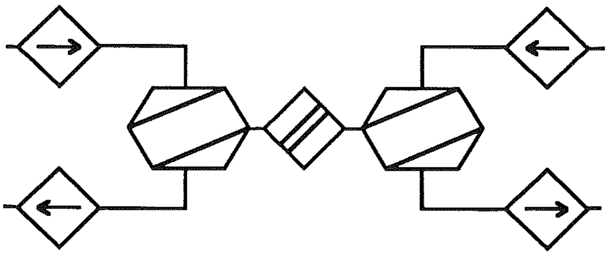


Figure 3.20 A macro flow function for the mass flows of a heat exchanger. Flow structures like this could be put into a component library.

The construction of macro and component libraries can be taken a step further. It would be possible to use this information in building flow models also. If a topological or geographical model is available, a large part of the mass, energy, and information flows are already available. If all components in the topological flow descriptions had corresponding macro flow functions associated to them, the flow networks could be automatically generated, and the MFM model designer would only have to provide the connections between the levels, i.e., the goals and the achieve and condition relations.

Equivalence of Flow Structures

It is often the case that the same physical flow path can be described by different flow structures. Take the process part shown in Figure 3.21 as an example.

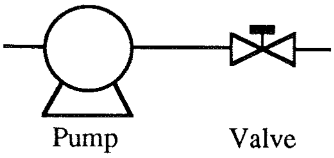


Figure 3.21 A pump and valve on a water pipe. This process part may be modeled in several different ways in MFM.

This part of the process could be described with two transport functions, corresponding to the pump and valve respectively, with a balance function in between, for syntactical reasons, see Figure 3.22.

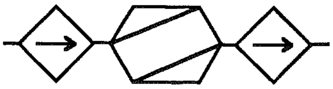


Figure 3.22 One flow structure describing the pump and valve. The pump and valve are modeled as separate transport functions. The balance must be present for syntactical reasons. In this model, the pump and valve may be conditioned by different subgoals.

This way of modeling is good, e.g., when the pump or valve have different working conditions, so that it is useful to describe them separately. However, a formally equivalent model is shown in Figure 3.23.

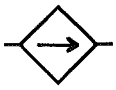


Figure 3.23 Another flow structure describing the pump and valve. Here, the pump and valve are seen as a single transport. They can no longer have different working conditions.

It is important to notice that this formal equivalence holds only if the separate flow functions of Figure 3.22 are not differently conditioned by subgoals, as described in the next section.

One possibly useful extension of the MFM syntax would be to allow transport-to-transport connection. This would be beneficial under the following circumstances:

- A flow line is made up of several parts of equipment, such as pumps, valves, etc., which are connected in series.
- The parts are modeled as transport functions.
- The different parts should be separated and treated distinctly from each other in the diagnosis.

In this case it would be desirable to model the flow line as a series of transport functions, each with its own conditions. In order to avoid a large number of balances, which appear because of syntactical reasons only, direct transport-to-transport connection might be allowed. This thread has not been investigated further in the current work, but it is left here as a suggestion.

The next step on the way to a complete MFM model of the tanks process is the connection of the separate flow structures into an MFM model graph. For that, several relations are needed.

3.6 *Means-End and Part-Whole Relations*

So far, one type of relations have been described, the links that connect flow functions into flow structures. However, another type of relations is more important for MFM; the *means-end* relations. These help connect the flow structures into new structures with several abstraction levels.

It is from this the name *multilevel* comes. Another type of relations are the *part-whole* relations, which describes hierarchical decomposition of entities into subentities.

Part-Whole Relations

The part-whole relations describe a hierarchical decomposition of a system into subsystems, and of subsystems into further subparts. This is a well known structuring concept and will not be further described. It is important to note that it applies to all levels in the means-end hierarchy, thus there are several categories of part-whole relations:

- Goal-subgoal
- Function-subfunction
- Component-subcomponent

An example of goal and subgoals was given in Chapter 1, see Figure 1.3, where, among other things, the goal hierarchy of a tanks process is shown. It is equally common that functions and components have part-whole relations, i.e., that they have an inner structure. This type of decomposition is commonly accepted and there will be no further examples given in this work.

Means-End Relations

The means-end relations are used to connect the different flow structures, and are thus the most important original characteristic of MFM. The means-end relations are of three kinds:

- Achieve relations
- Achieve-by-control relations
- Condition relations

The simple *achieve relations* connect a goal with the flow structure provided for the achievement of the goal.

EXAMPLE 3.4
The flow structure of Figure 3.17 is used to fulfill the goal of keeping the pump running, see Figure 3.24.

The implicit interpretation of an achieve relation is that if all the flow functions in the structure are currently in working order, the goal will be fulfilled. It would, however, be possible to define a more complex interpretation of the achieve relations. This will be further dealt with in Section 3.12 of this chapter. □

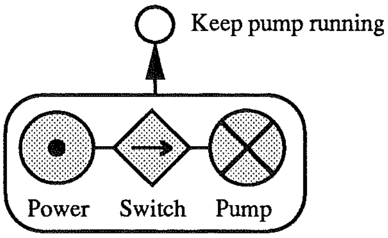


Figure 3.24 A means-ends structure with an achieve relation. The goal of keeping the pump running is achieved by a network of flow functions describing a flow of electrical energy, from a source, the power supply system, via a transport, the power switch, and to a sink, the motor of the pump. If all the flow functions are available, the goal will be satisfied.

The *achieve-by-control* relations also connect a goal with the flow structure that achieves the goal. The difference is that a manager function is also involved. This represents the fact that an active management of resources is needed to achieve the goal.

EXAMPLE 3.5
The water flow structure of Figure 3.16 achieves the goal of keeping the correct level in the upper tank, but a control system is also involved in this, by controlling the actual value of the flow, see Figure 3.25. □

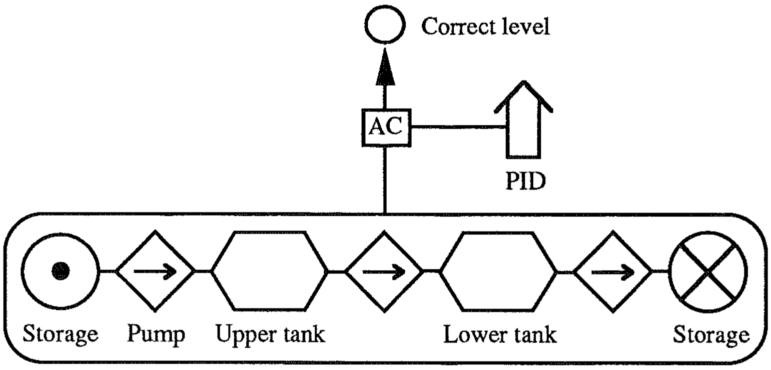


Figure 3.25 A means-end structure with an achieve-by-control relation. The goal of keeping the correct level is achieved by a network of mass flow functions, but the resources of that mass flow are also controlled by a manager function, in this case a PID controller controlling the flow through the pump.

EXAMPLE 3.6
The manager function depends on the information flow of Figure 3.18, and thus that flow structure could be connected to the manager, via a subgoal or directly, as suggested in Figure 3.26.

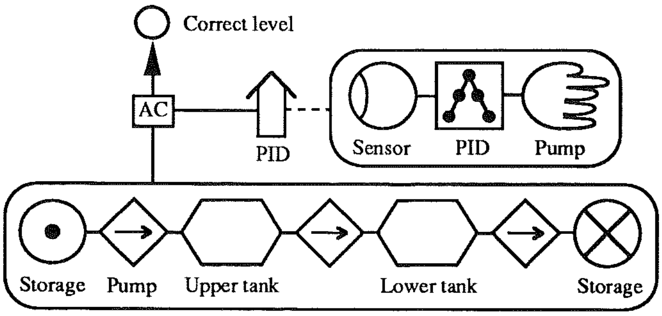


Figure 3.26 The manager function connected to an information flow. The dashed line marks that the manager function, PID, contains an information flow, from the sensor via the controller to the pump.

It should be observed that Lind never makes the connection between manager function and information flow structure. This way of representing a control system is new to the current work. □

The third kind of means-end relation, the *condition,* is used to connect a flow function to a goal it depends on. This means that the function will only be available if the goal is fulfilled. In this way it is possible to describe that the existence of one flow function depends on a set of other functions in the system.

Example 3.7

The transport function of the pump is dependent on the goal that the pump power support is working, see Figure 3.27. □

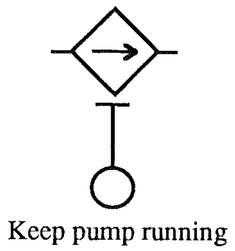


Figure 3.27 A transport function conditioned by a subgoal. The fulfillment of the goal, (keep pump running), is a condition for the transport function to be available.

Flow functions can sometimes be conditions for themselves. This is the case with, e.g., a chemical reactor which uses energy produced in the reaction in order to keep the reaction going. In these cases the flow model will not be tree-like but contain loops.

3.7 The Structure of MFM Models

MFM models are built with the structures presented in the earlier sections. Basically, an MFM model is a graph of flow structures, connected into several levels via achieve and achieve-by-control relations, over goals, and further on via condition relations to flow functions belonging to higher level flow structures.

EXAMPLE 3.8
The full MFM model of the tanks process can be found in Figure 3.28. It should be noted that the figure does not show the components, which is the standard way of presenting MFM models.

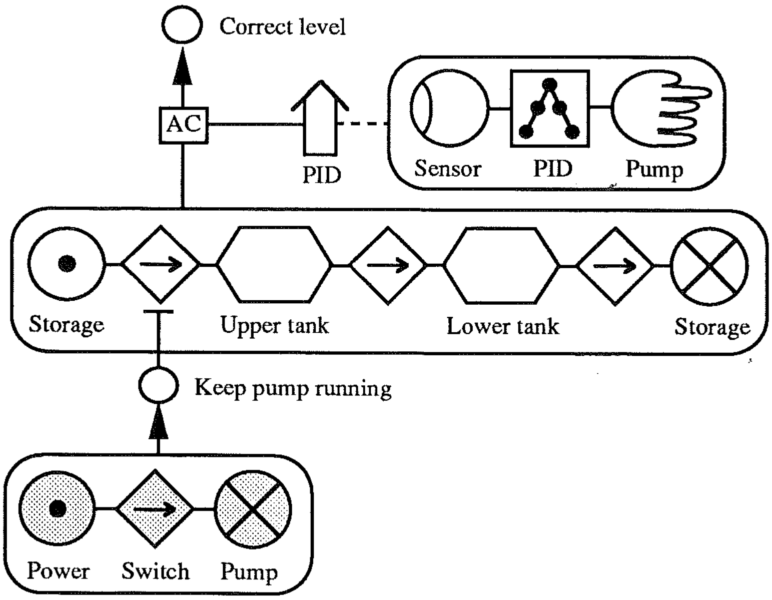


Figure 3.28 An MFM model of the tanks process. All networks have been connected, with achieve, achieve-by-control, and condition relations. Here, the *multilevel* structure is clearly seen.

It is easy to recognize all the different goals, flow structures, and relations binding the parts together. □

In Figure 3.28 it is easy to see the typical multilevel structure of MFM models, with different layers of goals and functions. The general form of an MFM model is the following:

- One or several top level goals
- Achieve relations from the goals to networks of flow functions
- Condition relations from functions to subgoals
- Achieve relations from the subgoals to the next level, etc.

This concludes the basic description of MFM. The rest of this chapter will describe the theoretical contributions made in this work. A large part of this is in the form of sketches or ideas, rather than developed theory.

3.8 Building MFM Models

The first problem for someone coming in contact with MFM is trying to understand what it means, i.e., what it can represent and what an available model can be used for. Hopefully, the previous parts of this chapter and Lind (1990 a) will serve as a description of what MFM can represent, and the following chapters provide examples of what it can do. However, another question is just as important; how does one build MFM models?

MFM as a Modeling Tool

Modeling in general is a complex task, which to a large extent rests on experience, and must be learned by doing. The following observations are based on the limited practical experience of the author, and should be considered with this fact in mind:

- The basic difficulty in multilevel flow modeling is that the concepts are quite different both from topological and behavioral ones. Thus, the means-end nature of MFM must always be kept in mind, and instead of starting to put together a set of mass and energy balances, one should try to focus on the goal hierarchy.
- MFM is well suited for a *top-down* analysis. Actually, the first action when constructing an MFM model should always be to define the top level goals of the entire process. Then either the goal hierarchy could be developed downwards, or the top level functions defined and the next level of goals looked for.
- Functions are often ascribed to systems and subsystems without explicit mentioning of goals. For each such function, a goal should be searched. Sometimes one knows that one function is a condition for another on a higher level. In this case an intermediate "connection" goal should be defined.

- Each topological subpart of the process, i.e., each subsystem and component, probably has a subgoal associated to it, together with one or several functions. By analyzing the subsystems, more goals can be found. But it is important to observe that the topological and means-end hierarchies have no simple correspondence, and many goals will not be found in this way.
- Each flow of mass and energy should be investigated, as they could correspond to MFM flow structures. Here it is important to observe, however, that MFM is *not* thermodynamics. The purpose of thermodynamic models is to accurately represent what is actually happening in physical reality, while MFM represents a normative view of the process. This leads to two important differences. First, the level of abstraction needed can be very different. Secondly, the models might actually be quite different, as the thermodynamics would treat flows not supposed to be in the design.
- No heuristic modeling strategy will be strictly followed in practise, no matter what its academic merits. Thus, the actual strategy most oftenly used would be a combination of a more or less top-down construction of a goal tree and building flow structures and putting them in place in the tree structure; and this would sometimes be interrupted by some bottom-up construction and global connection of larger parts of MFM models.

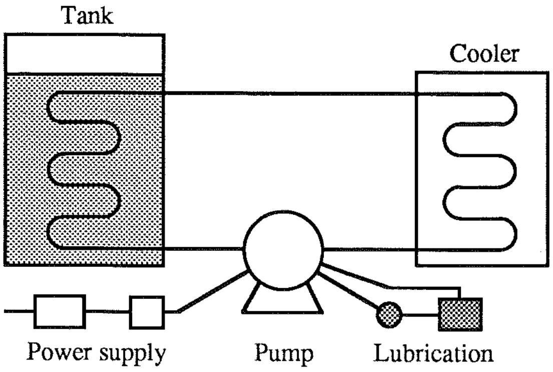


Figure 3.29 A cooling system. The tank is cooled by water, which is circulated through the tank and a cooler. The water is driven by a circulation pump, which needs electrical power and lubrication. The lubrication system consists of an oil tank and a circulation pump.

General Modeling Considerations

Building MFM models is similar to other types of modeling, and several concepts can be "borrowed" from those disciplines. Thus, MFM models may be *simpler than reality* and parts may be *ignored; approximative* models may be useful, *model reduction* is possible; and the flow models will strongly depend on the *supposed use*. To see some examples of this, consider the small process shown in Figure 3.29.

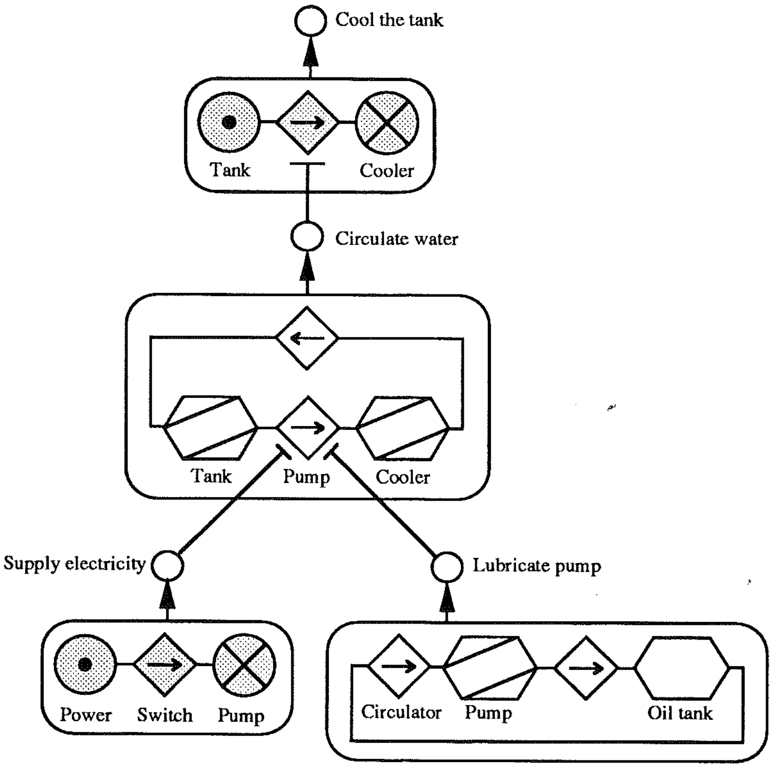


Figure 3.30 An MFM model of the cooling system. Here the electrical energy network is much simplified, consisting of just a source, a transport, and a sink, while the power supply of the real process consists of many more parts. This is an example of the use of an *approximative* model.

An MFM model of this process is shown in Figure 3.30. The top level goal is to cool the tank, and this is achieved by an energy network. In this network, the energy transport is conditioned by the subgoal of keeping water circulating. The active transport function is realized by the pump, while the tank and cooler is modeled as balances. The second transport function is needed for syntactical reasons. The active transport, (the pump), is conditioned by two subgoals, that electrical energy is supplied and that the pump is lubricated. The power supply and lubrication networks are also shown.

Here the power supply network is an example of how an MFM model may be simpler than reality. The power supply has been modeled as a source, a transport, and a sink, corresponding to the external power system, a switch, and the pump motor. The real process is much more complicated and contains a power plant, power lines, transformers, a power plug, cords, more transformers, switches, safeties, and so forth. The consequence is that the MFM model is much simplified, and that the diagnostic resolution capabilities are lessened. A system using this simpler model cannot recognize any fault in the unmodeled parts of the power system. Thus, this also shows the use of an approximative model, where the simplification depends on the model's supposed use.

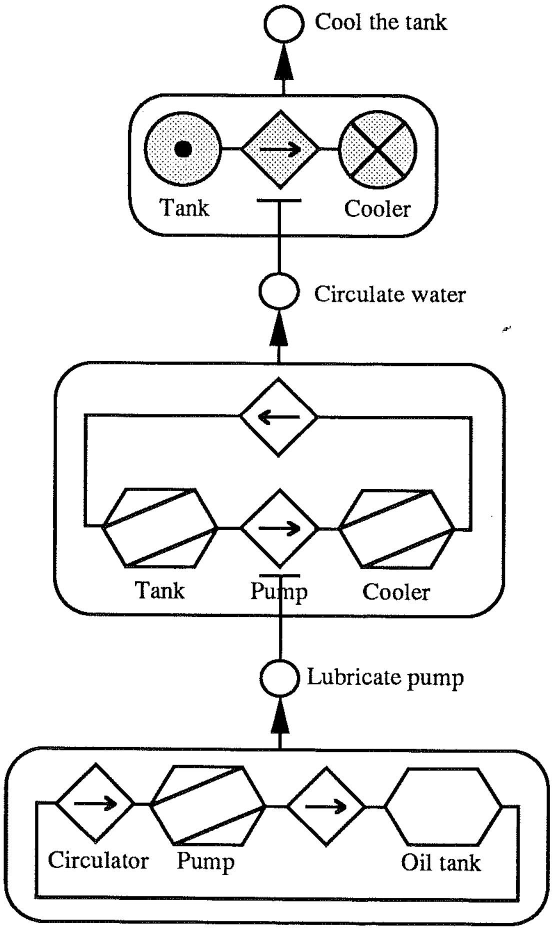


Figure 3.31 Another MFM model of the cooling system. Here the electrical energy network has been completely taken away, thus giving an example of how parts of an MFM model may be *ignored.*

Figure 3.31 shows a somewhat smaller version of the MFM model. Here the power supply system has been completely ignored, which is another form of simplification, once again dependent on the supposed use. With this model, the system of course cannot diagnose any fault in the power supply system.

In Figure 3.32 the lubrication network has been simplified. The full model contains a transport, a balance, another transport and a storage, corresponding to the oil pump, the main water pump, the return pipe, and the oil tank. If the diagnostic resolution demanded is only to find that the lubrication system is at fault, but not exactly where the fault has occurred, it is possible to replace the more complicated network with the rudimentary circulation shown in Figure 3.32, in which is only found the lubrication pump and a balance modeling the rest of the equipment. This could be called the MFM equivalent of *model reduction.*

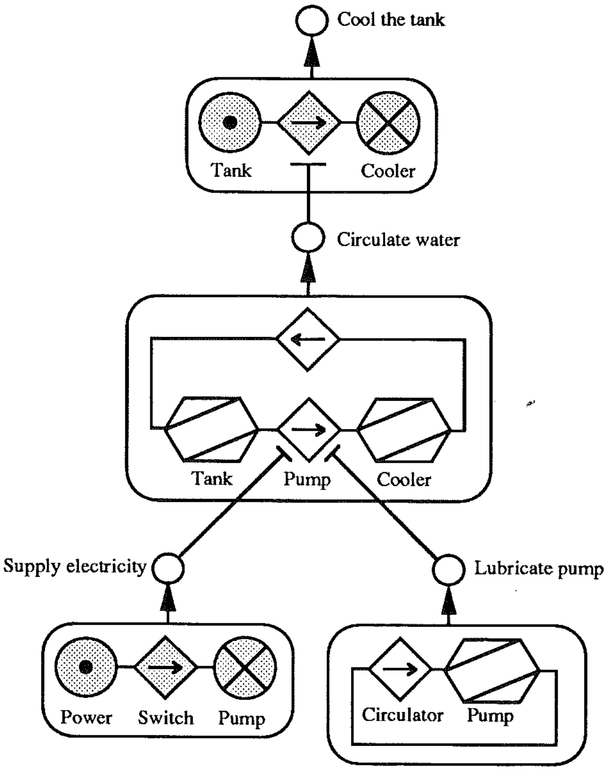


Figure 3.32 A third MFM model of the cooling system. Here the lubrication oil network has been simplified to a rudimentary circulation. This is an example of MFM *model reduction.*

It is important to keep in mind the approximative nature of MFM models. But once it is understood that MFM may use simplifications similar to those used for other model categories, these properties may be utilized to make the models smaller and thus simpler both to build and understand. Model reduction is not a goal in itself, however, but should only be used when there is an obvious gain in simplification. Often it is beneficial to keep those parts of MFM models that help the user to interpret them. This may be illustrated by an example. In Figure 3.33 is shown an MFM network describing the biological life cycle of micro organisms growing in milk. Bacteria are created from spores, resides for a while in the milk, but are then killed by heating, i.e., sterilizing it.

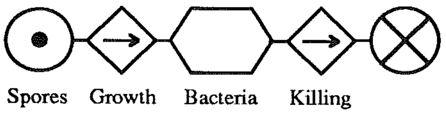


Figure 3.33 An MFM model of the life cycle of milk bacteria. The bacteria are created from spores, reside for a while in the milk, and are then killed by a sterilization procedure.

Under the assumption that the growth of bacteria can be neither affected nor directly measured, it may seem reasonable to perform a model reduction by removing the growth part, see Figure 3.34, where only the killing part of the network remains.

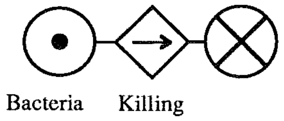


Figure 3.34 A simplified model of the bacterial life cycle. As the growing of bacteria cannot be affected, it has been removed. Instead the model only shows that bacteria are present and that they are killed. However, this simplification gives a less clear view of the bacteria's life cycle, and it is possible that the model reduction should not be performed.

This kind of simplification may not be very useful, though, as the descriptive power of the model is reduced. The picture of the total life cycle of the bacteria is lost, while the gain from simplification is quite small. Thus, it is sometimes a good idea not to use model reduction in MFM.

MFM as a Graphical Programming Language

It is also fruitful to view MFM as a graphical programming language. In this way, all the knowledge about programming tasks can be used in the building of flow models:

- Knowledge on how to structure the construction task can be used "off the shelf." This includes structured and object-oriented programming, etc. The part-whole dimension of MFM lends itself immediately to an object-oriented way of thinking. The goal-oriented nature of MFM will actually make the use of such techniques easier than with conventional programming languages.
- The special techniques of building knowledge-based systems, i.e., knowledge acquisition and formalization can also be utilized. Here the model-based nature of MFM will make the tasks easier than with most other tools.

- There is a danger in using MFM simply as a programming language. A conventional programming language is quite general, and mainly adapted to the computer, rather than to the problem task. Thus, a complex interpretation is needed in order to go from the problem formulation to the implementation. It would certainly be a misuse of MFM to do the similar thing, as MFM is specifically aimed at describing a class of flow processes; i.e., MFM is a *modeling* language, not a *programming* language.
- MFM lends itself to a graphical implementation, and the toolbox described in Chapter 7 allows an easy construction of MFM model graphs. With its help the user can quickly build the graphical representations in a window and mouse based fashion, and then write rules and procedures to connect the MFM data structures to sensors, other algorithms, presentation code, etc. In fact, with the toolbox the actual implementation effort can hopefully become quite modest, compared to the time needed for problem formulation, knowledge acquisition, and model construction.

3.9 Modeling Control Systems

MFM is a young and not fully developed modeling technique. One area where more work is needed is the modeling of control systems. The tasks of these systems may be provided by physical hardware, by external computer systems, or by human operators. The systems can be viewed as supervisory systems, situated outside the process proper, observing its states and making the right decisions and control actions. At the same time they are part of the process itself and may thus also need supervision, fault diagnosis, and so on. There is an inherent problem of self reference here, as the control system is part of any complete model of the system it may be using. Lind has given his view in Lind (1990 d).

For this project, a simple but useful solution was needed. The developed solution differs slightly from the suggestions of Lind. It provides the following good properties:

- A simple way of modeling low level controllers
- A uniform way of mixing information flows with other MFM models
- The use of the manager function

The solution is to use Lind's manager function to describe *any* control or management of resources. As opposed to Lind, this allows the manager to describe the most low level control tasks, e.g., single loop PID control. Agreeing with Lind, the manager is also used to represent high level decision making.

EXAMPLE 3.9
Figure 3.35 gives a small example. A circulation flow is controlled by a simple controller. The goal is to keep a constant flow, and this task is performed by, say, a PID controller. The flow itself is represented in a mass flow network, while the control task is represented by a manager function. The goal, mass network and manager are connected by an achieve-by-control relation, all according to Lind.

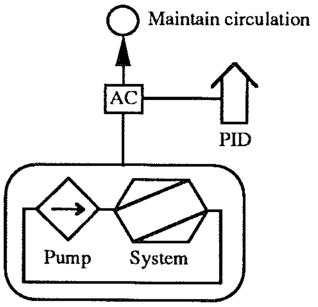


Figure 3.35 A circulating flow controlled by a controller. The flow is represented by the mass network, while the control task is represented by the manager.

The implementation of a control task involves an information flow, and MFM provides a way of modeling such flows, Lind (1988). The simplest control tasks use the information flow shown in Figure 3.36.

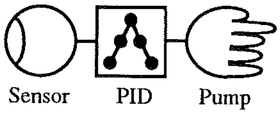


Figure 3.36 The simplest information flow of a control task. Some state of the process is observed, control decisions are made, and the proper actions performed. This may correspond to an operator seeing an alarm state, deciding to shut down the plant, and the shut down procedure, but it may also correspond to a sensor, a PID controller, and a servo.

Lind's group have made some tests with mixing the observers, decision functions, and actors with other flow functions, putting them into the networks, in close contact to the functions connected to observed states. The results were not fully satisfactory.

The solution used in this project is as follows. The information flow corresponding to the task of a manager function is placed "inside" that manager, i.e., the manager is treated like a network, and may contain flow functions. An example is shown in Figure 3.37

Here, the "inside" relation is shown with a dashed line, thus differing from the presentation of a network containing flow functions. In the G2 toolbox implementation, however, the two are exactly alike. Here, one selects to zoom in on either a network or a manager, and a new window with the inner structure of flow functions is shown.

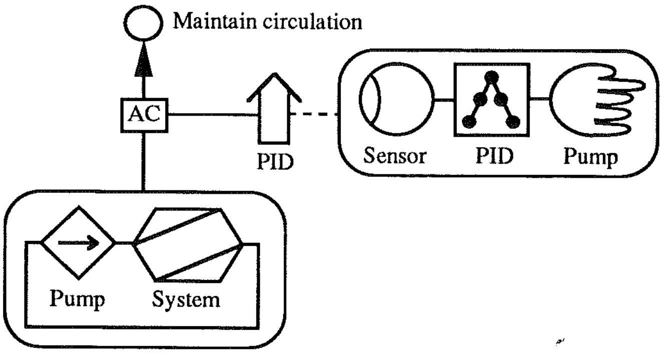


Figure 3.37 The manager and its information flow are connected with a relation. This is the same "inside" relation that connects a network with the flow functions it contains, and is a new contribution of this project.

The choice of putting information flows inside managers provides a simple solution to the problem of connecting information and other flows, and it makes all implemented diagnostic algorithms work, in exactly the same way as they do concerning a network containing flow functions.

The solution violates the principles of Lind in that it views low level PID control as decision making. Bearing this difference in mind, the solution seems to be a viable one, however. □

The connection of information flow functions to subgoals is trivial, see Figure 3.38. Here the standard condition relations may be used without problems.

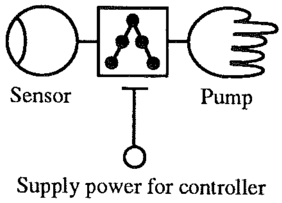


Figure 3.38 Information flow functions may be conditioned by subgoals in the same way as other flow functions. Here the decision function is implemented as a PID controller, which needs a power supply network in order to work.

3.10 Modeling Biochemical Reactions

MFM is primarily concerned with mass, energy, and information flows. However, other types of processes may take place in a system, and be vitally important for supervision and diagnosis. Thus, one objective of MFM research is to extend the set of function types, in order for MFM to be able to accommodate more types of processes.

One very important extension would be to include descriptions of *biological* and *chemical reactions,* together with operations such as mixing, separation, phase changes, etc. Some new ideas will be suggested in this work.

Using MFM Mass Flows

The first suggestion, which has also been investigated by Lind, is to use MFM mass flow functions to represent some biological and chemical reactions. An example may illustrate this idea, see Figure 3.39.

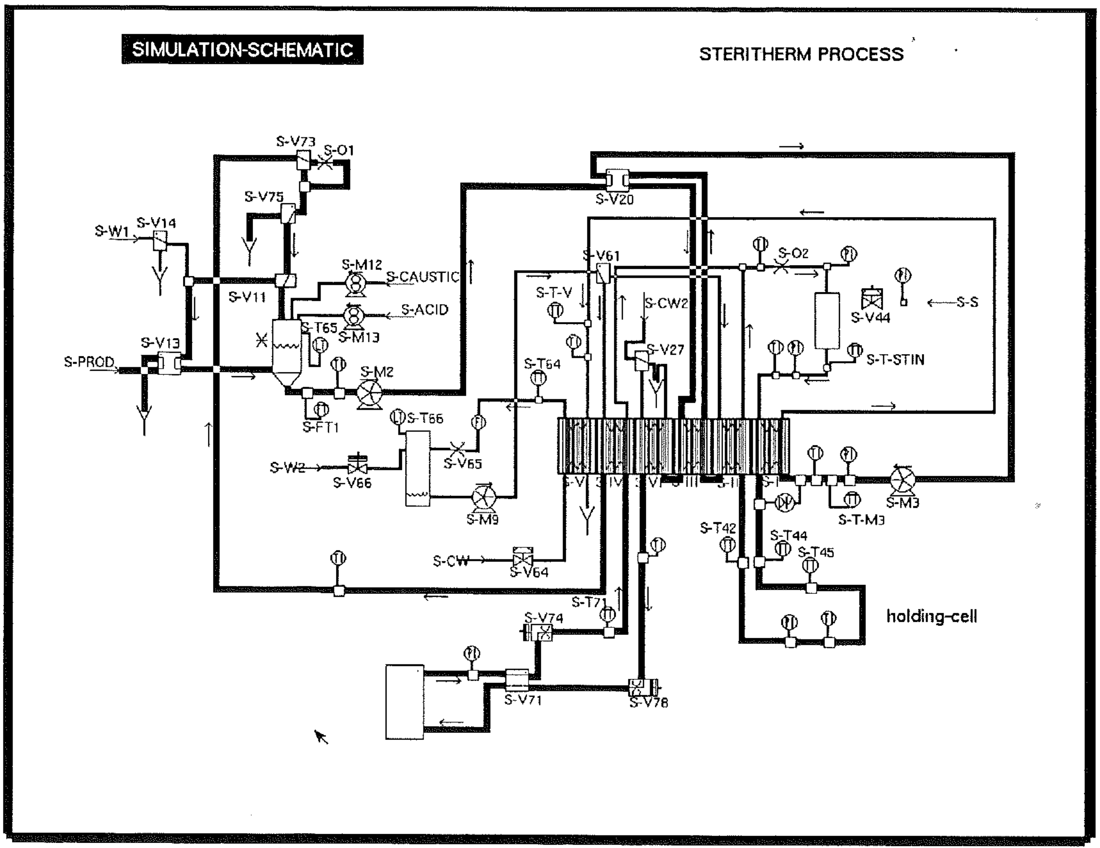


Figure 3.39 Steritherm is a small process which sterilizes a product, e.g., milk, by heating it to a high temperature for a short while. Thus, all the micro organisms in the product are killed.

Steritherm is a small-scale mass and energy flow process for ultra-high temperature, (UHT), treatment of dairy products such as milk or cream. It has been developed by the Alfa-Laval company and is the target process used in the Swedish IT4 project "Knowledge-Based Real-Time Control Systems," see Asea Brown Boveri AB *et al* (1988, 1990, 1991), Larsson (1990 a), Årzén (1989, 1990, 1993), and Årzén *et al* (1990). Steritherm is an operating process and large enough to provide a good target for testing diagnostic algorithms, but it is important to notice that it is still small compared to many chemical processes and power plants.

The top level production goal of Steritherm is to deliver sterile product. Unsterilized product contains spores, from which micro organisms grow. The bacteria reside in the milk until they are killed by heating. This bacterial life cycle is represented by the MFM mass flow in Figure 3.40. Here, the spores are modeled by a mass source, the growing of bacteria by a transport, the living bacteria by a storage, representing the total amount of live bacteria in the milk, the killing by another transport, and the dead bacteria by a sink. Lind has used this methodology for developing models of water treatment plants.

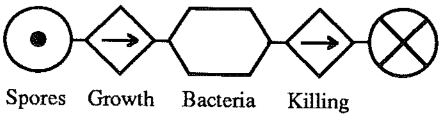


Figure 3.40 The bacterial flow in Steritherm. Micro organisms are created from spores, reside in the milk, and are killed by heating. In this case, an MFM mass flow may be used to describe a biological process.

Using the CAMBIO Graphical Representation

The group of Chéruy *et al*, the Biosystems Group in the Automatic Control Laboratory of the Institut National Polytechnique de Grenoble, has developed a graphical language for describing biological and biochemical processes. It is suggested that this language may be used as a basis for developing new flow functions, and thus extending MFM to incorporate such processes.

The intended use of CAMBIO is modeling and simulation. The graphical language is used to describe a biochemical reaction. The user enters a functional scheme, i.e., a graphical description of the reaction, and the computer system can check the syntactic legality and then produce differential equations, e.g., mass balance equations. The user supplies some of the equations that cannot be deduced from the graphical description, after which the system is able to perform a simulation. Thus, this system provides the user with an easy and uniform way of setting up new biochemical reactions, as well as with a representation that is both easy to read, free from inconsistencies, and which works well as documentation.

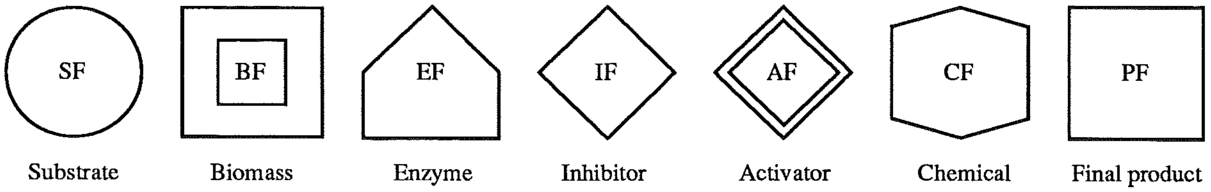


Figure 3.41 The 7 different objects used to build CAMBIO reactions. These are the symbols available to construct the functional schemes.

CAMBIO recognizes 7 different basic building objects, or variables of the functional scheme, see Figure 3.41. They are *substrates* from which biomass may grow and enzymatic reactions may take their needed material, *biomass,* i.e., micro organisms, *enzymes, inhibitors* which hinder or slow down a reaction, *activators* which start or speed up a reaction, *chemicals,* and *final products.*

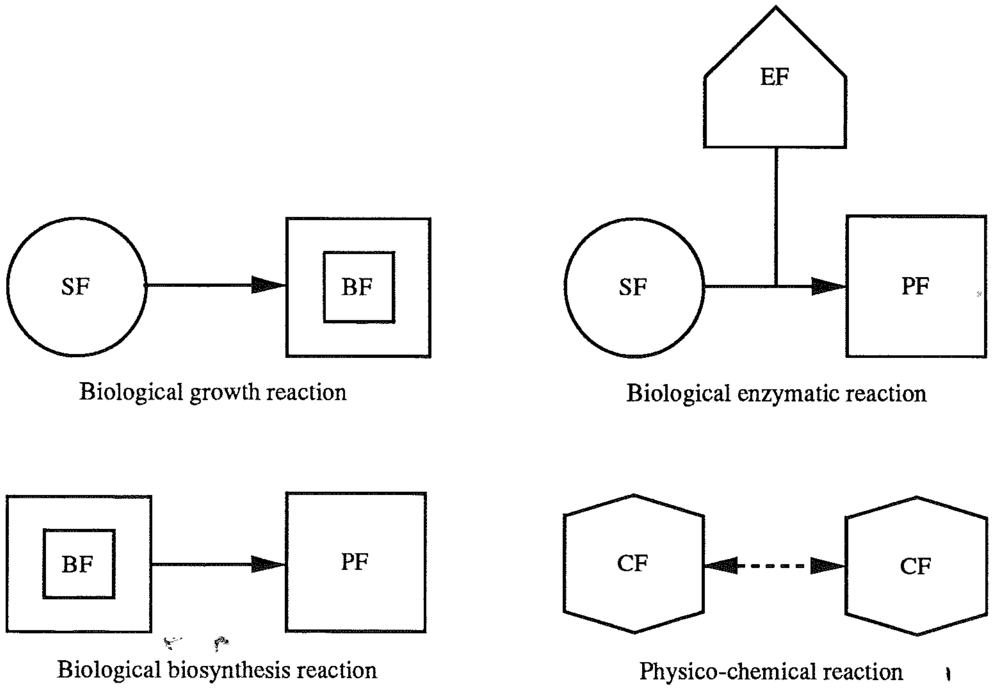

EF
SF
BF
Biological growth reaction
SF
PF
Biological enzymatic reaction
BF
PF
Biological biosynthesis reaction
CF
CF
Physico-chemical reaction

Figure 3.42 The 4 basic CAMBIO reactions. These are the legal connections of the basic objects, but inhibitors and activators may also be involved.

The basic building objects may only be connected into certain allowed biochemical reactions, and the computer checks this at construction time. There are four basic reactions: *growth, enzymatic, biosynthesis,* and *physiochemical reactions,* see Figure 3.42.

The variables and reactions may be combined to build more complex reactions. In Figure 3.43 is shown a description of a yeast fermentation, producing ethanol from starch. The starch is broken down into glucose by an enzymatic reaction. The yeast grows from the glucose in a growth reaction, and produces ethanol in a biosynthesis reaction. The diagram in Figure 3.43 gives a functional description of the three reactions, with inhibition, and from this representation CAMBIO can produce simulation equations.

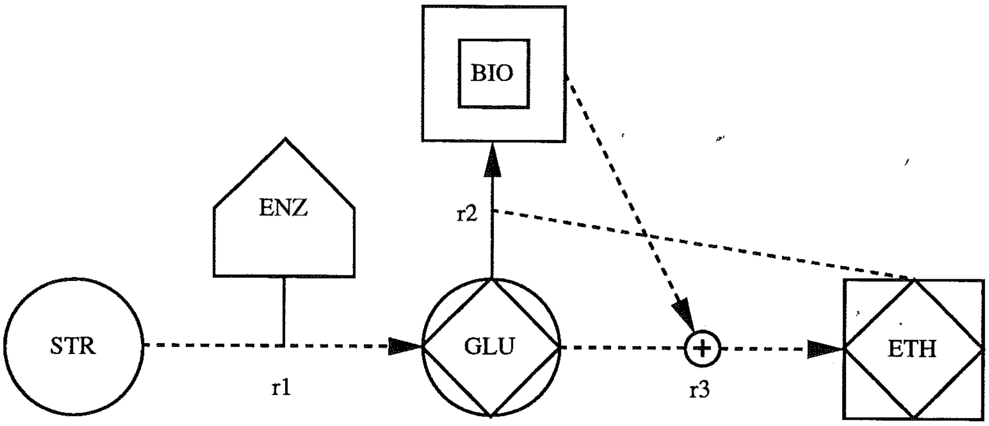


Figure 3.43 A CAMBIO description of yeast fermenting from starch. Glucose, (GLU), is produced from starch, (STR), via an enzyme, (ENZ). The glucose functions as a substrate for the yeast, (BIO), and as an inhibitor for the enzymatic reaction, (r1), if its concentration becomes too large. The yeast grows from the glucose in a simple growth reaction, (r2), and uses the glucose to produce ethanol, (ETH), in a biosynthesis reaction, (r3). The ethanol works as an inhibitor to the yeast growth when its concentration becomes too high. The combination of symbols used in CAMBIO is also seen here.

Here, it is suggested that the CAMBIO language may be used together with MFM, to describe biochemical reactions. The idea is that the variables of the CAMBIO functional schemes should form the basis for new functions, *biochemical functions,* and that the processes, inhibitions, and activations be added to the set of relations. In this way it seems possible to capture a large part of all biochemical reactions and enable MFM to handle them.

A CAMBIO representation of the Steritherm example can be found in Figure 3.44. Here bacteria grow from spores in milk, then to be killed by heating. The sterilization has been modeled as an inhibitor, which goes beyond the ideas of Chéruy's group.

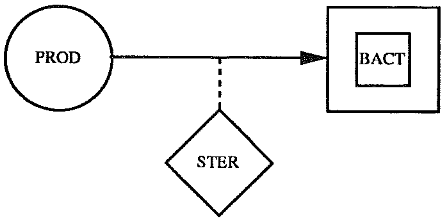


Figure 3.44 A CAMBIO description of bacterial growth in milk. The bacteria, (BACT), grow from the milk, (PROD), in a simple growth reaction, but this growth is inhibited by the sterilization, (STER).

It seems to be straightforward to incorporate the CAMBIO models in MFM. Figure 3.45 shows how the reaction of Figure 3.44 may be enclosed in a network and connected to an achieved goal, and how the sterilization function can be conditioned by a subgoal. This subgoal would then be achieved by a thermal energy network.

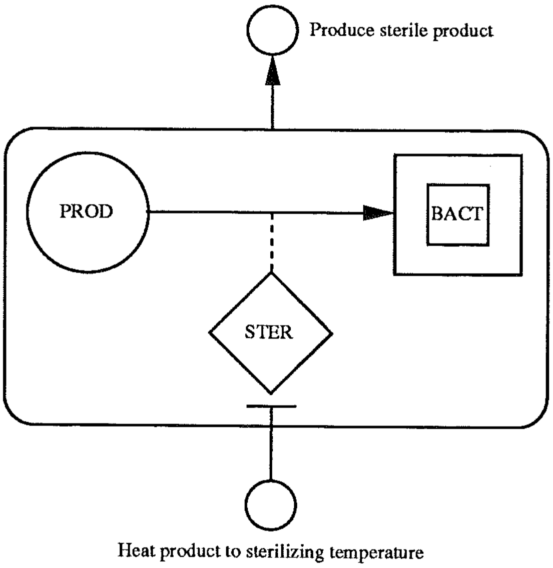


Figure 3.45 The CAMBIO functions can be enclosed in an MFM network and connected to goals and subgoals.

Tomita *et al* (1989) describes a system for automatic generation of operating procedures, based on models of functional hierarchy. These functions include *reaction, separation, blending,* and *storage.* Furthermore, Padalkar *et al* (1991) describes a system for alarm analysis and fault diagnosis, based on structural and functional hierarchies. Among these functions *chemical treatment* and *separation* is mentioned. 'It is possible that some of these functions may form another platform for new chemical functions to be used in an extended MFM representation.

3.11 Extensions of MFM

It would be valuable to extend the flora of MFM functions further, beyond that of biochemical functions. The following categories immediately suggest themselves:

- Chemical reactions, which often occur together with mass and energy flows.
- Geometrical functions fulfilling physical constraints, for example the function of the wheels of a car to keep it at at certain distance from the ground.
- Other physical functions, described by differential equations, e.g., momentum.

It is possible that the CAMBIO representation is extendible to encompass purely chemical reactions, but there may also be a possibility to produce new function concepts from scratch. These may be based on the mass flow functions of MFM, (viewing the reactions as either transports or balances), or on other functional schemes similar to those of CAMBIO. As CAMBIO has been developed with the help of much biological competence, it is clear that a similar effort is needed to solve the description of pure chemical reactions.

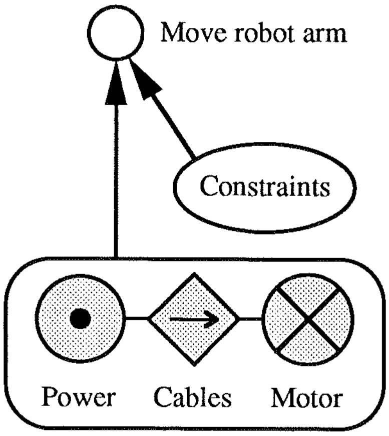


Figure 3.46 To solve geometrical problems from, e.g., robotics, MFM may be extended with sets of constraints.

Another area which MFM currently does not handle is the problem of geometrical or physical demands of a system, e.g., an industrial robot. Here it is important to reason about pure geometrical demands, such as the position of the robot arm, etc. MFM can describe some of this as flows of mass and energy, but this is probably quite unintuitive and apart from that, many concepts still cannot be captured at all. For example, a robot arm may provide the same mass and energy transport, while having very different geometrical positions.

A possible solution is outlined in Figure 3.46. Here, a goal is achieved by the ordinary mass, energy, and information flow networks, but in order for it to be satisfied, a set of constraints must also be fulfilled. The constraints could be represented in the form of mathematical or logical equations. Then the algorithms could work on these constraints too. The development of such a constraint representation demands an effort of its own, however.

3.12 Other Developments of MFM

During the project, several ideas for changes and enhancements of MFM has come up. Some of these may be worth mentioning.

The Isolation Problem

Sometimes a barrier function is important for a system, while there is no tightly coupled flow or flow path. An example of this would be an airtight container for sterile milk. The package clearly has a barrier function, but there is no associated flow. In this case and other cases it would be useful to allow the simple connection shown in Figure 3.47.

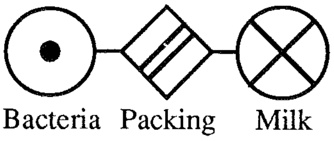


Figure 3.47 A simple network showing a barrier function. This construction is illegal in current MFM, but could be useful to describe some simple cases of isolation.

An easy solution would be to *allow barriers wherever transports would be syntactically correct.* But sometimes, the picture is still complicated, as in the isolation of sterile milk in Steritherm, see Figure 3.48. Here extra transports must normally be inserted between source and balance, and balance and sink. It may seem a bit awkward to put in extra barriers for syntactical reasons, however.

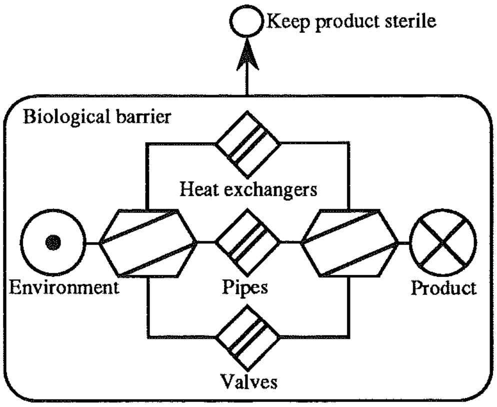


Figure 3.48 The isolation of sterilized milk from the environment, as is needed in the Steritherm process. Several components must be able to isolate the product from the environment, and it is unclear what a flow network describing this should look like.

No solution has been given to this problem in the current work. However, it is important to notice that this problem does exist.

Complex Goal Satisfaction

In real processes, the logical function, f, from the status, F_i, of the flow functions to the status or satisfaction of an achieved goal, G_s, is complicated:

$$G_s = f(F_1, \ldots, F_n).$$

The simple assumption currently used in MFM is that all the functions must be working, i.e., all the F_i's must be true:

$$G_s = F_1 \wedge \ldots \wedge F_n.$$

This guarantees that when the goal is fulfilled, all the functions are working, and thus ensures the validity of any conclusions using this assumption. However, it is sometimes too strong a demand.

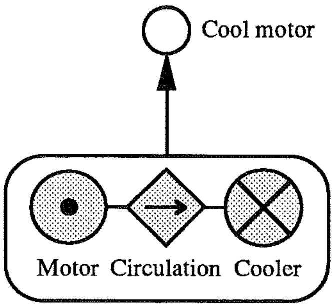


Figure 3.49 A simple energy network for a cooler. The goal is achieved if all three functions are available, but it would also be satisfied if the source function stopped working. This could be described if more complex goal satisfaction functions were used.

The energy network of a simple cooling system can be seen in Figure 3.49. The source produces extra energy, which must be transported to the sink in order to achieve the goal. But the goal of keeping the process cool would be satisfied if the source was not working, regardless of the status of the other functions. This could be described in MFM, but would demand a more complex goal satisfaction function:

$$G_s = \neg F_1 \vee F_2 \wedge F_3.$$

This subject will not be further elaborated here, but it may be interesting to note that Walseth *et al* (1992) outlines a system where goals are connected to continuous variables, and thus a different goal satisfaction algorithm is used.

3.13 A Summary of the Theoretical Contributions

In this chapter, several changes and extensions of MFM have been described, and more have been proposed or outlined.

Differences in Syntax

Some minor changes in syntax have been used in this project. Lind allows a storage function to have several inputs and outputs, which has been disallowed here. The reasons are simplicity and an implementation that fits more closely to G2. The change is not an important one, however, as a multiply connected storage may always be replaced by a storage, one or two transports, and balance functions, where the multiple connections are made.

Two more syntactical changes has also been proposed, transport-to-transport connection and allowing barriers where transports are allowed. The former would be convenient in describing long flows of connected transport functions, to avoid the use of several balances to be put in for syntactical reasons only. The latter proposal addresses a real problem, however, that of how to model a barrier function without an associated flow. The proposal is to allow barriers to be allowed wherever transports are allowed. Some examples have been shown, but the problem has not been pursued further.

Modeling of Control Systems

In the original version of MFM, the modeling of control systems is unclear. Two simple decisions was taken in this project: to use the manager function for representing all control systems, low level, as well as high level, and to let the manager contain information flow functions, just like a network. This allows for a simple and uniform solution of how to join the information flows with the rest of the MFM models.

New Types of Flow Functions

Several suggestions of how to extend MFM with new types of flow functions have been given. The most elaborated proposal concerns biochemical processes, where the solution is either to use MFM mass flow functions to represent the processes, or to use the CAMBIO functional language, adapted to the syntax of MFM.

Some further extensions have also been outlined. Purely chemical reactions may possibly be described by MFM mass flow functions or by a CAMBIO-like language, while it is suggested that geometrical functions are best described by constraints, to be connected to the goals.

Other Contributions

A small example has been given to show the advantage of more complex goal satisfaction rules. The current MFM assumption is that all functions in a network must be available for the goal achieved by the network to be fulfilled, while in practise, this is seldom the precise truth. Thus, more complex conditions for goal satisfaction could be used. No further investigations have been performed in this project, however.

Finally, there are some remarks on the building of MFM models. So far, this area is largely uninvestigated, and every user of MFM must learn it by himself. The sections given here do by no means remedy this situation, but they may have some value in that they described the experiences gathered during this project.

CHAPTER 4

MEASUREMENT VALIDATION

The problem of measurement validation or data reconciliation is the following: given a set of redundant and possibly inconsistent measurements, decide whether there are any inconsistencies, and find out which measurements are correct and which ones are not.

By propagating the measured values into the networks of flow functions of an MFM model of the process, the problem of measurement validation can be solved. The proposed solution will find all inconsistencies, and give as a result all the hypotheses that can explain the current state. That is, the method will take care of all theoretical possibilities instead of guessing on the most probable.

4.1 Introduction

Most industrial processes are equipped with a large number of sensors, of which several directly or indirectly measure the same variables. Especially when material and energy balance equations are taken into account, the total set of measurements commonly gives rise to redundancy, which can be used to check the consistency of the signals, i.e., to *validate* them.

If a process is described by an MFM model, a simple version of measurement validation can be automatically implemented. The available measurement values are treated, for example by different forms of filtering. Then they are sent to the corresponding flow functions. As the MFM model describes the process as a set of flow networks, where each flow should obey some simple flow balances, inconsistent values of mass and energy flows can easily be found. Through further propagation of consistent information, a subset of singularly inconsistent measurement points may be computed. Unknown flow values can be supplied by guessing; a flow propagation algorithm.

4.2 Flow Semantics

In order to use MFM as a basis for measurement validation, a semantics has been defined that assigns flow values and grouping information to the different flow functions. Four of the flow functions have their attributes in common, and have thus been grouped into one class, called *flow carrier.* Sources, transports, non-forking balances, (i.e., balances with only one input and one output), and sinks are all flow carriers. Storages, barrier functions, and forking balances are given a separate treatment. The following attributes are used:

- Flow carriers have one flow value; a quantitative variable that corresponds to a physical flow of mass or energy. Its unit of measure could be, e.g., m^3/s or J/s. A flow carrier is connected to a single measurement device, and its flow attribute is set to the value of the measured signal.
- Storages have three flow values. There are input and output flows, which are connected to corresponding measurements. There is also a third attribute, corresponding to the rate of change of the mass or energy contained in the storage, i.e., the derivative of the storage's volume. Thus, a storage can be connected to at most three measurements.
- Barriers have no flow value, as they do not transport any matter or energy in working states, and they serve only as borders between separately analyzed flows.
- Forking balances have no flow value. Instead, the sum of the inflows should equal the sum of the outflows.

In addition to the flow attributes, each flow function contains information about whether it belongs to a group of several flow functions with consistent flow values, and also a validated, or reconciled, flow value, which can be different from the measured one.

4.3 Generation of Flow Measurements

The measurement validation method takes a set of flow signals as inputs. These inputs can be obtained in several different ways. They can be direct or filtered signals from sensors. It would be more probable, though, that they were the outputs of some low level data filtration on the direct signals, e.g., outputs from a Kalman filter or from some statistical algorithm. Figure 4.1 shows the general architecture of a system using the measurement validation algorithm. An example of the use of a Kalman filter that can also detect sensor failures is given in Section 5.3 of Källström (1979). The signals could also come from any other hardware or software that produced flow values; the origin of the flow values does not matter for the method.

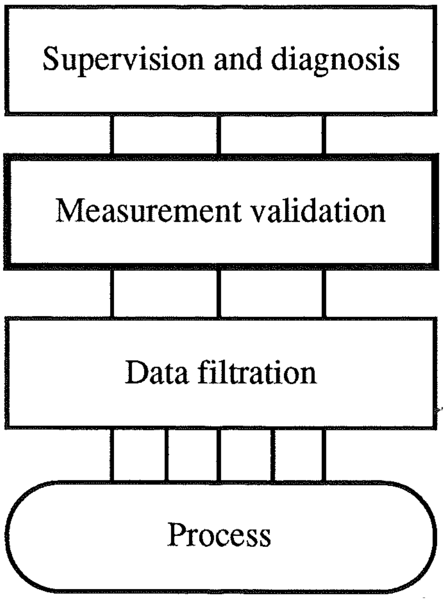


Figure 4.1 The architecture of a system using measurement validation. Low level data filtering and treatment should be done before the algorithm is used. The results of the method may be passed on to higher level algorithms.

The measured flow values are assigned to the attributes of the appropriate flow functions. It is these flow values that the algorithm operate on. The validated output values could be used in high-level supervision and diagnosis.

4.4 Consistent Subgroups

With use of the semantics above it is possible to split an MFM model, (i.e., a set of connected flow functions), into internally consistent subgroups. This is done via use of the following rules:

1. For the flows, F_1 and F_2, of two connected flow carriers the following inequality should hold:

$$|F_1 - F_2| \leq \varepsilon_1.$$

If it does, the two flow carriers belong to the same, (consistent), subgroup; they support each other. If, however, the flow values should disagree, they belong to separate subgroups. This latter situation indicates that at least one of the measurements is in error.

2. If the input flow F_i of a storage is equal to the flow F of the flow carrier connected at the input of the storage, i.e., the following inequality holds:

$$|F_i - F| \leq \varepsilon_1,$$

the input part of the storage belongs to the same subgroup as the flow carrier. Should the two flows disagree, the storage and the flow carrier belong to different subgroups. The corresponding is true for the output flow, F_o, of a storage and the flow, F, of the flow carrier connected to it.

3. For each storage, with volume V, inflow F_i, and outflow F_o, the following inequality should hold:

$$|\frac{dV}{dt} - F_i + F_o| \leq \varepsilon_1.$$

If it does, the input and output parts of a storage belong to the same subgroup, if not, they belong to separate subgroups. Note that this requires some separate measurement of the derivative. If this is not available, the flows connected to the inlet and the outlet must be treated as two completely separate flows.

4. For each balance, with inflows and outflows F_i, the following inequality should hold:

$$|F_1 + F_2 + \cdots + F_n| \leq \varepsilon_1.$$

If it does, the flow carriers connected to the balance all belong to the same consistent subgroup. If not, at least one of the connected flow carriers belong to another subgroup. In this case, the balance should be given a special marking, telling the user that one or more of the flow carriers are not consistent, while others may be. However, all connected flow carriers will be marked as belonging to different groups.

5. If the flow values of two flow functions agree, and the flow functions are in the same flow path, but separated by one or more inconsistent subgroups, they still belong to the same subgroup. However, flow functions that are not in the same flow path do not form subgroups.

Application of the five rules above will enable a splitting of each flow into smaller groups with consistent measurement values. It should be noted that the fifth rule means that there can be groups with holes in them; they need not be directly consecutive. This is the case in Figure 4.2. Here the flow values are shown above the flow functions and the subgroup information is shown with a shading of the graphical symbols. There are two subgroups, and one encloses the other.

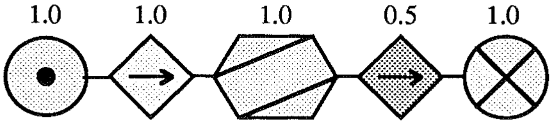


Figure 4.2 An example of a flow path. Here the flow functions form two consistent subgroups, where one group is surrounding the other. The three possible fault hypotheses are that the four measurements are correct and the one is faulty, that the one is correct and the four are faulty, or that all measurements are faulty.

The last rule uses information about several flow functions connected in line; the other four only look at directly connected flow functions. This enables the algorithm implemented to be incremental and almost local.

Each flow function has a group attribute, (a storage has two). The algorithm goes through the flow values of the connected flow functions, and assigns every function to a subgroup of the flow path, with use of the rules described above. This is done by incrementally updating the group attributes.

The fact that all flows obey simple balance equations and are single component, and the local nature of the algorithm means, among other things, that there will be no local nor global ambiguities. Thus, for example, loops and forks pose no particular problems.

4.5 Flow Propagation

The description so far has used the assumption that all flow functions have measurements. This is quite seldom the case. Many of the flow values needed in the algorithm will usually be unknown.

The MFM flow paths can be used, however, to propagate flow information, i.e., to guess the unknown flow values. The idea is quite simple: if an unknown flow value is connected to a known one, the known value is propagated to the unknown. However, as different flow values can be more or less trustworthy, it is necessary to have several propagation rules. The implemented algorithm uses the following rules.

1. A flow value from a subgroup of more than one flow function is said to be *validated.* It has precedence over all flow values supplied by single flow functions, when propagated both upstream and downstream in flow paths.
2. A single measured flow value will be propagated to unmeasured flows, both upstream and downstream, if there is no other information available.
3. If two validated flow values should meet, the one which is propagated downstream has precedence over the one propagated upstream. This does not imply that downstream propagation is better in any sense; it only serves to make the flow propagation behave consistently.
4. If two flow values from single flow functions should meet, the one which is propagated downstream has precedence over the one propagated upstream. Once again, this is only to make the propagation work.
5. Guessed values will not be propagated over balance functions, except when only one value is currently unknown or unguessed.

Using these propagation rules, the system can provide both validated flows whenever there is enough redundant information, and guessed values for most flow functions in a network. The only case when guesses will not be available is when several forking paths, i.e., paths between balances, have no measured value connected to them.

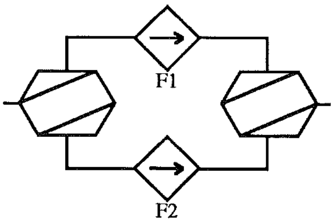


Figure 4.3 A forking flow path. If neither of the flows of `F1` and `F2` are measured, it is not possible to guess the flow values in the flow propagation algorithm.

This would be the case in Figure 4.3, if neither flow functions `F1` and `F2` had any measurement connected to them. In such cases it is impossible to tell how much of a flow that goes through one forking path, and how much goes through the other.

4.6 Validation

Each flow value has a corresponding validated flow attribute. This is set according to the following rules.

1. If a flow value is the only one in its subgroup, and it is surrounded by a consistent subgroup, its own flow value is overridden, and the flow of the surrounding group becomes the validated flow of the flow function.
2. In all other cases, the validated flow is equal to the corresponding measured flow.

In addition to the presentation of validated flow values, the implementation also presents some subgroup information to the user.

- A coloring scheme is used to separate the inconsistent subgroups in the graphical representation of the MFM model. Thus, the symbols of the flow functions in the different subgroups receive a light gray, gray, or dark gray rim, depending on which group they belong to.
- Each flow function that is alone in its group is highlighted in red.

The decision to explicitly mark all single subgroups is only one possible alternative of many. It is derived from the obvious possibility that the measured value in question probably is in error. It is very important to observe, however, that this is only probable, not certain. It is also possible, albeit with a lower probability, that all measurements of a larger, consistent subgroup is in error, while the single value is correct. The third possibility is that all the measurements are wrong. It is very important that these cases be taken into account when the results of the analysis are presented to the operator or higher level algorithm. This is the reason why the implemented system primarily displays information about the different consistent subgroups.

In situations with many inconsistent subgroups, the number of hypotheses grows very quickly due to the combinatorial nature of the problem. However, the recognition of the less likely fault hypotheses may be *vitally important* in order to make a reliable fault diagnosis, that any situation with more than one consistent subgroup is so suspicious that often the only reasonable action will be to perform an emergency shutdown, and that *if* a normative steady state is known, all hypothesis but one may be discarded.

It should be noted that this measurement validation algorithm only uses information about the flow paths, not the means-end relations.

Thus, each flow network is treated separately, and only part of the information of the MFM model is used.

4.7 Implementation

The algorithm has been implemented with two rule groups. One is concerned with the measurement validation and consists of 59 rules, the other takes care of the flow propagation and consists of 12 rules. These rule bases use a database of connected flow functions to yield a incremental and basically local algorithm, i.e., the flow values are updated as soon as new values are available, and only local information about neighboring functions is needed, except in case one consistent subgroup encloses another. Further information about the G2 implementation is given in Chapter 7. The algorithm works by updating a set of attributes of each flow function, see Table 4.1.

`flow`	*quantitative flow value*
`group`	*first, second, or third*
`measured`	*true or false*

Table 4.1 The attributes of a flow carrier. Storages has a double set of attributes, for input and output, together with a derivative value. Barriers and balances have no flow values.

For simplicity, the current version of the method uses a single value to describe a flow measurement, and one global value, ε_1, for comparison. If the signals differ less, they are considered to be equal. It would be possible, however, to include more information about, e.g., uncertainties, variances, etc. This would also enable more reliable comparisons of signals.

Due to the local and incremental nature of the algorithm, it is very efficient. Whenever a new flow measurement is received, updated values spread in the local environment around the concerned flow function. The search starts at the points where new data enters and follows static links along single flow paths. This makes the method very unsensitive to increasing complexity. In the worst case, where the MFM model consists of one long flow path, the effort needed grows *linearly* with the model complexity, (i.e., the number of flow functions). In the normal case, however, the model grows by comprising more and more flow paths, and then the effort stays *constant* in spite of the growing complexity.

4.8 Examples of How the Method Works

Let us now demonstrate the method on a small example. The process used is the same one as in Chapter 3; the tanks process. For simplicity, the control system will be ignored here, see Figure 4.4.

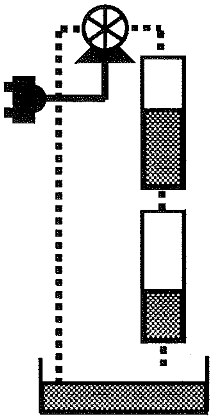


Figure 4.4 The tanks process. Water is pumped from a storage tank, to a cylindrical tank, from where it flows down into another tank, and then back to the storage again.

The main goal of the process is to keep the water level in the tanks at a specified level. The MFM model of the process can be seen in Figure 4.5.

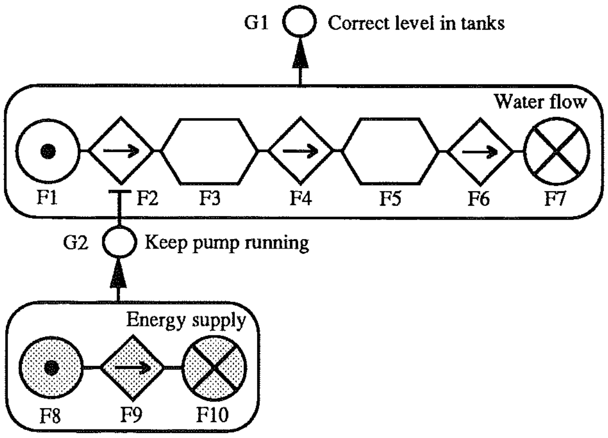


Figure 4.5 An MFM model of the tanks process. The main goal is to keep the level of the upper tank correct, and it is achieved by a water flow. In order for the transport function **F2** to be available, i.e., to keep the pump running, energy must be supplied.

Example 4.1
Now assume that flow measurements are available from the outflow of the storage tank, the throughput flow of the pump, and the inflow, derivative, and outflow of the upper tank, and that these flows have the following values.

flow of `F1`	20×10^{-6} m^3/s	(outflow from storage)
flow of `F2`	10×10^{-6} m^3/s	(flow through pump)
inflow of `F3`	20×10^{-6} m^3/s	(upper tank inflow)
deriv of `F3`	0×10^{-6} m^3/s	(volume change)
outflow of `F3`	20×10^{-6} m^3/s	(upper tank outflow)

Table 4.2 A set of flow measurement values for Example 4.1.

The situation described in Table 4.2 is also shown in Figure 4.6, where only the concerned flow functions are found. The flow values are shown above the flow function symbols; the storage function realized by the upper tank has three values, corresponding the inflow, derivative of volume, and outflow.

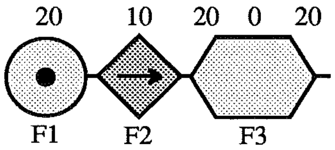


Figure 4.6 A flow path corresponding to Table 4.2. The flow of `F2` disagrees from the rest, and there are two consistent subgroups, whereof ones is single and surrounded.

The flow of `F1` and all the flows of `F3` agree, and thus they form a consistent subgroup. The flow of `F2` disagree, however, forming another subgroup, with only one function in it. The system marks the two subgroups in different colors, thus notifying that there is an inconsistency. In this case it will also mark `F2` specially, as it is a single function group, and the validated flow value of `F2` will be set to 20×10^{-6} m^3/s, (the flow of the surrounding group). □

The consistent subgroup information has been shown in Figure 4.6 with a shading system. In addition to this, the flow function `F2` should also have a special marking, for forming a single function group.

EXAMPLE 4.2

Now assume instead that the flow value of F1 is 10×10^{-6} m^3/s and that the flow of F2 is not measured; a situation which is shown in table 4.3.

flow of F1	10×10^{-6} m^3/s	(outflow from storage)
flow of F2	unmeasured	(flow through pump)
inflow of F3	20×10^{-6} m^3/s	(upper tank inflow)
deriv of F3	0×10^{-6} m^3/s	(volume change)
outflow of F3	20×10^{-6} m^3/s	(upper tank outflow)

Table 4.3 A set of flow measurement values for Example 4.2.

The situation in Table 4.3 corresponds to the flow function description of Figure 4.7.

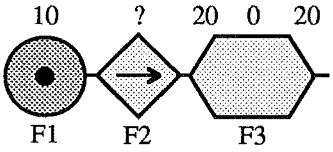


Figure 4.7 A flow path corresponding to table 4.3. The flow value of F2 is propagated from the storage, where the measurements form a multiply validated and consistent subgroup. Then it can be seen that F1 forms a single, (but not surrounded), subgroup.

In this case the system will propagate the validated value of 20×10^{-6} m^3/s from the inflow, derivative, and outflow of F3 upstream to F2, (the value from the validated subgroup has precedence over the value of the single function F1). It then observes that the flow of F1 does not agree with the guessed value of F2, and F1 is marked as a single failure. Its validated flow is not reset, however, since it is not *surrounded* by other flow values. □

EXAMPLE 4.3

Further assume the situation described by Table 4.4.

flow of F1	20×10^{-6} m^3/s	(outflow from storage)
flow of F2	20×10^{-6} m^3/s	(flow through pump)
inflow of F3	10×10^{-6} m^3/s	(upper tank inflow)
deriv of F3	5×10^{-6} m^3/s	(volume change)
outflow of F3	5×10^{-6} m^3/s	(upper tank outflow)

Table 4.4 A set of flow measurement values for Example 4.3.

This situation is also found in Figure 4.8.

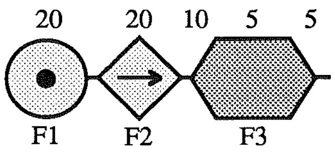


Figure 4.8 A flow path corresponding to table 4.4. Here there are two consistent subgroups which both consists of more than one measurement. The algorithm signals that the flows do not agree, but it cannot guess as to which ones that are correct.

Here we have two consistent subgroups, each with more than one measurement to support it. This situation is difficult to assess, as many sensor values must be wrong. The system will mark the two inconsistent groups, but will take no further action. Marking any particular value as wrong could be misleading and potentially dangerous. □

4.9 A Comparison with Classical Data Reconciliation

In Chapter 2 some other forms of data reconciliation were described. Basically, one type of method uses least-squares estimation to compensate for small errors, assuming known covariances of the measurements. The other type of method uses statistical hypothesis testing to find gross errors, once again assuming known measurement covariances and significance levels. The method presented in this chapter is more qualitative in nature and therefore, a comparison may be quite interesting. These are the main points:

- The MFM method is designed to find large deviations. Thus there is no use to compare it with the first type of method mentioned above.
- The MFM method demands knowledge of how to set the comparison limits for the tests of whether one flow value is equal to another or not, while the hypothesis testing methods must have information about the auto- and covariances of all measurements in order to perform the statistical tests.
- The hypothesis testing demands a large computational effort, especially during the fault isolation by elimination and retesting, and it must be started by outside actions, e.g., by an operator or at specified intervals. The MFM method is simple and efficient, and works incrementally by updating its results as soon as new measurements are available.

- The hypothesis testing finds only the most probable fault hypothesis, while the MFM method is able to display information about all possible hypotheses.

4.10 Conclusions

This chapter describes a method for using redundancy in measurements to find measurement errors and validated measurement values. An important aspect of the method is that it will not make any guesses as to which measurement that may be in error when inconsistencies are found. Instead it will present the different subsets of measurements that are internally consistent. The operator will then be able to investigate all possible alternatives, which is quite important for safe operation and decision-making. Currently the method uses simple quantitative flow values and a global comparison uncertainty, but the algorithms could be extended to include more complex information about the flow values, such as noise levels, variances, etc.

It should be noted that the presentation of single subgroups as less trustworthy than validated groups rests on the assumption that all failure likelihoods are in the same order of magnitude. However, the method could be extended to use explicit measures of failure likelihood, and then a single value could be more trustworthy than a group of values. The same assumption is found in the flow propagation, where validated values have precedence over single values. These guessed values are only used to isolate single groups, however, and the assumption has no other effects on the method.

The method has been tested on two G2 simulations of processes, the small laboratory process shown in the example above, and Steritherm. It was successful in both cases.

The method works under rather general conditions; there should be an MFM model of the process and this model should capture the important aspects of the process. Under these conditions the method provides a simple solution to the problem of measurement validation.

CHAPTER 5

ALARM ANALYSIS

Alarm analysis is the task of analyzing an alarm situation in a process, and separating out primary alarms, i.e., those directly connected to faults, from secondary ones, which are only consequences of the primary ones.

MFM models describe how different functions of the process are connected and depend on each other. By propagating alarm information into this relational structure, it is possible to find out which of the alarms that must be primary, as opposed to those that may be primary or caused by consequential faults. This yields an algorithm that can handle multiple faults in a theoretically correct way.

5.1 *Introduction*

Most industrial processes are equipped with a large number of alarms. In a failure state it is quite usual that many of the alarms will trigger. Some of them will be directly connected to the primary sources of error, but others may be secondary, i.e., not connected to any failed equipment, but due only to consequential effects of the primary failures. In a failure state it is vital for the operator to separate the primary from the secondary alarms. The rest of chapter describes a new method, based on MFM, for automatically recognizing the primary failures.

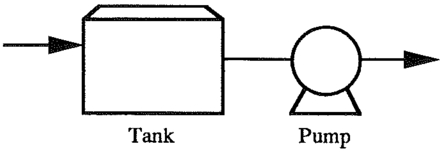


Figure 5.1 A simple process, consisting of a connected tank and pump. A tank underflow may cause a pump low flow, a tank overflow may cause a pump high flow, a pump low flow may cause a tank overflow, and a pump high flow may cause a tank underflow. The alarm analysis algorithm builds on a generalization of causation rules like this.

An example of how the algorithm works is given in Figure 5.1. Here it is reasonable to assume that a too low volume in the tank may cause a too low flow though the pump; that a too high volume in the tank may cause a too high flow through the pump; that a too low flow through the pump may cause a tank overflow, and that a too high flow through the pump may cause a tank underflow. These four *causation* rules describe how one fault may cause another.

Assume that the tank becomes empty, and that a level alarm is activated. As a consequence of the low level in the tank, the pump flow will become too low, and activate another alarm. With the use of the first causation rule above, it is possible to conclude that the tank underflow is the cause of both alarms, the *primary* fault, while the pump low flow is a consequential or *secondary* fault. The alarm analysis algorithm uses an MFM model of a process to find out which alarms that are primary and which that may be secondary.

Of course it is possible that the pump motor has stopped for another reason, at the same time as the tank became empty. Thus, a secondary fault may always hide a primary one. Still, the alarm analysis method will be valuable in separating out those alarms that *must* be primary. In case one alarm is potentially much more important than another one, it is always possible to inform the operator to treat that alarm first, even though it is presented as secondary.

5.2 Failure Conditions for Flow Functions

Every flow function may or may not be alarmed, i.e., be connected to a corresponding part of the process, in such a way that a measurement tells whether the function is currently available or not. However, the alarm conditions are limited according to the following rules:

- A source is working if the current outflow F is less than the source's maximum capacity F_{cap}:

$$F \leq F_{cap}.$$

  If this condition is not fulfilled, the alarm *locap* is true.

- A transport is working if the current flow F lies within an interval, specified in the design:

$$F_{lo} \leq F \leq F_{hi}.$$

If the flow F is below F_{lo} the alarm *loflow* is true; if it is above F_{hi} *hiflow* is true.

- A barrier is working if the current flow F is low enough, (approximately zero):

  $$F \leq \varepsilon_1.$$

  If this condition is not fulfilled, the alarm *leak* is true.

- A storage is working if the current volume V lies within a specified interval:

  $$V_{lo} \leq V \leq V_{hi},$$

  and the following inequality is fulfilled:

  $$|\frac{dV}{dt} - F_i + F_o| \leq \varepsilon_1.$$

  If the volume V is lower than V_{lo}, the alarm *lovol* is true, if it is higher than V_{hi}, *hivol* is true. If the expression within bars is less than $-\varepsilon_1$ the alarm *leak* is true; if it is larger than ε_1 the alarm *fill* is true.

- A balance is working if the following inequality is fulfilled:

  $$|F_1 + F_2 + \cdots + F_n| \leq \varepsilon_1.$$

  If the expression within bars is less than $-\varepsilon_1$ the alarm *leak* is true; if it is larger than ε_1 the alarm *fill* is true.

- A sink is working if the current inflow F is less than the sink's maximum capacity F_{cap}:

  $$F \leq F_{cap}.$$

  If the condition is not fulfilled, the alarm *locap* is true.

It is important to observe that an MFM model is *normative* instead of *descriptive,* i.e., it describes how the process is intended to work, and failures in the process are found by noting differences between the intended model and the actual state. This guarantees completeness, i.e., we will catch every failure if the model captures the important aspects of the process. However, all differences from the intended state are treated as failures, even if there can be other states in which the process is partly or fully operational. Thus, if several flow functions violate their conditions, they are failed by definition, even though the process might still be running in a new state.

5.3 Generation of Alarms

The method needs a set of measured flow signals as inputs. These inputs can be obtained in several different ways. They can be direct or filtered signals from sensors. It would be more probable, though, that they were the outputs of some low level data filtration on the direct signals, e.g., outputs from a Kalman filter or from some statistical algorithm, as already described in Chapter 4.

A model-based validation of the flow values could also be performed before the values were sent to the algorithm, see Figure 5.2. Of course, the method described in the previous chapter may be used for measurement validation.

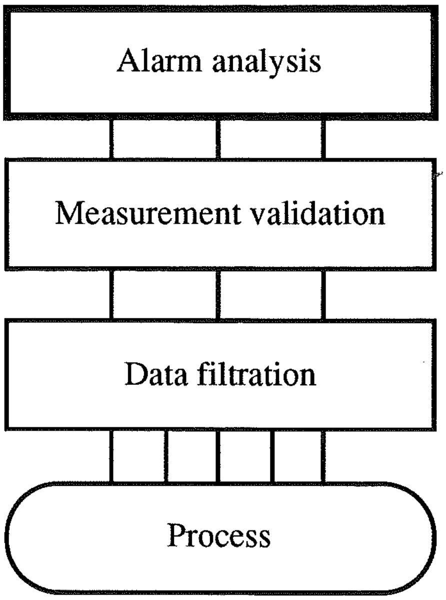


Figure 5.2 The architecture of a system using the alarm analysis algorithm. The incoming data must be treated in several ways before it is turned into qualitative alarm information. This could involve low level filtration as well as measurement validation algorithms.

It is also important that the alarm limits be set properly, with respect to the properties of the supervised signal as well as with the limits of other alarms. The alarm limits should obviously depend on the specific properties of the signal in question, such as time derivative and variance, and several methods have been suggested for taking these into consideration, some of which allow the limits to vary in time, see for example De Marë (1980), Lindgren (1985), and Hansson and Nielsen (1991).

The implemented method uses crisp alarm limits, i.e., there is a fixed value where each alarm condition is activated. It would be interesting to consider the use of fuzzy limits, but the qualitative nature of the rest of the algorithm makes this a complicated task. This is clearly an area that deserves further research, however.

The limits of different alarms also depend on each other. Assume that both the volume and the outflow of a tank are alarmed. As the outflow is largely a function of the volume, the limits of the two alarms should be tuned so that the alarms are activated at approximately the same time. If they are set otherwise, so that, e.g., the outflow alarm can be activated while the volume is still inside the limits, the method will not know that there is anything wrong with the volume, and, thus, it cannot recognize the flow alarm as a consequence of the fault in volume.

5.4 A Method for Alarm Analysis

Failures can only propagate from flow function to flow function in certain ways. This is a consequence of the failure conditions described above. Thus, some primary failures in some types of flow functions may cause secondary failures in the connected functions, while failures in others may not.

EXAMPLE 5.1
An example of fault causation is given in Figure 5.3, where a source **F1** is connected to a transport function **F2**. This could correspond to, e.g., a tank connected to a pump.

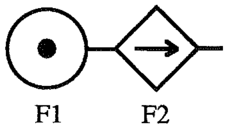


Figure 5.3 A connected source and transport. The source has a maximum outflow capacity, F_{cap}, and the transport has a working interval, $F_{lo} \leq F_t \leq F_{hi}$. If the wanted outflow of the source goes over its capacity or if the actual flow trough the transport leaves the working interval, alarms occur.

The source has an output flow F_s, which must lie beneath the maximum capacity, F_{cap}, of the source. Thus, the following inequality must hold:

$$F_s \leq F_{cap}.$$

If it does not, either because the capacity, F_{cap} has fallen or because the output flow F_S has risen, the source will have a *locap* alarm.

The transport has a throughput flow F_t, which must lie in between a lower and an upper limit, F_{lo}, and F_{hi}, which are set during the design. Thus, the following inequalities must hold:

$$F_{lo} \leq F_t \leq F_{hi}.$$

If they do not, the transport will cause one of two alarms. If $F_t \leq F_{lo}$, the alarm will be *loflow;* if $F_t \geq F_{hi}$, the alarm will be *hiflow.*

Note the *normative* character of these assumptions. Here, the working interval of the transport must be decided during the design phase.

Assume further that the output flow of the source is controlled by the throughput flow of the transport, so that during normal operation:

$$F_s = F_t,$$

and that the working interval of the transport:

$$F_{lo} \leq F_t \leq F_{hi},$$

is small enough so that F_{cap} is always outside it during normal working conditions. Then the following analysis can be performed:

- If the capacity F_{cap} of the source should fall below the desired outflow, the transport will not get enough flow medium, and its throughput flow F_t will be forced out of and below the working interval. Thus, if the source loses its capacity, the transport cannot keep its flow within the correct limits. This implies that a *locap* alarm of a source will *force* a *loflow* alarm in the connected transport function.
- If, on the other hand, the current throughput flow of the transport should become higher than the upper limit of the working interval, i.e., F_t rises above F_{hi} because of some fault, this may or may not lead to the output flow of the source going above the maximum capacity $F_{cap.}$ Thus, a *hiflow* alarm of a transport *may* cause a *locap* alarm in a connected source.
- If the current throughput flow of the transport should fall below the lower limit of the working interval, the flow demanded from the transport will still be below the source's capacity, and the source will not be affected. Thus, a *loflow* alarm of a transport will not cause any alarm in a connected source.

This analysis can be expressed very simply and crisply in two rules, where all the quantitative information is suppressed and only the alarm information used:

1. A source *locap* alarm will force the connected transport to have a *loflow* alarm.
2. A transport *hiflow* alarm may cause a connected source to have a *locap* alarm.

With the use of these two rules for how faults may cause other faults, and thus, how alarms may cause other alarms, any alarm situation concerning a source connected to a transport may be analyzed. If either function is in an alarmed state but the other is not, that alarm is a primary one. If both functions are in an alarmed state, however, one of the situations above will apply, and accordingly, it is possible to tell which of the alarms that must be primary, and which that may be either primary or a consequence, a secondary alarm.

It is also interesting to note that some alarms will *force* consequential alarms to occur, while some *may* cause them, a fact that can be used to recognize some situations where the measurements must be at fault. □

EXAMPLE 5.2
An analysis like the one above can be performed for every combination of connected flow functions. Take for example a transport connected to the input of a storage, as shown in Figure 5.4.

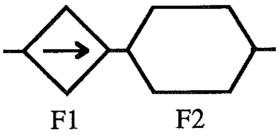


Figure 5.4 A connected transport and storage. The transport is working when its flow lies within the interval $F_{lo} \leq F_t \leq F_{hi}$, and the storage is working when its volume lies within the interval $V_{lo} \leq V_s \leq V_{hi}$. Otherwise, alarms occur.

Here, the working interval of the transport, i.e., the interval within which the throughput flow F_t must lie, is:

$$F_{lo} \leq F_t \leq F_{hi},$$

and the working interval of the storage, the interval within which the volume V_s must lie, is:

$$V_{lo} \leq V_s \leq V_{hi}.$$

The design must have the property that a flow value of F_t within the allowed interval will keep the volume V_s in its allowed interval. This has the consequence that if the flow F_t should be too low, (less than F_{lo}), the volume V_s may also be forced out of and below the allowed interval, while if it should be too high, this could cause the volume to become too high also. This can be described by two rules:

1. A transport *loflow* alarm may cause a storage *lovol* alarm.
2. A transport *hiflow* alarm may cause a storage *hivol* alarm.

One more fault causation may be possible in this situation. This concerns the case when the storage is *overfull.* If the storage is realized by a balance tank with a finite volume, say, this tank may become completely filled up. If so, the transport may not be able to keep its flow value. In that case the following rule should also be added:

3. A storage *hivol* alarm may cause a transport *loflow* alarm.

In other cases an overfill will lead to an overflow of the storage, while the transport is not affected. This would be the case for an open tank, where the medium could spill over at the top. Thus, a choice has to be made as to whether the last rule should belong to the set of analysis rules or not. □

In the following, a whole set of rules for analysis of alarm situations will be presented. They have been arrived at in the same fashion as shown in the previous examples, and it is important to notice that they assume more specific properties in the flow functions than given in MFM. For example, it is assumed that a storage overfill can cause a *loflow* in a transport connected at the inlet of the storage, i.e., the third rule above has been included.

As there are several possible rule sets, a choice has been made. This is probably not the final solution to the problem. Instead, several rules sets could be used, each better fitted for certain types of processes and situations.

5.5 Assumptions of Flow Function Behavior

In the examples above, the different working conditions gave rise to a set of assumptions of how the flow functions involved will react when connected to each other. Such assumptions have been listed below for all flow functions, where the faults they can cause and the reactions they have on faults are given:

- A source can send a low flow downstream, and will react with a *locap* on a high flow wanted from downstream.
- A transport can send low and high flows downstream and it can demand low and high flows upstream. It will react with a *loflow* on a low flow demanded from downstream and on a low flow provided from upstream. It will react with a *hiflow* on a high flow demanded from downstream and a high flow provided from upstream.
- A barrier can send high flows upstream and downstream and demand high flows upstream and downstream. It will not react to any demand.
- A storage can send low and high flows downstream and it can demand a low flow upstream. It will react with a *lovol* on a high flow demanded from downstream and a low flow provided from upstream. It will react with a *hivol* on a low flow demanded from downstream and on a high flow provided from upstream.
- A balance can send low and high flows downstream and it can demand low and high flows upstream. It will not react to any demand. It will pass on demands upstream and downstream.
- A sink can demand a low flow upstream, and it will react with a *locap* on a high flow provided from upstream.

Using these assumptions, a small G2 program was written to automatically generate rules for all possible alarm causations, see Larsson (1992 c). In fact, a set of rules was first generated by hand; and when the automatic rule generation program was ready, the previous hand-generated rules where checked and found to be correct.

5.6 A Rule Set for Possible Secondary Alarms

The examples in Section 5.4 can be extended to all the allowed connections of flow functions. This will give a set of rules for how an alarm in one flow function may or will cause consequential alarms in the connected functions. A complete set of rules is as follows:

1. A source *locap* will force the connected transport to have a *loflow.*
2. A transport *loflow* may cause a storage connected at the inlet of the transport to have a *hivol,* and a storage connected at the outlet to have a *lovol.* It may cause another transport connected in the same

direction via a balance to have a *loflow.* If the balance has no other connections the same alarm will be forced.

3. A transport *hiflow* may cause a connected source or sink to have a *locap.* It may cause a storage connected at the inlet of the transport to have a *lovol,* and a storage connected at the outlet to have a *hivol.* It may cause a transport connected in the same direction via a balance to have a *hiflow.* If the balance has no other connections, the same alarm will be forced. It may cause another transport connected in the opposite direction via a balance to have a *loflow.*
4. A barrier *leak* may cause a transport connected via a balance to have a *loflow* or a *hiflow.*
5. A storage *lovol* may cause an outgoing connected transport to have a *loflow.*
6. A storage *hivol* may cause an incoming connected transport to have a *loflow,* and it may cause an outgoing connected transport to have a *hiflow.*
7. A storage *leak* may cause the same storage to have a *lovol.*
8. A storage *fill* may cause the same storage to have a *hivol.*
9. A balance *leak* may cause a connected outgoing transport to have a *loflow,* and a connected incoming transport to have a *hiflow.*
10. A balance *fill* may cause a connected incoming transport to have a *loflow,* and a connected outgoing transport to have a *hiflow.*
11. A sink *locap* will force the connected transport to have a *loflow.*
12. An alarm in a network will force a function depending on this network to fail.

5.7 Different Rule Sets for Different Domains

The rule set presented above is one reasonable solution of how to choose the possible secondary alarms. However, it rests on several assumptions about the behavior of flow functions. If these assumptions are changed, new rule sets are needed. This point may be illustrated by some examples:

- One possible assumption is that active transports will not be forced into a *hiflow* state, even if they are connected via a balance to another transport which has a *hiflow.* This corresponds to the case of two rotor pumps connected with a pipe, and the assumption is that the first pump cannot force the medium to flow quicker through the second one. The obvious counter-assumption is to allow one transport *hiflow* to cause another further down the line. The latter is the normal case in the rules above.

○ Another assumption is that storages have a limited volume and are closed, i.e., a storage *hivol* may cause a *loflow* in an incoming transport. If a storage may overflow, this causation will not take place. The first assumption was used in the rules above.

The solution to these ambiguities is to allow several types of flow functions, e.g., as has been shown above, in the separation of *active* and *passive* transports. Likewise, it would be possible to allow *closed* and *overflowing* storages. However, this may lead to a plethora of almost similar flow functions and make it more difficult to understand the MFM models.

Another possibility is to design different rule sets, and use the most appropriate one for specific processes, equipment, and design tasks. In this case, the choice of flow function assumptions is made once and for all in the beginning of the modeling process. This would still leave the problem of how to mix MFM models from different domains, however.

EXAMPLE 5.3

In the presented rules there is the implicit assumption that a transport can be forced to have a *hiflow*. However, it is often reasonable to assume that an *active* transport cannot be forced to such an alarm state. In this case, three of the rules above only concern passive transports, not active ones.

3.b A transport *hiflow* may cause a passive transport connected via a balance to have a *hiflow.*If the balance has no other connections, the same alarm will be forced.

4.b A barrier *leak* may cause an outgoing passive transport connected via a balance to have a *hiflow.*

9.b A balance *leak* may cause a connected incoming transport to have a *hiflow.*

10.b A balance *fill* may cause a connected outgoing passive transport to have a *hiflow.* □

5.8 Higher Order Rules

The rule set shown contains only rules involving two or three tightly connected flow functions. At a first glance, it would seem possible that there could exist cases where a fault causation either involved more flow functions, or involved flow functions separated by two or more other functions.

There are no such cases, however, as can easily be seen from the fact that the only type of flow function that allows propagation of a faulty flow value without having an alarm of its own is the balance. As balances may not be connected to each other, the longest chain of fault causation is three flow functions long, and there is a balance in the middle. All other causations involve only pairs of directly connected flow functions.

5.9 An Alarm Analysis Algorithm

The rules in Section 5.6 can be used for automated alarm analysis. Given a set of alarms, it is possible to decide which of the alarms that must be primary ones, and which ones that *may* be secondary. It is important to observe, however, that one cannot be certain that a fault is indeed secondary; there might be multiple faults. Thus, the method will differentiate between positively primary alarms, and alarms that may be either primary or secondary.

As soon as a new alarm value is discovered, the corresponding alarm of the concerned flow function is set to an alarm value, e.g., a transport *loflow,* a storage *hivol,* or a balance *fill.* Then all rules that can be applied to the new alarm are tried, in order to see if they match the new situation. If so, the failure state of one or several flow functions may change, from *normal* to *primary failed* or *secondary failed.* It should be noted that the failure state *secondary* really means *primary or secondary.*

The rules will only be applied locally and involve two or three closely connected flow functions. The only way global information is propagated is via the connections in the MFM graphs. Thus, the efficiency is proportional to the length of the networks and the depth of the abstraction hierarchies. For any reasonable MFM model, this will make the algorithm very fast. Furthermore, the rules can be applied in any order and in a “real-time” fashion. The implementation is actually done as a real-time algorithm, acting as soon as new values drop in.

5.10 Unknown Alarm States

When some alarm states are unknown, the flow networks can be used to guess the missing values. This method will be called *consequence propagation.* The idea is simple. Given a set of known, (primary and secondary), alarms and a set of unknown alarm states, the unknown values are filled in with secondary alarms according to a set of rules.

The rules used for this are very similar to the ones used for alarm analysis. An example is shown in Figure 5.5.

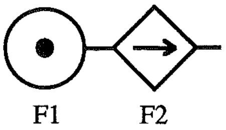


Figure 5.5 A connected source and transport. If the source has a *locap* while the alarm state of the transport is not measured, it is still possible to know that its alarm state is *loflow.* Further, if the alarm state of the source is not measured, while the transport has a *hiflow,* it is reasonable to guess that the source has a *locap* alarm.

Here we, once again, have a source connected to a transport. Assume that the alarm state of the source is known, while the alarm state of the transport function is not measured. If the source is working, i.e., is in a *normal* state, there is no way of knowing whether the transport is working or not. Unless any other information is available, a reasonable guess here is to assume that the transport is actually working.

If the source would have a *locap* alarm, the transport would not be able to move enough of the medium, and the only reasonable guess is to assume that it would have had a *loflow* alarm.

Likewise, if the transport's alarm state is known, while the source's is not measured, the situation is as follows. A *normal* state or a *loflow* alarm is no reason to believe other than that the source is working. A *hiflow,* on the other hand, may or may not force the output flow of the source above its maximum capacity. In this situation, the source *may* have a *locap* alarm.

Thus, two consequence propagation rules may be formulated:

- If there is a source *locap* alarm, then guess a *loflow* alarm in a connected transport.
- If there is a transport *hiflow* alarm, then guess a *locap* alarm in a connected source.

As can be seen, these rules exactly correspond to the alarm analysis rules. Every alarm analysis rule can be converted to a guessing rule, to be used in case the flow function in question is not given an alarm state from measurements.

The guesses can have two different degrees of certainty. In the case of a source *locap,* we know that, assuming a correct design of the working intervals, the connected transport *must* have a *loflow.* On the other hand, a *hiflow* in a transport *may* cause a *locap* in the connected source, but this is not necessary, as the actual flow may or may not rise enough as to go over the source's capacity.

However, both types of guesses are equally useful for the alarm analysis. The result of the algorithm is a separation of the alarms into two subsets, one containing alarms which are known to be primary, and one with alarms that are either primary or secondary. This interpretation of the results allows for uncertain guesses.

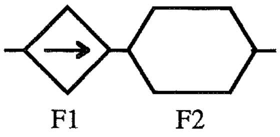


Figure 5.6 A connected transport and storage. The guessed alarm state of **F1** is loflow, and the known alarm state of **F2** is lovol. The alarm of **F2** may thus be consequential, i.e., it is either primary or secondary. If the guess is wrong, the alarm state would be known to be primary. However, the first case covers the second one. Thus, an erroneous guess may mean loss in diagnostic resolution, but will never give a faulty result.

Assume that the alarm state of the transport **F1** in Figure 5.6 is not measured, and that the storage **F2** has a *lovol* alarm. An guess is made that **F1** has a *loflow* alarm. The algorithm uses the guess and decides that the *lovol* of the storage may be caused by the guessed *loflow* of the transport. The conclusion is that the storage's *lovol* is either primary or secondary. If **F1** had been known to be working, i.e., the guess was wrong, the conclusion would have been that the *lovol* alarm was primary.

Thus, an erroneous guess by the consequence propagation results in a loss of discrimination; a primary alarm will be presented as either primary or secondary. It is important to notice, however, that the interpretation of the result is still correct. The primary case is covered by the primary or secondary case. Thus, the consequence propagation will never give rise to an erroneous result.

Here it was also shown how the consequence propagation mixes with the alarm analysis. As soon as a new alarm value has been guessed, it can be utilized in the alarm analysis. The two methods interleaves in a local, real-time fashion, to form a single algorithm. Thus, unknown alarm states may also be treated.

5.11 Conflict Resolution

There are cases where conflicting guesses are possible. Consider the situation shown in Figure 5.7.

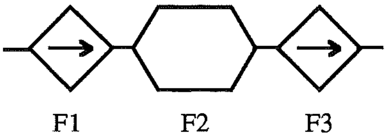


Figure 5.7 A storage connected to two transports. If the alarm state of the storage is not measured, while both transports receive *loflow* alarms, the order of the alarms will determine which of them that will be considered primary. This can be solved by a conflict resolution mechanism, however, which recognizes the case and sets both transport alarms to *secondary failed.*

Assume that the alarm state of the storage `F2` is not measured, while those of the transports `F1` and `F3` are. Further assume that first `F1` and then `F3` has *loflow* alarms. The guess for the alarm state of `F2` would be a *lovol,* and the alarm of `F3` could be a consequence of this.

If, however, the *loflow* of `F3` came before that of `F1,` the guess would be a *hivol* for `F2,` and now the alarm of `F3` would be considered primary instead. Thus, there are cases where the order of the alarms would give rise to different results.

This situation can be remedied with a conflict resolution strategy, which is quite simple; it gives preference for as many secondary alarms as possible. For example, in the case above, the first alternative interpretation of the situation should be preferred. This conflict resolution will give rise to some further diminishing of the diagnostic resolution and more complexity in the rules, but the resulting interpretation will always be correct.

At a glance, there may seem to be several possibilities for conflicts. There are, however, only two types of cases, concerning storages and passive transports. This follows from the fact that, for a conflict to be possible, there must be multiple and different alarm causations for the same flow function. Storages may be caused to have both *lovol* and *hivol* alarms, and passive transports may be caused to have *loflow* and *hiflow* alarms, but no other type of flow function may be caused to have more than one type of alarm. Thus, there are possibilities of conflict in transport-storage-transport connections, as well as around passive transports, but nowhere else. These cases can be recognized and treated with special rules.

It is an open question whether this conflict resolution should be used or not. Sometimes it may be good to use timing information and the order of incoming alarms in the diagnosis. A reasonable solution would probably be to allow conflict resolution only if the alarms appeared within a small time interval, but not otherwise.

5.12 Measurement Faults

Sensor faults are common sources of false or uncorrect alarms. In such cases the state of the process will not be correctly shown in the alarm presentation, and situations where this occur can be very dangerous.

The method described is not aimed at discovering uncorrect alarms. Instead it is sensitive to false alarms and largely dependent on correct sensor values. However, some cases can be detected as situations where the alarm state contains a sensor fault. This is true when one alarm will *force* another. If the latter alarm is not active, something is wrong with the measurements leading to one, (or more), of the alarm states.

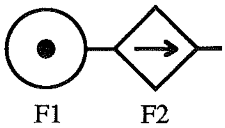


Figure 5.8 A connected source and transport. If the source has a *locap,* the transport must have a *loflow.* If this is not the case, something is wrong with the measurements.

If, for example, a source and a transport is connected, as shown in Figure 5.8, and the source has a *locap* alarm, the transport must have a *loflow* alarm. Should this not be the case, then either of the flow functions is in an incorrect alarm state, and the algorithm can issue a warning about this. However, it can give no answer as to which alarm that may be incorrect, and a more thorough measurement validation will have to be performed with other methods, see for example Chapter 4.

5.13 Using the Results

The result from applying the described method is that alarms will be marked as either primary or primary/secondary. An operator or algorithm can then use this information in the task of finding the primary faults in a fault situation.

An obvious strategy for the user is to first check the faults known to be primary. When these have been remedied, most of the secondary faults will also be gone, while some may instead have become primary. These should then be taken care of.

It is possible, however, to use other knowledge to guide the search through the alarms. If failure likelihoods are known or can be guessed, they can be used for ordering the actions of the operator. Likewise, if one alarm is more important or dangerous than another, it may be investigated first, although it is secondary and other primary alarms remain. This knowledge is heuristic in its nature, and should be clearly separated from the results of the alarm analysis. It may be implemented using a standard rule base to discriminate between different situations and giving advice on which actions to take first.

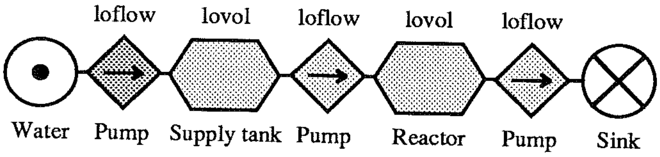


Figure 5.9 An MFM model of the main water flow through a nuclear reactor. The water is pumped from a source to a supply tank, and from there on through the reactor tank itself, and further on to a sink. In the situation shown, all pumps have *loflow* alarms, and both the supply and reactor tanks have *lovol* alarms. The alarm analysis recognizes that the first pump must have a primary fault, while the others may be consequences of this. However, the *lovol* of the reactor tank is by far the most dangerous alarm, and this may be treated first by the operator, by activating a reserve filling system.

An MFM model of the main moderating and cooling water flow through a nuclear reactor is shown in Figure 5.9. In the current situation all pumps have *loflow* alarms and both the supply tank and the reactor itself have *lovol* alarms. The alarm analysis algorithm singles out the first pump as the only primary fault, while all the other alarms may be consequences of this. The normal operating procedure could now be for the operators to investigate and fix the pump fault, and then see whether any other alarms remain.

In this example, however, the low volume of water in the reactor may be a much more dangerous fault than the others, and the operators may use this knowledge to first investigate the condition of the reactor, checking whether there are any leaks or other problems, and using separate water systems to remedy the situation in the reactor, before taking care of the other alarms.

5.14 Implementation

The algorithm is implemented with two rule groups. One is concerned with the alarm analysis and consists of 39 rules, the other takes care of the consequence propagation including conflict resolution and consists of 25 rules. These rule bases use a database of connected flow functions to yield an incremental and local algorithm, i.e., the alarm states of the flow functions are updated as soon as new values are available, and only local information about neighboring functions is needed. Further information about the G2 implementation is given in Chapter 7.

The algorithm works by updating a set of attributes, see Table 5.1.

`alarm state`	*locap, loflow, hiflow, lovol, hivol, leak, or fill*
`failure state`	*normal, primary failed, or secondary failed*
`alarmed`	*true or false*

Table 5.1 The alarm analysis attributes of a flow function. The **`alarm state`** is set by external measurements or guessing, and the algorithm sets the *failure state* with the help of the causation rules. The *alarmed* attribute tells whether the flow function is question is connected to an alarm, or if its *alarm state* must be guessed by the flow propagation algorithm.

The local and incremental nature of the algorithm makes it very efficient. The updating starts at the points of data entry and follows static links though the MFM model. Thus, the effort is at worst *linear* in model complexity, i.e., how many objects the model consists of. In the normal case, however, only smaller parts of the model is traversed, and the algorithm is even more efficient.

5.15 Examples of How the Method Works

Let us now demonstrate the method on an example. Once again the small tanks process will be used, see Figure 5.10.

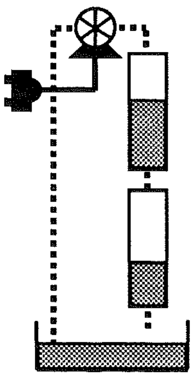


Figure 5.10 The tanks process. Water is pumped from a storage tank, to a cylindrical tank, from where it flows down into another tank, and then back to the storage again.

The MFM model of this process is shown in Figure 5.11.

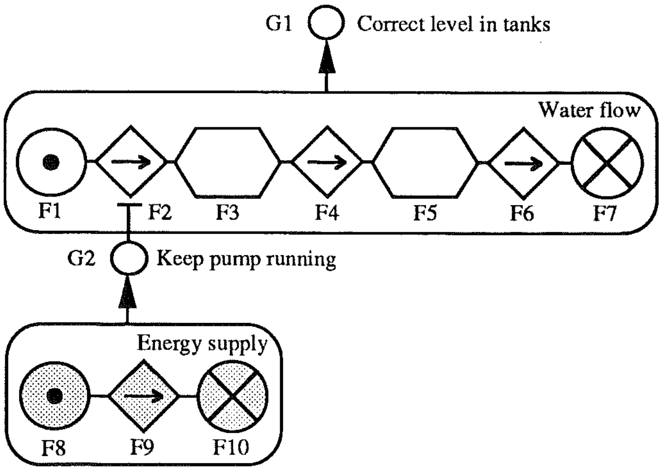


Figure 5.11 An MFM model of the tanks process. The main goal is to keep the level of the upper tank correct, and it is achieved by a water flow. In order for the transport function **F2** to be available, i.e., to keep the pump running, energy must be supplied.

EXAMPLE 5.4
Assume that the functions **F1, F2, F3,** and **F5** have measurements connected to their alarm states, while **F4** and **F6** have not. Further assume that **F1** has a *locap,* **F2** a *loflow,* **F3** a *lovol,* and **F5** a *lovol* alarm. This alarm situation is shown in Figure 5.12. The shading of

the flow function symbols is used to indicate the failure state, (a dark shade means a primary alarm, a lighter shade a primary or secondary, and white a normal or unalarmed state).

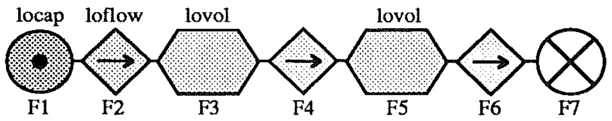


Figure 5.12 An alarm analysis situation. The functions **F1, F2, F3,** and **F5** have *locap, loflow,* and *lovol* alarms. The alarms of **F4** and **F6** have been guessed. In this situation, the *locap* of **F1** is the only primary alarm.

This would correspond to the plethora of alarms that could appear in, say, a complicated fault situation in a larger plant, although this situation is, of course, far simpler.

An application of the presented method will result in that the *locap* of **F1** must be a primary alarm, while the *loflow* of **F2** and the *lovol* of **F3** may be secondary. The consequence propagation implies that **F4** and **F6** might have had *loflow* alarms, had they been measured. Thus, assuming that **F4** has a *loflow* alarm, the alarm analysis can also conclude that the *lovol* of **F5** may be a secondary alarm. The result is that the *locap* of **F1** is the only primary alarm, while all the others may be consequences of it. **F1** is the source function of the storage tank, and the sole cause of the fault situation could thus be that there is too little water in that tank. □

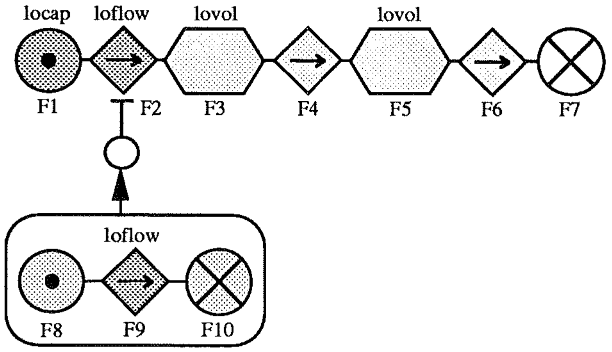


Figure 5.13 Another alarm analysis situation. Here the *loflow* of **F2** must be primary, as there is an alarm in the achieving network, (the *loflow* of **F9**).

EXAMPLE 5.5

If the function **F9,** (transport of electrical energy to the pump motor), was to have an alarm also, the last rule in the rule set, (the rule concerning causation via the *achieve* and *condition* relations), would imply that **F2,** (the pump), also had had a primary fault, i.e., there would now be at least two primary faults: no power supply for the pump and too little water in the storage tank, see Figure 5.13. □

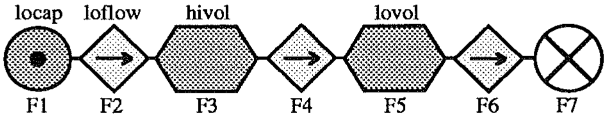


Figure 5.14 A third alarm analysis situation. Here there are three primary alarms. The situation is highly dynamic.

EXAMPLE 5.6

If instead **F3** had a *hivol* alarm, the algorithm would conclude that there were three primary alarms, the *locap* of **F1,** the *hivol* of **F3,** and the *lovol* of **F5,** see Figure 5.14. This would correspond to a dynamic process state, where the pump flow and the volumes of the lower and storage tanks were too low, while the volume of the upper tank was too high. □

5.16 A Comparison with MIDAS

The MIDAS system of Kramer *et al*, see Finch (1989) and Oyeleye (1989), analyzes measurements to find deviations from a nominal steady state. All measurements are turned into alarms, which are grouped in clusters, each connected to one primary fault, or *root cause.* MIDAS uses several types of models, where each type is translated into the next one more or less automatically. An event interpreter uses the final representation to construct an on-line graph of the actual alarm events, their links, and root causes. Thus, the alarms are analyzed and, if possible, connected. The root causes in the on-line graph represent the actual faults.

MIDAS has similarities with the alarm analysis algorithm presented here. There are several important differences, however, and the main ones are:

- MIDAS allows equations of any form to be the basis of the system representation, i.e., the Signed Directed Graphs, while MFM imposes a strong structure, allowing flow balance equations only. This means that MIDAS gives a lot of freedom, but that it is quite complicated to handle the equations with satisfactory results. The corresponding MFM algorithms may be much simpler and more efficient, as they have a simpler problem to solve.

- MIDAS has a memory of events, while the MFM alarm analysis algorithm, (like the DMP system of Petti), only shows a snapshot of the current situation, and updates its outputs whenever new events occur.
- MIDAS demands a large computational effort, both in model transformation and during runtime, while the MFM algorithms are very efficient.

5.17 Conclusions

This chapter describes a method that can distinguish positively primary alarms from those that may or may not be primary. An important aspect of the method is that it will not make any guesses as to which interpretation is the most likely; instead the resulting separation of alarms is guaranteed to be correct if the modeling assumptions and measurements are valid.

The algorithm has been implemented and tested on the same two G2 simulations as the measurement validation: the small laboratory process and Steritherm. It could successfully distinguish between primary and secondary alarms in all test situations.

The method works under rather general conditions; there should be an MFM model of the process, this model should capture the important aspects of the process, and there should be no faults in the alarm measurements. Under these conditions the method provides a simple solution to the problem of alarm analysis.

CHAPTER 6

FAULT DIAGNOSIS

Expert systems have been used with good results in fault diagnosis. However, a large rule database is not easy neither to build, maintain, nor document. Furthermore, a standard rule-based system can only diagnose faults that were considered during the design of the knowledge database. This chapter presents a method for fault diagnosis that uses MFM models instead of specially designed rule bases. This allows the knowledge database to be built, updated, and documented in the MFM graphical language, and internal consistency is also guaranteed. In addition, the normative nature of MFM enables the system to detect any fault that is a deviation from the working state of the process.

6.1 Introduction

The classical use of knowledge-based systems in process control is to aid the process operator in diagnosing faults. This is usually done by a rule-based expert system, running the rules in backward chaining. The techniques are well-known, a good example being MYCIN, see Shortliffe (1976).

The development of an expert system for fault diagnosis comprises several phases, each demanding a large effort:

- An initial phase consisting of problem definition and the search for knowledge sources, i.e., written information and, most important, the right experts.
- A knowledge acquisition phase, where the available information is retrieved and structured so that it can be used for the implementation of a rule base.
- An implementation and testing phase, where the knowledge is written in rule form and the resulting system is tested.
- A validation phase, where the system is further tested and evaluated, to see whether it meets all desired specifications.

These phases are all quite demanding in effort, and for each new project they must be performed once more, i.e., each process and task needs a new rule base, and therefore new knowledge acquisition and implementation.

6.2 An Example of a Rule-Based Expert System

A small example of a target process for fault diagnosis is the tanks process, which is shown in Figure 6.1.

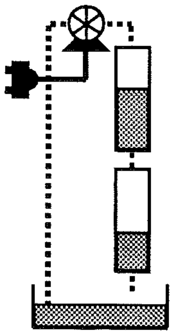


Figure 6.1 The tanks process. Water is pumped from a storage tank, to a cylindrical tank, from where it flows down into another tank, and then back to the storage again.

A small rule base for fault diagnosis using backward chaining is shown in Figure 6.2–4. The rules are written in the format used by the small expert system shell MESS, Larsson (1988). This shell is written in the Lisp dialect Scheme and allows for simple pattern matching, and forward and backward chaining.

The goal satisfaction rules, see Figure 6.2, simply define the relations between the goals and the supporting functions, i.e., the *achieve* relations. It should be observed that one rule is needed for each goal, and that every flow function in a supporting network must be mentioned in the premisses.

```
(rule goal-1
  (if
    (F1 is working)
    (F2 is working)
    (F3 is working)
    (F5 is working))
  (then
    (G1 is satisfied)))
```

```
(rule goal-2
  (if
    (F8 is working)
    (F9 is working))
  (then
    (G2 is satisfied)))
```

Figure 6.2 Rules for goal satisfaction. These rules correspond to the achieve relations of MFM, and are used to guide the backward chaining search downwards from goals to flow functions. Note that goals and functions have the same names as in the MFM examples.

The rules of Figure 6.3 are responsible for going through the single flow functions of the water flow network and finding out their fault status. They also contain the remedies needed for fixing them, should these functions be at fault.

```
(rule waterflow-1-a
  (if
    (* there is water in the storage tank))
  (then
    (F1 is working)))

(rule waterflow-1-b
  (if
    (not (there is water in the storage tank)))
  (then
    (action (fill water in the storage tank))
    (remove (not (there is water in the storage tank)))
    (there is water in the storage tank)
    (F1 is working)))

(rule waterflow-2-a
  (if
    (* the pump is pumping water))
  (then
    (F2 is working)))

(rule waterflow-2-b
  (if
    (not (the pump is pumping water))
    (G2 is satisfied))
  (then
    (remove (not (the pump is pumping water)))
    (the pump is pumping water)
    (F2 is working)))

(rule waterflow-3-a
    (if
      (* the upper tank water level is correct))
    (then
      (F3 is working)))

  (rule waterflow-3-b
    (if
      (not (the upper tank water level is correct)))
    (then
      (remove (not (the upper tank water level is correct)))
      (the upper level is fixed by other actions)
      (the upper tank water level is correct)
      (F3 is working)))

  (rule waterflow-4-a
    (if
      (* the lower tank water level is correct))
    (then
      (F5 is working)))

  (rule waterflow-4-b
    (if
      (not (the lower tank water level is correct)))
    (then
      (remove (not (the lower tank water level is correct)))
      (the lower level is fixed by other actions)
      (the lower tank water level is correct)
      (F5 is working)))
```

Figure 6.3 Rules for diagnosing the water flow. These rules take care of diagnosing each flow function, by asking questions, suggesting remedies, and concluding that the flow function in question is working. Two rules are needed for every flow function; one for the working case and one for the corrective action case.

In a like manner, the rules of Figure 6.4 go through the flow functions of the power support network for the pump, finding faults and giving remedies.

```
  (rule power-support-1-a
    (if
      (* power is supplied))
    (then
      (F8 is working)))

  (rule power-support-1-b
    (if
      (not (power is supplied)))
    (then
      (action (power must be supplied))
    (remove (not (power is supplied)))
    (power is supplied)
    (F8 is working)))

(rule power-support-2-a
  (if
    (* the power switch is on))
  (then
    (F9 is working)))

(rule power-support-2-b
  (if
    (not (the power switch is on)))
  (then
    (action (switch on the power))
    (remove (not (the power switch is on)))
    (the power switch is on)
    (F9 is working)))
```

Figure 6.4 Rules for diagnosing the electrical energy flow. These rules diagnoses each flow function in turn, and once again two rules are needed for each function.

The rules can be used to perform a simple fault diagnosis, by asking questions about the state of the equipment and issuing remedies for how to fix the possible faults. The backward chaining tries to prove the hypothesis "`G1 is satisfied`" and moves down via rule `goal-1,` goes through the water flow rules, and via the rule `goal-2` further down into the electrical energy flow rules. An example run is found in Figure 6.5.

```
> (diagnose)
Is it true that there is water in the storage tank? (y/n)
n
PERFORM:           fill water in the storage tank
Deduction/advice:  there is water in the storage tank
Deduction/advice:  f1 is working
Is it true that the pump is pumping water? (y/n)
n
Is it true that power is supplied? (y/n)
y
Deduction/advice:  f8 is working
Is it true that the power switch is on? (y/n)
n
PERFORM:           switch on the power
Deduction/advice:  the power switch is on
Deduction/advice:  f9 is working
Deduction/advice:  g2 is satisfied
Deduction/advice:  the pump is pumping water
Deduction/advice:  f2 is working
```

```
Is it true that the upper tank water level is correct? (y/n)
n
Deduction/advice:  the upper level is fixed by other actions
Deduction/advice:  the upper tank water level is correct
Deduction/advice:  f3 is working
Is it true that the lower tank water level is correct? (y/n)
n
Deduction/advice:  the lower level is fixed by other actions
Deduction/advice:  the lower tank water level is correct
Deduction/advice:  f5 is working
Deduction/advice:  g1 is satisfied

ok
```

Figure 6.5 A fault diagnosis session. The expert system shell MESS uses the rules presented above to perform the diagnosis. The search starts from the top level goal and moves down via the water flow rules and into the electrical energy flow rules. `PERFORM` means that an action should be taken, while `Deduction/advice` is a more general comment or advice.

In the session shown in Figure 6.5 several faults are found. The system goes through the different parts of the process, discovers the faults, and offers remedies. As the pump is not working, the diagnosis must be taken down into the pump power support system.

6.3 Problems with Rule-Based Expert Systems

The example above was included to show how a standard, rule-based expert system would be used for fault diagnosis. The techniques of building and using such systems are now more or less mature. They have, however, several shortcomings, some of which MFM can be used to remedy:

- Each rule base is specific for a certain process and task. Thus, a new system must be designed, built, and validated each time fault diagnosis is needed for a new process.
- A rule base may contain inconsistencies, and a large rule base most probably will do so. It is very difficult to guarantee consistency between the rules in an automatic fashion.
- A large rule base is difficult to overview, both in building and updating.
- A rule-based system can only diagnose the faults anticipated in the design of the rule base.

These shortcomings can be solved to a large degree by using MFM. The whole process of design, construction, and updating of the knowledge database becomes much more efficient, as the MFM models are easy to present graphically, and thus quicker to build and change. The other definite advantage of MFM is that consistency within the model is guaranteed. The graphical syntax makes inconsistencies in the database impossible. In addition, MFM can find any deviation from the working state.

Thus, MFM gives the following advantages over a standard rule-based system:

- Each task can be handled by a general rule base, using an MFM model as data. Thus, the same generic rules may be used for many different target processes.
- The graphical syntax of MFM guarantees internal consistency of the database; only correct couplings can be made. Of course, the modeling can differ from reality, and thus cause faults is the diagnosis, but there cannot be any conflicting rules in the constructed model.
- The MFM model can be presented graphically to the designers and users. Thus, the problem of orientation in a maze of rules in a large rule base is avoided.
- Any fault that means a deviation from the working state can be discovered by the MFM algorithm.

6.4 The MFM Data Structure for Fault Diagnosis

In MFM the means-end dependencies are explicitly represented, so when a certain control goal fails, i.e., a fault occurs, the model will provide information on which functions that may be in error, and thus, in which component sub-systems the reasons for the failure can be found.

The working conditions for flow functions used in the fault diagnosis algorithm are the same as used in the alarm analysis, see Chapter 5. Thus, each flow function might be in a normal, (or working), state, or have a fault, more or less directly corresponding to a *locap, loflow, hiflow, lovol, hivol, leak,* or *fill.*

The fault diagnosis algorithm must have a way of finding out the failure states of the physical components corresponding to the different flow functions. Thus, each flow function may have a question to be asked, or a check or test to be performed, in order to investigate the failure state of the function.

Each flow function can also have a remedy of the fault, in the form of a text string to be output by the algorithm.

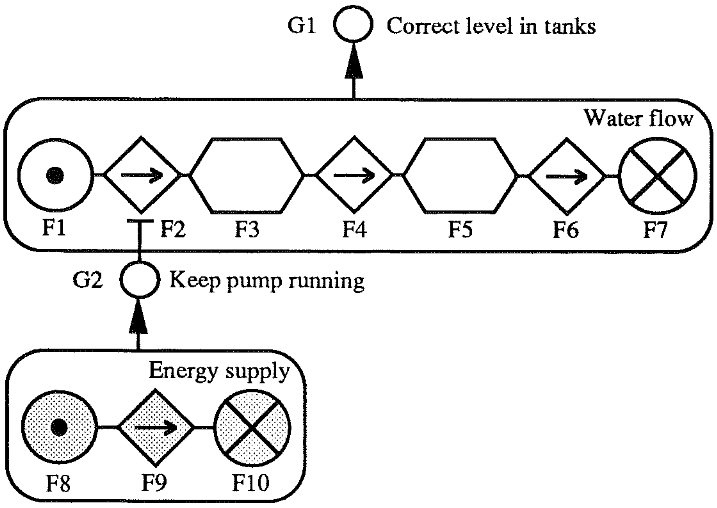


Figure 6.6 An MFM model of the tanks process. The main goal is to keep the level of the upper tank correct, and it is achieved by a water flow. In order for the transport function `F2` to be available, i.e., to keep the pump running, energy must be supplied.

In Figure 6.6 we once again see the MFM model of the tanks process. In order to enable a fault diagnosis, the different flow functions should be assigned questions:

- The source `F1` could have the question "`Is there enough water in the storage tank?`" associated to it, together with the remedy "`Fill water in the storage tank.`"
- The transport `F2` could have the question "`Is the pump transporting water?`" but no remedy, as the pump has a supporting system enabling it to run. The remedies would probably be taken care of down in that system.
- The storage F3 could have the question "`Is the water level of the upper tank correct?`", but it could also have a special rule associated to it. This rule could be activated by the algorithm and use an external measurement to set the failure state of the flow function. The remedy could be "`The water level of the upper tank will be corrected by other actions.`" This implies that once the pump and controller is working, the level will soon be regulated to its desired value.

- The rest of the flow functions in the water network could have similar questions and remedies. It is also possible, however, that some of the flow functions had no questions or remedies. These would simply be skipped by the algorithm.
- The source `F8` in the electrical energy network could have the question "`Is power available?`"
- The transport `F9` could have the question "`Is the power switch on?`" and the remedy "`Switch on the power.`"

These are the data necessary for the fault diagnosis algorithm to work; to give it information from the outer world, either by asking questions to the operator of the process, or by activating rules, which in turn can cause measurements to be made or demon processes, (data event activated processes), to be executed. The final result of the activation should be that the failure state of the flow function concerned is set correctly.

6.5 The Search Strategy

An MFM model consists of information about the goals of a process, how these goals are achieved by networks of functions, how the functions depend on subgoals, and how they are realized by physical components. In a standard rule-based expert system, this information structure is implemented in rules, but in MFM it is explicitly described. Thus, a fault diagnosis can be easily implemented, as a search in the model graph. The strategy used for this search is as follows.

The fault diagnosis algorithm traverses the MFM graph, and when it comes to single flow functions it uses the questions or rules to find the failure state of those flow functions. Depending on the answers to the diagnostic questions, parts of the MFM model may not have to be traversed. The algorithm is combined with the alarm analysis and consequence propagation, which is performed incrementally as information comes in, and interleaves with the fault diagnosis algorithm. The simple rule for successful matching of diagnosis and consequence propagation, (i.e., guessing of consequences), is that every flow function should have *either* a diagnostic question *or* be subject to guessing.

The specifics of the diagnostic search are as follows:

- The user chooses a goal for diagnosis. If this is a top-level goal, the whole model, (and thus the whole process), will be investigated.

However, the goal chosen can also be a subgoal, in which case only part of the process will be diagnosed.

- The search propagates downwards from the goal, via achieve relations, into the connected network of flow functions, each of which is now investigated.
- Each flow function may have a diagnostic question, which is asked in order to find out whether the corresponding physical component is currently realizing the function, i.e., whether the function is available or not. Alternatively, there can be a rule or relation to a physical component, whereby information about the working order of the function may be found.
- The appropriate alarm state of the flow function is set, and the alarm analysis and consequence propagation algorithms are activated.
- If a flow function conditioned by a subgoal is found to be at fault, or has no means of being checked, the connected subgoal is recursively investigated. If, however, a function is working, that part of the subtree is skipped.

The fault diagnosis method can be implemented quite easily, with a group of generic rules, which yields an incremental and local algorithm. It should be noted that the search propagates along static connections. Thus neither global search, pattern matching, nor conflict resolution is needed, and the algorithm is quite efficient.

6.6 Implementation

The diagnostic search is implemented with 12 rules and 7 procedures, and uses the MFM model as database. The algorithm works by updating a set of attributes, see Table 6.1.

`diagnose`	*true or false*
`ask`	*true or false*
`question`	*a text string*
`explain`	*true or false*
`explanation`	*a text string*
`fault state`	*locap, loflow, hiflow, lovol, hivol, leak, or fill*

Table 6.1 The flow function attributes used by the fault diagnosis algorithm. The `diagnose` attribute is set to true by a downwards search along condition

> and achieve relations and is used to tell which parts of the MFM model that should be investigated. The *ask* attribute tells whether there exists a diagnostic question for the concerned flow function, and *question* contains that question string itself. Likewise, the *explain* attribute tells whether there exists an explanation or remedy, and *explanation* contains that explanation or remedy. The *fault state* is used to track the fault condition of the flow functions during the search.

The local and incremental nature of the algorithm makes the method more efficient than a rule-based expert system. The search starts at the goals selected for diagnosis by the user, and follows static links though the MFM model, as a standard depth-first search. When subtrees are skipped, which happens when a conditioned flow function is known to be working, the search uses the geographical placement of the objects on the screen to select the next goal or function to be investigated. This makes the effort *linear* in model complexity, except during skip operations. In the normal case, a large part of the time is spent in the quick, linear search, and the algorithm is quite efficient.

The method uses the the same alarm state variables as the alarm analysis algorithm. As the latter is activated whenever an alarm state changes, the two algorithms interleaves without any additional communication needed. The fault diagnosis traverses the MFM graph and sets the alarm states according to the questions and test results, and as soon as new information is available, the failure states are updated by the alarm analysis and consequence propagation algorithms. Thus a picture of the fault causation develops during the diagnostic search.

The method of this chapter has been described as an off-line algorithm, using queries to the operator of the process. Apart from that, however, there is nothing in the algorithm that demands execution off-line. In fact, if all queries can be settled by automatic tests instead of questions, the method works fine on-line. In this case, the top-down search makes the algorithm be as efficient as possible, testing only faulty parts of the MFM model.

6.7 Examples of how the Method Works

Let us now demonstrate the method on a small example. Ohce again, the tanks process, see Figure 6.1, will be used. An MFM model of the process is shown in Figure 6.6.

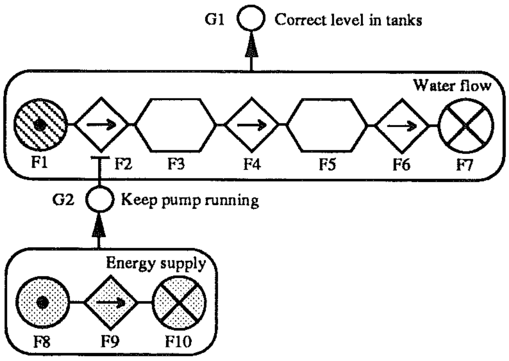


Figure 6.7 The diagnostic search has started from the goal **G1**, followed the achieve relation down into the water flow network, and reached the source **F1**.

EXAMPLE 6.1

Assume that the level of the upper tank is not correct, i.e., the goal **G1** is violated, and that the user asks for a diagnosis of that goal. The algorithm starts at the goal **G1**, i.e., the topmost goal, and moves down into the network describing the mass, (water), flow and checks the flow functions in turn.

The source **F1** describes the source function of the storage tank, and is the first flow function to be reached by the search algorithm. The current position is marked in the graphic representation of the MFM model, see Figure 6.7. The source **F1** has the following question associated to it:

Q: **Is there water in the storage?**

The user checks this and discovers that there is almost no water left in the storage tank. Thus he gives the answer '**no**' and **F1** is marked with a *locap*. The alarm analysis is activated but can draw no further conclusions, see Figure 6.8.

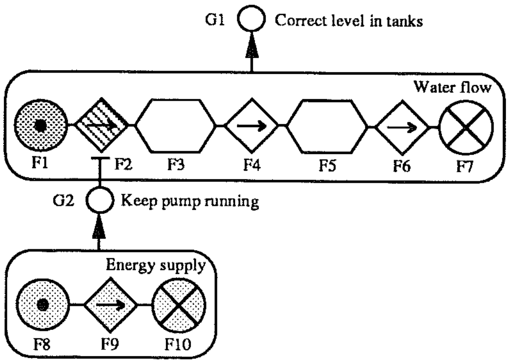


Figure 6.8 The diagnostic search has concluded that the source **F1** is faulty. Then it has moved on to reach the transport **F2**.

The algorithm now moves on to `F2` and asks the following question:

Q: `Is the pump running?`

Once again, the answer is 'no' and `F2` is marked with a *loflow* alarm. The alarm analysis is activated and deduces that the *locap* of `F1` must be a primary fault, while the *loflow* of `F2` may be caused by the fault of `F1`, see Figure 6.9.

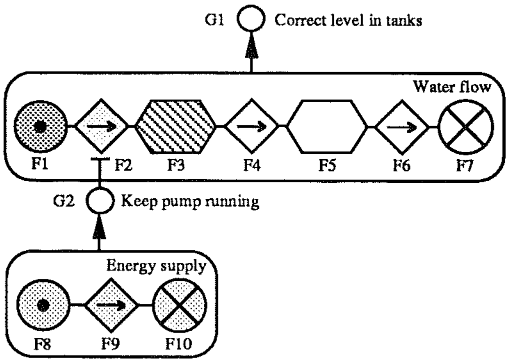


Figure 6.9 The diagnostic search has concluded that the transport *F2* was at fault, and the alarm analysis signals that the fault of `F1` is primary, while the fault of `F2` may be secondary. Then the search has reached the storage `F3`.

The storage function `F3` corresponds to the upper tank. It could have a question associated to it, that asked whether the volume of that tank was within the correct limits. Assume, however, that it is connected to a level alarm sensor. It will automatically be assigned a *lovol* alarm. Before the diagnosis has started, this alarm was considered primary, but once the *loflow* of `F2` has been established, the alarm analysis algorithm decides that it may be a consequential fault.

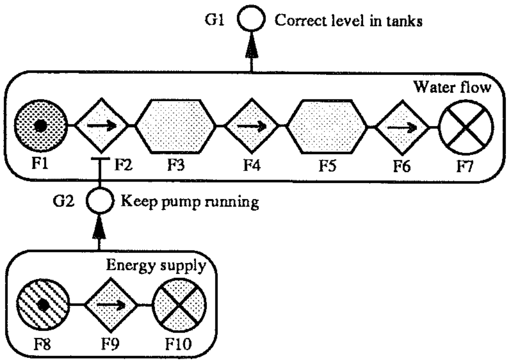


Figure 6.10 The diagnostic search has finished its investigation of the water flow network, and the alarm analysis concludes that there is one primary fault, three secondary, and two guessed faults, at `F4` and `F6`. As the transport `F2` was at fault and there is a condition relation, the search has continued down to the energy supply network and has reached the source `F8`.

The transport function **F4** corresponds to the gravitationally caused outflow from the upper tank. It has no alarm and no question, so here the consequence propagation will be used to guess the alarm state, which will be a *loflow.*

The storage **F5** corresponds to the lower tank, and is also connected to an alarm sensor. Under the reasonable assumption that the level of this tank is too low, it is automatically marked with a *lovol,* and the algorithm can now use the guessed *loflow* of **F4** to decide that the *lovol* of **F5** also is a consequential fault.

The flow functions **F6** and **F7** are neither alarmed or have questions; thus the algorithm would guess that **F6** ha a *loflow* alarm, while **F7** is in a normal state. All this can be seen in Figure 6.10.

As there was a fault in **F2,** the algorithm now goes down in the subtree below it, and starts diagnosing the goal **G2.** It moves further down, finds the source **F8,** see Figure 6.10, and asks the question that belongs to it:

Q: **Is power supplied?**

The user checks that the power line is connected to the wall, and that other equipment seems to have electrical power; then he gives '**yes**' as the answer.

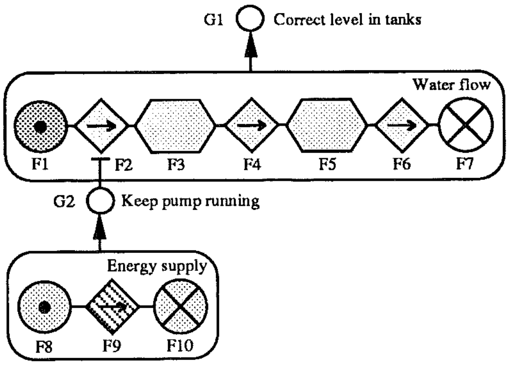


Figure 6.11 The diagnostic search has concluded that the source **F8** is working and reached the transport **F9.**

When the algorithm comes to the transport **F9,** see Figure 6.11, which corresponds to the power switch of the pump, it asks the following question:

Q: **Is the power switch on?**

The user discovers that the power switch is in the 'off' position and answers '**no**' to this question. The transport function **F9** is marked with a **loflow** alarm. The alarm analysis now deduces that the fault of **F2** was indeed primary, as there is a fault in its support system. The total fault situation is thus that there are two independent causes of the level being to low; there is not enough water in the storage tank, and the pump power switch is not on, see Figure 6.12.

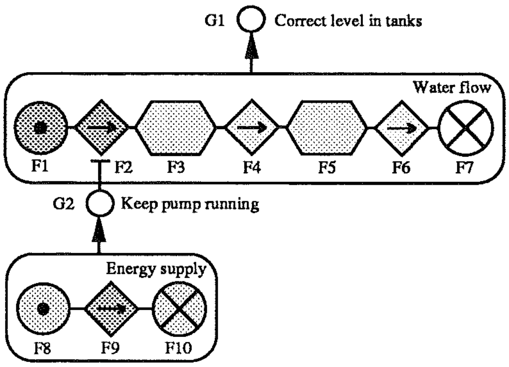


Figure 6.12 The state after the fault diagnosis. As the transport **F9** is at fault, the fault in **F2** is also a known primary fault. The user may now ask for explanations and remedies, and the algorithm will search through the graph and output the appropriate text strings from the failed flow functions.

This system state will probably not last very long, however, as the water presently in the two cylindrical tanks will flow down into the storage tank and fill it up, thus making the source function **F1** available again.

The implemented system also allows the flow functions to have explanations and remedies associated with them, and these can now be asked for. In the example, the algorithm would go through the different primary faults. When **F1** is reached, it would output the following remedy:

R: **Fill water in the storage tank.**

When it reaches the power switch, **F9,** it would output another remedy:

R: **Switch on the power.**

Instead of writing out a remedy in text form, the system could also activate rules or procedures to perform any actions needed. These rules and procedures should be written in the general G2 rule format. □

6.8 A Comparison with PERFECT

The PERFECT project of Sassen *et al* has several similarities with the method presented in this chapter, but it also differs in many important aspects:

- The PERFECT system operates continuously on-line. The method of this chapter may either be activated by the operator or a supervision algorithm, or operate continuously.
- As it is an on-line method, PERFECT uses only measured values, while the method presented here can use either questions asked to the operator or measurement values. Thus, the latter method allows for more freedom in input data and greater diagnostic resolution.
- PERFECT continually checks the working condition of *every* leaf node in the MFM graph, which may become computationally expensive, and then propagates the fault information upwards. The method presented here searches downwards, skipping large parts of the MFM model; thus only a fraction of all leaf nodes must be examined. While PERFECT tests every possible fault, the latter method performs a knowledge-based search directly aimed at the faulty parts of the process. However, the PERFECT strategy has the advantage that sometimes it may discover a fault which has not yet lead to the violation of a goal.

6.9 Conclusions

This chapter describes how MFM models can be used for fault diagnosis. The method works under rather general conditions; there should be an MFM model which captures the important aspects of the process. The advantages of the method are that the knowledge database is easy to construct, modify, and describe, that consistency is guaranteed, and that the search performed is quite efficient.

The method has been implemented and tested on G2 simulations of two processes, the small laboratory process and Steritherm. The method was successful in all test situations.

The efficiency of the method is shown by a simple implementation in C, executing on-line using only tests and no questions, see Larsson (1992 c). Searching through the whole Steritherm model, (more than 100 MFM objects), takes only 200 microseconds on a SPARC station 2, while finding a fault in the thermal energy network takes less than 50 microseconds. The worst case of executing time can easily be found out; for the Steritherm it is 200 microseconds. Clearly, the presented method enables knowledge-based fault diagnosis in an on-line control algorithm.

CHAPTER 7

AN MFM TOOLBOX

The algorithms described in the previous chapters have all been implemented in G2, and tested on the tanks process and Steritherm. This implementation is a starting point for a toolbox for building and using MFM models in G2.

The implementation consists of definitions of data structures and graphics for building topological models and MFM graphs, a set of rule bases that perform the diagnostic reasoning tasks, and a set of equations used for simulation of the processes. In total, the program contains the following:

- Definitions of a class hierarchy of the objects in an MFM model.
- Rule bases for syntax control of MFM models, measurement validation, alarm analysis, consequence propagation, and fault diagnosis.
- Definitions of a class hierarchy of the components in the processes.
- Generic simulation equations for the physical components.

The implemented program may be taken to pieces and the MFM specific parts saved separately, with no definitions left for any specific process. Thus, only the MFM model definitions and the rule bases are kept, as an MFM toolbox. The user may define his own components and construct a topological simulation model, which is the standard way of using G2. He then builds an MFM model using the toolbox definitions, and when that model is ready, he immediately has available the diagnostic algorithms described in previous chapters, instead of having to construct a rule base for the specific process. Thus, the general ideas is to replace the construction of a rule-based expert system with the building of an MFM model using the toolbox, see Figure 7.1.

MFM model + Toolbox = Expert System

Figure 7.1 With the help of the toolbox, it is possible to replace the large effort of building a rule base for an expert system with the construction of an MFM model.

This chapter will describe a suitable general architecture for a system for diagnostic reasoning with MFM: it will give an overview of the different rule bases, it will describe G2, and it will go through the different parts of the implemented toolbox.

7.1 Architecture of a Supervisory and Control System

In the previous chapters only the methods or algorithms for the three diagnostic reasoning tasks have been described. However, to use such methods, they must be successfully implemented and built into a system which also provides data collection, user presentation, other algorithms, and a systematic way of connecting the different parts with each other. In order to design such a system, the architecture of it must be defined. This is a special case of the standard problem of writing a computer program, and several well known concepts can come to use.

A monitoring and supervisory system must perform many different types of tasks, in widely different time scales, and preferable implemented with different techniques:

- Data collection at sensor level
- Data filtration and other low level treatment
- Estimation, reconstruction, and statistical treatment
- Data reconciliation and measurement validation
- Supervision and diagnosis, automated reasoning
- Advice generation
- Presentation for the user or operator
- User help and support

Here, the data collection and filtration tasks will usually be distributed to smaller processors and performed in fast software or even specially designed hardware, and be working in time scales from microseconds to seconds. The more complicated statistical analysis is usually performed on computers, and operates in the time scale of seconds.

Tasks on the supervision and diagnosis level is usually performed by humans, but can also be taken care of by knowledge-based systems, running on larger workstations and operating in the time scale of seconds or minutes. The presentation for the human operator, and any help and support for him must also operate in this time scale.

Thus, a system that can handle all these different tasks efficiently should be constructed out of several hardware and software techniques, and be designed in several layers, see Figure 7.2.

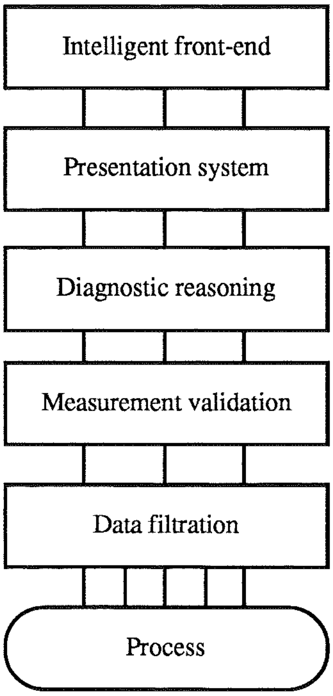


Figure 7.2 A control and supervisory system should have a layered architecture, as the tasks of different levels are performed at different time scales and by different hardware and software.

In such an architecture, the different layers may be implemented in different hardware and software. The only strong provision is that there be clear interfaces between each level, and that the whole system is built in a clear and systematic fashion, which certainly implies a strong hierarchical design.

All the standard concepts for structuring a design task may and should be used here. These are, for example, the use of *class hierarchies, object-oriented programming,* or at least an object-oriented programming style, and the clear separation of different programming concepts such as *task driven* and *data driven* activation, the use of *demons,* and so on.

It should be noted that on top of the diagnostic level there should be a presentation and communication level. The use of more advanced knowledge-based diagnosis methods demands and enables a better and more user-friendly interface. Thus, it is in no way certain that the standard supervisory system communication is good enough. It would certainly be useful to have available several new ways of presenting process information to the operators in a human way. Here, at least two problems are worthy of notice:

- The problem of presenting *means-end* information
- The possibility of giving *knowledge-based* help and support

Presentation of Means-End Information

An MFM model of a process may, as have been shown in the previous chapters, be useful in that it allows a simple and uniform way to build a database for different diagnostic reasoning tasks. In this case the model must be constructed but is only used internally by the algorithms. When the process is redesigned or changed, however, the MFM model must also be updated.

However, the MFM models contain more information than is used by any of the different algorithms, and this information remains hidden in the MFM graphical data structure. This means-end information is not available through any other type of model; it is a separate view containing original information, but which may be of vital importance for the process operators in their tasks.

Thus, it would most certainly be of great benefit if the means-end information could be presented in an understandable way. On the other hand, means-end information is very different from physical topology, and its nature is often rather complex and hard to understand at a first glance. Therefore, it is not clear that the current graphical language used to represent MFM models is a good alternative for presentation of means-end information for the operator. Chapter 8 of this work will treat these problems closer, and give some examples of alternate presentations of MFM models. It is clear, however, that a solution to this problem would be very valuable.

Intelligent Help Systems

The tasks handled by process operators usually require both knowledge and experience, and the production efficiency and safety depends to a large degree on how well the operators perform. Therefore, the following points are of great importance:

- The operator should be given information on as high a level as possible. This will make it easier for him to use it in the problem solving tasks.

- The operator should be provided with as much help as he may wish for, not only about the available commands of the computer system, which is usual today, but also with knowledge about the *process itself* and about the *operating sequences.*
- The operator should be able to look at the process from several different *views,* and to have help in interpreting all these presentations.
- The operator should be helped with the operating sequences of the process, i.e., the help should take into consideration the action sequence history, and be able to reason and give information about past and future.
- The help should be *non-invasive,* so that the operator can use it if he wants, but can ignore it if he finds that appropriate.

The ascent of expert systems has provided a new technology which is very well suited for handling knowledge not only about the computerized control system's commands, but also about the process, the operating sequences, the possible past and future, and about causes and effects of different actions. Thus, they form a good basis for constructing new and better help systems.

Intelligent Front-Ends

A standard rule-based expert system is not very well adapted to be used as a *help system.* It usually lends itself to some kind of question and answer dialog, and thus it fails in the demand of non-invasiveness. Also, it is usually rather cumbersome to handle planning and plan recognition in this way.

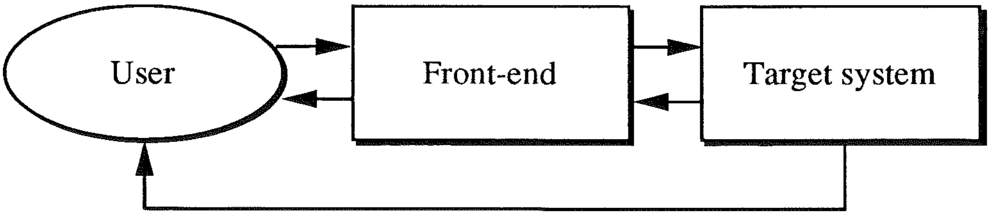


Figure 7.3 The command spy idea. By putting an expert system between the user and the target system, as an intelligent front-end, it is possible to make the help system non-invasive.

In order to overcome these drawbacks, the concept of an intelligent front-end has been created. The idea is simple; put the help system as an interface between the user and the target system, and let the intelligent front-end analyze the communication and act according to the information obtained, see Figure 7.3.

This setup enables the system to monitor the communication between the user and the target system. It may then use the command history to understand what has happened and why the operator has done certain actions, *plan recognition.* This means that the system will know the goals of the operator's actions and the action or command sequences necessary to reach these goals. This, in turn, will enable the system to give dynamical help about past and future, and about the necessary operating sequences. By making the intelligent front-end quiet until asked for help, the system becomes non-invasive, the so called *command spy* idea.

A typical example of a system for plan recognition is the intelligent help system for process identification, `(ihs)`, see Larsson and Persson (1987, 1988 a, b, 1991; the latter to be found in this volume). This system uses an intelligent front-end architecture to implement a command spy for Idpac, an interactive program that performs numerical calculations for fitting mathematical models to pairs of measured input and output signals.

7.2 The Toolbox Rule Bases

The algorithms described in the previous chapters have been implemented in G2. The MFM graph structure is built with G2 graphical objects and connections, thus giving both a graphical presentation and an underlying data structure on which the G2 rules and procedures can operate. The construction of the flow models is simple and user friendly, as it uses G2's possibilities of graphical editing, creation, and cloning of objects, etc.

There are several groups of rules and procedures, each implementing one or a part of an algorithm. Thus, the main contribution in the implementation is to be found in these rule groups. It should be noted that G2's rules and procedures can be mixed and work well together. Thus, a rule set may really be a set of rules *and* procedures. The different rule sets will now be described.

Syntax Control

There are many rules of syntax that say how the MFM objects may or may not be connected. These have been stated in Chapter 3. The G2 connections cannot be mixed, and therefore mixing of flow types is prohibited even from the initial design of the graphical structure. Within a flow path, however, flow functions may only be connected according to certain rules; for example, a source may only be connected to a transport. A small rule set is used to check this part of the syntax. These rules must be invoked by the user, however, and without doing so, it is possible to violate the MFM syntax. It should also be noted that the rules do not check the more unclear syntax rules that state that a node may not be filled or emptied only, see Section 3.5 of Chapter 3.

Measurement Validation

The measurement validation algorithm is implemented with four rule sets. One handles the updating of group information, i.e., it keeps track of consistent and inconsistent subgroups. Another rule set controls the flow propagation. The flow values of the flow functions with no measurement connected to them must be guessed, and known flow values are propagated to them. Multiply supported flow values have precedence over singularly supported ones, and downstream propagation has precedence over upstream propagation.

A third rule set handles the recognition of singular inconsistent subgroups, which should be specially marked. If the single group is surrounded by another, consistent group, its flow value is also changed. The fourth and last group handles dynamic color coding of the information. Thus, the different inconsistent subgroups are shaded in different hues of gray, and the singular failures are marked with red.

Alarm Analysis

The alarm analysis rule set handles the recognition of the different alarm situations and sets the failure states accordingly. These rules only look at the flow functions two and two, or in some cases three and three. Thus, they work locally and efficiently.

Consequence Propagation

The consequence propagation rules handle the guessing of alarm states of flow functions which are not connected to physical alarm sensors. Similar causation rules to those in the alarm analysis are used. This rule set works locally and mixes with the alarm analysis rules.

Fault Diagnosis

The fault diagnosis algorithm performs a downward search in the MFM graphs and uses a dialog of questions and answers. The dialog part has had the consequence that most of the implementation is done with procedures. Thus, rules handle the search and decisions about which subtrees that need further investigation, while procedures handle the dialog and setting of fault values. The implemented search is not strictly local, (the diagnosis may skip parts of the graph altogether), but as the trees are graphically represented, the geographical coordinate values are used to make the search move in a left to right direction over the screen, and no additional reasoning over global paths is needed.

7.3 The Implementation Tool G2

The data structures and algorithms have all been implemented in G2. Some extra insight into the problems of developing AI software may be gained if some details of this implementation are described. First, a short description of G2 itself will be given. The following description is based on Nilsson (1991).

The expert system tool G2 has been developed by Gensym Corporation, and is probably the most advanced real-time expert system tool currently available. It is implemented in Common Lisp, which is automatically translated to C, and it runs on many different computers. A Sun Sparcstation 2 was used in this project. G2 consists of several main parts:

- A knowledge database
- A real-time inference engine
- A procedure language interpreter
- A simulator
- A development environment

- An operator interface
- Optional interfaces to external on-line data servers

Classes and Objects

G2 is an object-oriented programming environment. All G2 components, including rules, procedures, graphs, buttons, objects, etc., are *items,* and organized into a class hierarchy with *single inheritance.* All items have a graphical representation through which they may be manipulated by mouse and menu operations. Operations exist for moving an item, cloning it, changing its size and color, etc. The user defined items are called *objects* in G2.

Objects are used to represent the different concepts needed in a specific application. The object definition defines the attributes specific to the class and the look of the icon. Attributes of many types are supported, such as constants, variables, parameters, lists, arrays, and G2 objects. The constants, variables, and parameters may be quantitative, (integer or real), symbolical, logical, or text strings.

Objects are either static, i.e., they are explicitly created by the developer, or dynamic, i.e., created and deleted dynamically during runtime. G2 contains operations for moving and rotating an object, and changing its color. With these facilities, simple animations can be created. Each G2 object may have an associated *subworkspace.* Here arbitrary items may be positioned, and thus the internal structure of an object may be represented on its subworkspace.

Relations Between Objects

G2 has different ways of defining relations between objects. One way is to have lists containing other objects as attributes. Another way is to use *connections,* which have a graphical representation and may have attributes. Connections can be used in G2 expressions for reasoning about interconnected objects in a variety of ways.

A third way of relating objects is to use *relations.* These may only be created at runtime and have no graphical representation. They have no corresponding relation hierarchy and cannot have attributes. Relations are used in G2 expressions in the same way as connections.

The Inference Engine

G2 rules can be 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
- Unconditionally rules
- Whenever rules

If rules my be invoked by forward and backward chaining, by scanning at a specified time interval, and by explicit invoking. *When* rules are a variant of *If* rules that may not be invoked through forward chaining or cause backward chaining. *Initially* rules are executed at initialization time only. *Unconditionally* rules are equivalent to *If* rules with a true premiss. *Whenever* rules trigger when 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, i.e., they acts as *demons.*

The rules contain references to objects and their attributes in a natural language style syntax. Objects may be referenced through connections with other objects, thus utilizing the connection structure of the objects instead of explicit names. G2 supports generic rules that apply to all instances of a class.

In addition to forward and backward chaining, rules may be invoked explicitly in several ways. A rule may be scanned at even time intervals. A *focus* statement invokes all rules associated with a certain focal class or object. An *invoke* statement triggers all rules belonging to a specified rule category.

Internally the G2 inference engine is based on an agenda of actions to be performed by the system. 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.

Procedures

G2 contains a Pascal-style procedural programming language. The procedures are started by rule actions, they are reentrant and each invocation executes as a separate task, and they may have input parameters and return one or several values.

The set of procedure statements include all rule actions, assignment, branching, (*if-then-else* and *case*), iteration, (*repeat* and *for*), *exit if* to exit loops, the infamous *go to, call* to call a procedure and wait for its result, and *start* to call a procedure without waiting. The *for* loops may be either numeric or generic for a class, i.e., they execute a statement or set of statements once for each instance of the class.

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. It allows for differential, difference, and algebraic equations on explicit form. These may be specific to a certain variable of apply to all instances of a variable class. Each first-order differential equation is integrated individually with an individual, user defined stepsize. The numeric integration algorithms available are a simple Forward Euler algorithm and a fourth order Runge-Kutta, both with fixed stepsize.

7.4 Using G2 to Implement the MFM Toolbox

MFM models have a strong graphical nature and consist of objects connected together and collected in different networks. Diagnostic algorithms using MFM will use information about these objects and how they are interconnected. Thus, G2 is ideally suited for implementing the basic MFM data structures, as well as the diagnostic algorithms. The MFM concepts have the corresponding implementations shown in Table 7.1.

The animation facilities of G2 have been used to present results of the algorithm. For example, the primary and secondary failure states from the alarm analysis are shown in red and bright red, and a quick glance will give the operator a good idea of the total failure state of the process.

Goals and flow functions	*G2 objects*
Relations and links	*G2 connections*
The "inside" relation	*Subworkspaces*
Diagnostic methods	*Rules and procedures*

Table 7.1 MFM concepts and the corresponding G2 concepts used to implement them. The "inside" relation refers to the case when a path of flow functions resides in a network or manager function.

The time efficiency has not posed any problem during the project. As all the algorithms themselves are linear in effort, the main obstacle to overcome when scaling up the size of the knowledge database is the internal G2 representation. However, the G2 inference engine uses static links, and no decrease in efficiency has been observed when tackling somewhat larger processes as Steritherm.

All in all, it can be concluded that G2 is a very good programming tool, provided it is used to solve a fitting problem, and MFM is one such problem.

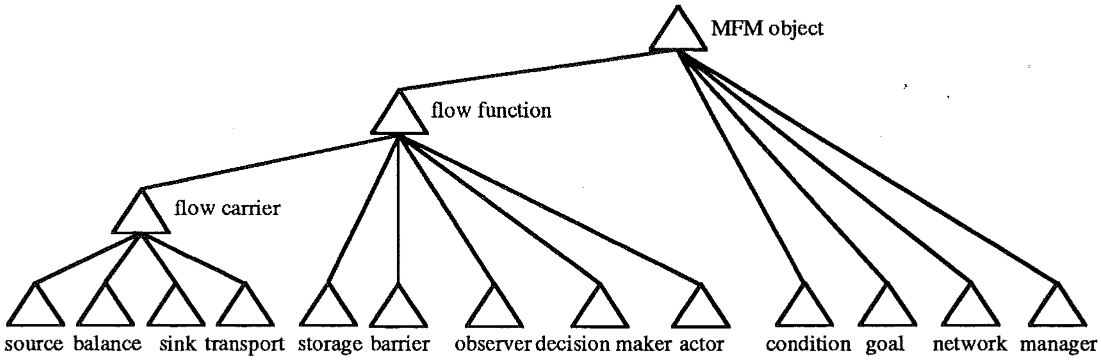


Figure 7.4 The MFM class hierarchy, which is a part of the G2 object hierarchy. Thus, the superior class of `MFM object` is the G2 `object`.

The Data Structure and Class Hierarchy of Objects

The MFM objects lend themselves to be put in a class hierarchy with single inheritance, see Lind (1990 b). This is easily done in G2, and the resulting classes are shown in Figure 7.4. Actually, it would be quite natural to use *multiple inheritance* to express that each flow function is either of type mass, energy, or information, but this is not supported in G2.

Rules

The different algorithms are easily implemented with G2 rules. As an example, consider once again, connected source and transport, see Figure 7.5.

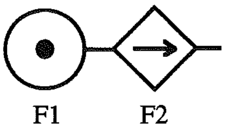


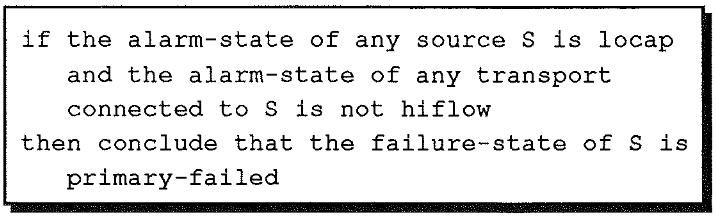
Figure 7.5 A connected source and transport. In the alarm analysis algorithm, this connection is treated with three rules.

In Chapter 5 the following rules to find out the failure state of the source were presented:

- A transport *hiflow* alarm may cause a connected source to have a *locap* alarm.
- An alarm in a network will force a function depending on this network to fail.

Taking the normal situation into consideration also, three rules for the failure state of a source may be formulated.

1. If a source has a *locap* alarm and any connected transport does not have a *hiflow* alarm, then the alarm of the source is primary.
2. If a source has a *locap* alarm and any connected transport has a *hiflow* alarm and there is no alarm in the network that the source may depend on, then the alarm of the source may be secondary.
3. If a source is not alarmed, its failure state is working.



```
if the alarm-state of any source S is locap
   and the alarm-state of any transport
   connected to S is not hiflow
then conclude that the failure-state of S is
   primary-failed
```

Figure 7.6 A rule from the alarm analysis rule set, as it looks in the G2 implementation. It handles the case when there is a primary failure in the source, and it is quite similar to the text version of the same rule, presented earlier in this chapter.

EXAMPLE 7.1
These rules are surprisingly easily implemented in G2. The possibility to reason about objects and their connections make the G2 rules look very much like the textual rules above. For example, the G2 rule corresponding to the first rule is shown in Figure 7.6. □

EXAMPLE 7.2
The G2 version of the second rule is shown in Figure 7.7. Note that the *locap* alarm is secondary only if there is no fault in the supporting network, i.e., a fact from the second rule. This test must be included in the G2 rule, as otherwise two rules could be in conflict and trigger each other infinitely. □

```
if the alarm-state of any source S is locap
   and the alarm-state of any transport
   connected to S is hiflow and not (there
   exists a condition C connected to S
   such that (the alarm-state of C is
   alarmed))
then conclude that the failure-state of S is
   secondary-failed
```

Figure 7.7 Another rule from the alarm analysis rule set. It corresponds to the second rule above, and treats the case when the source has a primary *or* secondary failure.

EXAMPLE 7.3
If the source should be in a normal state, (i.e., not alarmed), the failure state must be set to working. This is handled by a more general rule, valid for all flow functions, see Figure 7.8. □

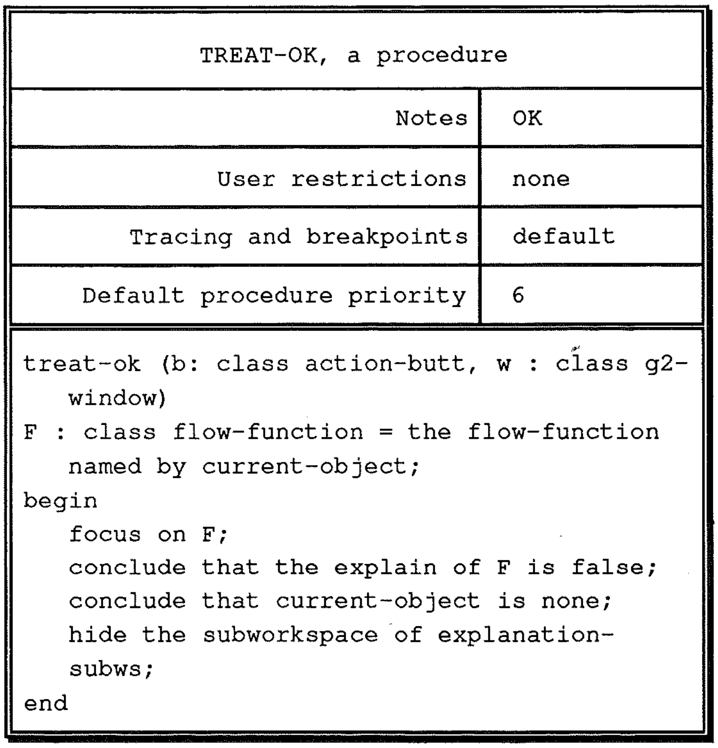
```
if the alarm-state of any mfm-object M is
   normal
then conclude that the failure-state of M is
   working
```

Figure 7.8 A third rule from the alarm analysis rule set. This rule is valid for any MFM object and shows the use of the "any" construction allowed in G2 rules. This enables the writing of rules that apply to whole classes of objects. The rule is used to set all MFM objects which are not alarmed to the working state.

Procedures

Parts of the algorithms are more easily expressed with procedures than rules. This is the case especially in the fault diagnosis.

EXAMPLE 7.4
A short procedure from the fault diagnosis algorithm is shown in Figure 7.9. This particular procedure treats the case when the user has been given an explanation or remedy and clicks the **OK** button. □



TREAT-OK, a procedure	
Notes	OK
User restrictions	none
Tracing and breakpoints	default
Default procedure priority	6

```
treat-ok (b: class action-butt, w : class g2-
   window)
F : class flow-function = the flow-function
   named by current-object;
begin
   focus on F;
   conclude that the explain of F is false;
   conclude that current-object is none;
   hide the subworkspace of explanation-
   subws;
end
```

Figure 7.9 A procedure from the fault diagnosis algorithm. This specific procedure is called when an explanation or remedy has been shown and the user clicks the **OK** button. Then the **explain** attribute is set to false, so that the explanation will not be activated again, the search variable **current object** is set to **none** so that the search may go on to look for more explanations, and the subworkspace upon which the explanation was given is hidden.

Programming Effort

It is somewhat difficult to estimate the time or effort used for "coding" the toolbox, as this work has been intertwined with algorithm development and testing. With this in mind, it can be stated that the implementation time is about three months, spread out over a period of about a year. The G2 database is not very large; around 350 Kilobytes.

7.5 The Simulation Model

In order to develop and test the MFM toolbox, two target processes were used, the tanks and Steritherm. G2 simulation models were built for these two processes. This includes a class hierarchy of physical component definitions, generic simulation equations, and some rule bases for transferring values between the simulations and the MFM models. The Steritherm simulation model was constructed in a master's project, see Christiansson and Ericsson (1989).

The construction of the physical component and simulation model is a standard G2 task and will not be further described here. Some of the resulting representations, (such as physical topology views), can be seen in Chapter 8.

7.6 Conclusions

It is necessary to implement and test algorithms in order to investigate their value and to see further possibilities of development. Usually, the implementation of diagnostic algorithms for control systems is a major effort in itself. However, as MFM is very well suited for implementation in G2, the implementation has been a minor part of the current project. All the algorithms have been tested and work well, and the test implementation is easily turned into a G2 toolbox for MFM modeling and diagnosis.

CHAPTER 8

GRAPHICAL PRESENTATION

So far, MFM has only been used as a representation language for building knowledge databases for computerized algorithms. However, it is also important to *show* means-end information to users. This chapter presents some different ways of using MFM and other representations for presentation.

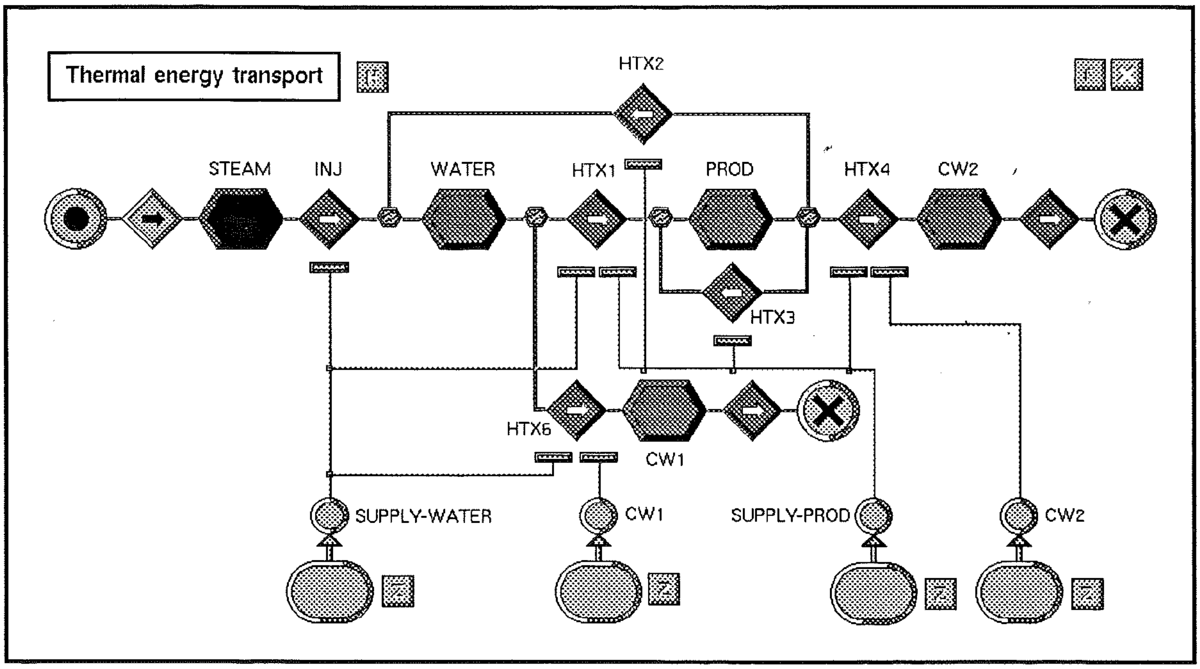


Figure 8.1 An alarm analysis situation in the main energy flow of Steritherm. Primary faults are shown in a darker and secondary faults in a lighter shading. This picture gives a compact overview of the fault situation, and it would be useful to present it to an operator. There are several faults, but a single primary fault in the steam system is able to explain the whole situation.

8.1 Introduction

Once the importance of means-end models has been understood, it is also obvious that the information they contain should be presented to operators, in order to help them understand the processes more thoroughly, see Figure 8.1. It is not clear, however, how means-end information should be presented, as it is a difficult task. In order for a user to understand MFM, the ideas behind means-end relations *must* be grasped, and there are no easy ways around this. Thus there is a basic difficulty in making any means-end information understood. Several alternative ways of presentation are possible:

- Use the concepts of flow sheets and other presentations, which are already well-known to the operators. In this way, user acceptance and easy introduction will benefit. However, the danger is that the means-end information is not seen as such, but mistaken in different ways.
- Use MFM itself. Means-end information is inherently different from topological information, and must be presented with special means. MFM is, so far, the only graphical language available, and it is possible for operators to learn to use it. This standpoint have been proposed by Lind and by Duncan and Prætorius, see Duncan *et al* (1989). Critics, for example Rasmussen, have proposed that user acceptance may be low, though.
- Use computer graphics to develop new ways of presentation, which are more well-suited to show means-end information. This may include the choice of a *metaphor*, i.e., a complete framework of interpretation. The Apple Macintosh metaphor of viewing the screen as a writing desk with documents on it is probably the most successful example of this. However, no such metaphor of means-end models have been presented so far.

8.2 Presentation of MFM

The suggestion of this project is that it is indeed possible to present means-end information by using MFM graphs, provided they are integrated in a full presentation system in which several other representations of the process are also available; a multiple view system. This implies that together with the MFM graphs there should be available topological, geographical, and behavioral descriptions, see Section 1.4 of Chapter 1, and that the user should be able to navigate through the models and get help with interpreting and connecting the different parts of them to each other. A system like this has been described in Larsson (1990 a), Årzén (1989, 1990, 1993), and Årzén *et al* (1990). This system contains several models of Steritherm, including an MFM model built with the toolbox presented in Chapter 7. Basically, such a system should fulfill the following points:

- The different kinds of models should be integrated together into one environment, making it easy for the user to use any type of representation he wishes.
- The presentation interface must be designed so that it is clearly visible what kind of view the user is currently looking at. For example, it is vitally important that the user knows whether the presented pictures contains geographical, topological, or means-end information.
- Navigation tools must be available. For example, it should be possible to select an object or set of objects in one view and have the corresponding objects in other views highlighted. In the same way, it should be possible to move from one object or environment in one view to the corresponding objects or environment in other views, and the possible paths must be clearly and simply presented.
- Knowledge-based help should be available in all environments and concerning all changes of views.

With these aids, a multiple view system should be able to provide support for using MFM for showing means-end information. The system described in Larsson (1990 a), Årzén (1989, 1990, 1993), and Årzén *et al* (1990) gives a good demonstration of the possibilities, once the whole system has been designed for a full integration of several model types.

8.3 Different Abstraction Hierarchies

An integrated multiple view system will contain a large amount of data. This data should be structured, in order to make it practically possible for the users to navigate through it, and find the information important for specific tasks. This implies that several different abstraction hierarchies should be used:

- Part-whole decomposition. This is the simple aggregation of smaller components into large building blocks, and the possibility of zooming in on a composite object and see its inner parts.
- Specialization. This is the standard class-subclass structure, with inheritance and specialization of subclasses.

- Functional abstraction. This is the decomposition of a model into means-end hierarchies described in Chapters 1 and 3.
- Multiple views. This is not a proper hierarchical decomposition as the three others, but it still deserves to be treated in a similar way. There is a definite advantage in designing a system so that all orientation and moving through the models is done with the same type of techniques. Only then will a view change be as natural as moving up and down in the other abstraction hierarchies.

This general structure, based on several information hierarchies and multiple views, implies the following navigation operations:

- `Up` and `down` in the part-whole hierarchies. For example, if the current presentation is centered on a certain machine, the `down` operation means zooming in on the machine and moving to a presentation of the inner parts of the machine in question, while going `up` form inside the machine is the reverse operation.
- `Go to class` and `go to instances.` This means that the user should be able to move to the definitions of any object and from any definition to the corresponding instances.
- `Up` and `down` in the means-end hierarchies. This means moving along the condition and achieve relations of MFM, from goals vias networks and flow functions to subgoals, and vice versa.
- `Other view` is the operation of staying in more or less the place in the part-whole and class hierarchies, but changing the model type, for example from topological to means-end representation. This operation is not always possible, as the different models may have parts not found in other models. Where there is a possible connection, there should also be a possibility to change view, a so called *bridge,* see for example Mariño *et al* (1990).
- In addition, other obvious operations should be provided, such as quick commands for moving to the top levels, to the previous screen, etc.

8.4 The Implemented System

The MFM toolbox described in Chapter 7 has been used in the implementation of the system described in Larsson (1990 a), Årzén (1989, 1990, 1993), and Årzén *et al* (1990), and to implement a demonstration system for two processes, the tanks process used in the previous chapters, and Steritherm. The latter system includes several examples of how to use graphics and different other ideas to present means-end information:

- The MFM graphical language has been changed from the definitions given by Lind, so that color is now used to separate between mass, energy, and information flows. The new graphical language is also more clearly readable, (on screen), using icons with depth effects and shadowing, etc.
- Colors are used to show the results of the diagnostic algorithms. Thus, primary and secondary alarms are shown with deeper and lighter shades of red respectively, and the object currently under fault diagnosis investigation is highlighted in grey. In the pictures shown in this work, the colors have been changed to gray scales instead. The front page illustration shows how the pictures look on a color screen.
- The mass flows of MFM may be shown in the corresponding topological presentations, i.e., the pipes and equipment where the mass flow is flowing may be highlighted, see Figure 8.9.
- The energy flows can be shown with arrows in the topological view, describing how energy is transported through the system, see Figure 8.8.
- The energy networks may also be shown with some of the details of MFM, such as conditions and subgoals, taken away, and with more natural icons. An example of this has been developed for the top level energy network of Steritherm, see Figure 8.7. The same solution could of course be used for mass flows, but as these are more easily shown in a topological presentation, it has not been done here.
- The goal-subgoal hierarchy may be broken out of MFM and shown separately, see Figure 8.5.

8.5 Some Examples of Graphical Presentation

All the ideas described in the last section above have also been implemented in a demonstration system built on the toolbox described in Chapter 7. The following shows how the different models and results of algorithms actually look on the computer screen. Hopefully, the examples serve both as a documentation of the diagnostic algorithms and as a collection of ideas of how to present means-end information.

EXAMPLE 8.1
Figure 8.2 is a simple, geographical presentation of the tanks process.

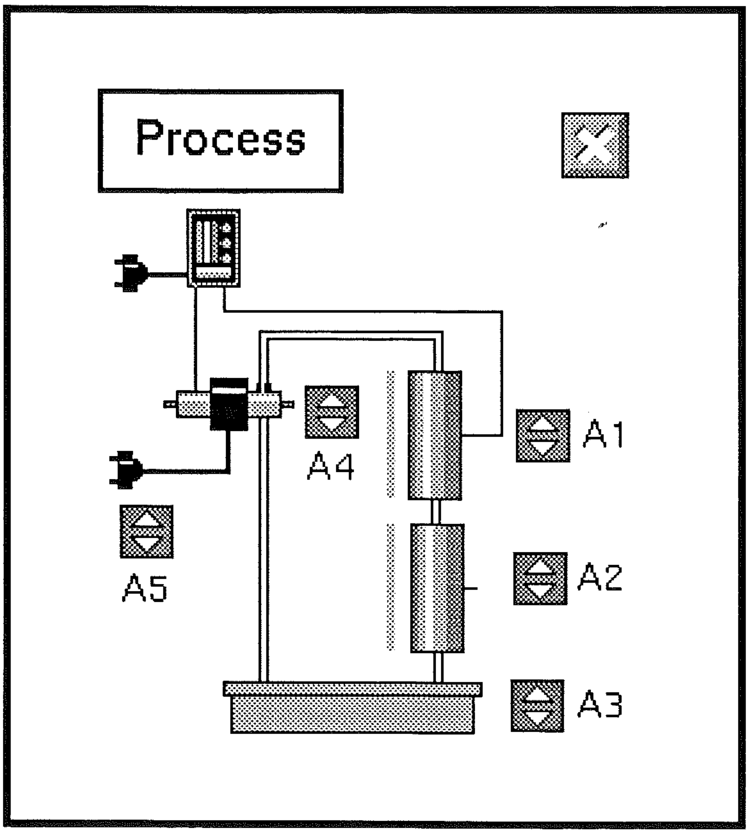


Figure 8.2 A flow sheet of the tanks process. The storage tank, the two cylindrical tanks, the pump and the controller are shown, together with alarms used by the alarm analysis algorithm. The icons look quite similar to the real process; thus, this is a good example of a geographical view. Some animation is also used, e.g., in the level pipes immediately to the left of the smaller tanks. Also, the pipes changes their color depending on whether they are empty or contain water.

The picture gives a good example of the graphical possibilities of G2. Thus, the icons look quite similar to the real process. Some animation is also used in this picture. The level pipes beside the cylindrical tanks use color to indicate the level of the tanks; the pipes change color when they contain water, and the alarms light up when they are activated.

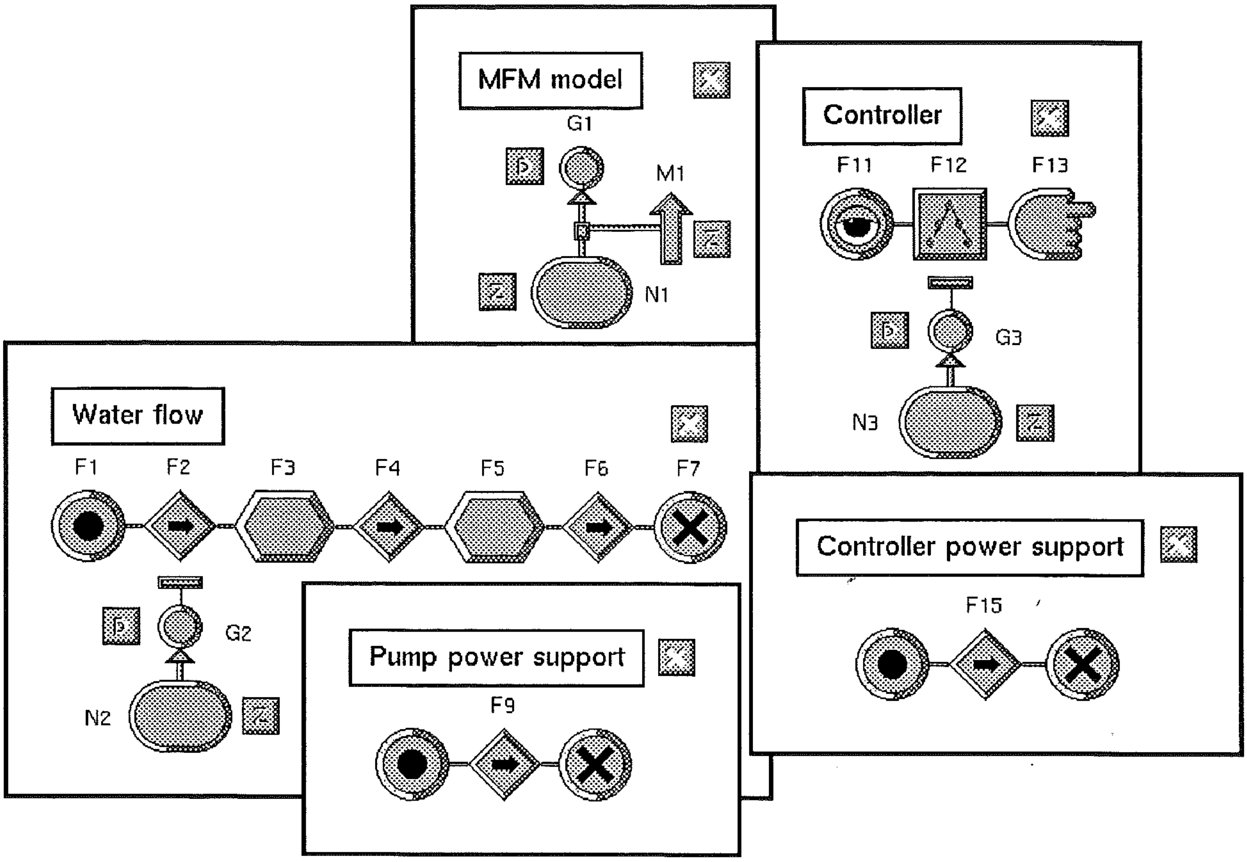


Figure 8.3 An MFM model of the tanks process. The top level goal is shown, as well as the water flow, pump energy, control information, and control power supply networks. The networks and manager are shown as icons, while their contents appear on new workspaces.

The MFM model of the tanks process has been shown several times before in this work. The toolbox version is found in Figure 8.3. Note that the contents of the networks and manager appear on new workspaces, the subworkspaces of the corresponding objects. □

EXAMPLE 8.2

Now we move to the Steritherm process. A topological presentation of the Steritherm process is shown in Figure 8.4. It is the standard flow sheet, but with more elaborated icons, as it appears in the G2 implementation.

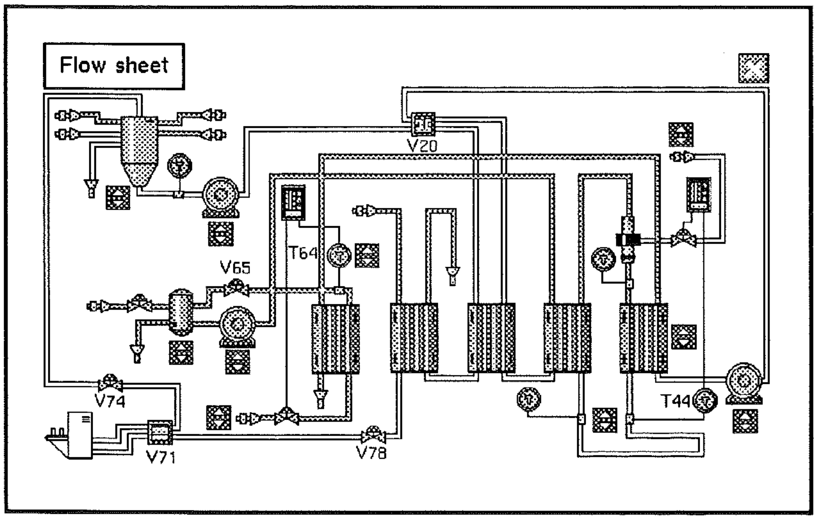


Figure 8.4 The Steritherm flow sheet, as it appears in the MFM toolbox implementation. An older version, more like the technical drawings, may be found as Figure 1.5 in Chapter 1.

The five heat exchangers, (numbered from right to left), can be seen in the lower right of the diagram, with the product balance, tank in the upper left and the packing machine in the lower left. The product flow is white while the water flows are shown in a darker shade. □

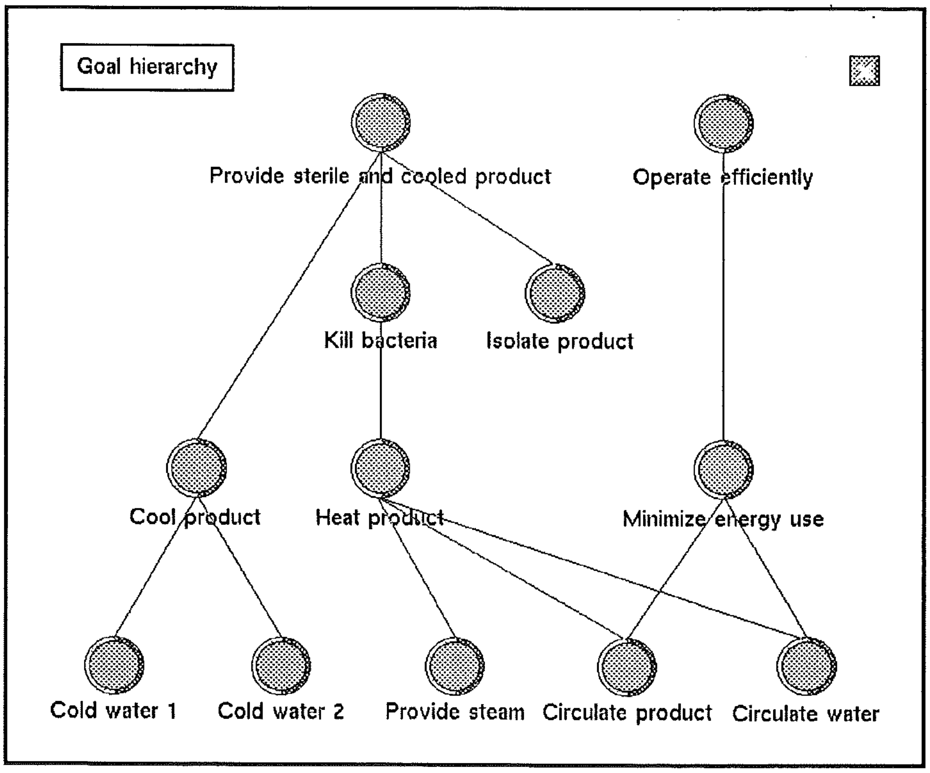


Figure 8.5 The Steritherm goal hierarchy. Here, both production and economy goals are shown, while in the following examples, only the hierarchy under the production goal is used. The goal hierarchy can be found by breaking out a part of the full MFM model.

EXAMPLE 8.3

The Steritherm goal hierarchy is found in Figure 8.5. This is a picture of the goal-subgoal hierarchy of the MFM model, broken out from the rest of the MFM representation. In this figure, all the different goals of Steritherm, down to a certain level of detail, are found. □

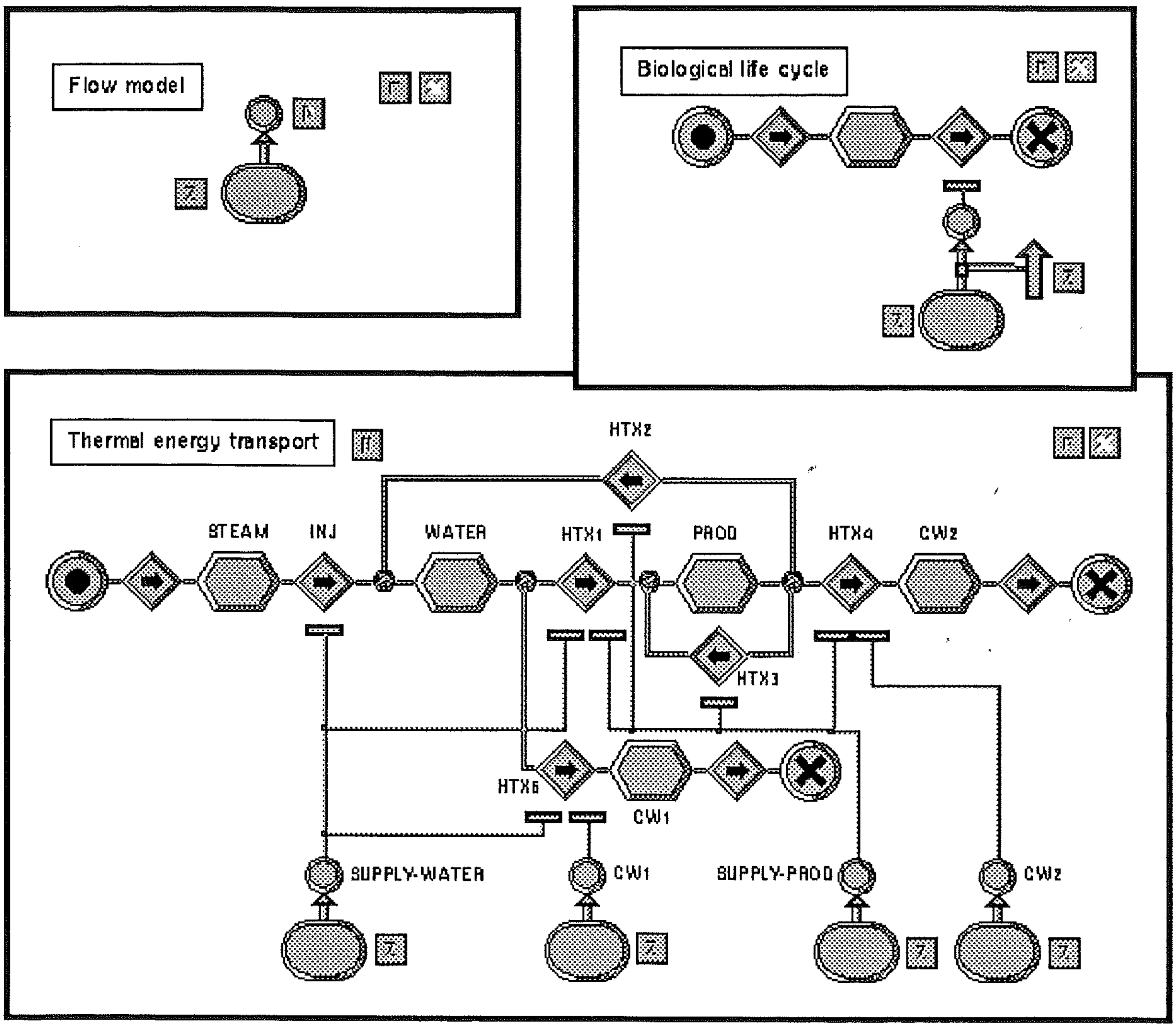


Figure 8.6 The top levels of the Steritherm MFM model. The top level production goal resides on one workspace, the bacterial life cycle network on the next, and the main thermal energy network on the third.

EXAMPLE 8.4

In Figure 8.6 the three top levels of an MFM model of Steritherm can be found. Here only the most important operational goal, that of sterilizing the product, has been put into the model. The topmost network describes the bacterial life cycle, and the next level is the main energy transportation network. □

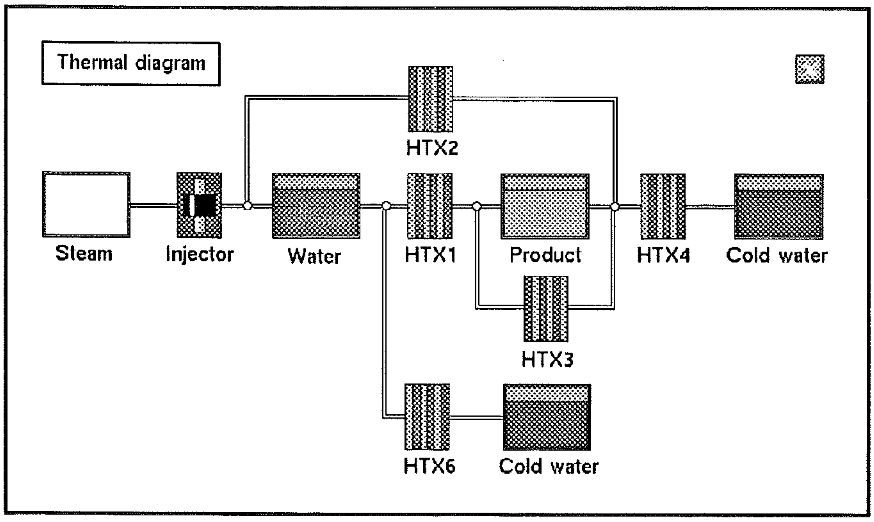


Figure 8.7 The energy transportation in Steritherm. The MFM icons have been changed to symbols of the physical equipment, and the condition and achieve relations have been omitted together with the subgoals. Thus, an easily interpreted diagram has been produced.

Example 8.5

An equivalent representation of the energy network is shown in Figure 8.7. The idea of this "thermal flow sheet" is to exemplify how MFM can form a basis for pictures that might indeed be successfully shown to a process operator. Here some detail has been taken away, and the MFM symbols have been exchanged for icons that show the corresponding physical object. The result is an easily read and understood diagram that shows the energy transport and reuse in the process. □

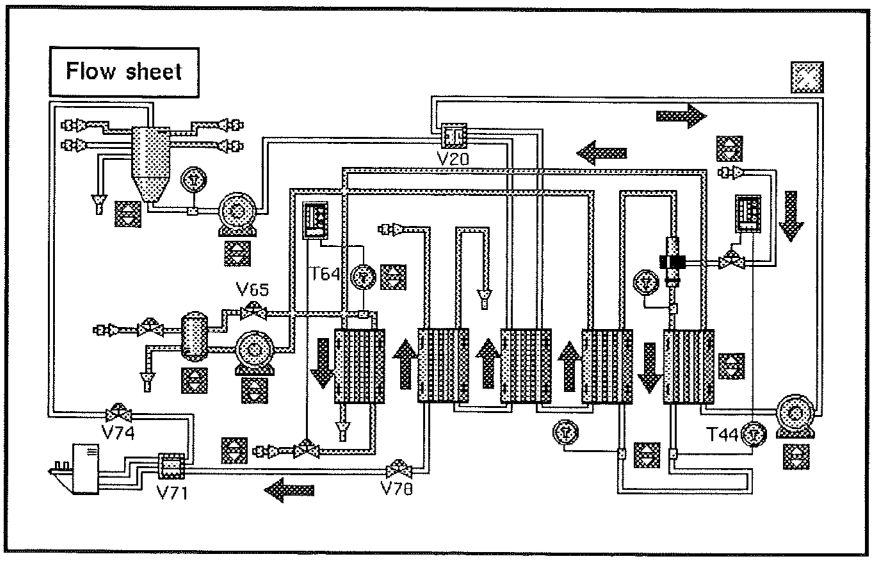


Figure 8.8 The flow sheet of Steritherm with the energy flow marked. This is an alternative way of showing the energy flow through the process.

Example 8.6

The energy flow through Steritherm can also be shown directly in the flow sheet, see Figure 8.8, where the energy transportation is marked with arrows along pipes and heat exchangers. In the implemented system, both this and the presentations of Figures 8.6 and 8.7 are available, side by side, so that the user can switch between them and have as much help as possible in understanding the energy flow. □

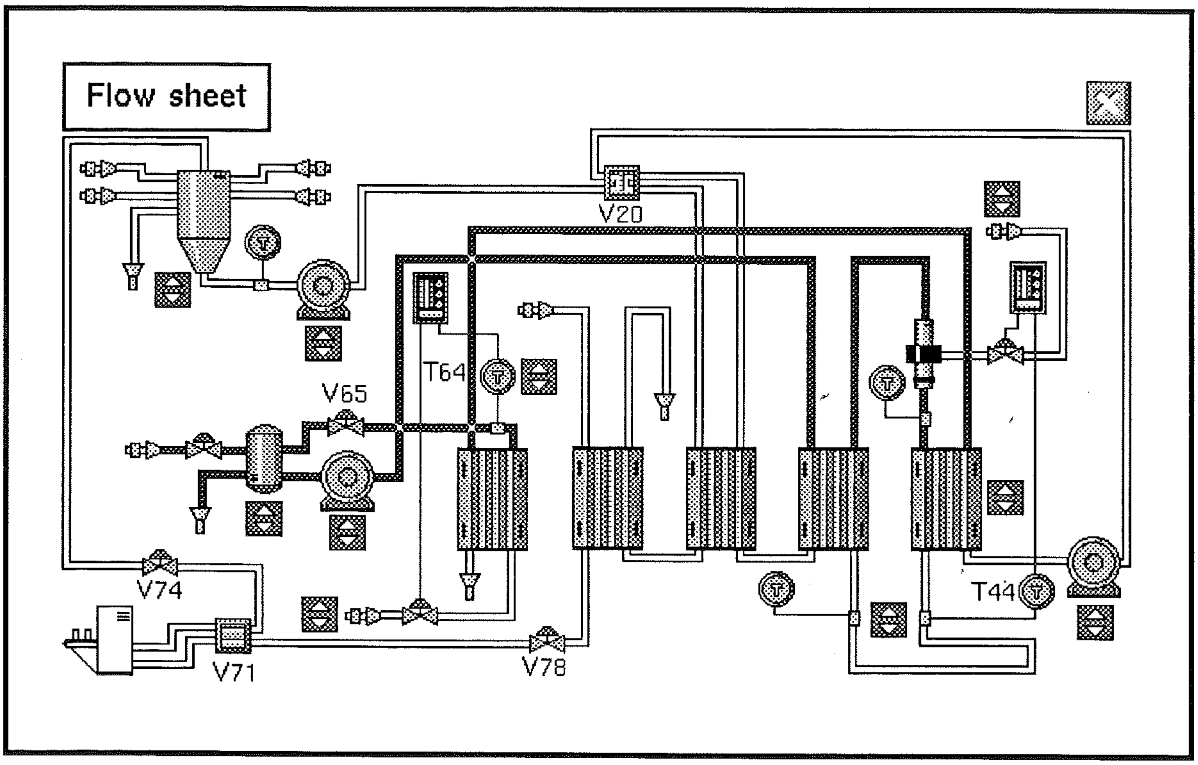


Figure 8.9 The flow sheet of Steritherm with the water flow marked. Showing mass flows is usually a simple question of highlighting in a topological or geographical presentation.

Example 8.7

Flows of mass can also be shown in flow sheets. In Figure 8.9 the primary water flow of Steritherm has been highlighted. It is, of course, easier to show a mass flow, as it will always be confined in pipes, etc. □

Example 8.8

There are two control loops in Steritherm. One measures the product temperature in the holding tube and controls the steam inlet valve to the steam injector; the other measures the return water temperature and controls the cooling water valve to heat exchanger number 5. The MFM model of these two control loops is shown in Figure 8.10.

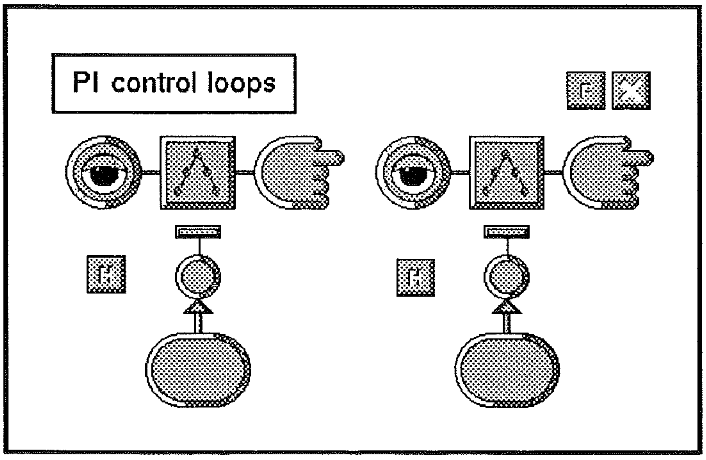
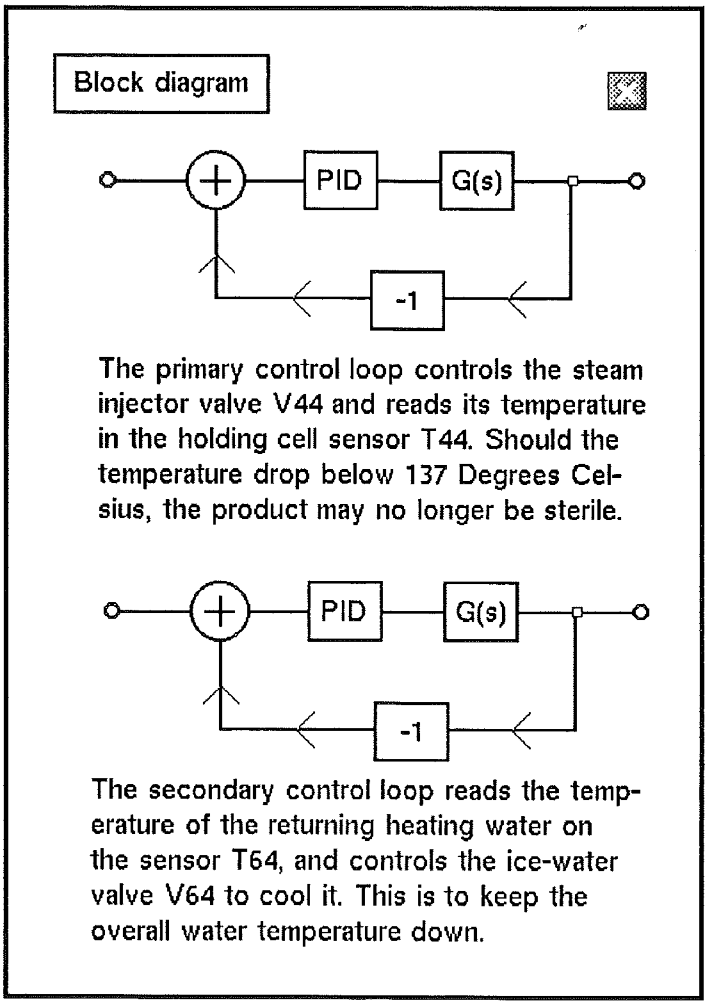


Figure 8.10 An MFM model of the control loops in Steritherm. The two temperature loops, regulated by PI controllers, are modeled as information flows, with observers, decision functions, and actors.



The primary control loop controls the steam injector valve V44 and reads its temperature in the holding cell sensor T44. Should the temperature drop below 137 Degrees Celsius, the product may no longer be sterile.

The secondary control loop reads the temperature of the returning heating water on the sensor T64, and controls the ice-water valve V64 to cool it. This is to keep the overall water temperature down.

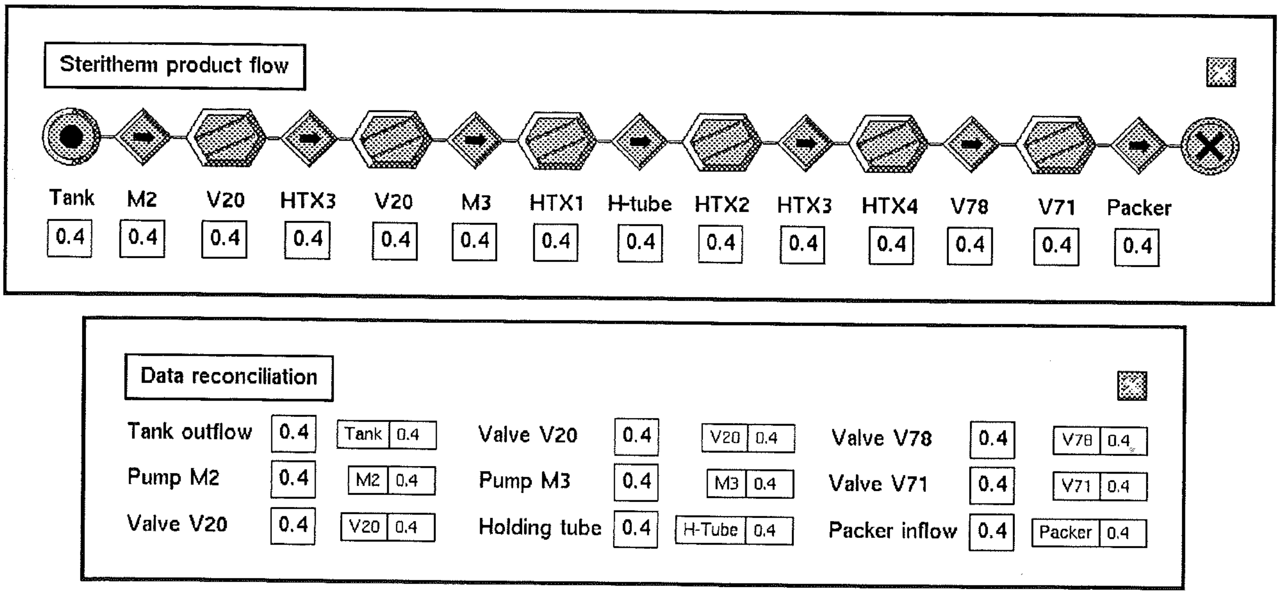
Figure 8.11 A block diagram of the control loops in Steritherm. This is an alternative diagram for presenting information about the control loops, more easily accepted by engineers.

Figure 8.11 shows an alternative way of presenting control loop information. Here a simple “control theory” style block diagram is used. The idea here is to give a more familiar way of presenting the same system as in Figure 8.10. □

8.6 Diagnostic Algorithms

The three methods described earlier has been implemented and tested. In earlier chapters they were demonstrated using the tanks process. Here follows some further examples, this time using Steritherm.

Example 8.9
A detailed MFM model of the product flow of Steritherm is shown in Figure 8.12, together with a panel of flow values, to be used for measurement validation of the product flow. Note that the example uses several measurements that are usually not present in a real Steritherm process. In the first snapshot, all flow measurements are equal and form one single, consistent group. Some flow values, e.g., those of the heat exchangers, have been guessed with the flow propagation algorithm.



Steritherm product flow

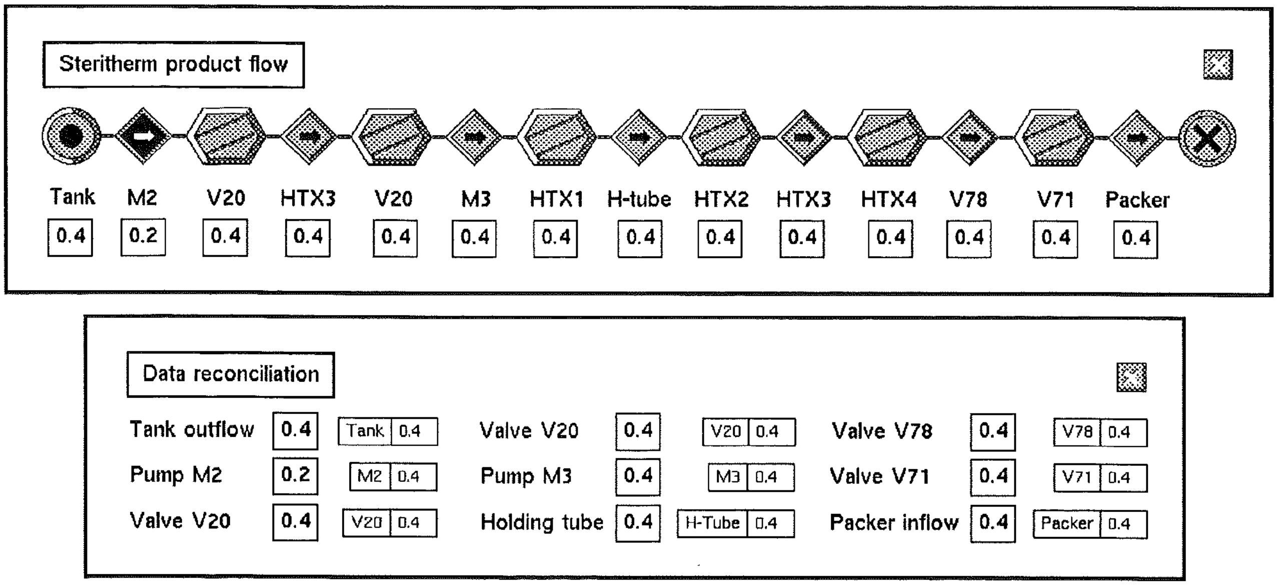
Tank	M2	V20	HTX3	V20	M3	HTX1	H-tube	HTX2	HTX3	HTX4	V78	V71	Packer
0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4

Data reconciliation

Tank outflow	0.4	Tank	0.4	Valve V20	0.4	V20	0.4	Valve V78	0.4	V78	0.4
Pump M2	0.4	M2	0.4	Pump M3	0.4	M3	0.4	Valve V71	0.4	V71	0.4
Valve V20	0.4	V20	0.4	Holding tube	0.4	H-Tube	0.4	Packer inflow	0.4	Packer	0.4

Figure 8.12 A flow situation in the Steritherm product flow, together with a control panel, for use with the measurement validation method. This is the normal situation and all flow values agree.

In Figure 8.13 we see the case of a single, surrounded subgroup, the flow through the pump **M2**. It has been highlighted in red and its validated flow value set to that of the surrounding group.



Steritherm product flow

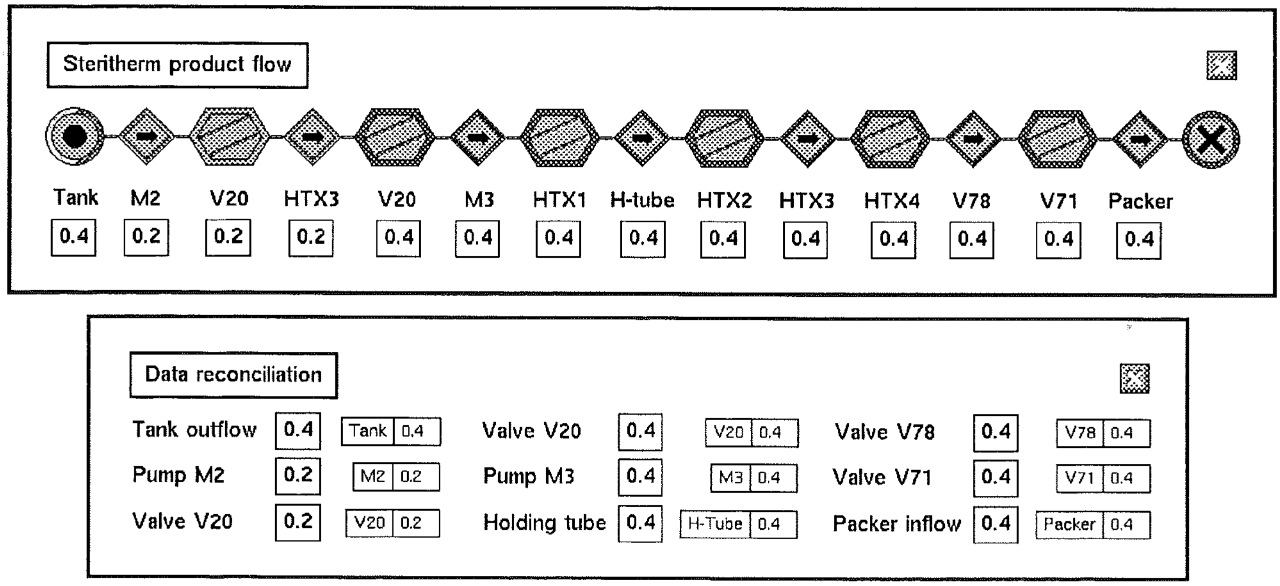
Tank	M2	V20	HTX3	V20	M3	HTX1	H-tube	HTX2	HTX3	HTX4	V78	V71	Packer
0.4	0.2	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4

Data reconciliation

Tank outflow	0.4	Tank	0.4	Valve V20	0.4	V20	0.4	Valve V78	0.4	V78	0.4
Pump M2	0.2	M2	0.4	Pump M3	0.4	M3	0.4	Valve V71	0.4	V71	0.4
Valve V20	0.4	V20	0.4	Holding tube	0.4	H-Tube	0.4	Packer inflow	0.4	Packer	0.4

Figure 8.13 A second flow situation in the Steritherm product flow. Here is an example of a single, surrounded subgroup, the transport **M2**.

In Figure 8.14 the pump **M2** and the valve **V20** deviates from the rest of the measurements, to form a consistent subgroup of their own. The flow value of the heat exchanger **HTX3** is propagated downstream, and these three flow functions forms one subgroup, while the rest forms the other. In this case the system only notices the discrepancy, but makes no guess as to what may be the most likely fault hypothesis. □



Steritherm product flow

Tank	M2	V20	HTX3	V20	M3	HTX1	H-tube	HTX2	HTX3	HTX4	V78	V71	Packer
0.4	0.2	0.2	0.2	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4

Data reconciliation

Tank outflow	0.4	Tank	0.4	Valve V20	0.4	V20	0.4	Valve V78	0.4	V78	0.4
Pump M2	0.2	M2	0.2	Pump M3	0.4	M3	0.4	Valve V71	0.4	V71	0.4
Valve V20	0.2	V20	0.2	Holding tube	0.4	H-Tube	0.4	Packer inflow	0.4	Packer	0.4

Figure 8.14 A third flow situation in the Steritherm product flow. Here both **M2** and **V20** deviates, to form a consistent subgroup. In this complicated situation, subgroup information only is presented.

EXAMPLE 8.10

The alarm analysis algorithm is easily demonstrated in the Steritherm energy network. The normal situation is shown in Figure 8.15.

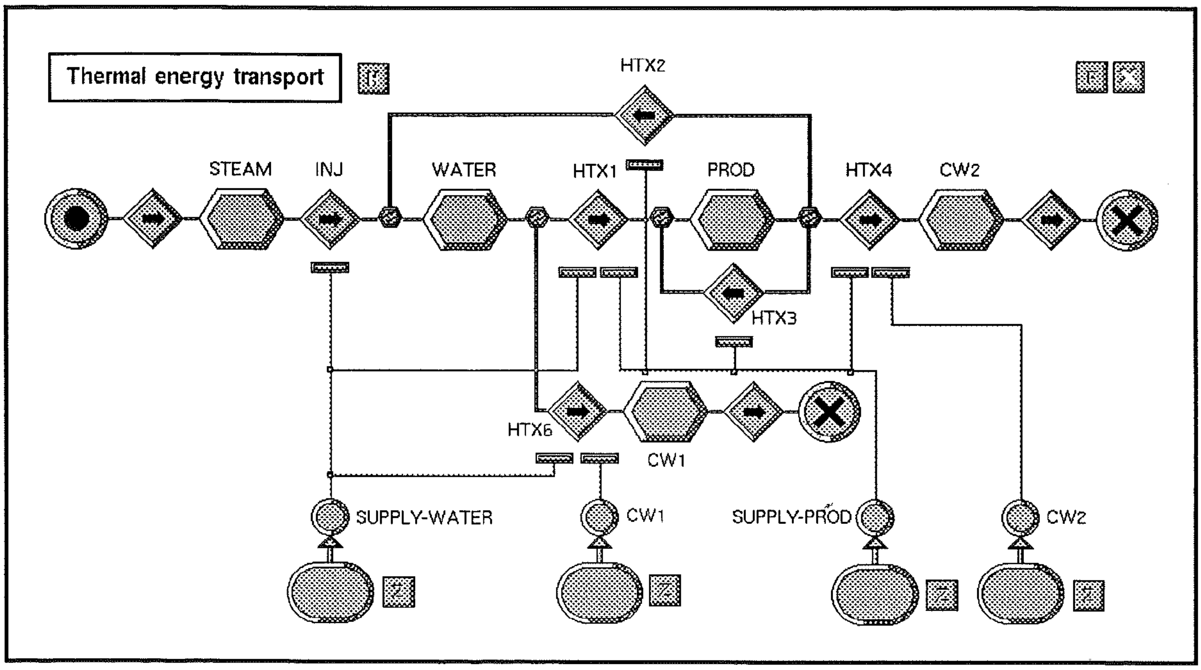


Figure 8.15 The main energy network of Steritherm. In this figure, the alarm situation is normal, i.e., there are no alarms.

An alarm situation is shown in Figure 8.16. The active alarms are:

- A low temperature in the holding tube
- A low temperature out from **HTX1**
- A low temperature in the primary water flow, measured by **T64**
- An indication that the steam system is out of order

The steam system alarm is marked as positively primary, while the others may be secondary effects. It should be noted that all these alarms are not normally available in a standard Steritherm process, but they have been added in the modeling to enable a more complicated example. Without them the methods still work, but most of the diagnostic resolution is lost.

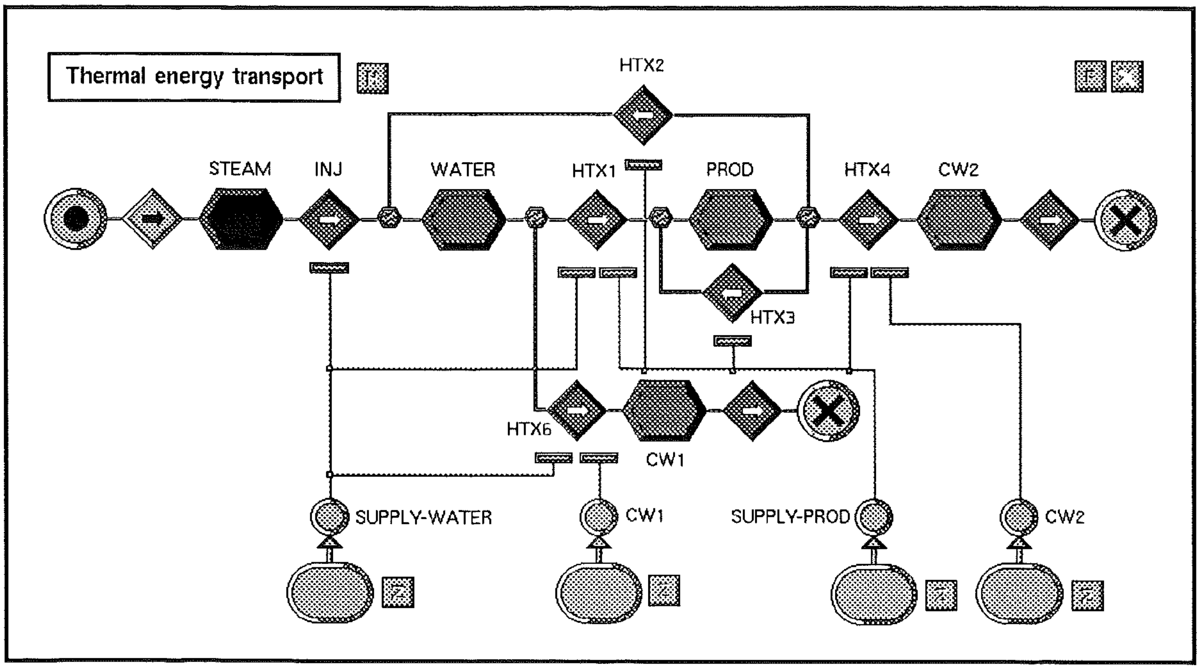
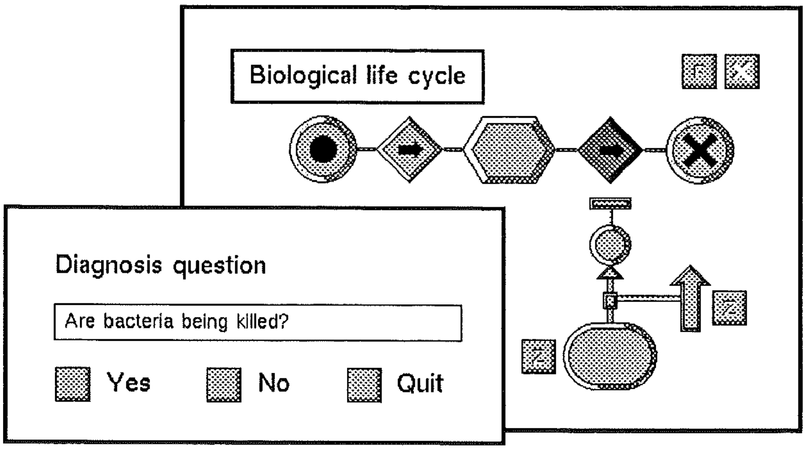


Figure 8.16 An alarm situation shown in the Steritherm energy network. Four alarms are present: low temperature in the holding tube, a low temperature out from **HTX1**, a low temperature in the primary water flow, and a steam system fault. The alarm analysis method concludes that the steam system fault may explain all the other alarms.

In Figure 8.16 it can be seen how the steam system alarm may be the cause of all the other malfunctions. □



Diagnosis question

Are bacteria being killed?

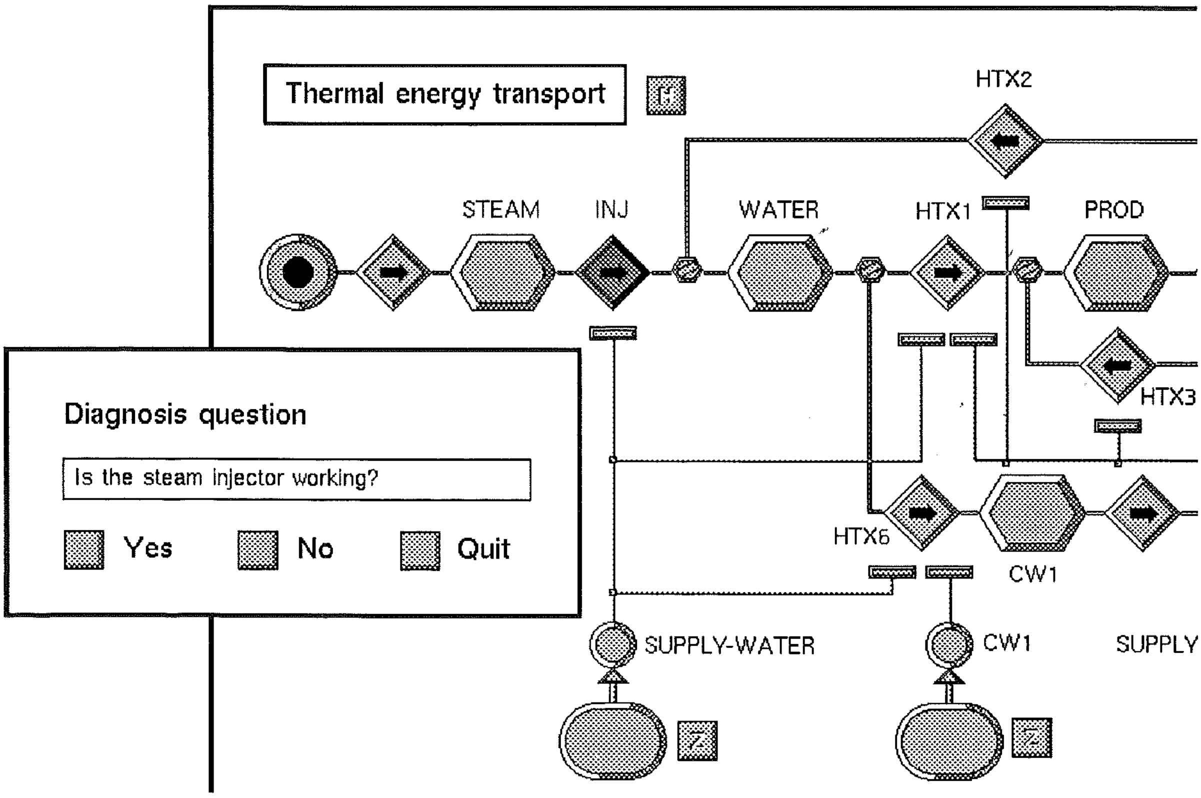
Figure 8.17 A snapshot of a fault diagnosis on Steritherm. The top level goal is under investigation, and the search has reached the first flow function, which is the transport corresponding to the killing of bacteria in the bacterial life cycle network.

Example 8.11

At last we have the fault diagnosis applied to Steritherm. In Figure 8.17 the bacterial life cycle network of Steritherm is shown. The product has not been properly sterilized, and a fault diagnosis is started from the topmost goal. The first question is:

> Q: **`Are bacteria being killed?`**

and the answer is `no`. The search now moves down into the thermal energy network.



Diagnosis question

Is the steam injector working?

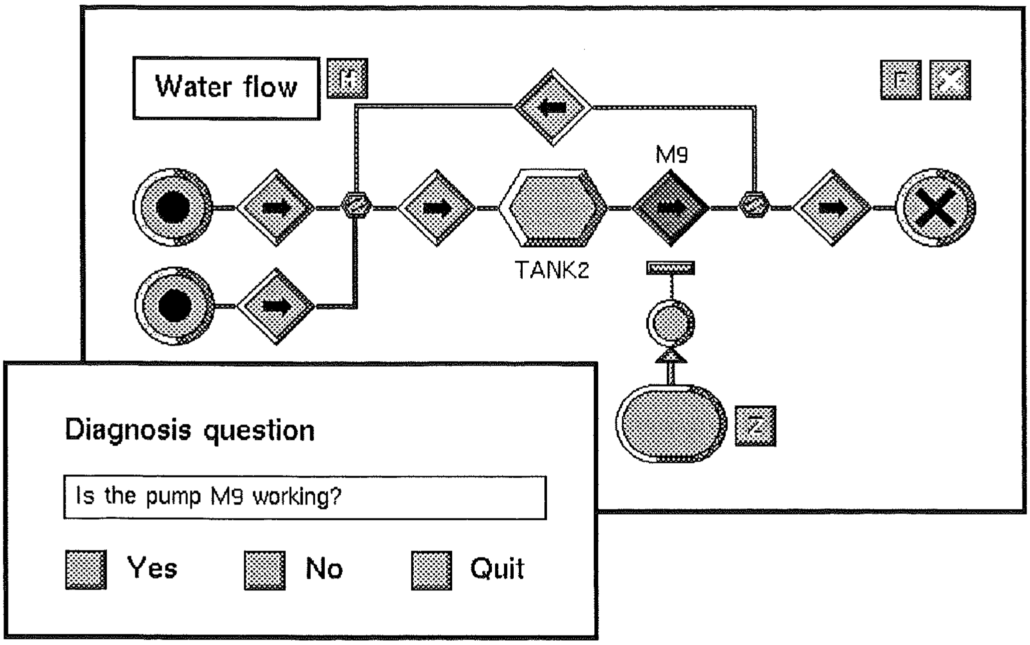
□ Yes □ No □ Quit

Figure 8.18 A second snapshot of the fault diagnosis on Steritherm. The search has moved down into the thermal energy network and found the transport corresponding to the steam injector.

In Figure 8.18 the diagnosis search has reached the steam injector and asks:

> Q: **`Is the steam injector working?`**

The answer is `no`, which means that the search must continue down via the condition relation to the water mass flow.



Diagnosis question

Is the pump M9 working?

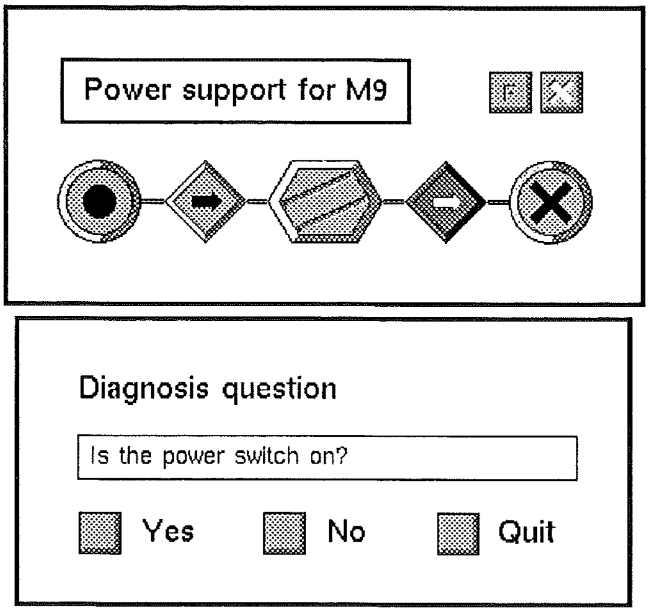
□ Yes □ No □ Quit

Figure 8.19 A third snapshot of the fault diagnosis on Steritherm. The search has moved down from the steam injector into the primary water flow and reached the water pump **M9**.

In Figure 8.19 the search has reached the pump **M9** and asks:

> Q: **Is the pump M9 working?**

Here the answer is **no,** and the search continues down into the energy support network of the pump.



Diagnosis question

Is the power switch on?

□ Yes □ No □ Quit

Figure 8.20 A fourth snapshot of the fault diagnosis on Steritherm. The search has reached the transport corresponding to the power switch of the pump **M9**.

The search now reaches the power switch, see Figure 8.20, and asks:

Q: `Is the power switch on?`

Here the answer is `no`. The current MFM model has no further levels, and the diagnostic search stops this branch.

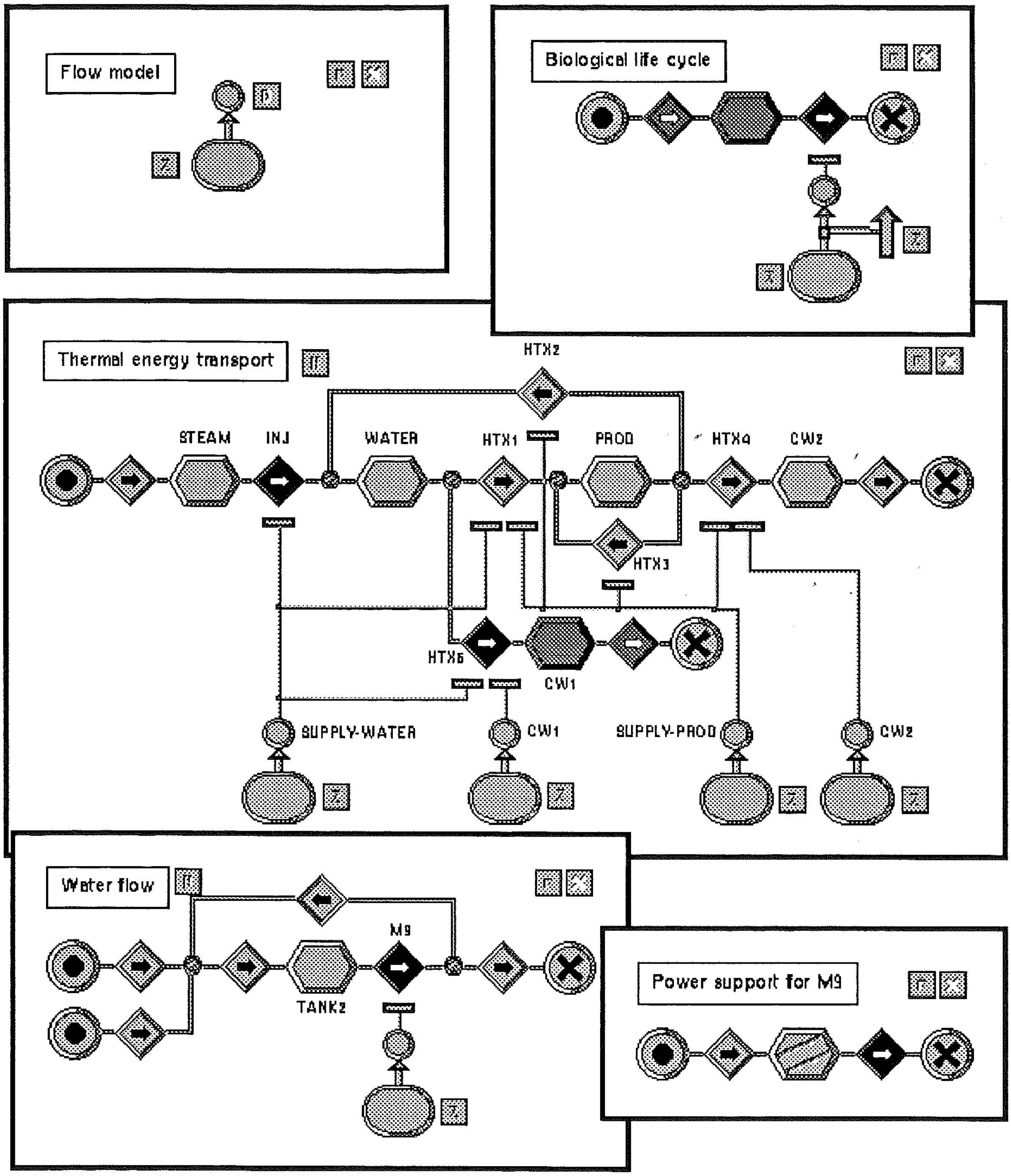
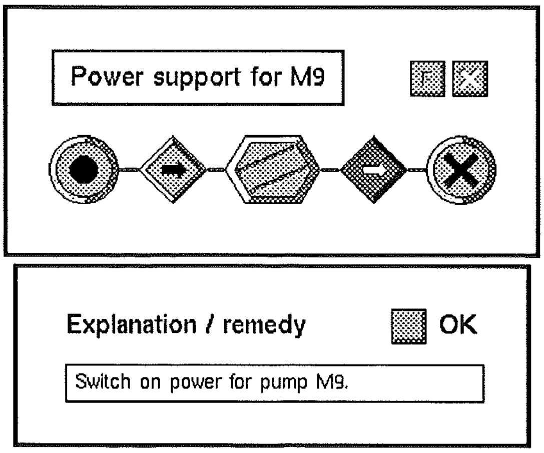


Figure 8.21 A fifth snapshot of the fault diagnosis on Steritherm. Here, all the levels of the MFM model is shown, and the fault situation is clearly and compactly described by the shading of the faulty flow functions.

When the diagnostic search is over, the whole fault situation has been investigated, and the result is shown in Figure 8.21.



Power support for M9

Explanation / remedy

Switch on power for pump M9.

Figure 8.22 A sixth snapshot of the fault diagnosis on Steritherm. The remedy to be found after the diagnosis is to switch on the power for the pump **`M9`**.

After the search, the user may ask for remedies and explanations. In this case the output is simple, see Figure 8.22:

R: **`Switch on power for pump M9.`**

This concludes the examples of measurement validation, alarm analysis, and fault diagnosis on Steritherm. □

8.7 Conclusions

Means-end information is both important to use and difficult to understand, while the MFM language may be hard to interpret for an unexperienced user. In spite of the latter, MFM may form a basis for presenting information about goals and functions, *if* the graphs themselves are enhanced by different other pictures, such as goal hierarchies, thermal flow sheets, etc., and *if* the MFM models are integrated into a multiple view system. Examples of how this could look has been given in this chapter. In this case, the user should be able to learn the MFM graphical language gradually, and understand both what information it conveys, and what the MFM models can be used for.

CHAPTER 9

CONCLUSIONS

This work describes model-based diagnosis using *multilevel flow models,* MFM. The first chapter is an introduction to the idea of means-end models, and to the concept of *multiple views.* The second chapter gives an overview of related work and Chapter 3 contains an introduction to the MFM language, together with some new ideas. Then three new diagnostic methods are described, followed by a chapter about the implemented toolbox, and a last chapter concerning presentation of means-end information for the user.

9.1 Diagnostic Methods

The main contributions of this work are the invention, implementation, and testing of three new diagnostic methods. The methods use MFM as a database and performs measurement validation, alarm analysis, and fault diagnosis. They have been implemented in G2, and tested on two target processes. One of these, Steritherm, is a small industry process with about 30 sensors and more than 100 major components. The implementation is of moderate size, see Table 9.1, and forms a toolbox for MFM models. Note that the rules are *generic* for MFM, and thus the same rules can be used for all processes.

Measurement validation	*71 rules*
Alarm analysis	*64 rules*
Fault diagnosis	*19 rules*

Table 9.1 The three diagnostic methods have been implemented with small G2 knowledge databases. The rules are generic for all processes.

The algorithms are local and incremental. They work in real-time, and propagate information along static links. This makes them quite efficient, and the execution effort increases at worst linearly with the size of the models. A C implementation of the fault diagnosis takes 200 microseconds to search through an MFM model of Steritherm. The other methods could be equally efficiently implemented.

The implementation has given a valuable test on how the methods work in practise. It also works as a test experience in implementing a knowledge database and in using G2. The conclusion is that G2 is excellently suited for the task. There is probably no other tool that would come close to the same ease and efficiency.

9.2 Secondary Contributions

In addition to the primary conclusions, several smaller contributions can also be found in the work.

One goal has been to evaluate MFM. It is quite clear that means-end models are important and useful, both in diagnostic tasks in general and for presentation to operators. Thus they are a valuable complement to other types of models. MFM works reasonably well in modeling of many processes, but enhancements are needed. The work suggests several additions and changes to the syntax and semantics. These include the modeling of biochemical processes and of control systems. The main conclusion here is that the new diagnostic algorithms show the strength of the MFM framework. However, it is also possible to generalize the three methods to use other model frameworks than MFM.

The project also shows the strength of and the need for multiple view representation and presentation. It is not true that one type of model will suffice for all the tasks demanded of supervision and control systems. Thus several model types must be allowed to live side by side.

9.3 Further Developments

The results of the work certainly need further corroboration. The implemented rule databases need testing and development, both concerning theory and specific details of implementation.

A major issue in MFM is to include more types of functions, and to describe processes not entirely based on flows. In that case the three diagnostic methods should also be further developed to handle the new functions.

The ideas of the three diagnostic methods could also be extended to use models other than MFM. For example, a new graphical language with more differentiated objects could be developed. The graphical language in question should probably be closer in look to existing topological and geographical descriptions.

Several issues within MFM modeling need further attention. The modeling of control systems, and the inclusion of non flow-based processes are two obvious cases. Many minor details should also be further investigated, e.g., concerning the connection syntax. It is also important to develop a practise for modeling with MFM.

MFM may be used for several other methods within diagnostic reasoning. One example is the automatic generation of simulation equations from the flow networks and another is planning. It would also be possible to generate data for different methods in qualitative reasoning, etc.

The implementation leaves several questions unanswered. For an implementation like the one done in this project, G2 has proved excellent. However, to make the methods available in realistic control systems, it is necessary to look for other, more ordinary ways of implementation, for example in a standard programming language like C.

Finally, the diagnostic methods proposed should be further compared with other methods, both concerning similar, competing properties, and concerning how the different methods could be combined, to provide better diagnostic resolution.

In spite of all that remains to be done, however, it is the hope of the author that this thesis will bring knowledge-based diagnosis in general and MFM in particular forward, and thus form another small step among all the steps taken by science in the long but steady march towards the future.

CHAPTER 10

REFERENCES

The purposes of this reference collection are threefold: to give a background to the methods developed in this work, to give a selected overview of the model-based diagnosis and means-end model research area, and to give a complete list of references for the project behind the third part of the thesis.

ALLEN, D. J. and M. S. M. RAO (1980): "New Algorithms for the Synthesis and Analysis of Fault Trees," *Ind. Eng. Chem. Fundam.*, **19**, 1, 79–85.

ALMASY, G. A. and T. SZTANO (1975): *Problems of Control and Information Theory*, **4**, 1, 57–69.

ANTSAKLIS, P. J., K. M. PASSINO, and S. J. WANG (1991): "An Introduction to Autonomous Control Systems," *IEEE Control Systems*, **11**, 4, 5–13.

ASEA BROWN BOVERI, SATTCONTROL, TELELOGIC, and DEPARTMENT OF AUTOMATIC CONTROL, LUND INSTITUTE OF TECHNOLOGY (1988): *Knowledge-Based Real-Time Control Systems—IT4 Feasibility Study*, Studentlitteratur, Lund.

ASEA BROWN BOVERI, SATTCONTROL, and DEPARTMENT OF AUTOMATIC CONTROL, LUND INSTITUTE OF TECHNOLOGY (1990): *Knowledge-Based Real-Time Control Systems—IT4 Project: Phase 1*, Internal report, Lund.

ASEA BROWN BOVERI and DEPARTMENT OF AUTOMATIC CONTROL, LUND INSTITUTE OF TECHNOLOGY (1991): *Knowledge-Based Real-Time Control Systems—IT4 Project: Phase 2*, Internal report, Lund.

ASHBY, W. R. (1956): *Introduction to Cybernetics*, Chapman & Hall, London.

BENVENISTE, A. (1991): "Meetings Held with Thomson-CSF, GEC-Alsthom, and Siemens-AG," *IEEE Control Systems Magazine*, June 1991, 86–92.

BODDY, M. and T. DEAN (1989): "Solving Time Dependent Planning Problems," *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, pp. 979–984.

BROWNSTON, L., R. FARRELL, E. KANT, and N. MARTIN (1985): *Programming Expert Systems in OPS5: An Introduction to Rule-Based Programming*, Addison-Wesley, Reading, Massachusetts.

CHÉRUY, A., R. MONTELLANO, and M. P. BERNIER (1989): "Computer-Aided Design in Modeling of Biotechnical Processes," in Breedveld, P. *et al* (Eds.): *Modeling and Simulation of Systems*, J. C. Baltzer AG, Scientific Publishing Co., pp. 235–237.

CHRISTIANSSON, M. and P. ERICSSON (1989): *Knowledge-Based Control and Modeling with G2*, Master's thesis, TFRT–5411, Department of Automatic Control, Lund Institute of Technology, Lund.

CHUNG, D. T. and M. MODARRES (1989): "GOTRES: An Expert System for Fault Detection and Analysis," *Reliability Engineering and System Safety*, **24**, 113–137.

COGSYS (1990): *COGSYS Manual*, COGSYS Ltd., Salford, United Kingdom.

COHEN, G. (1991): "Meetings with Électricité de France," *IEEE Control Systems Magazine*, June 1991, 92–94.

CRESPO, A., J. L. NAVARRO, R. VIVÓ, A. GARCÍA, and A. ESPINOSA (1992): "RIGAS: an Expert Server Task in Real-Time Environments," *Proceedings of the 1992 IFAC/IFIP/IMACS International Symposium on Artificial Intelligence in Real-Time Control*, Delft, Nederland, pp. 631–636.

CRESPO, A., J. L. NAVARRO, R. VIVÓ, A. ESPINOSA, and A. GARCÍA (1991): "A Real-Time Expert System for Process Control," *Proceedings of the 3rd IFAC International Workshop on Artificial Intelligence in Real-Time Control*, Rhonert Park, Sonoma, California.

CREUTZFELDT, J. (1990): *Sensorvalidering, alarmbehandling, og fejldiagnose i større processanlæg, (Sensor Validation, Alarm Analysis, and Fault Diagnosis in Large Processes)*, Ph. D. thesis, preliminary status report, Institute of Automatic Control Systems, Technical University of Denmark, Lyngby, Denmark, in Danish.

DAVID, R. and H. ALLA (1992): *Petri Nets and Grafcet: Tools for Modeling*

Discrete Event Systems, Prentice Hall, New York.

Davis, R. and W. Hamscher (1988): "Model-Based Reasoning: Troubleshooting," in H. Shrobe, (Ed.): *Exploring Artificial Intelligence,* Morgan Kaufmann Publishers, Inc., San Mateo, California, pp. 297–346.

Dean, T. and M. Boddy (1988): "An Analysis of Time Dependent Planning," *Proceedings of the AAAI '88,* pp. 49–54.

de Kleer, J. (1986 a): "An Assumption-Based Truth Maintenance System," *Artificial Intelligence,* **28**, 2.

de Kleer, J. (1986 b): "Problem Solving with the ATMS," *Artificial Intelligence,* **28**, 2.

de Kleer, J. and J. S. Brown (1984): "A Qualitative Physics Based on Confluences," *Artificial Intelligence,* **24**, 1–3, 7–83.

De Maré, J. (1980): "Optimal Prediction of Catastrophes With Applications to Gaußian Processes," *The Annals of Probability,* **8**, 4, 841–850.

Dennett, D. C. (1987): *The Intentional Stance,* Bradford Books, Cambridge, Massachusetts.

Duncan, K. D. and N. Prætorius (1989): "Flow Displays Representing Complex Plant for Diagnosis and Process Control," *Proceedings of the 2nd European Meeting on Cognitive Science Approaches to Process Control,* Siena, Italy.

Dvorak, D. L. (1992): *Monitoring and Diagnosis of Continuous Dynamic Systems Using Semiquantitative Simulation,* Doctor's Dissertation, AI 92–170, Artificial Intelligence Laboratory, University of Texas at Austin, Austin, Texas.

Dvorak, D. L. and B. Kuipers (1991): "Process Monitoring and Diagnosis," *IEEE Expert,* June 1991, 67–74.

Farza, M. and A. Chéruy (1991): "CAMBIO: A Software for Modeling and Simulation of Bioprocesses," *Computer Applications in the Biosciences,* **7**, 3, 327–336.

Fikes, R. E. and N. J. Nilsson (1971): "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving," *Artificial Intelligence,* **2**, 189–208.

Finch, F. E. (1989): *Automated Fault Diagnosis of Chemical Process*

Plants Using Model-Based Reasoning, Doctor's thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts.

FORBUS, K. D. (1984): "A Qualitative Process Theory," *Artificial Intelligence,* **24**, 85–168.

FORBUS, K. D. (1988): "Qualitative Physics: Part, Present, and Future," in H. Shrobe, (Ed.): *Exploring Artificial Intelligence,* Morgan Kaufmann Publishers, Inc., San Mateo, California, pp. 239–296.

FRANK, P. M. (1990): "Fault Diagnosis in Dynamic Systems Using Analytical and Knowledge-Based Redundancy—A Survey and Some New Results," *Automatica,* **26**, 3, 459–474.

FRANK, P. M. (1991): "Enhancement of Robustness in Observer-Based Fault Detection," *Proceedings of the IFAC/IMACS Symposium SAFEPROCESS '91,* Baden-Baden, pp. 275–287.

FRANK, P. M. (1992): "Robust Model-Based Fault Detection in Dynamic Systems," *Proceedings of the IFAC Symposium on On-Line Fault Detection and Supervision in the Chemical Process Industries,* University of Delaware, Newark, Delaware.

GREPA (1985): *Le Grafcet — de Nouveaux Concepts,* Groupe Equipment de Production Automatisée réuni à l'ADEPA, Cepadues-éditions, 111, rue Nicolas-Vauquelin, 31100 TOULOUSE, France.

HAMSCHER, W., L. CONSOLE, and J. DE KLEER, (Eds.) (1992): *Readings in Model-Based Diagnosis,* Morgan-Kaufmann Publishers, Inc., San Mateo, California.

HANSSON, A. and L. NIELSEN (1991): "Control and Supervision in Sensor-Based Robotics," *Proceedings of Robotikdagar 1991,* Linköping University, Linköping, Sweden.

HARMON, P. and D. KING (1985): *Expert Systems, Artificial Intelligence in Business,* John Wiley & Sons, New York.

HAYES-ROTH, B. (1985): "A Blackboard Architecture of Control," *Artificial Intelligence,* **26**, 251–321.

HAYES-ROTH, B. (1990): "Architectural Foundations for Real-Time Performance in Intelligent Agents," *Real Time Systems,* **2**, 1, 2.

HAYES-ROTH, B., R. WASHINGTON, R. HEWETT, M. HEWETT, and A. SEIVER (1989): "Intelligent Real-Time Monitoring and Control," *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence.*

HAYES-ROTH, F., D. A. WATERMAN and D. B. LENAT (1983): *Building Expert Systems*, Addison-Wesley, Reading, Massachusetts.

HEWETT, R. (1990): "ICE Manual," BB1 Internal report.

HIMMELBLAU, D. M. (1987): "Interval Analysis as a Tool for Data Rectification," *Proceedings of the AIChE Annual Meeting*, Houston, Texas.

HO, Y. C. (Ed.) (1989): *Proceedings of the IEEE Special Issue on the Dynamics of Discrete Event Systems*, **77**, 1.

ISERMANN, R. (1984): "Process Fault Detection Based on Modeling and Estimation Methods—A Survey," *Automatica*, **20**, 4, 387–404.

JAGER, R. (1990): "Direct Real-Time Control using Knowledge-Based Techniques," *Proceedings of the ESS '90 Intelligent Process Control Design*, Ghent, Belgium.

JAMES, J. R. and C. J. HERGET (1991): "Software Tools for Distributed Intelligent Control Systems," *Proceedings of the 3rd IFAC International Workshop on Artificial Intelligence in Real-Time Control*, Rhonert Park, Sonoma, California.

KARNOPP, D. and R. ROSENBERG (1975): *System Dynamics: A Unified Approach*, John Wiley and Sons, New York.

KIM, I. S. and M. MODARRES (1987): "Application of Goal Tree — Success Tree Models as the Knowledge-Base of Operator Advisory Systems," *Nuclear Engineering and Design*, 104, 67–81.

KRIJGSMAN, A. J. and R. JAGER (1992): "DICE: a Real-Time Toolbox," *Preprints of the 1992 IFAC/IFIP/IMACS International Symposium on Artificial Intelligence in Real-Time Control*, Delft University of Technology, Delft, the Netherlands, pp. 637–641.

KRIJGSMAN, A. J., R. JAGER, H. B. VERBRUGGEN, and P. M. BRUIJN (1991): "DICE: a Framework for Real-Time Intelligent Control," *Proceedings of the 3rd IFAC International Workshop on Artificial Intelligence in Real-Time Control*, Rhonert Park, Sonoma, California.

KRIJGSMAN, A. J., H. B. VERBRUGGEN, P. M. BRUIJN, and E. G. M. HOLWEG (1990): "DICE: a Real-Time Intelligent Control Environment," *Proceedings of the ESS '90 Intelligent Process Control Design*, Ghent, Belgium.

KUIPPERS, B. J. (1984): "Commonsense Reasoning About Causality: Deriving Behavior From Structure," *Artificial Intelligence*, **24**, 169–204.

KUIPPERS, B. J. (1986): "Qualitative Simulation," *Artificial Intelligence*, **29**, 289–338.

KUIPPERS, B. J. (1989): "Qualitative Reasoning: Modeling and Simulation with Incomplete Knowledge," *Automatica*, **25**, 4, 571–585.

KÄLLSTRÖM, C. G. (1979): *Identification and Adaptive Control Applied to Ship Steering*, Doctor's thesis, TFRT–1018, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

LARSSON, J. E. (1984): *An Expert System Interface for Idpac*, Master's thesis, TFRT–5310, Department of Automatic Control, Lund Institute of Technology, Lund.

LARSSON, J. E. (1988): "MESS—A Minimal Expert System Shell," Technical report, TFRT–7380, Department of Automatic Control, Lund Institute of Technology, Lund.

LARSSON, J. E. (1990 a): "A Knowledge-Based Control System Concept," *Workshop on Methods for Measuring, Monitoring, and Controlling Aseptic Processing*, The Swedish Institute for Food Research, Lund Institute of Technology, Lund.

LARSSON, J. E. (1990 b): "A Multilevel Flow Model of Steritherm," *Proceedings of the SAIS '90 Workshop*, Institute of Computing and System Science, Stockholm University / Royal Institute of Technology, Stockholm.

LARSSON, J. E. (1990 c): "An Expert System for Frequency Response Analysis," Technical report, TFRT–7469, Department of Automatic Control, Lund Institute of Technology, Lund.

LARSSON, J. E. (1990 d): "A Multilevel Flow Model of Steritherm," *Proceedings of the ESS '90 Intelligent Process Control Design*, Ghent, Belgium.

LARSSON, J. E. (1990 e): "A Multilevel Flow Model of Steritherm," *Proceedings of Reglermöte '90*, Division of Automatic Control, Linköping University, Linköping, p. 41.

LARSSON, J. E. (1990 f): "A Multilevel Flow Model of Steritherm," *Proceedings of the Nordic CACE Symposium,* Technical University of Denmark, Lyngby, Denmark.

LARSSON, J. E. (1991 a): "Model-Based Alarm Analysis Using MFM," Technical report, TFRT–7470, Department of Automatic Control, Lund Institute of Technology, Lund.

LARSSON, J. E. (1991 b): "Model-Based Fault Diagnosis Using MFM," *Proceedings of the SAIS '91 Workshop,* Computer Science Department, Uppsala University, Uppsala.

LARSSON, J. E. (1991 c): "Model-Based Alarm Analysis Using MFM," *Proceedings of the 3rd IFAC International Workshop on Artificial Intelligence in Real-Time Control,* Rhonert Park, Sonoma, California.

LARSSON, J. E. (1991 d): "FREX—An Expert System for Frequency Response Analysis," *Proceedings of the 11th Triennial IFAC World Congress 1990,* Tallinn, Estonia, pp. 41–45.

LARSSON, J. E. (1992 a): "Model-Based Measurement Validation Using MFM," *Proceedings of the IFAC Symposium on On-Line Fault Detection and Supervision in the Chemical Process Industries,* University of Delaware, Newark, Delaware.

LARSSON, J. E. (1992 b): "Model-Based Fault Diagnosis Using MFM," *Proceedings of the IFAC Symposium on On-Line Fault Detection and Supervision in the Chemical Process Industries,* University of Delaware, Newark, Delaware.

LARSSON, J. E. (1992 c): "An MFM Toolbox," Technical report, TFRT–7493, Department of Automatic Control, Lund Institute of Technology, Lund.

LARSSON, J. E. and P. PERSSON (1986): "Knowledge Representation by Scripts in an Expert Interface," *Proceedings of the 1986 American Control Conference,* Seattle, Washington.

LARSSON, J. E. and P. PERSSON (1987 a): *An Expert System Interface for Idpac,* Licentiate thesis, TFRT–3184, Department of Automatic Control, Lund Institute of Technology, Lund.

LARSSON, J. E. and P. PERSSON (1987 b): "The (ihs) Reference Manual," Technical report, TFRT–7341, Department of Automatic Control, Lund Institute of Technology, Lund.

LARSSON, J. E. and P. PERSSON (1987 c): “A Knowledge Database for System Identification,” Technical report, TFRT–7342, Department of Automatic Control, Lund Institute of Technology, Lund.

LARSSON, J. E. and P. PERSSON (1988 a): “An Intelligent Help System for Idpac,” *Proceedings of the 8th European Conference on Artificial Intelligence*, Technischen Universität München, München, pp. 119–123.

LARSSON, J. E. and P. PERSSON (1988 b): “The Knowledge Database Used in an Expert Interface for Idpac,” *Proceedings of the IFAC Workshop on Artificial Intelligence in Real-Time Control*, Swansea, Wales, pp. 107–112.

LARSSON, J. E. and P. PERSSON (1991): “An Expert System Interface for an Identification Program,” *Automatica*, **27**, 6, 919–930.

LARSSON, J. E. and K. J. ÅSTRÖM (1985): “An Expert System Interface for Idpac,” *Proceedings of the 2nd IEEE Control Systems Society Symposium on Computer-Aided Control System Design*, Santa Barbara, California.

LEES, F. P. (1983): “Process Computer Alarm and Disturbance Analysis: Review of the State of the Art,” *Computers and Chemical Engineering*, **7**, 6.

LIND, M. (1979): “The Use of Flow Models for Design of Plant Operating Procedures,” *IWG/NPPCI Specialists Meeting on Procedures and Systems for Assisting an Operator During Normal and Anomalous Nuclear Power Plant Operation*, Garching, Deutschland.

LIND, M. (1987): “Multilevel Flow Modeling—Basic Concepts,” Technical report, Institute of Automatic Control Systems, Technical University of Denmark, Lyngby, Denmark.

LIND, M. (1988): “Issues in Modeling Information Flow,” *Workshop on New Technology, Distributed Decision Making, and Responsibility*, Bad Homburg.

LIND, M. (1989): “Human-Machine Interface for Diagnosis Based on Multilevel Flow Modeling,” *Proceedings of the 2nd European Meeting on Cognitive Science Approaches to Process Control*, Siena, Italy.

LIND, M. (1990 a): “Representing Goals and Functions of Complex Systems—An Introduction to Multilevel Flow Modeling,” Technical report, Institute of Automatic Control Systems, Technical University of Denmark, Lyngby, Denmark.

LIND, M. (1990 b): "Abstractions Version 1.0—Descriptions of Classes and Their Use," Technical report, Institute of Automatic Control Systems, Technical University of Denmark, Lyngby, Denmark.

LIND, M. (1990 c): "An Architecture for Real-Time MFM Diagnosis," Technical report, Institute of Automatic Control Systems, Technical University of Denmark, Lyngby, Denmark.

LIND, M. (1990 d): "Modeling Control Tasks in Complex Systems," Technical report, Institute of Automatic Control Systems, Technical University of Denmark, Lyngby, Denmark.

LIND, M., E. HARDER, H. JENSEN, and S. AGGER (1987): "Systembeskrivelse og präsentation i proceskontrol, (System Representation and Presentation in Process Control)," SIP project technical report, Institute of Automatic Control Systems, Technical University of Denmark, Lyngby, Denmark, in Danish.

LINDGREN, G. (1985): "Optimal Prediction of Level Crossings in Gaußian Processes and Sequences," *The Annals of Probability*, **13**, 3, 804–824.

MACLEOD, I. M. and V. LUN (1991): "Towards Distributed Real-Time Intelligence," *Proceedings of the 3rd IFAC International Workshop on Artificial Intelligence in Real-Time Control*, Rhonert Park, Sonoma, California.

MAH, R. S. H. (1990): *Chemical Process Structures and Information Flows*, Butterworths, Boston, Massachusetts.

MAH, R. S. H., G. M. STANLEY, and D. M. DOWNING (1976): "Reconciliation and Rectification of Process Flow and Inventory Data," *I & EC Proc. Des. Dev.*, **15**, 175–183.

MAH, R. S. H. and A. C. TAMHANE (1982): "Detection of Gross Errors in Process Data," *AIChE Journal*, **28**, 828–830.

MARIÑO, O., F. RECHENMANN, and P. UVIETTA (1990): "Multiple Perspectives and Classification Mechanism in Object-Oriented Representation," *Proceedings of the 9th European Conference on Artificial Intelligence*, Stockholm, pp. 425–430.

MINSKY, M. (1965): "Models, Minds, Machines," *Proceedings of the IFIP Congress*, pp. 45–49.

MODARRES, M. and T. CADMAN (1986): "A Method of Alarm System Analysis for Process Plants," *Computers and Chemical Engineering*, **10**, 6, 557–565.

MONTELLANO, R., M. P. BERNIER, A. CHÉRUY, and M. FARZA (1990): "A Knowledge-Based System in Modeling and Control for Biotechnological Processes," *Preprints of the 1990 IFAC World Conference*, Tallinn, Estonia, pp. 54–58.

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," *Proceedings of the 10th IFAC World Congress*, München, pp. 274–281.

MOORE, R. L., H. ROSENOF, and G. STANLEY (1991): "Process Control Using a Real-Time Expert System," *Proceedings of the 11th Triennial IFAC World Congress 1990*, Tallinn, Estonia, pp. 241–245.

MÕTUS, L. (1991): "Artificial Intelligence in Hard Real-Time—A New Paradigm Needed?," *Proceedings of the 3rd IFAC International Workshop on Artificial Intelligence in Real-Time Control*, Rhonert Park, Sonoma, California.

MURDOCK, J. L. and B. HAYES-ROTH (1991): "Intelligent Monitoring and Control of Semiconductor Manufacturing Equipment," *IEEE Expert*, December 1991, 19–31.

NAGY, P. A. J. (1992): *Tools for Knowledge-Based Signal Processing with Applications to System Identification*, Doctor's thesis, No. 280, Department of Electrical Engineering, Linköping University, Linköping, Sweden.

NAGY, P. A. J. and L. LJUNG (1989): "An Intelligent Tool for System Identification," *Proceedings of the 1989 IEEE Control System Society Workshop on Computer-Aided Control System Design*, Hyatt Regency, Tampa, Florida, pp. 58–63.

NAGY, P. A. J. and L. LJUNG (1991): "An Intelligent Tool for System Identification," *Proceedings of the 1991 IFAC Symposium on Identification and System Parameter Estimation*, Budapest, pp. 918-923.

NG, H. T. (1991): "Model-Based, Multiple-Fault Diagnosis of Dynamic, Continuous Physical Devices," *IEEE Expert*, December 1991, 38–43.

NILSSON, A. (1991): *Qualitative Model-Based Diagnosis—MIDAS in G2*, Master's thesis TFRT–5443, Department of Automatic Control, Lund Institute of Technology, Lund.

NORBY LARSEN, M. (1990): "Strips as a Planning Method Within Abstractions and MFM Modeling," Technical report, Institute of Automatic Control Systems, Technical University of Denmark, Lyngby, Denmark.

OSMAN, A. (1990): "The Interface Substrate," Technical report, Institute of Automatic Control Systems, Technical University of Denmark, Lyngby, Denmark.

OYELEYE, O. O. (1989): *Qualitative Modeling of Continuous Chemical Processes and Applications to Fault Diagnosis*, Doctor's thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts.

PADALKAR, S., G. KARSAI, C. BIEGL, J. SZTIPANOVITS, K. OKUDA, and N. MIYASAKA (1991): "Real-Time Fault Diagnosis," *IEEE Expert*, June 1991, 75–85.

PAYNTER, H. M. (1961): *Analysis and Design of Engineering Systems*, MIT Press, Cambridge, Massachusetts.

PETERSON, J. L. (1981): *Petri Net Theory and the Modeling of Systems*, Prentice Hall, New York.

PETTI, T. F. (1992): *Using Mathematical Models in Knowledge-Based Control Systems*, Ph. D. Dissertation, University of Delaware, Newark, Delaware.

PETTI, T. F. and P. S. DHURJATI (1991): "Object-Based Automated Fault Diagnosis," *Chemical Engineering Communications*, **102**, 107–126.

PETTI, T. F., J. KLEIN, and P. S. DHURJATI (1990): "Diagnostic Model Processor: Using Deep Knowledge for Process Fault Diagnosis," *AIChE Journal*, **36**, 4, 565–575.

POLLACIA, L. F. (1989): "A Survey of Discrete Event Simulation and State-of-the-Art Discrete Event Languages," *Simulation Digest*, **20**, 3, 8–25.

RAMADGE, P. J. G. and W. M. WONHAM (1989): "The Control of Discrete Event Systems," *Proceedings of the IEEE*, **77**, 1, 81–98.

RASMUSSEN, J. (1986): *Information Processing and Human Machine*

Interaction: An Approach to Cognitive Engineering, North Holland, New York.

RASMUSSEN, J. and L. P. GOODSTEIN (1988): "Information Technology and Work," in M. Helander (Ed.): *Handbook of Human-Computer Interaction*, Elsevier Science Publishers B. V. North-Holland, New York.

RASMUSSEN, J. and M. LIND (1981): "Coping with Complexity," Technical report, Risø National Laboratory, Roskilde, Denmark.

REITER, R. (1987): "A Theory of Diagnosis from First Principles," *Artificial Intelligence*, **32**, 1, 57–95.

SANDEWALL, E. (1988): "Future Developments in Artificial Intelligence—A Personal View," *Proceedings of the 8th European Conference on Artificial Intelligence*, Technischen Universität München, München, pp. 707–715.

SARIDIS, G. N. (1977): *Self-Organizing Control of Stochastic Systems*, Marcel Dekker, Inc., New York.

SASSEN J. M. A. AND R. B. M. JASPERS (1992): "Designing Real-Time Knowledge-Based Systems with PERFECT," *Preprints of the 1992 IFAC/IFIP/IMACS International Symposium on Artificial Intelligence in Real-Time Control*, Delft University of Technology, Delft, the Netherlands, pp. 625–630.

SASSEN J. M. A., A. OLLONGREN, and R. B. M. JASPERS (1992): "Predicting and Improving Response-Times of PERFECT Models," *Preprints of the 1992 IFAC/IFIP/IMACS International Symposium on Artificial Intelligence in Real-Time Control*, Delft University of Technology, Delft, the Netherlands, pp. 709–714.

SASSEN J. M. A., P. C. RIEDIJK, AND R. B. M. JASPERS (1991): "Using Multilevel Flow Models for Fault Diagnosis of Industrial Processes," *Proceedings of the 3rd European Conference on Cognitive Science Approaches to Process Control*, Cardiff, United Kingdom, pp. 207–216.

SCHANK, R. C. and R. P. ABELSON (1977): *Scripts, Plans, Goals and Understanding*, Lawrence Erlbaum Associates, Hillsdale, New Jersey.

SCHANK, R. C. and C. K. RIESBECK (1981): *Inside Computer Understanding*, Lawrence Erlbaum Associates, Hillsdale, New Jersey.

SHORTLIFFE, E. H. (1976): *Computer Based Medical Consultations: MYCIN*, Elsevier Science Publishers B. V. North-Holland, New York.

STEFIK, M. *et al* (1982): *The Organization of Expert Systems—A Prescriptive Tutorial*, Palo Alto Research Centers, Palo Alto, California.

STRUSS, P. (1987): "Multiple Representation of Structure and Function," in Gero, J. (Ed.): *Expert Systems in Computer-Aided Design*, Elsevier Science Publishers B. V. North-Holland, New York.

STRUSS, P. (1991): "What's in SD? Towards a Theory of Modeling for Diagnosis," "Working notes of the Second International Workshop on Principles of Diagnosis," CISE-Tecnologie Innovative, Milano.

STRUSS, P. (1992): "What's in SD? Towards a Theory of Modeling for Diagnosis," in Hamscher, W., L. Console, and J. de Kleer, (Eds.): *Readings in Model-Based Diagnosis*, Morgan-Kaufmann Publishers, Inc., San Mateo, California.

SZAFNICKI, K. and S. GENTIL (1991): "An Object-Oriented Knowledge-Based System for Process Identification," *Preprints of the IFAC Symposium on Computer-Aided Design in Control Systems*, Swansea, Wales, pp. 188–193.

TERPSTRA, V. J., H. B. VERBRUGGEN, and P. M. BRUIJN (1991): "Integrating Information Processing and Knowledge Representation in an Object-Oriented Way," *Proceedings of the Workshop on Computer Software Structures Integrating AI/KBS Systems in Process Control*, Bergen, Norway, pp. 19–29.

TERPSTRA, V. J., H. B. VERBRUGGEN, M. W. HOOGLAND, and R. A. E. FICKE (1992): "A Real-Time, Fuzzy, Deep-Knowledge Based Fault Diagnosis System for a CSTR," *Proceedings of the IFAC Symposium on On-Line Fault Detection and Supervision in the Chemical Process Industries*, University of Delaware, Newark, Delaware.

THOMA, J. U. (1975): *Introduction to Bond Graphs and Their Applications*, Pergamon Press, New York.

TOMITA, S., K. S. HWANG, E. O'SHIMA, and C. MCGREAVY (1989): "Automatic Syntheziser of Operating Procedures for Chemical Plant by Use of Fragmentary Knowledge," *Journal of Chemical Engineering of Japan*, **22**, 4, 364–372.

TOMITA, S., H. YUHKI, M. MOHRI, T. SAWA, and E. O'SHIMA (1986): "Development of Batch Process Operating System," *Proceedings of the 3rd World Congress of Chemical Engineering*, Tokyo.

TORASSO, P. and L. CONSOLE (1989): *Diagnostic Problem Solving*, North Oxford Academic, Kogan Page Ltd., London.

VAN DEN REE, R., H. KOPPELAAR, and E. J. H. KERCKHOFFS (1991 a): "Knowledge Management in Process Modeling. Engineering Systems with Intelligence: Concepts, Tools, and Applications.," *Proccedings of the European Robotics and Intelligent Systems Conference*, Kluwer Academic Publishers, Dordrecht, Nederland, pp. 83–90.

VAN DEN REE, R., H. KOPPELAAR, and E. J. H. KERCKHOFFS (1991 b): "Proposal for Process Modeling," *Proccedings of the IMACS MCTS '91, Modeling and Control of Technological Systems*, Lille, France, pp. 732–737.

VERBRUGGEN, H. B. and K. J. ÅSTRÖM (1989): "Artificial Intelligence and Feedback Control," *Proceedings of the 2nd IFAC International Workshop on Artificial Intelligence in Real-Time Control*, China.

VINA, A. and B. HAYES-ROTH (1991): "Knowledge-Based Real-Time Control: The Use of Abstraction to Satisfy Deadlines," *Proceedings of the 3rd IFAC International Workshop on Artificial Intelligence in Real-Time Control*, Sonoma, California.

WALSETH, J. Å., B. A. FOSS, M. LIND, and O. ÖGAARD (1992): "Models for Diagnosis—Application to a Fertilizer Plant," *Proceedings of the IFAC Symposium on On-Line Fault Detection and Supervision in the Chemical Process Industries*, University of Delaware, Newark, Delaware.

WATERMAN, D. A. (1986): *A Guide to Expert Systems*, Addison-Wesley, Reading, Massachusetts.

WELD, D. S. and J. DE KLEER (Eds.) (1990): *Readings in Qualitative Reasoning about Physical Systems*, Morgan Kaufmann Publishers, Inc., San Mateo, California.

WIENER, N. (1948): *Cybernetics*, MIT University Press, Cambridge, Massachusetts.

WOODS, E. A. (1991): "The Hybrid Phenomena Theory," in J. Mylopoulos and R. Reiter (Eds.): *Proceedings of the 12th International Joint Conference of Artificial Intelligence*, Morgan Kaufmann Publishers, Inc., San Mateo, California.

WOODS, E. A. and J. G. BALCHEN (1991): "Structural Estimation with the Hybrid Phenomena Theory," *Preprints of the 3rd IFAC International Workshop on Artificial Intelligence in Real-Time Control*, Rhonert Park, Sonoma, California.

ZIEGLER, B. P. (1990): *Object-Oriented Simulation with Hierarchical, Modular Models*, Academic Press, Inc., San Diego.

ÅRZÉN, K. E. (1989): "Knowledge-Based Control Systems: Aspects on the Unification of Conventional Control Systems and Knowledge-Based Systems," *Proccedings of the 1989 American Control Conference*, Pittsburgh, Pennsylvania.

ÅRZÉN, K. E. (1990): "Knowledge-Based Control Systems," *Proccedings of the 1990 American Control Conference*, San Diego, California.

ÅRZÉN, K. E. (1991): "Knowledge-Based Applications in the Process Industry: Current State and Future Directions," *Proceedings of the Workshop on Computer Software Structures Integrating AI/KBS Systems in Process Control*, Bergen, Norway.

ÅRZÉN, K. E. (1993): "A Model-Based Control System Concept," manuscript, to appear.

ÅRZÉN, K. E., C. RYTOFT, and C. GERDING (1990): "A Knowledge-Based Control System Concept," *Proceedings of the ESS '90 Intelligent Process Control Design*, Ghent, Belgium.

ÅSTRÖM, K. J., J. J. ANTON, and K. E. ÅRZÉN (1986): "Expert Control," *Automatica*, **22**, 3, 277–286.

ÅSTRÖM, K. J., A. BENVENISTE, P. E. GAINES, G. COHEN, and L. LJUNG (1993): "Facing the Challenge of Computer Science in the Industrial Applications of Control," to appear as an IFAC report at the 12th Triennial World Congress of IFAC in Sydney 1993.

ÅSTRÖM, K. J., A. BENVENISTE, P. E. GAINES, G. COHEN, L. LJUNG, and P. VARAIYA (1991): "Facing the Challenge of Computer Science in the Industrial Applications of Control," *IEEE Control Systems Magazine*, June 1991, p. 86.

ÅSTRÖM, T. HÄGGLUND, C. C. HANG, and W. K. HO (1992): "Automatic Tuning and Adaptation for PID Controllers—A Survey," *Proceedings of the 4th IFAC International Symposium on Adaptive Systems in Control and Signal Processing,* Ecole Nationale Superieure D'Ingenieurs Electriciens de Grenoble, Grenoble, France, pp. 121–126.