

Diagnosis Based on Explicit Means-End Models

Jan Eric Larsson

Department of Automatic Control
Lund University
Box 118, 22100 Lund, Sweden
E-mail: janeric@it.lth.se

This is a *preprint* of an article published in *Artificial Intelligence* 1996, and it is made available as an electronic reprint by permission of *Artificial Intelligence*. The full reference to the published article is:

Larsson, J. E., "Diagnosis Based on Explicit Means-End Models," *Artificial Intelligence*, vol. 80, no. 1, pp. 29-93, 1996.

Diagnosis based on explicit means-end models

Jan Eric Larsson*

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

Submitted to *Artificial Intelligence* on May 24th 1993

Revised version submitted in March 1994

Final version submitted in June 1994

Abstract

This article 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 real-time expert system tool G2, in C, and in Common Lisp, and successfully tested on simulations of several processes.

The knowledge representation ontology used is based on the notion of *flows*, of mass, energy, and information, which are used to describe physical systems. The relationships between structure and function of a system is described by teleological relations, which connect the flow structures into a graph, built at model construction time. This allows the diagnostic reasoning to be implemented as searches in a static graph structure, and it can thus be performed extremely rapidly. As with other model-based approaches, general algorithms are used over a representation with generative capacities. The representation gains strength from being functional with a very abstract physical level, more abstract than most qualitative physics models. It works well with systems that can be described using flows, while it currently lacks the capability of capturing important aspects of other types of systems, for example, electronic circuits.

1. Introduction

Diagnosis of industrial processes can be performed with different types of models. However, most model types in use today focus on physical structure and behavior, and contain little or no *means-end*, i.e., teleological, information.

* This article is based on the author's doctor's thesis from the Department of Automatic Control, Lund Institute of Technology, Lund, Sweden, and on work done during a postdoctoral fellowship visit at Knowledge Systems Laboratory, Stanford University, California.

† Current address: Knowledge Systems Laboratory, Department of Computer Science, Stanford University, 701 Welch Road, Building C, Palo Alto, CA 94304, USA.

The latter is important for diagnosis, though, and it seems to be worthwhile to try and incorporate more teleological knowledge in model-based diagnosis.

This article describes diagnosis using one type of explicit means-end models, *multilevel flow models*, (MFM), as developed by Lind [73]. These are functional models with a very high level of abstraction, combined with a teleological representation of goals, or purposes, of the modeled system. Lind has suggested a syntax for a formal language and given general ideas on how to use the MFM representation. Note that there are several minor differences between different versions of MFM, as presented in [73], in Lind's newer version, [79, 80], and in [67]. The following work is based on the version described in [67], while the latest definition of MFM appears in [79, 80].

The contributions of this article are descriptions of three methods for diagnostic reasoning using MFM:

- Measurement validation
- Alarm analysis
- Fault diagnosis

The measurement validation algorithm takes a set of measured flow values and uses any available redundancy to check consistency. A single erroneous flow measurement is marked and corrected. If there are several conflicting values, the consistent subgroups of measurements are marked but no flow value is 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 the primary alarms, while the other alarms are *either* primary *or* consequences of the primary ones.

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 or from measured signals and triggering of rules. The system looks for faults, provides explanations, and gives remedies.

2. The Basic Concepts of MFM

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 concept types of 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 level of the tank high enough.

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 delivering 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.

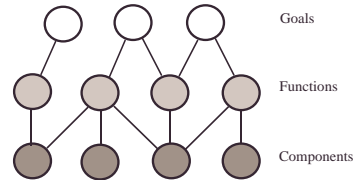


Figure 1. Goals, functions, components, and relations. The functions are connected to goals via achieve relations, while the components are connected to functions via realize relations. Subgoals may be connected to functions via condition relations, but this is not shown in the figure. All three kinds 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 Fig. 1.

MFM demands that goals, functions, and physical components be viewed as separate entities. The assumption that functions are separate from components is similar to the *no function in structure* assumption of qualitative physics. In addition to this, MFM also assumes that the goals are not given by separate functions. Instead they must be stated by the designer during model construction.

3. Goals

The concept of a goal is central to MFM, as it is the representational device for teleological information. 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 is useful to narrow it down to more specific descriptions. Thus, three different types of goals are 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:

$$x_0 \leq x \leq x_1.$$

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.

Economy Goals

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

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

where G_0 and G_1 are numerical limits inside which the system is working with satisfactory efficiency. 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.

4. Functions

The second important concept of MFM is that of a *function*. In the context of processes it is possible to find several interpretations of the function concept. In MFM a function is always associated with a goal, and correspondingly, goals are always associated with functions.

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 flow 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 as well. 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. Most of the graphical symbols used to build MFM models are shown in Fig. 2.

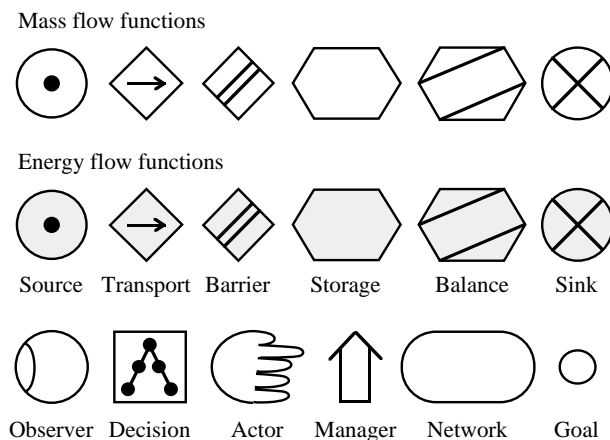


Figure 2. The symbols for the different MFM objects.

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 is general and useful enough to be interesting. MFM could be extended with other types of functions, see [67] for a discussion.

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. 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.

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. 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.

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. A barrier function is characterized by a throughput flow F , which 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.

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. 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, a storage

function may have several inputs and outputs, [73]. For simplicity, this has been disallowed in the current work.

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

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

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.

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. 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.

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 human operators. 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.

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 are modeled with decision functions, which differs from the intentions of Lind, [73]. Lind does not view simple control algorithms as decision making. 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.

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. 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.

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. A network function can be connected to goals via achieve relations.

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. 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.

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 later sections.

This concludes the description of the semantics of MFM. It would certainly be possible to use a representation based on, for example, predicate logic, but no such representation has been developed. For the simple and fast diagnostic algorithms to be presented in this article, the graphical representation with a simple, underlying data structure based on objects and links is quite sufficient.

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:

- 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 only be connected to outgoing transports.
- Decision functions may only be connected to transports.
- Actors may only be connected to incoming transports.

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.

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 flow 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.

The given description of MFM is based on [73], while the following is original work. It should be noted that later versions of MFM differ in the descriptions of control systems and information flow, [79, 80].

6. An Example of a Flow Model

The following example will be used to explain the basic concepts of MFM. The target process consists of a plate heat exchanger, see Fig. 3.

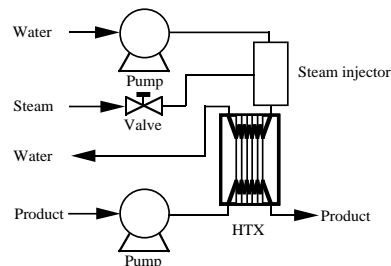


Figure 3. A heat exchanger system. The flowsheet shows how water is pumped through a steam injector, where it is heated with steam, and through a plate heat exchanger, where it heats the product.

This simple system serves quite well to explain the concepts of MFM. The primary goal is to heat the product to a certain temperature, but a brief analysis shows that there are two subgoals also: having water and product available, i.e., bringing the media to the heat exchanger:

- G1: Heat product to certain temperature
- G2: Bring product to heat exchanger
- G3: Bring water to heat exchanger

The given example process is rather small, but there are many functions present:

- 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 steam
- F9: Transport steam
- F10: Mix water and steam

The third type of objects are the physical components. Note that the product and water tanks do not actually appear in Fig. 3:

- 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. 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. In Fig. 4, both the goal hierarchy and the means-end relations are shown in a graph.

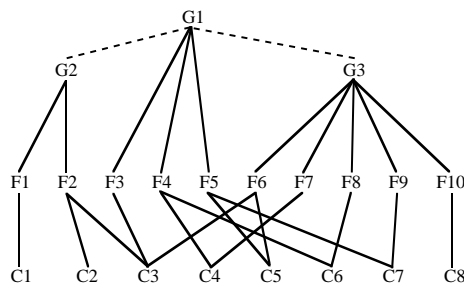


Figure 4. Goals, functions, components, and relations of the heat exchanger system.

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 Fig. 5.

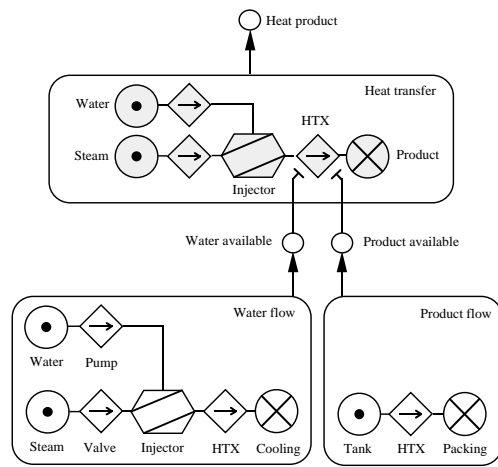


Figure 5. A flow model of the heat exchanger system. The goals and goal hierarchy are shown in the tree structure of the graph, while the functions are connected into flow paths in three networks. The topmost goal, to heat the product, is achieved by the heat transfer flow path, (upper network), while 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.

7. Diagnostic Algorithms

Before we proceed to describe the diagnostic methods, it will be useful to mention a few facts about the implementations performed.

The algorithms have been implemented in the real-time expert system tool G2, [89, 90], as a part of the doctor's project described in [67]. All three algorithms were successfully tested on simulations of two processes, a simple lab process and Steritherm. The latter is a real-world process for ultra-high temperature sterilization of liquid food products.

The algorithms have also been implemented in C, and together with a graphical editor and an ASCII file system they form an MFM Toolbox for Macintosh systems. This version has been tested with several processes, including all used with the G2 and Common Lisp implementations. A simpler console version of this program will run on any system with a C compiler, [69].

The alarm analysis and fault diagnosis algorithms have also been implemented in Common Lisp for use in the Guardian system, [40]. Here, the algorithms have been tested on several small examples of systems of the human body. Currently, a large MFM model of the body of an intensive-care unit patient is being verified by medical experts.

8. The Architecture of a Monitoring and Control System

The three methods for diagnostic reasoning fit into different parts of a monitoring and control 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 6.

The inputs needed can be obtained in several different ways. They can be direct or filtered signals from sensors. It would also be possible, 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. For example, in the Guardian system, the alarm analysis and fault diagnosis operates on qualitative data, supplied by several routines that perform data filtration and abstraction based on the current resources available to the system.

The MFM algorithms can, of course, be used in a stand-alone fashion, presenting their results directly to the user or operator, but it is more realistic to use them as integral parts of a larger monitoring and control system. The latter approach was used in the project "Knowledge-Based Real-Time Control Systems," [3–9, 63], and is currently being investigated in the Guardian project. Here, the MFM algorithms are used alternatively with other methods, depending on the available resources and needs, and the results of several methods may be combined to obtain higher quality diagnosis than any single method can provide.

When the operating conditions of the target system change considerably, a new and different MFM model may be needed. This would, for example, be the case when a nuclear power plant switches from production to emergency shutdown mode. In such a case the monitoring and control system must also switch to a new MFM model.

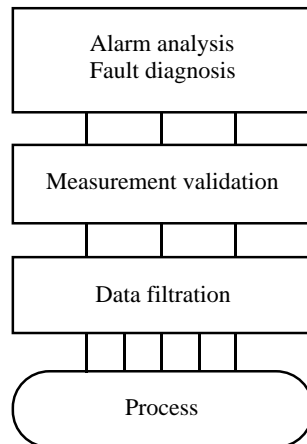


Figure 6. The places for the diagnostic methods in a monitoring and control system. Raw data from the process is treated by filters and low level numerical routines before it is sent on to the other algorithms. The measurement validation algorithm works on an intermediate level, while the alarm analysis and fault diagnosis algorithms belong in the topmost, supervisory level of the system.

9. Measurement Validation

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.

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, 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., m³/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 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.

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:

- 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.

- 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.

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

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

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.

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

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

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, in the current implementations, all connected flow carriers are marked as belonging to different groups.

- 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 enables a splitting of each flow into smaller groups with consistent measurement values. It should be noted that the last rule means that there can be groups with holes in them; they need not be directly consecutive. This is the case in Fig. 7. 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.

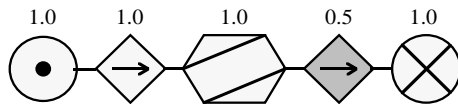


Figure 7. An example of a flow path. Here the flow functions form two consistent subgroups, where one group is surrounding the other. The three possible fault hypotheses are that the four measurements are correct and the one is faulty, that the one is correct and the four are faulty, and that all measurements are faulty.

The last rule uses information about several flow functions connected in line; the other four only look at directly connected flow functions and no global analysis is needed. This enables the algorithm to be very efficient.

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:

- A flow value from a subgroup of more than one flow function is said to be *validated*. It has precedence over flow values supplied by single flow functions, when propagated both upstream and downstream in flow paths.
- A single measured flow value is propagated to unmeasured flows, both upstream and downstream, if there is no other information available.
- 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 some sense; it only serves to make the flow propagation behave consistently.
- 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 an arbitrary decision to make the propagation work.
- Guessed values are not 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.

Validation

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

- 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.
- 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.

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. This is aimed at giving some reliability to the guessed flow values of functions not connected to sensors. However, if there exists two inconsistent groups of measurements, each based on more than one sensor value, the guessing strategy of downstream propagation is arbitrary.

The above description is based on the G2 implementation. In the C version, single downstream propagation is used instead. This still allows the method to detect any inconsistencies, but does not provide guessed flow values with even a minimal degree of reliability. In either case, the method is able to detect any possible inconsistency between flow measurements.

Note also that the measurement validation method use neither achieve nor condition relations. Instead, each flow structure is treated as a separate part, and no global interactions are considered. This makes the method behave very favorably when the problem size increases. Furthermore, if the MFM model is only to be used for measurement validation, neither goals nor means-end relations are needed, but the process may be described as a list of separate flow structures. This is not true of the other algorithms, however.

Examples of Measurement Validation

Let us demonstrate the method on a small example. The process consists of a storage tank, a pump, and two cylindrical tanks, see Fig. 8.

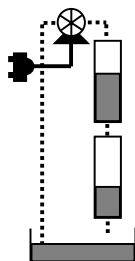


Figure 8. The tanks process. Water is pumped from a storage tank, to a cylindrical tank, from where it flows down into another tank, and then back to the storage again.

The main goal of the process is to keep the water level in the tanks at a specified level. This is achieved by the primary mass flow, i.e., the circulation of water. Here, the storage tank has been modeled both as a source and a sink. One of the transport functions, (the one that corresponds to the pump), depends on the subgoal that the pump motor has power, and this goal in turn is achieved by a secondary flow of electrical energy. The power support system is quite complicated but has been modeled simply as a source, a transport, and a sink. All this can be seen in the MFM model in Fig. 9.

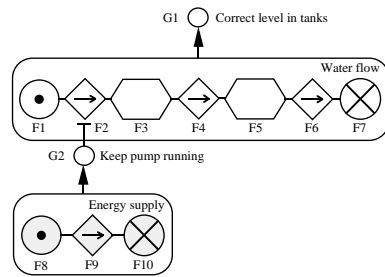


Figure 9. An MFM model of the tanks process. The main goal is to keep the level of the upper tank correct, and it is achieved by a water flow. In order for the transport function F2 to be available, i.e., to keep the pump running, energy must be supplied.

EXAMPLE 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	$20 \times 10^{-6} \text{ m}^3/\text{s}$	(outflow from storage)
flow of F2	$10 \times 10^{-6} \text{ m}^3/\text{s}$	(flow through pump)
inflow of F3	$20 \times 10^{-6} \text{ m}^3/\text{s}$	(upper tank inflow)
deriv of F3	$0 \times 10^{-6} \text{ m}^3/\text{s}$	(volume change)
outflow of F3	$20 \times 10^{-6} \text{ m}^3/\text{s}$	(upper tank outflow)

Table 1. A set of flow measurement values.

The situation described in Table 1 is also shown in Fig. 10, 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.

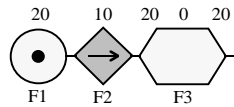


Figure 10. A flow path corresponding to Table 1. The flow of F2 disagrees from the rest, and there are two consistent subgroups, whereof one is single and surrounded.

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 also marks F2 specially, as it is a single function group, and the validated flow value of F2 is set to $20 \times 10^{-6} \text{ m}^3/\text{s}$, (the flow of the surrounding group).

The consistent subgroup information has been shown in Fig. 10 with a shading system. In addition to this, the flow function F2 should also have a special marking, for forming a single function group. □

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

Table 2. A second set of flow measurement values.

EXAMPLE 2

Now assume instead that the flow value of F1 is $10 \times 10^{-6} \text{ m}^3/\text{s}$ and that the flow of F2 is not measured; a situation which is shown in table 2. It would correspond to the flow function description of Fig. 11.

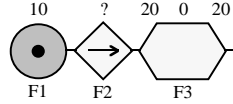


Figure 11. A flow path corresponding to table 2. The flow value of F2 is propagated from the storage, where the measurements form a multiply validated and consistent subgroup. It can then be seen that F1 forms a single, (but not surrounded), subgroup.

In this case the system propagates the validated value of $20 \times 10^{-6} \text{ m}^3/\text{s}$ 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. □

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

Table 3. A third set of flow measurement values.

EXAMPLE 3

Further assume the situation described by Table 3, which is also found in Fig. 12.

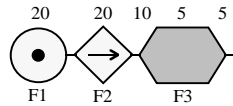


Figure 12. A flow path corresponding to table 3. Here there are two consistent subgroups which both consists of more than one measurement. The algorithm signals that the flows do not agree, but it cannot guess which ones that are correct.

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 marks the two inconsistent groups, but takes no further action. Marking any particular value as wrong could be misleading and potentially dangerous. □

10. Alarm Analysis

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.

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:

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

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:

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

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.

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 will not. An example is given in Fig. 13, where a source F1 is connected to a transport function F2. This could correspond to, e.g., a tank connected to a pump.

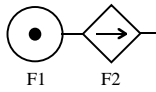


Figure 13. A connected source and transport. The source has a maximum outflow capacity, F_{cap} , and the transport has a working interval, $F_{lo} \leq F_t \leq F_{hi}$. If the wanted outflow of the source goes over its capacity or if the actual flow through the transport leaves the working interval, 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 $F_t < F_{lo}$, the alarm will be *loflow*; if $F_t > F_{hi}$, 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 and modeling 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 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 can not 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:

- A source *locap* alarm will force the connected transport to have a *loflow* alarm.
- 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.

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. Using such assumptions for all flow functions, a small G2 program was written to automatically generate rules for all possible alarm causations, [68]. 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 were checked and found to be correct.

Possible Secondary Alarms

The examples can be extended to all the allowed connections of flow functions. This gives 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:

- A source *locap* will force the connected transport to have a *loflow*.
- 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.
- 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*.
- A barrier *leak* may cause a transport connected via a balance to have a *loflow*, or a *hiflow*.
- A storage *lovol* may cause an outgoing connected transport to have a *loflow*.
- 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*.
- A storage *leak* may cause the same storage to have a *lovol*.
- A storage *fill* may cause the same storage to have a *hivol*.
- A balance *leak* may cause a connected outgoing transport to have a *loflow*, and a connected incoming transport to have a *hiflow*.
- A balance *fill* may cause a connected incoming transport to have a *loflow*, and a connected outgoing transport to have a *hiflow*.
- A sink *locap* will force the connected transport to have a *loflow*.
- An alarm in a network will force a function depending on this network to fail.

Note that the final rule makes use of means-end relations. Thus, even if most of the algorithm is concerned with comparing alarms of functions in a single flow structure, information may propagate upwards in the model graph, and a single alarm may ultimately affect the failure states of all goals and networks above it all the way up to the toplevel goals of the entire model.

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.

The possibility of different assumptions on flow function behavior and, consequently, of different rule sets, rises several interesting questions. For example, what different rule sets could be proposed, and how would they differ in properties and in applicability to different domains or processes? And given some different rule sets, how should the best one for a certain task be chosen? It might even be possible to provide several rule sets and have the supervisory monitoring and control system choose the appropriate one as a part of the diagnosis.

The choice of assumptions, i.e., the choice of flow function semantics, as well as the choice of flow functions types, is made from a very large number of possible abstractions of behaviors of flow systems, and rather than being a theoretical problem of enumerating all possibilities and investigating their different properties, it is the author's opinion that the selection of flow functions and their semantics is ultimately an engineering problem. The first step on the way to deciding which rule sets are preferable is to gain practical experience of construction and use of FM models, and this is an ongoing effort.

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.

An Alarm Analysis Algorithm

The rules above 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, and a primary fault may be “hidden” by a causation, i.e., look like a secondary fault. Thus, the method differentiates between positively primary alarms, and alarms that may be either primary or secondary, so called *indeterminate* alarms. In this article, the concept “secondary” is always taken to mean the same as “failed and indeterminate.” It should be noted that this is not a peculiarity of MFM. It is a basic property of all causal analysis.

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*.

The rules are only 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. This makes the execution very fast.

Unknown Alarm States

When some alarm states are unknown, the flow networks can be used to guess the missing values. This method is 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, (i.e., indeterminate), alarms according to a set of rules.

The rules used for this exactly correspond to the alarm analysis rules. Every such 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.

For example, two consequence propagation rules may be formulated for the source and transport in Fig. 13:

- 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.

Conflict Resolution

There are cases where conflicting guesses would be possible. Consider the situation shown in Fig. 14.

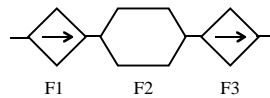


Figure 14. A storage connected to two transports. If the alarm state of the storage is not measured, while both transports receive *lowflow* alarms, the order of the alarms will determine which of them that will be considered primary. This can be solved by a conflict resolution mechanism, however, which 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 *lowflow* alarms. The guess for the alarm state of F2 would be a *lowflow*, and the alarm of F3 could be a consequence of this.

If, however, the *lowflow* of F3 came before that of F1, the guess would be a *highflow* 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, (i.e., indeterminate), alarms as possible. For example, in the case above, the first alternative interpretation of the situation should be preferred. This conflict resolution gives rise to some further diminishing of the diagnostic resolution and more complexity in the rules, but the resulting interpretation is always correct. This actually corresponds to a “wait and see” step, because if the fault of F3 was indeed a primary one, this would be discovered if the known primary fault of F1 was remedied.

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.

Implementations

The description above is based on the G2 implementation, which uses generic rules to implement the alarm analysis. The method has also been implemented in C and Common Lisp. The C version uses a table driven algorithm instead of rules. This has resulted in even faster execution, see Section 14.

Using the Results

The result from applying the described method is that alarms will be marked as either primary or indeterminate. 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 indeterminate 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 indeterminate and other primary alarms remain. This knowledge is heuristic in nature, and should be clearly separated from the results of the alarm analysis. It may, for example, be implemented using a standard rule base to discriminate between different situations and giving advice on which actions to take first.

Examples of Alarm Analysis

Let us now demonstrate the method on an example. Once again the tanks process will be used.

EXAMPLE 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 Fig. 15. The shading of the flow function symbols is used to indicate the failure state, (a dark shade means a primary alarm, a lighter shade an indeterminate alarm, and white a normal or unalarmed state).

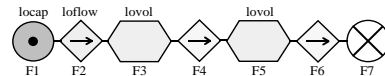


Figure 15. An alarm analysis situation. The functions F1, F2, F3, and F5 have *locap*, *loflow*, and *lovol* alarms. The alarms of F4 and F6 have been guessed. In this situation, the *locap* of F1 is the only primary alarm.

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 methods 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. This has been shown with the shading in Fig. 15. 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. □

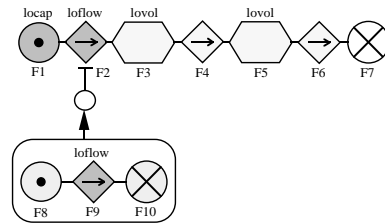


Figure 16. Another alarm analysis situation. Here the *loflow* of F2 must be primary, as there is an alarm in the achieving network, (the *loflow* of F9).

EXAMPLE 5

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 Fig. 16. □

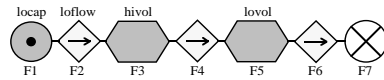


Figure 17. A third alarm analysis situation. Here there are three primary alarms. The situation is highly dynamic.

EXAMPLE 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 Fig. 17. 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. □

11. Fault Diagnosis

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, [112]. However, rules-based systems have several shortcomings, for example that each rule base is specific for a certain process and task, a rule base may contain inconsistencies, and a large rule base most probably will do so, a large rule base is difficult to overview, both in building and updating, and 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 *model-based approaches*. The process of design, construction, and updating of the knowledge database can be made more efficient, if the models are intuitive and easy to use for the domain expert. Consistency within the model can often be guaranteed, and the model can be used to find any deviation from the working state and not only pre-specified faults. MFM shares all these properties with other model-based approaches

The MFM Data Structure for 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 alarm analysis. 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 test to be performed, in order to investigate the failure state of the function.

Alternatively, automatic sensor readings may be used. Each flow function can also have a remedy of the fault, in the form of a text string to be output by the algorithm, or an automatic procedure to call.

As an example, we will once again use the flow 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 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 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.*”
- 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.*”

Note that the above assumptions are for use in the following example. When the algorithm is used as part of a monitoring and control system, that system would handle both questions and remedies. Thus, it would be a better solution to connect the MFM model to a model of physical structure, and use the latter to formulate questions, receive measurements, and to provide remedies. This solution is used in the Guardian project.

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 questions or sensor readings to find the failure state of those flow functions. Depending on the answers or sensor readings, parts of the flow 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 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 / sensor reading 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 has been implemented as 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 very efficient.

The description above is, once again, based on the G2 implementation. The method has also been implemented in C and Common Lisp, in the form of recursive procedures performing a depth-first search. This enables even faster execution.

As is shown in Section 14, the speed of the fault diagnosis algorithm is so high that it may seem unnecessary to select which parts of the MFM model to investigate. Instead, why not simply check all goals and functions, one after the other? This approach has been taken in the PERFECT project, [108–111]. However, the speed of the diagnosis is not determined by the search algorithm alone. The questions and sensor readings may be costly in time and other resources, or involve simulation and other computations, etc. Due to this, it will often be important to keep the number of questions and sensor readings down and to direct the diagnosis to as small parts as possible of the process.

An Example of Fault Diagnosis

Let us now demonstrate the method on a small example. Once again, the tanks process will be used.

EXAMPLE 7

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.

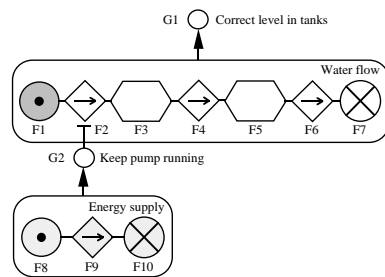


Figure 18. The diagnostic search started from the goal G1 , followed the achieve relation down into the water flow network, and has reached the source F1 .

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 flow model, see Fig. 18. The source F1 has the following question associated to it:

Q: *Is there water in the storage tank?*

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 Fig. 19.

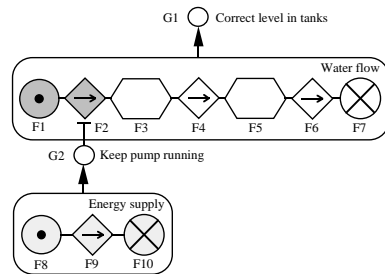


Figure 19. The diagnostic search has concluded that the source F1 is faulty. Then it has moved on to reach the transport F2 .

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 Fig. 20.

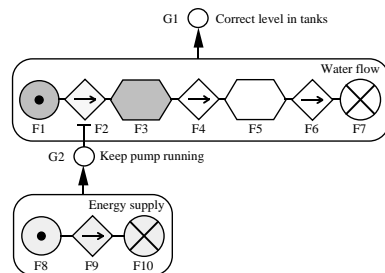


Figure 20. The diagnostic search has concluded that the transport *F2* was at fault, and the alarm analysis signals that the fault of F1 is primary, while the fault of F2 may be secondary. Then the search has reached the storage F3 .

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.

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 can also be a consequential fault.

The flow functions F6 and F7 are neither alarmed or have questions; thus the algorithm would guess that F6 has a *loflow* alarm, while F7 is in a normal (working) state. All this can be seen in Fig. 21.

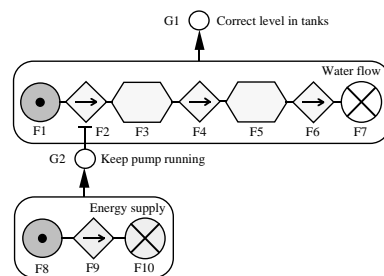


Figure 21. The diagnostic search has finished its investigation of the water flow network, and the alarm analysis concludes that there is one primary fault, three secondary, and two guessed faults, at F4 and F6. As the transport F2 was at fault and there is a condition relation, the search has continued down to the energy supply network and has reached the source F8.

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 Fig. 21, 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.

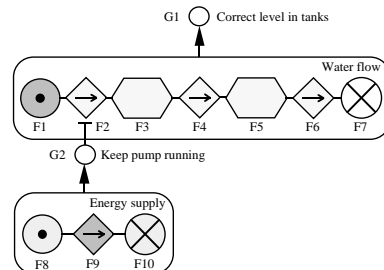


Figure 22. The diagnostic search has concluded that the source F8 is working and reached the transport F9.

When the algorithm comes to the transport F9, see Fig. 22, 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 *lowflow* 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 Fig. 23.

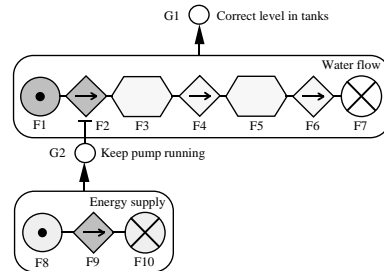


Figure 23. The state after the fault diagnosis. As the transport F9 is at fault, the fault in F2 is also a known primary fault. The user may now ask for explanations and remedies, and the algorithm will search through the graph and output the appropriate text strings from the failed flow functions.

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. □

EXAMPLE 8

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.*

So far, the examples built within the G2 implementation has used this simple strategy of outputting one remedy text for each failed function. Another possibility would be to have the system activate rules or procedures to produce explanations and remedies using information from the whole MFM model, or to actually perform remedying actions. The rules and procedures needed should be written in the general G2 rule format. □

Generation of Explanations

The C and Common Lisp versions of the fault diagnosis were enhanced with an algorithm for automatic generation of explanations in a restricted natural language style. This algorithm is implemented as a recursive depth-first search, that reads the fault states of the MFM model objects and builds a textual explanation as it moves down the graph.

This algorithm uses information about the fault state of the whole MFM model, and can produce reasonable causal explanations of why goals and

functions depending on other goals and functions have failed. An example output is shown in Figure 24.

“The diagnosis is that the goal G1, (maintain the water level of the upper tank), is not fulfilled. This is due to the failure of F1, (the storage tank’s ability to provide water), and F2, (the pump), which are not working. F2’s failure is caused by the fact that the goal G2, (provide the pump with power), is not fulfilled. This is due to the failure of F9, (the pump’s power switch), which is not working.”

Figure 24. An explanation of the failure state in Examples 7 and 8. The text was copied from the screen of the MFM Toolbox in C. The expressions within brackets are textual descriptions associated with each goal and function. Some formulations are clumsy, but the example is intended to show the system’s ability to generate explanations, not to produce correct natural language.

When the fault diagnosis algorithm is used as a part of a larger monitoring and diagnosis system, the explanations and remedies should be handled in cooperation with other algorithms. For example, in the Guardian project, the automatically generated explanations will be used in communication with the intensive-care unit personnel, but once the fault diagnosis has computed a set of failed functions, i.e., diseases, treatments for these will be handled by a separate treatment database and a treatment plan generator. Guardian also monitors the execution of these plans.

12. Experimental Experiences

No matter what the theoretical properties of diagnostic algorithms, their utility will ultimately be decided by their practical usefulness. Different properties of the methods will add up to determine their relative merit.

MFM is a new and largely untested modeling concept, and it differs from most other model-based approaches, conceptually as well as in detail. Thus, before any of the abovementioned properties can be assessed, further experiences of modeling and use are needed. Therefore, the best we can do currently, is to present the experiences gained from some of the models that have been constructed.

Three Test Processes

The algorithms described above have all been tested on simulations of the tanks process, Steritherm, and the Guardian model of the human body.

The tanks process is a small laboratory process used in teaching basic control theory at the Department of Automatic Control, Lund Institute of Technology Lund, Sweden. It was used as the generic “toy” example during the development of the MFM algorithms, and to produce the examples used in this article. As has been shown, it suffices for demonstrating all three algorithms.

Steritherm is a widely used, moderately sized process for ultra-high temperature treatment, (UHT), of dairy products, see Figure 25.

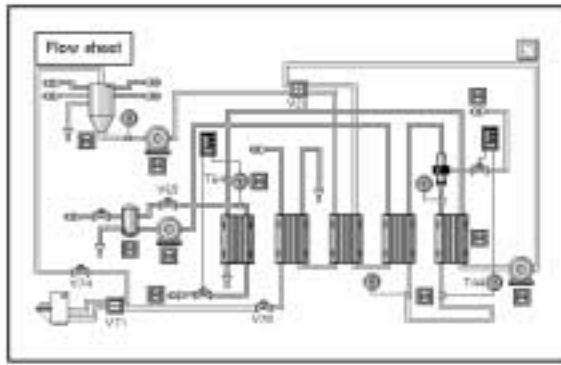


Figure 25. The Steritherm flow sheet, (screen dump from the G2 Toolbox implementation).

Steritherm is a real process in world-wide use, but it is still small enough to be of manageable size for an academic research project. It was the target process used in the Swedish project “Knowledge-Based Real-Time Control Systems, [3–9, 63], of which the author’s doctor’s project was an informal part.

The MFM model of Steritherm describes the toplevel goal of sterilizing the liquid foodstuff, and how this is done by heating the product to 137 °C for a few seconds. The main flow of thermal energy is modeled in detail, and in addition to this, the supporting flows of product and water for heating and cooling are described. Some other support systems, such as pressurized air and the 220 and 380 Volt electrical systems have not been included in the model.

The measurement validation algorithm has been verified using the fairly complex product mass flow of Steritherm. The alarm analysis was tested on the the thermal energy flow, and the fault diagnosis on faults which had effects on the entire plant. All tests were on realistic fault situations. The conclusion was that all three algorithms worked accurately and gave correct and useful information. They all managed to handled multiple faults without problems. It may also be noted that no changes whatsoever were necessary when transferring the algorithms from the tanks process to Steritherm.

The Guardian project aims at developing a monitoring and diagnosis system for use with post-operative intensive-care patients, [40]. The system is implemented as an intelligent agent, which can use several different reasoning methods, according to the changing demands and available resources. It is built on the BB1 blackboard system architecture, [38, 39].

For the Guardian project, a large MFM model of the human body has been developed. It covers all the systems needed for intensive-care unit monitoring, such as the heart, circulation, the body fluid volume, the nutrition, respiration, body temperature, acid-base balance, the concentrations of sodium, potassium, calcium, and magnesium, and the, often multiple, regulatory mechanisms for all these systems.

The two implemented algorithms, (alarm analysis and fault diagnosis), have been verified and found to work without any problems on this model, which is probably the largest MFM model in existence. Judging from these experiences, it is clear that MFM algorithms scale favorably, without changing any qualitative properties in behavior or results.

Reasonable assessments of the efforts needed to construct the three MFM models are shown in Table 4. The knowledge engineering includes the time needed to study the process and to find out the physical facts of how it works, while the model construction time was used for identifying the goals and

functions and constructing the actual MFM model. Since it is difficult in practice to completely compartmentalize the different phases from each other, the figures in Table 4 are approximations. As can be seen, the MFM specific effort is quite modest. Although no comparative studies are available, it is the author's experience that the MFM modeling effort is much smaller than the effort needed to turn the same knowledge into a rule base. Furthermore, the model-based and graphical nature of MFM makes it considerably easier to handle MFM models, than to use knowledge encoded in rules.

Process	Size	Knowledge Engineering	Model Construction
Lab Tanks	"toy"	0	1 hour
Steritherm	medium	2 months	1 week
Guardian	large	4 months	1 week

Table 4. Three modeling tasks of different size. The times are fairly reasonable estimates of the man-hours used. Note, though, that the Guardian modeling effort is still not entirely finished.

13. Properties of the Algorithms

Using MFM to represent diagnostic knowledge has several clear advantages over using an ordinary rule-based system, and it may also retain some of these advantages in comparison with other model-based techniques. This is due to the fact that MFM models consist of static graphs.

The algorithms are local and incremental. They work in real-time, and propagate information along static links only. This makes them very efficient, and the effort as a function of model complexity increases at worst linearly with the size of the MFM models. The local nature also has the benefit that feedback and recirculation loops pose no problems for the algorithms.

Worst-Case Estimates

Diagnostic reasoning tasks like the ones described above can be implemented as searches in MFM graphs. The fault diagnosis method uses a depth-first search, and as the size of every subtree is known, it is simple to obtain a worst-case estimate of the time or computational effort needed to diagnose a certain subgoal. Similar statements are true for the other two algorithms. In general, it is possible to write diagnostic algorithms for MFM which will provide reliable and precise information of worst-case efforts. It is important to note, however, that in case the target process may shift operating mode, a different MFM model may be needed to describe the new goals and functions. In such cases, the monitoring and control system using the MFM algorithm must have a whole set of MFM models available, and worst-case estimates for each of these.

Hard Real-Time Demands

The ability of giving worst-case estimates of the time needed for a diagnostic subtask makes MFM well suited for diagnosis under hard real-time constraints. Whenever a subtask must be finished in a fixed time, the known worst-case can be used to test whether it is indeed possible to comply with the timing demands. If not, the subproblem in question can not be solved.

Here, the hierarchical structure of MFM models is quite valuable. The general structure of an MFM model is such that the toplevel goals and flow

structures describe plant-wide goals and functions. The functions in these flow structures are conditioned by subgoals, each concerned with support systems for the main systems, and the functions here are in turn conditioned by smaller subsystems.

For example, in an MFM model of a nuclear power plant, the toplevel flow structures would describe the flow of energy from reactor, via primary water system, to the generators, and the primary flow of water. On the next level there would be flow models of reactor, generators, etc., and on the next lower level models of reactor and generator subsystems. Measurements are usually available on all these different levels.

The fault diagnosis algorithm presented above starts from the topmost goals, i.e., those goals that are most general and concern production and safety demands of the entire plant. Then the search moves downwards in the MFM graph and attacks more specific subproblems. Currently, the fault diagnosis search is implemented as a simple depth-first search, which investigates all levels to obtain a result. It would be a simple matter to use a breadth-first, (or staged depth-first), search instead. This would allow the algorithm to first handle the toplevel, plant-wide diagnosis, and then gradually search deeper, making the diagnostic resolution finer and finer as more time was available. Whenever the time would run out, a more or less coarse diagnosis would be available. Thus, MFM allows graceful degradation under hard real-time constraints.

Large Scaling of Problems

MFM's graphical structure makes the presented algorithms behave favorably when the problems are scaled up. All three methods increase at worst linearly with the size of the MFM model. However, most incoming data cause only local changes in the diagnosis, and very often, the computational effort increases less than linearly. For example, the cost of performing a fault diagnosis of one subpart of a model is not affected at all by adding another, (separate), subpart.

Parallel Execution

The local, incremental updating of diagnostic information used in the three methods also enables them to be efficiently parallelized. Since the methods use only local information, there is no global bottleneck in the system. The distribution can be done in two different ways:

- Each processor can be responsible for a specific subpart of the model. This corresponds to a standard distributed control and supervisory system, where each part of the process and thus of the MFM model has a designated processor. The topmost levels of the model will have to be handled by a globally responsible processor that gathers the information from the other processors.
- Processors may be assigned to tasks on a server basis. When a new subpart of the MFM model must be traversed, this task is sent to a new processor. In this version, a fault diagnosis moving downwards in the MFM graph splits into several parallel branches.

Once again note that although the algorithms themselves are so fast that a parallel implementation may seem unnecessary, other activities such as sensor readings, lower level simulations, asking questions, etc., may require enough time or other resources to motivate further increase of efficiency.

14. Implementation

The algorithms presented above have been implemented and tested in the real-time expert system tool G2, in C for Macintosh systems, and in Common Lisp for use in the Guardian project. All examples shown in this article have been tested in both the G2 and C implementations.

The first implementation was done in G2 and used this tool's special graphical data structures and generic rules. The MFM models are built with G2's standard graphical interface and each algorithm is implemented as a generic rule base, each of which is reasonably small, see Table 5. This implementation does not show the speed of the methods. Since G2 has a minimal sampling period of one second, each call to a diagnostic algorithm takes precisely 1 second. The system is commercially available and has been sold to CERN. G2 was developed by Gensym Corporation, [89, 90].

Measurement validation	<i>66 rules</i>
Alarm analysis	<i>93 rules</i>
Fault diagnosis	<i>19 rules</i>

Table 5. The three methods have been implemented as G2 knowledge databases.

The second implementation has been done in C, [69]. The MFM data structures and algorithms are portable, but the graphics interface and file system is currently only available for Macintosh systems. This implementation enables the algorithms to be used in conventional monitoring and control systems, and it also highlights the efficiency of the algorithms. Since the fault diagnosis is the most time consuming of the algorithms, some performance data for it is shown in Table 6. Here, the fault diagnosis was executed using simulated sensor readings from a pregenerated fault situation. Note that the times given are for the worst case, i.e., when all objects of the model must be investigated. Often, a diagnosis is orders of magnitude faster, and the measurement validation and alarm analysis are faster still. The central part of the C code implementation of the fault diagnosis algorithm is given in Appendix B.

Process	Objects	Rules	Worst Case	Rules per Second
Lab Tanks	27	39	480 μ s	81 000
Steritherm	99	150	630 μ s	240 000
Guardian	331	544	1100 μ s	500 000

Table 6. Performance data for fault diagnoses of the three models using the C implementation. 'Objects' are the total number of MFM objects used in the model, 'Rules' are the number of rules needed to perform an equivalent diagnosis with backward chaining, 'Worst Case' is the longest possible execution time for a fault diagnosis of the model, and 'Rules per Second' gives the apparent speed of the algorithm, should it have been performed by a rule-based system. More than half the time is spent on sensor readings and bookkeeping operations.

The Lisp implementation described below includes a small translator that reads an MFM model and produces a rule base for performing the fault diagnosis algorithm with a standard backward chaining system. Each goal is translated into one rule, and each function into two, handling the cases when the function is working and failed, respectively. These rule bases were used to give a rough estimate of the complexity of the MFM models and the speed of the algorithms compared to that of standard rule-based systems.

The relative difference in efficiency between the models is probably due to their different size. For the smaller models, some initializing and bookkeeping operations affect the execution times.

It should be noted that the Guardian model poses a test with significant difficulties. It contains numerous loops in the means-end dimension, (i.e., functions on one level depends on sub-functions, and these in turn depend on the original functions), and most of the flow structures consist of loops, (since most of the flows in the human body are closed loops). The algorithms can handle all the classically difficult cases of feedback, recirculation loops, and circular dependencies without problems, though.

15. An Overview of Related Work

The main contributions to MFM have been made by Morten Lind and his group. Lind [73, 79, 80] describes the basics of MFM, while [74, 76, 77] contains Lind's suggestion for a diagnostic system. Lind has also treated real-time diagnosis, [75], and design of operator interfaces, [72]. Lind's group has developed a graphical interface, [24, 94, 95], a STRIPS planning system, [61, 62], a fault diagnosis system for ship engines [45], and a system for alarm analysis and fault diagnosis, [17]. [78] contains a comparative discussion of MFM and other types of models.

MFM has also been used in nuclear safety research [11, 20], in operator interfaces for fault diagnosis, [23], for constructing COGSYS diagnostic systems, [108–111], for fault diagnosis in process industry, [124, 125], and in intelligent man-machine systems for nuclear plants, [87].

The methods described in this article has been presented in earlier papers; measurement validation in [65], alarm analysis in [64], and fault diagnosis in [66]. They have all been thoroughly described in the Doctor's thesis [67], which is the basis for this article. The report [68] contains more detailed descriptions of the implementation of the algorithms, while [69] reports on the C implementation.

Model-Based Diagnosis

The most thorough work in model-based diagnosis has arguably been done in research areas related to *qualitative physics*. The main bulk of the work of the AI community has concerned diagnosis of analog and digital electronic circuits, but there has also been applications in neurophysiology, hydraulic systems, process industry, and other domains, see [37].

Since automatic reasoning usually does not need all 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. Common approaches are to use representations based on, e.g., logics, constraints, or directed graphs. Sussman and Steele, [116], describes a system based on hierarchical constraints and consistency checking. Davis, [18], describes the use of both structural and functional models in diagnosis. The area of qualitative physics is described in [30] and [126]. A good overview of model-based diagnosis is given in [37]. [19] also gives a good overview, while [122] gives a more theoretical overview and relates to standard expert system techniques.

The theoretical foundation for several of these diagnostic methods is Reiter's algorithm, [35, 107]. A model of physical structure and component behavior is used to generate a description of the target system in logic formulae. Together with information about measurements, Reiter's algorithm is then used to compute the minimal set of possible hypotheses that is sufficient to explain the current, (failed), state of the system.

The method outlined above has been implemented in a domain independent architecture called the General Diagnostic Engine, (GDE), [54]. GDE uses an assumption-based truth maintenance system, (ATMS), [50, 51], and is a direct, incremental implementation of Reiter's algorithm.

Most of the results in this area concern digital circuits, but analog circuits and circuits containing components with internal states have also been addressed. There are also probabilistic approaches. All of this is excellently covered in [37].

Other important approaches to qualitative reasoning have been taken by de Kleer and Brown, [53], which points out the need for both topological and functional models and formulates a theory based on confluences, by Forbus, [30], which describes a modeling method starting with a physical description of a system, giving a set of constraints, and using the *process* concept to model behavior, and by Kuipers, [58–60], describing a qualitative simulation method. The system starts with a set of qualitative constraints and an initial state, and can predict the set of possible futures for the system. Dvorak's project, [25, 26] describe the MIMIC fault diagnosis system, based on the QSIM language for qualitative simulation, [58–60]. The Inc-Diagnose system, [92], also uses a QSIM representation. The DATMI program, [22], uses a qualitative representation to maintain a set of measurement interpretations, and can diagnose sensor failures by tracing dependencies. This can be compared with the measurement validation algorithm presented in this article, although there are considerable differences.

A Comparison

It is important to note that MFM differs from the "classical" kinds of model-based diagnosis, as exemplified by Reiter's algorithm and GDE, in the typical target domains as well as in the basic problem formulation of the diagnostic task and in the results produced. The main contribution of MFM is that it extends the model-based research with new methods, well-suited for application in process industry and control systems.

The classical AI version of model-based diagnosis is clearly geared towards diagnosis of electronic circuits and similar domains. The models describe how components are connected into a physical structure, and the behavior of the more or less complex components are described by analog or discrete equations or constraints. From faulty behavior of the entire circuit, the diagnosis tries to isolate faulty components, often by using varying input signals, i.e., tests.

MFM only handles a special class of behaviors; those which can be described by a small set of very abstract flow functions. Furthermore, it is assumed that the failure state of each flow function can be observed from direct measurements or tests, and a diagnosis consists of a description of the fault state of the whole target system, *and* the casual reasons for this fault state. MFM is geared towards systems in process and nuclear industry and similar domains. For example, both the alarm analysis and fault diagnosis algorithms assume that there is a known desired state of the process. On the other hand, the use of varying inputs is quite often not possible in these domains.

It is interesting to note, though, that the project of Walseth *et al* uses MFM to generate input to Reiter's algorithm, [124–125]. This is thus a very interesting effort with the goal of joining MFM and more classical model-based diagnosis.

Another difference is that in MFM, the concept of goals is central, while most classical approaches are not explicitly concerned with teleology. A

notable exception is [49], where a teleological analysis is applied to electronic circuits. In this domain, de Kleer shows how the purposes of components can be obtained from an analysis of structure and function. However, in a more general and unconstrained case, this is not possible. Qualitative physics has proposed the principle of *no function in structure*, i.e., it is impossible to conclude what function a component has from observation of its physical properties alone. In MFM, this principle is observed, and there is the equivalent principle that the goals of a system cannot be concluded solely from an analysis of its functions. For example, with no additional information, it is not possible to tell whether the purpose of a telephone is to enable electro-mechanical communication or to stop a door from closing.

An interesting question is to what extent MFM models are equivalent to other qualitative and quantitative models, and whether it would be possible to generate MFM models automatically from, say, a structure and behavior, QSIM, QPT, or other model-based representation. As with bond graphs, the answer is that most of these representation could provide the flow information needed for MFM, while they may not be sufficient in describing the means-end information, i.e., to define the goals of the target system.

An important and quite interesting possibility is that Forbus' Qualitative Process Theory, (QPT), [30], and similar representations may indeed provide teleological information in the *process* concept. If so, it may be possible to devise a translation between the two types of models. This is still a question open to research, however.

The flow descriptions of MFM are less detailed than those of other models. Thus, these other models might be translated into MFM, but MFM flows will not suffice to automatically produce other models. In the dimension of more or less abstraction, MFM is at the most abstract end. However, it should be clear from the results reported in this article, that it can still provide valuable diagnostic results. The idea that abstraction yields efficiency is also presented in Hamscher's XDE, [36], a program that can diagnose complex circuits by using abstractions of behavior.

It is also worth noting that, although no comparative study is available, the construction of MFM models most probably demands a *lesser* effort than many other model-based approaches. This tentative conclusion is based on the author's experiences from the project "Knowledge-Based Real-Time Control Systems," where models of Steritherm were built using quantitative simulation equations for the G2 simulator, quantitative constraint equations for DMP, and qualitative causal graphs for MIDAS. All these tasks demanded efforts larger than that needed to construct the MFM model of Steritherm, mainly because MFM is more abstract than the behavior models used, and it does not demand any detailed knowledge such as equations, constraints, etc.

A practical difference between MFM and most other model-based techniques is that MFM methods can be very efficient. The MFM algorithms presented in this article perform diagnostic tasks in the time scale of microseconds, while a typical system based on Reiter's algorithm is quite slow. Indeed, computations may become intractable for systems with more than a few tenths of components. In [52], de Kleer shows how a GDE system can increase its efficiency by several orders of magnitude, by concentrating the diagnosis to the probably faulty sections of the circuit, and to focus on the most probable faults only. This algorithm may perform poorly for certain symptoms, though, and, (temporarily ignoring the difficulties in comparing two very different algorithms), it is still orders of magnitude slower than the MFM algorithms.

A fuller overview of related work is given in Appendix A.

16. Conclusions

The article has presented three newly invented and implemented diagnostic methods for use with multilevel flow models, MFM. The methods use MFM as a database and performs measurement validation, alarm analysis, and fault diagnosis. They have been implemented in G2, C, and Common Lisp and successfully tested on several processes. The search algorithms are all very efficient and work in real-time, and their sensitivity to large scaling of models is at worst linear. Together with implemented toolboxes and demonstrations, the project shows a good example of the usefulness and power of means-end models.

17. Appendix A: An Overview of Related Work

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

	<i>MFM</i>	<i>Functional</i>	<i>Qualitative</i>	<i>Quantitative</i>	<i>Integrated</i>
<i>General</i>	Lind De Kjær-Hansen		de Kleer Forbus Kuipers Woods	Isermann Frank Woods	Årzén Krijgsman Crespo
<i>Validation</i>	Larsson Creutzfeldt		DeCoste	Mah	
<i>Alarms</i>	Larsson	Modarres Padalkar	Kramer	Lees	
<i>Diagnosis</i>	Lind Creutzfeldt Sassen Larsson Walseth Jørgensen	Modarres Padalkar Allen	Davis Reiter de Kleer Dvorak Ng	Petti	Viña Struss Mariño
<i>Planning</i>	Larsen	Tomita			
<i>Presentation</i>	Lind Duncan Businaro Monta Larsson	Rasmussen Modarres			
<i>Simulation</i>		Chérury Bond graphs	Kuipers		

Figure 26. 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 26.

Projects Using MFM

Lind. Morten Lind is the creator of MFM. The most important documents about MFM are [73, 79, 80], where the basic ideas, the syntax, and semantics are defined. These reports give an introduction to the background of MFM,

and works as the definition of Lind's current version. Some examples are also given. [71] is the original paper of MFM.

Lind's solution for the data structures and diagnostic algorithms to be used under the MFM top layer is described in [74, 76, 77]. The implementation is done in Smalltalk 80. Together with a graphical interface for building MFM graphs, also written in Smalltalk, [24, 94, 95], this forms the main effort of Lind's group.

De et al. Westinghouse Corporation used MFM to perform a functional analysis of a new control room concept for a pressurized water reactor in the early 1980s, [20].

Kjr-Hansen. Kjær-Hansen has used MFM in the development of models of decision-making processes, [48].

Measurement Validation

Larsson. The only MFM method that treats measurement validation as a separate problem is the one described in Section 9 of this article.

Creutzfeldt. The project described in [17] 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 Section 10 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 [75], where it is argued that the structure of MFM models is well suited for fault diagnosis under hard real-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*, [10, 21].

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

Jrgensen. The thesis of Jørgensen [45] treats the problem of building general MFM models for diagnosis of ship engines, thereby aiding supporting reuse of modeling knowledge.

Sassen et al. The Dutch project PERFECT uses a preprocessor to translate MFM models into COGSYS programs for diagnosis, [108–111]. 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 Section 11 of this article. 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, [14]. 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 [105, 106, 117, 118].

Larsson. Section 11 of this article 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, [124, 125]. One contribution is the idea of connecting MFM goal satisfaction to tests on quantitative state variables in the functions. The MFM model is then used to produce input to Reiter's algorithm, which is the diagnostic vehicle for this project.

Planning

Larsen. One project of Lind's group uses MFM to control the production of STRIPS plans for startup of plants, [61, 62]. 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 [28].

Presentation

Lind. Lind has also treated presentation of means-end information and the design of operator interfaces, [72]. 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, [81], 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, [24, 94, 95]. Lind has also cooperated with CEC-JRC Ispra in Italy.

Duncan and Praetorius. Duncan and Praetorius [23] 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 Praetorius developed a graphical presentation system for MFM. Sassen has later performed similar experiments.

Businaro. Businaro *et al* performed some studies of man-machine interfaces using MFM, in the mid 1980s, [11].

Monta et al. Toshiba is developing a supervisory control system for boiling water nuclear reactors. The system is based on design principles from cognitive science and uses MFM to represent plant knowledge for diagnosis. The system is scheduled to appear in the next generation of operator room software, and this may very well be the most comprehensive and interesting project so far.

Larsson. Some ideas about and examples of presentation of means-end information are given in [67]. 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.

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, [86]. The system is written in Common Lisp 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, [97]. 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, [13]. 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.*, [97], 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 [1].

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, [120].

Another system is specifically aimed at batch processes, [121]. 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 [102, 103]. 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 [102] for an extensive overview. [104] 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, [47]. 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.

Cheruy 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 [67]. For readings on CAMBIO, see [12, 27, 88]. 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*, [46, 98, 119]. 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, [119].

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 condensators 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.

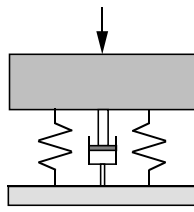


Figure 27. A simple mechanical system with a mass, a spring, and friction. The mass is seen as a point mass, and the excitation is a force independent of velocity. From [119].

With these building blocks, it is possible to describe electrical, mechanical, thermodynamical, and other systems. A mechanical system is shown in Figure 27.

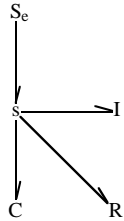


Figure 28. A bond graph of the system in Figure 27. The excitation is S_e . The spring is modeled as a capacity element, the friction as a resistance, and the mass as an inertia element. The connection is a series junction, as the velocities are equal and the different forces added. From [119].

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 28.

An augmented bond graph of the system is shown in Figure 29. Here the causality in the system is also described. The forces and velocities are shown in the graph.

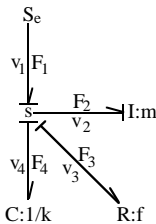


Figure 29. An augmented bond graph of the system. The force of the effort source acts on the mass which responds with the velocity v_2 , but after deducting the spring and friction forces, which are functions of the velocity. From [119].

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, and these flows may be used when building MFM models.

Projects Using Qualitative Behavioral Models

The most thorough work in model-based diagnosis has arguably been done in research areas related to *qualitative physics*. This area has already been overviewed in Section 15. Here, some further projects will be described under the appropriate headings.

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 [29, 96]. An alternative implementation in G2 of part of the MIDAS system is described in [93], 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.

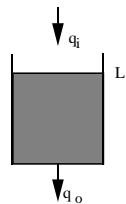


Figure 30. A small gravity tank. The level, L , of the tank is controlled by the inflow, q_i , and outflow, q_o , and there is a flow resistance, R , in the outflow pipe.

Consider the gravity tank in Figure 30. 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 31.

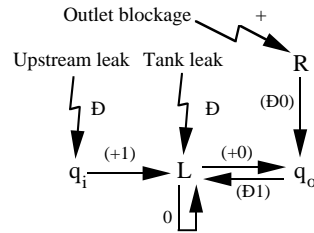


Figure 31. A Signed Directed Graph describing the gravity tank. The variables q_i , L , q_o , and R are shown, together with arcs that describe how a change in one of the variables will affect the others. From [93].

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 32.

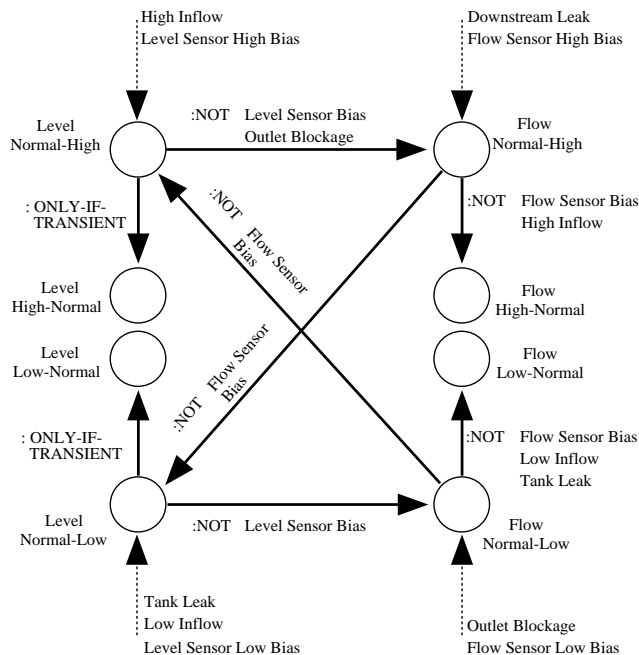


Figure 32. An event graph describing the gravity tank. There are eight events and four root causes. This is the data structure that MIDAS uses on-line. From [93].

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 33.

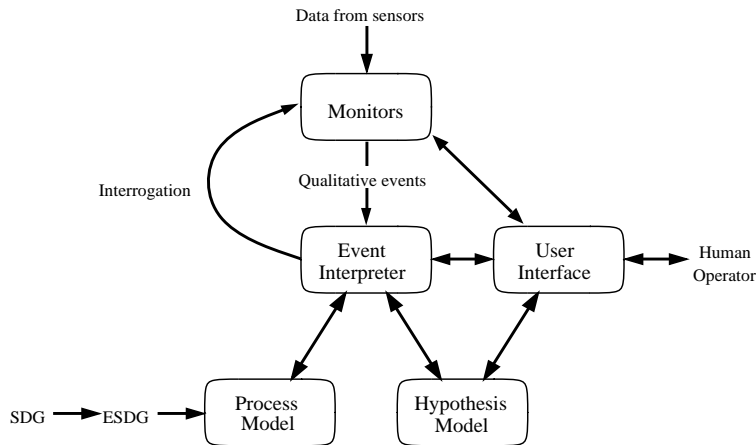


Figure 33. The architecture of the MIDAS on-line system. The input signals are sent to monitors, which turns them into qualitative events. The event interpreter receives all events and uses them to construct an on-line event graph, which represents the current situation in the process. From [93].

Fault Diagnosis

Several projects within AI have used model-based reasoning to perform fault diagnosis. This effort has been described earlier, in Section 15. Here, some projects with closer connections to MFM and process control domains will be described.

Dvorak and Kuipers. The MIMIC system is based on the QSIM modeling language, [25, 26]. 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 [92] uses a set of QSIM qualitative states and a new version of the diagnostic algorithm of Reiter, [107], 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, [58–60]. 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).

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 [43]. Fault detection with classical methods is given excellent overviews in [32–34]. In the Hybrid Phenomena Theory, (HPT), Woods describes a combination of qualitative and quantitative models, [127–129]. HPT is an attempt to combine state space models with a symbolic framework derived from the Qualitative Process Theory, (QPT), [30]. 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 [82] 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 [82]. 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, [42], 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,

[2], and nodal tests, [83], 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, [84], 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.

Alarm Analysis

There are many methods for alarm analysis based on quantitative techniques. See [70] for an overview.

Fault Diagnosis

Petti. The Diagnostic Model Processor method has been developed by Tom Petti at the University of Delaware, [99–101]. 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. [101] describes the DMP methodology, while [100] shows an implementation and some examples of the use of the method.

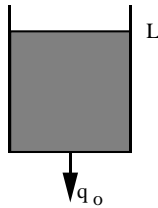


Figure 34. A tank with a gravity outflow. The outflow depends on the level as $q_{out} = \alpha\sqrt{2gh}$. Both the level and the outflow are measured, and can thus be used in a model equation.

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 34.

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 35.

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.

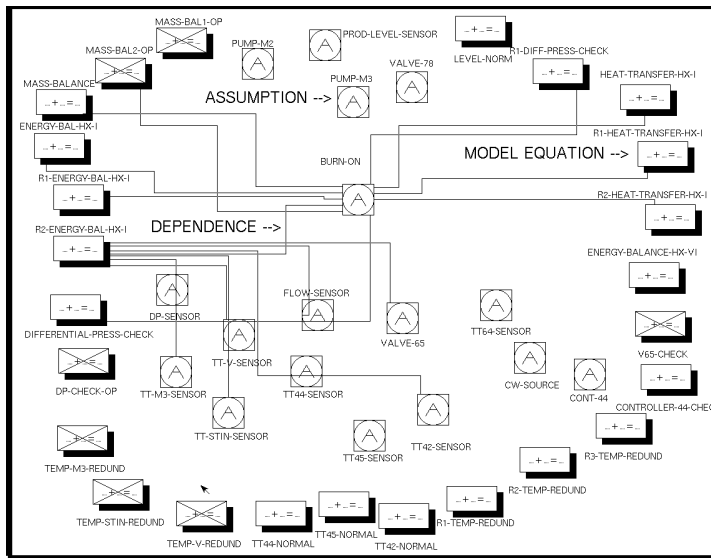


Figure 35. A DMP model of Steritherm. The picture contains the model equations and the assumptions, and some of the connections are also shown.

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.

Arzen et al. The “Knowledge-Based Real-Time Control System” project described in [3–9, 63], 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 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, [44, 55–57], 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, [15, 16], 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

Viña and Hayes-Roth. Viña and Hayes-Roth [123] 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.

Viña and Hayes-Roth has tested this system on two control systems, RCIC, an auxiliary subsystem for a power plant cooling system, [41], and GUARDIAN, a system for patient monitoring in a surgical intensive-care unit, [40]. 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, [38, 39]. 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, [91].

Struss. Struss, [113–115], 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), [50, 51], 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, [85], 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.

18. Appendix B: Fault Diagnosis Code

Here follows a listing of C code for the fault diagnosis algorithm. This code has actually been copied from the MFM Toolbox in C. Some variables and statements concerned with bookkeeping, updating the graphics, and the measurement validation and alarm analysis algorithms have been omitted, though.

```

/*--MFM Fault Diagnosis-----*/
/*                               */
/* Author: Jan Eric Larsson      */
/* Knowledge Systems Laboratory  */
/* Stanford University           */
/* 701 Welch Road, Building C, Palo Alto, CA 94304, USA */
/* Phone: +1 415 723 0948, E-mail: Larsson@KSL.Stanford.Edu */
/*                               */
/*--Global Definitions-----*/

struct GoalStruct {
    int No, State, Conditions;
    struct NetworkStruct * Network;
    struct ManagerStruct * Manager;
    struct ConditionStruct * Condition [MAXCONDS];
};

struct NetworkStruct {
    int No, State, Functions, Goals;
    struct FunctionStruct * Function [MAXFUNCS];
    struct GoalStruct * Goal [MAXGOALS];
};

struct ManagerStruct {
    int No, State, Functions;
    struct FunctionStruct * Function [MAXFUNCS];
    struct GoalStruct * Goal;
};

struct FunctionStruct {
    int No, Type, MType, State, Ins, Outs, Conditions;
    struct FunctionStruct * In [MAXLINKS], * Out [MAXLINKS];
    struct ConditionStruct * Condition [MAXCONDS];
    struct NetworkStruct * Network;
    struct ManagerStruct * Manager;
};

struct ConditionStruct {
    int No, State;
    struct GoalStruct * Goal;
    struct FunctionStruct * Function;
};

/*--Fault Diagnosis-----*/

/*
 * These procedures perform the depth-first search downwards in the
 * MFM graph, and sets the state values on the way up. DiagnoseGoal
 * performs the diagnostic search for a goal object. It checks that
 * the state of the goal is unknown, sets the state to undecided, (to
 * avoid loops in the means-end dimension), and calls DiagnoseNetwork

```

```

* and DiagnoseManager, (if a manager exists). Finally, the state is
* updated according to the results returned.
*
*/

int DiagnoseGoal (Goal)
struct GoalStruct * Goal;
{
    if (Goal -> State == UNKNOWN) {
        Goal -> State = UNDECIDED;
        Goal -> State = DiagnoseNetwork (Goal -> Network);
        if (Goal -> Manager != NULL) {
            if (DiagnoseManager (Goal -> Manager) == FAILED) {
                Goal -> State = FAILED;
            }
        }
    }
    return (Goal -> State);
}

/*
* DiagnoseNetwork and DiagnoseManager perform the downward search for
* network and manager objects. They simply go through the functions
* in the object, calling DiagnoseFunction for each, and then return
* working or failed, depending on the results returned.
*
*/

int DiagnoseNetwork (Network)
struct NetworkStruct * Network;
{
    int n;

    if (Network -> State == UNKNOWN) {
        Network -> State = UNDECIDED;
        Network -> State = DiagnoseFunction (Network -> Function [0]);
        for (n = 1; n < Network -> Functions; n ++) {
            if (DiagnoseFunction (Network -> Function [n]) == FAILED) {
                Network -> State = FAILED;
            }
        }
    }
    return (Network -> State);
}

int DiagnoseManager (Manager)
struct ManagerStruct * Manager;
{
    int n;

    if (Manager -> State == UNKNOWN) {
        Manager -> State = UNDECIDED;
        Manager -> State = DiagnoseFunction (Manager -> Function [0]);
        for (n = 1; n < Manager -> Functions; n ++) {
            if (DiagnoseFunction (Manager -> Function [n]) == FAILED) {
                Manager -> State = FAILED;
            }
        }
    }
    return (Manager -> State);
}

/*
* DiagnoseFunction performs the diagnostic search for a function object.
* First, it reads its state from the monitor array, which corresponds

```

```

* to a sensor reading. If the value is unknown or failed the search
* must be continued among the conditions of the function, but when it
* is normal, further search of the current branch can be skipped.
*
*/

int DiagnoseFunction (Function)
struct FunctionStruct * Function;
{
    int n;

    Function -> State = Monitor [Function -> No];
    if (Function -> State != WORKING && Function -> Conditions > 0) {
        for (n = 0; n < Function -> Conditions; n++) {
            if (DiagnoseCondition (Function -> Condition [n]) == FAILED) {
                Function -> State = FAILED;
            }
        }
    }
    return (Function -> State);
}

/*
* DiagnoseCondition performs the diagnostic search for a condition
* object. It simply propagates the search downwards to the next goal.
*
*/

int DiagnoseCondition (Condition)
struct ConditionStruct * Condition;
{
    Condition -> State = DiagnoseGoal (Condition -> Goal);
    return (Condition -> State);
}

```

19. Acknowledgements

I would like to thank my supervisor, professor Karl Johan Åström, and the originator of MFM, professor Morten Lind, and Doctor Karl-Erik Årzén for inspiration and support. This project has been influenced by the Swedish IT4 project “Knowledge-Based Real-Time Control Systems,” and I also wish to thank the members of the project group. The project has been supported by the IT4 project no. 3403 and the TFR project no. 92–956.

The Guardian project is taking place at Stanford University, and I would like to thank Doctor Barbara Hayes-Roth and the staff at the Knowledge Systems Laboratory for valuable help and support. The postdoctoral visit is supported by grants from the Swedish Research Council for Engineering Sciences, the Swedish Institute, the Royal Physiographic Society in Lund, and the Nils Hörjel Research Fund at Lund Institute of Technology.

Finally, I would also like to thank the anonymous referees for constructive suggestions that helped to clarify the paper.

20. References

1. 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.
2. ALMASY, G. A. and T. SZTANO (1975): *Problems of Control and Information Theory*, **4**, 1, 57–69.

3. ÅRZÉN, K. E. (1989): "Knowledge-Based Control Systems: Aspects on the Unification of Conventional Control Systems and Knowledge-Based Systems," *Proceedings of the 1989 American Control Conference*, Pittsburgh, Pennsylvania.
4. ÅRZÉN, K. E. (1990): "Knowledge-Based Control Systems," *Proceedings of the 1990 American Control Conference*, San Diego, California.
5. ÅRZÉN, K. E. (1992): "A Model-Based Control System Concept," Technical report, TFRT-3213, Department of Automatic Control, Lund Institute of Technology, Lund.
6. 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.
7. 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.
8. ASEA BROWN BOVERI, SATTCONTROL, and DEPARTMENT OF AUTOMATIC CONTROL, LUND INSTITUTE OF TECHNOLOGY (1990): *Knowledge-Based Real-Time Control Systems—IT4 Project: Phase 1*, Studentlitteratur, Lund.
9. ASEA BROWN BOVERI and DEPARTMENT OF AUTOMATIC CONTROL, LUND INSTITUTE OF TECHNOLOGY (1991): *Knowledge-Based Real-Time Control Systems—IT4 Project: Phase 2*, Studentlitteratur, Lund.
10. BODDY, M. and T. DEAN (1989): "Solving Time Dependent Planning Problems," *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, pp. 979–984.
11. BUSINARO, T., A. DI LORENZO, G. B. MEO, M. I. RABBANI, and E. RUBINO (1985): "An Application of MFM Method for Nuclear Plant State Identification," *Proceedings of the Enl. Halden Progr. Group Meeting on Computerized Man-Machine Communication*, Göteborg, Sweden.
12. 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.
13. CHUNG, D. T. and M. MODARRES (1989): "GOTRES: An Expert System for Fault Detection and Analysis," *Reliability Engineering and System Safety*, **24**, 113–137.
14. COGSYS (1990): *COGSYS Manual*, COGSYS Ltd., Salford, United Kingdom.
15. 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.
16. 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.
17. 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.
18. DAVIS, R. (1984): "Diagnostic Reasoning Based on Structure and Behavior," *Artificial Intelligence*, **24**, 1, 347–410.
19. 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.
20. DE, M. K., J. A. RUMANCIK, A. J. IMPINK, and J. R. EASTER (1982): "A Functional Design Approach to PWR Safety," *Proceedings of the International Meeting on Thermal Nuclear Reactor Safety*, Chicago, Illinois.
21. DEAN, T. and M. BODDY (1988): "An Analysis of Time Dependent Planning," *Proceedings of the 6th National Conference on Artificial Intelligence*, pp. 49–54.

22. DECOSTE, D. (1991): "Dynamic Across-Time Measurement Interpretation," *Artificial Intelligence*, **51**, 1-3, 273-341.
23. 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.
24. DUSCHEK, J. (1991): "Syntax Analysis of Modeling Languages and a Knowledge-Based System for Modeling," Master's thesis, Institute of Automatic Control Systems, Technical University of Denmark, Lyngby, Denmark, in Danish.
25. 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.
26. DVORAK, D. L. and B. KUIPERS (1991): "Process Monitoring and Diagnosis," *IEEE Expert*, June 1991, 67-74.
27. FARZA, M. and A. CHÉRUY (1991): "CAMBIO: A Software for Modeling and Simulation of Bioprocesses," *Computer Applications in the Biosciences*, **7**, 3, 327-336.
28. 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.
29. FINCH, F. E. (1989): *Automated Fault Diagnosis of Chemical Process Plants Using Model-Based Reasoning*, Doctor's thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts.
30. FORBUS, K. D. (1984): "A Qualitative Process Theory," *Artificial Intelligence*, **24**, 85-168.
31. 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.
32. 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.
33. 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.
34. 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.
35. GREINER, R., B. A. SMITH, and R. W. WILKERSON (1989): "A Correction to the Algorithm in Reiter's Theory of Diagnosis," *Artificial Intelligence*, **41**, 1, 79-88.
36. HAMSCHER, W. (1991): "Modeling Digital Circuits for Troubleshooting," *Artificial Intelligence*, **51**, 1-3, 223-271.
37. HAMSCHER, W., L. CONSOLE, and J. DE KLEER, (Eds.) (1992): *Readings in Model-Based Diagnosis*, Morgan-Kaufmann Publishers, Inc., San Mateo, California.
38. HAYES-ROTH, B. (1985): "A Blackboard Architecture for Control," *Artificial Intelligence*, **26**, 3, 251-321.
39. HAYES-ROTH, B. (1990): "Architectural Foundations for Real-Time Performance in Intelligent Agents," *Real Time Systems*, **2**, 1, 2.
40. HAYES-ROTH, B., R. WASHINGTON, D. ASH, R. HEWETT, A. COLLINOT, A. VIÑA, AND A. SEIVER (1992): "Guardian: A Prototype Intelligent Agent for Intensive-Care Monitoring," *Artificial Intelligence in Medicine*, **4**, 165-185.
41. HEWETT, R. (1990): "ICE Manual," BB1 Internal report.
42. HIMMELBLAU, D. M. (1987): "Interval Analysis as a Tool for Data Rectification," *Proceedings of the AIChE Annual Meeting*, Houston, Texas.
43. ISERMANN, R. (1984): "Process Fault Detection Based on Modeling and Estimation Methods—A Survey," *Automatica*, **20**, 4, 387-404.

44. JAGER, R. (1990): "Direct Real-Time Control using Knowledge-Based Techniques," *Proceedings of the ESS '90 Intelligent Process Control Design*, Ghent, Belgium.
45. JØRGENSEN, S. S. (1993): *Generic MFM Models for Use in Fault Diagnosis of Ship System's Machinery*, Ph. D. thesis, Institute of Automatic Control Systems, Technical University of Denmark, Lyngby, Denmark.
46. KARNOPP, D. and R. ROSENBERG (1975): *System Dynamics: A Unified Approach*, John Wiley and Sons, New York.
47. 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.
48. KJÆR-HANSEN, J. (1992): *Decision Structures of Multi-Agent Systems*, Ph. D. thesis, Institute of Automatic Control Systems, Technical University of Denmark, Lyngby, Denmark.
49. DE KLEER, J. (1984): "How Circuits Work," *Artificial Intelligence*, 24, 1–3, 205–280.
50. DE KLEER, J. (1986): "An Assumption-Based TMS," *Artificial Intelligence*, 28, 2, 127–162.
51. DE KLEER, J. (1986): "Extending the ATMS," *Artificial Intelligence*, 28, 2, 163–196.
52. DE KLEER, J. (1991): "Focusing on Probable Diagnoses," *Proceedings of the 9th National Conference on Artificial Intelligence*, Anaheim, California, pp. 842–848.
53. DE KLEER, J. and J. S. BROWN (1984): "A Qualitative Physics Based on Confluences," *Artificial Intelligence*, 24, 1–3, 7–83.
54. DE KLEER, J. and B. C. WILLIAMS (1987): "Diagnosing Multiple Faults," *Artificial Intelligence*, 32, 1, 97–130.
55. KRIJGSMAN, A. J. and R. JAGER (1992): "DICE: a Real-Time Toolbox," *Preprints of the 1992 IFAC/IFIP/TMACS International Symposium on Artificial Intelligence in Real-Time Control*, Delft University of Technology, Delft, the Netherlands, pp. 637–641.
56. 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.
57. 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.
58. KUIPPERS, B. J. (1984): "Commonsense Reasoning About Causality: Deriving Behavior From Structure," *Artificial Intelligence*, 24, 169–204.
59. KUIPPERS, B. J. (1986): "Qualitative Simulation," *Artificial Intelligence*, 29, 289–338.
60. KUIPPERS, B. J. (1989): "Qualitative Reasoning: Modeling and Simulation with Incomplete Knowledge," *Automatica*, 25, 4, 571–585.
61. LARSEN, M. N. (1990): "Strips as a Planning Method within Abstractions and MFM Modeling," Technical report, Institute of Automatic Control Systems, Technical University of Denmark, Lyngby, Denmark.
62. LARSEN, M. N. (1993): "Modeling Start-Up Tasks Using Functional Models," Final report, Project 4937–92–08–ED ISP DK, CRC JSP Ispra.
63. LARSSON, J. E. (1990): "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.
64. LARSSON, J. E. (1991): "Model-Based Alarm Analysis Using MFM," *Proceedings of the 3rd IFAC International Workshop on Artificial Intelligence in Real-Time Control*, Rhonert Park, Sonoma, California.
65. LARSSON, J. E. (1992): "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.

66. LARSSON, J. E. (1992): "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.
67. LARSSON, J. E. (1992): *Knowledge-Based Methods for Control Systems*, Doctor's thesis, TFRT-1040, Department of Automatic Control, Lund Institute of Technology, Lund.
68. LARSSON, J. E. (1992): "An MFM Toolbox," Technical report, TFRT-7493, Department of Automatic Control, Lund Institute of Technology, Lund.
69. LARSSON, J. E. (1994): "Hyperfast Algorithms for Model-Based Diagnosis," *Proceedings of the IEEE/IFAC Joint Symposium on Computer-Aided Control System Design*, Tucson, March 1994.
70. LEES, F. P. (1983): "Process Computer Alarm and Disturbance Analysis: Review of the State of the Art," *Computers and Chemical Engineering*, **7**, 6.
71. 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.
72. 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.
73. LIND, M. (1990): "Representing Goals and Functions of Complex Systems—An Introduction to Multilevel Flow Modeling," Technical report, 90-D-38, Institute of Automatic Control Systems, Technical University of Denmark, Lyngby, Denmark.
74. LIND, M. (1990): "Abstractions Version 1.0—Descriptions of Classes and Their Use," Technical report, 90-D-380, Institute of Automatic Control Systems, Technical University of Denmark, Lyngby, Denmark.
75. LIND, M. (1990): "An Architecture for Real-Time MFM Diagnosis," Technical report, Institute of Automatic Control Systems, Technical University of Denmark, Lyngby, Denmark.
76. LIND, M. (1991): "Abstractions for Modeling of Diagnostic Strategies," *Proceedings of the IFAC Workshop on Computer Software Integrating AI/KBS Systems in Process Control*, Bergen, Norway.
77. LIND, M. (1991): "On the Modeling of Diagnostic Tasks," *Proceedings of the Third European Conference on Cognitive Science Approaches to Process Control*, Cardiff, Wales.
78. LIND, M. (1992): "A Categorization of Models and Its Application for the Analysis of Planning Knowledge," *Proceedings of the POST ANP '92 Conference on Human Cognitive and Cooperative Activities in Advanced Technological Systems*, Kyoto.
79. LIND, M. (1993): "Functional Architectures for Systems Management and Control. Interactive Planning for Integrated Supervision and Control of Complex Plant.," Final report, Project 4937-92-08-ED ISP DK., CEC JRC Ispra.
80. LIND, M. (1994): "Modeling Goals and Functions of Complex Industrial Plants," *Applied Artificial Intelligence*, **8**, 2, 259-283.
81. LIND, M., E. HARDER, H. JENSEN, and S. AGGER (1987): "Systembeskrivelse og præsentation 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.
82. MAH, R. S. H. (1990): *Chemical Process Structures and Information Flows*, Butterworths, Boston, Massachusetts.
83. 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.
84. MAH, R. S. H. and A. C. TAMHANE (1982): "Detection of Gross Errors in Process Data," *AIChE Journal*, **28**, 828-830.
85. 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.

86. MODARRES, M. and T. CADMAN (1986): "A Method of Alarm System Analysis for Process Plants," *Computers and Chemical Engineering*, **10**, 6, 557–565.
87. MONTA, K., J. TAKIZAWA, Y. HATTORI, T. HAYASHI, N. SATO, J. ITOH, A. SAKUMA, and E. YOSHIKAWA (1991): "An Intelligent Man-Machine System for BWR Nuclear Power Plants," *Proceedings of the AI91—Frontiers in Innovative Computing for the Nuclear Industry*, Jackson, Wyoming.
88. 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.
89. 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*, Vol 6, München, pp. 274–281.
90. 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.
91. MURDOCK, J. L. and B. HAYES-ROTH (1991): "Intelligent Monitoring and Control of Semiconductor Manufacturing Equipment," *IEEE Expert*, December 1991, 19–31.
92. NG, H. T. (1991): "Model-Based, Multiple-Fault Diagnosis of Dynamic, Continuous Physical Devices," *IEEE Expert*, December 1991, 38–43.
93. NILSSON, A. (1991): *Qualitative Model-Based Diagnosis—MIDAS in G2*, Master's thesis TFRT-5443, Department of Automatic Control, Lund Institute of Technology, Lund.
94. OSMAN, A. (1990): "The Interface Substrate," Technical report, Institute of Automatic Control Systems, Technical University of Denmark, Lyngby, Denmark.
95. OSMAN, A. (1992): *Graphical Control Environment (Grace)*, Ph. D. thesis, Institute of Automatic Control Systems, Technical University of Denmark, Lyngby, Denmark.
96. OYELEYE, O. O. (1989): *Qualitative Modeling of Continuous Chemical Processes and Applications to Fault Diagnosis*, Doctor's thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts.
97. PADALKAR, S., G. KARSAI, C. BIEGL, J. SZTIPANOVITS, K. OKUDA, and N. MIYASAKA (1991): "Real-Time Fault Diagnosis," *IEEE Expert*, June 1991, 75–85.
98. PAYNTER, H. M. (1961): *Analysis and Design of Engineering Systems*, MIT Press, Cambridge, Massachusetts.
99. PETTI, T. F. (1992): *Using Mathematical Models in Knowledge-Based Control Systems*, Ph. D. Dissertation, University of Delaware, Newark, Delaware.
100. PETTI, T. F. and P. S. DHURJATI (1991): "Object-Based Automated Fault Diagnosis," *Chemical Engineering Communications*, **102**, 107–126.
101. 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.
102. RASMUSSEN, J. (1986): *Information Processing and Human Machine Interaction: An Approach to Cognitive Engineering*, North Holland, New York.
103. 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.
104. RASMUSSEN, J. and M. LIND (1981): "Coping with Complexity," Technical report, Risø National Laboratory, Roskilde, Denmark.
105. VAN DEN REE, R., H. KOPPELAAR, and E. J. H. KERCKHOFFS (1991): "Knowledge Management in Process Modeling. Engineering Systems with Intelligence: Concepts, Tools, and Applications.," *Proceedings of the European Robotics and Intelligent Systems Conference*, Kluwer Academic Publishers, Dordrecht, Nederland, pp. 83–90.
106. VAN DEN REE, R., H. KOPPELAAR, and E. J. H. KERCKHOFFS (1991): "Proposal for Process Modeling," *Proceedings of the IMACS MCTS '91, Modeling and Control of Technological Systems*, Lille, France, pp. 732–737.

107. REITER, R. (1987): "A Theory of Diagnosis from First Principles," *Artificial Intelligence*, **32**, 1, 57–96.
108. SASSEN, J. M. A. (1993): *Design Issues of Human Operator Support Systems*, Doctor's thesis, Faculty of Mechanical Engineering and Marine Technology, Laboratory for Measurement and Control, Delft University of Technology, Delft, Nederland.
109. 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.
110. 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.
111. 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.
112. SHORTLIFFE, E. H. (1976): *Computer Based Medical Consultations: MYCIN*, Elsevier Science Publishers B. V. North-Holland, New York.
113. 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.
114. STRUSS, P. (1992): "Diagnosis as a Process," in Hamscher, W., L. Console, and J. de Kleer, (Eds.): *Readings in Model-Based Diagnosis*, Morgan-Kaufmann Publishers, Inc., San Mateo, California.
115. 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.
116. SUSSMAN, G. J. and G. L. STEELE (1980): "Constraints: A Language for Expressing Almost-Hierarchical Descriptions," *Artificial Intelligence*, **14**, 1, 1–40.
117. 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.
118. 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.
119. THOMA, J. U. (1975): *Introduction to Bond Graphs and Their Applications*, Pergamon Press, New York.
120. TOMITA, S., K. S. HWANG, E. O'SHIMA, and C. MCGREAVY (1989): "Automatic Synthesizer of Operating Procedures for Chemical Plant by Use of Fragmentary Knowledge," *Journal of Chemical Engineering of Japan*, **22**, 4, 364–372.
121. 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.
122. TORASSO, P. and L. CONSOLE (1989): *Diagnostic Problem Solving*, North Oxford Academic, Kogan Page Ltd., London.
123. VIÑA, 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.
124. WALSETH, J. Å. (1993): *Diagnostic Reasoning in Continuous Systems*, Ph. D. thesis, ITK-rapport 1993: 164-W, Division of Engineering Cybernetics, Norwegian Institute of Technology, Trondheim, Norge.

125. 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.
126. WELD, D. S. and J. DE KLEER (Eds.) (1990): *Readings in Qualitative Reasoning about Physical Systems*, Morgan Kaufmann Publishers, Inc., San Mateo, California.
127. 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.
128. WOODS, E. A. (1993): *The Hybrid Phenomena Theory*, Ph. D. thesis, Institute of Engineering Cybernetics, Technical University of Norway, Trondheim, Norway.
129. 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.