CODEN: LUTFD2/(TFRT-5443)/1–58/(1991)

Qualitative Model-Based Diagnosis – MIDAS in G2

Anders Nilsson

Department of Automatic Control
Lund Institute of Technology
September 1991

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

Document name
Master Thesis

Date of issue
September 1991

Document Number
CODEN: LUTFD2/(TFRT–5443)/1-58/(1991)

Author(s)
Anders Nilsson

Supervisor
Claes Rytoft, ABB, Karl-Erik Årzén, LTH

Sponsoring organisation
IT4, NUTEK

Title and subtitle
Qualitative Model-Based Diagnosis – MIDAS in G2

Abstract

This report is the documentation of the master thesis project "Qualitative Model-Based Diagnosis – MIDAS in G2". MIDAS is a fault diagnosis methodology using a qualitative causal process model. It can be used for plant processes working at a nominal steady state.

The process model used for the diagnosis is an event graph, where events concerning the measured variables are causally related. An event is a transition of a variable between two of the qualitative states high, normal and low. During diagnosis such events are detected, and the system compares the detected events to the process model and creates hypotheses about the present faults. In the project, this diagnostic function has been developed in the expert system shell G2.

A structured way of obtaining an event graph for an arbitrary process is presented in the report. Basically, for each process unit a Signed Directed Graph (SDG) is constructed. The SDG expresses the qualitative relations between the variables describing the unit. Then the process SDG is built by connecting the unit SDGs according to the process flowsheet. When the feedback effects have been analyzed, the propagation of any fault can be determined, and the event graph can be constructed.

In the project, an event graph for the sterilization process Steritherm has been created manually and tested against the diagnosis module and a simulation model of Steritherm.

Key words

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title

ISBN

Language
English

Number of pages
58

Recipient's notes

Security classification

1.3 Specifications

This master thesis project can be divided into two parts. Firstly, the online part of the diagnostic methodology in MIDAS has been implemented in G2. The implementation has been done in a way that it can work against any possible event graph for any kind of industrial process. Basically, the diagnosis module follows the behavior of MIDAS as described by Finch (1989).

Secondly, a process model for a part of Steritherm has been developed, and tested against the diagnosis implementation and the existing simulation model of Steritherm in G2. The process model has been developed using the guidelines given by Oyeleye (1989). The process was first described as a signed directed graph (SDG). To deal with feedback effects, the SDG was modified to form an Extended SDG (ESDG). In the last step the ESDG was transformed into an event graph, which serves as the process model used by the diagnosis module.

1.4 Thesis outline

In Chapter 2 the derivation of the process model is described together with the diagnostic methodology of MIDAS. Chapter 3 deals with the implementation of these parts in G2. In Chapter 2 and 3, examples such as a simple tank process, are used to demonstrate process models and diagnostic behavior. The derivation of the Steritherm process model and the results from fault simulations using this model are described in Chapter 4. Chapter 5 contains some of the results and conclusions from the project, including some suggested improvements. A brief description of G2 can be found in Appendix A, and in Appendix B all class definitions in the G2 implementation are listed.

2. MIDAS – Qualitative Fault Diagnosis

2.1 Overview

MIDAS (Model Integrated Diagnosis Analysis System) is a computer program for diagnosing malfunctions in plant processes. It was developed at Massachusetts Institute of Technology by Kramer, Oyeleye and Finch and is fully described by Oyeleye (1989) and Finch (1989). Besides handling the diagnosis on-line, MIDAS also, more or less automatically, creates the process model used for the diagnosis. Since this master thesis project is an implementation of the MIDAS concept and not an exact copy, the word "MIDAS" is mostly referring to the methodology used in the program rather than to the program itself. In fact, only the diagnostic function was implemented, the actual process model was built manually following the guidelines by Oyeleye (1989).

MIDAS belongs to the diagnosis methodologies based on qualitative causal reasoning. The process model is an event graph where different events (e.g. "The level of the tank was NORMAL and is now LOW") and root causes (e.g. "Low inflow") are linked together by causal arcs. There are two types of causal arcs. A local cause link originates at a root cause and terminates at an event and it tells what will be the first symptom of the root cause. A compiled link connects two events and gives information about the propagation of disturbances.

During diagnosis all transitions between the qualitative states of the measured variables are detected. For each transition a copy of the corresponding event in the process model is created. Such a new recorded event is, together with the old ones, matched against the process model to see if they are somehow causally related. If this is the case, they are linked together in an event cluster. This way it is possible to create a hypothesis about what might have caused the detected behavior of the process.

2.2 Model building

To make a diagnosis it is necessary to have an adequate process model. The event graph MIDAS uses for this purpose could be created either from just heuristics or through a deeper knowledge of the process. Since it is hard for anyone to think of every little detail in a large process, the first method might lead to an improper model and probably to incorrect diagnosis. It would be better if one could derive the event graph systematically from basic physical equations on a low level of the process. Algorithms for doing this are presented in Oyeleye (1989) and will be described in this section. From physical equations a Signed Directed Graph (SDG) is built, containing all variables in the system and the qualitative relations between them. To resolve ambiguities due to feedback effects it is possible to add non-physical arcs in the SDG to form an Extended Signed Directed Graph (ESDG). Then the event

graph can be derived from the ESDG by eliminating unmeasured variables and expressing conditions for transitions between qualitative states of the measured variables.

2.2.1 SDG - Signed Directed Graph

The usual way to describe a process mathematically is to set up a system of first-order differential equations:

$$\dot{x}(t) = f(x(t), u(t)) \tag{2.1}$$

where x is the state vector and u is a vector of input variables and interesting process parameters. If the model is linearized around the steady state $x = x_0, u = u_0$, Equation 2.1 becomes:

$$\dot{\hat{x}}(t) = A\hat{x}(t) + B\hat{u}(t) \tag{2.2}$$

with $\hat{x} = x - x_0$ and $\hat{u} = u - u_0$. The signs and magnitudes of the elements in the matrices A and B give information of the behavior of the process near the steady state. Since the equations are based on physical relations it is quite easy to determine the signs of the coefficients, but it usually requires too much effort to get sufficiently exact numerical values. Thus it would have been useful if one could find a way to use just the qualitative information given by the process model.

The Signed Directed Graph (SDG) can be seen as a graphical representation of Equation 2.2 when only the signs of the matrix elements and the variables are taken into account. The state and input variables are represented by nodes which can have the qualitative states 0,- or + depending on if $\hat{x}_i$ (or $\hat{u}_i$) $= 0, \hat{x}_i < 0$, or $\hat{x}_i > 0$. The elements in the matrices A and B correspond to signed arcs initiating at any node and terminating at a node representing a state variable. Consider the i:th state equation (dropping the hats):

$$\dot{x}_i(t) = ... + a_{ij}x_j(t) + ... + b_{ik}u_k(t) + ... \tag{2.3}$$

The arcs from x_j and u_k to x_i bears the signs of a_{ij} and b_{ik} respectively. When $i = j$ one gets an arc initiating and terminating at the same node, a so called self cycle. In order to clear up the SDG as much as possible, some of the arcs are not shown. Zero-signed arcs just have a meaning for self cycles (indicating an integrator) and are otherwise omitted. Also, a negative self cycle attached to a node is not shown since it just indicates that the node is stable.

Along with the sign, an arc might also have a digit attached to itself, indicating the relative delay along the arc. If a variation of a node is transmitted via an arc instantaneously, or with very fast dynamics, this arc will have a zero delay, otherwise the delay is set to one. This can be useful when deriving the event graph to determine the causal order of events. The words fast and slow must be related to the detection time scale. If effects are transmitted faster than the process is sampled, then the transmission is quoted as fast, otherwise it is slow.

Figure 2.1 shows a simple gravity flow tank along with its equations and its SDG. The SDG is used in the following way: to determine the process behavior due to a disturbance, the arcs are followed from the origin of the disturbance to all other reachable nodes in the graph. If, for instance, an upstream leak occurs, then the negative sign at the arc to F_{in} gives that F_{in}

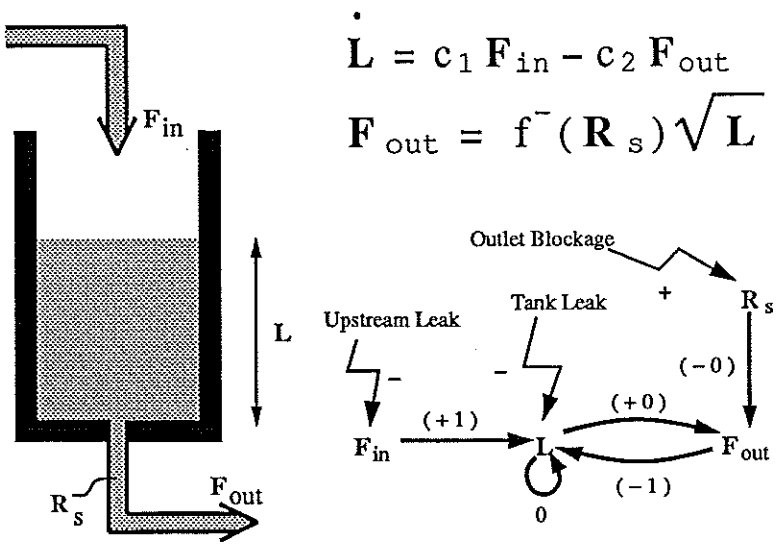

Figure 2.1 A gravity flow tank with its equations and SDG

will decrease. Now, the deviation of L can be determined by multiplying the sign of the deviation of F_{in} with the sign at the arc between F_{in} and L. In this case L will decrease, since the deviation of F_{in} is negative and the sign of the arc is positive. Now this can be done for all nodes downstream of the disturbance. Since L is decreasing, F_{out} will decrease too. This means that L will increase which compensates for the decrement predicted above. The graph must be analyzed more deeply to determine what the final values of nodes in negative loops like this will be, and this analysis is done in the following section.

In many cases it is not enough to use ordinary differential equations only. When partial differential equations occur they are discretized to ordinary differential equations. Algebraic equations mostly have a variable that can be considered the output of the equation. An example of this is shown in Figure 2.1. The second equation,

$$F_{out} = f^-(R_s)\sqrt{L} \tag{2.4}$$

where f^- denotes a positive, monotonically decreasing function, is written in this form since it is actually the outflow F_{out} that is given by the pipe resistance R_s and the level L. This indicates that causal arcs will initiate at R_s and L and terminate at F_{out} with signs given by the partial derivatives $\frac{\partial F_{out}}{\partial R_s}$ and $\frac{\partial F_{out}}{\partial L}$ respectively. When there does not exist one unique way to write the equations, a single process might lead to different SDGs.

The construction of the SDG for the whole process is done in a few minor steps. First the process is decomposed into units such as tanks, valves, pumps etc. Then the SDG for each component is derived from physical relations. The variables that describe a process unit must be chosen so that they specify the function of the unit sufficiently detailed. Further, adjacent units must have a consistent set of boundary variables. When the SDG for a process unit has been determined, it is possible to use this SDG for all units of the same class, regardless of the values of unit constants, as long as the underlying equations are valid. The set of root causes that might affect the unit must also be specified. All the faults concerning the unit that should be detectable

must be incorporated in this set. For each fault there must exist a primary deviation variable, i.e. a variable (or parameter) included in the SDG which is first affected by the root cause, and a sign that tells in what direction the primary deviation variable is deviated. For the tank possible root causes might include tank leak, affecting L with a negative sign, upstream leak, affecting F_{in} negative, and outlet blockage, affecting R_s positive.

When this is done all sub-SDGs are linked together so that shared variables overlap. It is possible to add more root causes to the SDG on the system level and to remove root causes that either are not interesting or cannot occur. Now that the process SDG has been developed it is possible to follow the disturbance propagation through the process just by reading the signs of the arcs on feed-forward paths. Though, when a negative feedback loop is involved, or when two or more paths with opposite signs exists between two variables, it is impossible to determine the actual process behavior from the graph only. The first case is discussed in the following section, and the second case requires some knowledge of the relative speed and strength of the paths to determine the behavior of the process.

2.2.2 ESDG - Extended Signed Directed Graph

Consider the tank in Figure 2.1 once again. Assume that the pipe resistance R_s is somehow decreased. Following the arcs gives that F_{out} will increase, L decrease, F_{out} decrease, L increase and so on. This leads to a chain of contradictions and the SDG does not tell whether the outflow and the level will ultimately be low or high, oscillate or return to their initial values.

One possible solution of this problem is to analyze the loops in the SDG and insert additional non-physical arcs into the graph. This will lead to the Extended Signed Directed Graph (ESDG), which can explain the behavior of the process through feed-forward paths only. To determine the set of additional arcs to be inserted, the following definitions must be made:

- A **path** is a directed sequence of nodes and arcs in the (E)SDG.
- An **acyclic path** is an open path where all nodes appear only once.
- A **loop** is a closed path where all arcs are traversed only once.
- A **cycle** is a loop where all nodes are traversed only once.
- A **strongly connected component** (SCC) is a subgraph of the SDG, such that a path exists from any node to any other node, and this subgraph is not a subgraph of another SCC.
- A **disturbance node** to a SCC is a node, not part of the SCC, such that there exists at least one arc from the node to any node in the SCC.
- The **complementary subsystem** of a path is the subgraph obtained when all nodes in the path are removed.
- The **acyclic subsystem** of a graph is the subgraph obtained when all nodes involved in cycles (excluding self-cycles) are removed.
- The **direct effect** of a disturbance on a node are the influences on the node following acyclic paths.
- **Feedback effects** are the influences on a node following loops containing the node.

- The **initial response** is the first detectable deviation from the nominal steady-state value of a node, due to a disturbance.
- The **ultimate** (or final) **response** of a node is the final qualitative state of the node when a disturbance is present.
- A node is subjected to **compensatory response** if feedback effects cause the node to return to its initial value.
- A node is subjected to **inverse response** if the final value of the node is in the opposite direction of the initial response.

The purpose of the SDG is to determine the initial and final response of every variable caused by any disturbance. The initial response is obtained by following acyclic paths. The ambiguities that may occur if acyclic paths with different signs exists must be resolved with a more quantitative knowledge of the process. This is discussed in the next section. The final response is the same as the initial response if no feedback loops are present. The variable is said to be monotonic with respect to the disturbance in this case. If a variable is a part of a loop it can either have a monotonic behavior or be subjected to compensatory or inverse response. It is possible to give criteria, in terms of the definitions above, for the cases where variables will or may undergo compensatory and inverse response. The criteria were first given by Oyeleye & Kramer (1988) but Rose (1990) has replaced the original criteria with simpler ones.

First the SDG is decomposed into strongly connected components. Then the criteria given below are applied to every node in each SCC with respect to any disturbance node of the SCC, including root causes pointing at any node in the SCC, to determine whether the node is a monotonic, compensatory or inverse variable. The criteria given by Rose are as follows:

1) Necessary and sufficient conditions for compensatory response of a variable with respect to a disturbance due to feedback effects:
   a. The variable is located in a negative feedback loop.
   b. The acyclic subsystems of the complementary subsystems to all acyclic paths from the disturbance to the variable each contain at least one zero self-cycle (integrator).

2) Necessary conditions for inverse response of a variable with respect to a disturbance due to feedback effects:
   a. The variable is located in a negative feedback loop.
   b. The complementary subsystem to one of the acyclic paths from the disturbance to the variable contains a positive cycle (or self-cycle).
   c. The acyclic subsystem of at least one of the complementary subsystems in (2b) must not contain a zero self-cycle.

Note that the conditions given for inverse variables are only necessary. When a variable satisfies the criteria for an inverse variable, additional quantitative information must be supplied to determine if the variable will actually display inverse, compensatory or monotonic response.

Now that the initial and ultimate response can be determined, it would have been interesting to know what happens in between. E.g. when a variable is compensatory due to a controller loop, will the value smoothly return to its nominal value, or will it oscillate before reaching the steady state? This depends on if the system is overdamped, underdamped or critically damped,

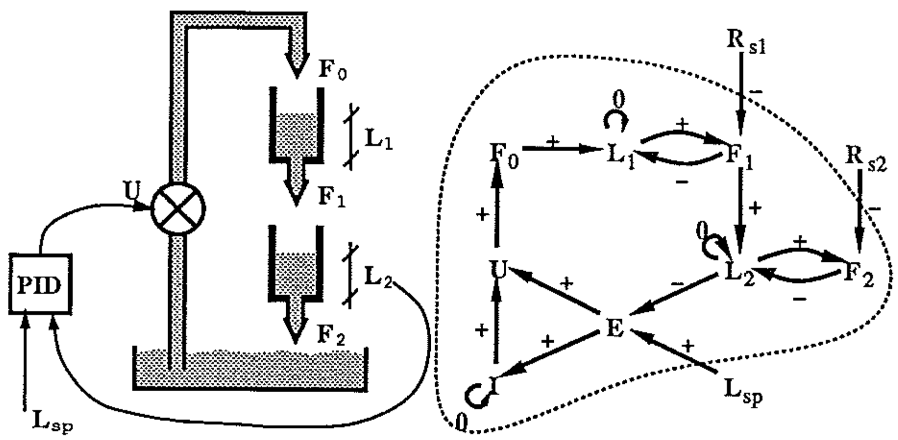

Figure 2.2 Two coupled tanks with a PID controller. For simplicity no root causes or time delays are shown in the SDG. All nodes inside the marked area belongs to the only SCC in the graph

and there does not exist any good way of modeling this qualitatively. Whenever this situation occur MIDAS will act as if the system is critically damped, and if it is not, the diagnosis might be inaccurate or at least loose resolution.

When the inverse and compensatory variables (IVs and CVs) have been discovered ESDG arcs will be added to the SDG. For each IV or CV an arc that initiates just upstream of the variable and terminates just downstream of the variable is created. When two or more nodes in a row displays the same type of response with respect to a disturbance, only one extra arc is added, from the node just upstream of the row to the node just downstream. For a string of IVs the sign of the arc is the product of the SDG arcs between the origin and the termination of the ESDG arc. For a string of CVs the ESDG arc sign will be of the form $\pm\delta([dx_j])$, where δ is the "qualitative Dirac delta function" and x_j is the node just upstream of the termination of the ESDG arc. The sign is determined the same way as for an IV. $[dx_j]$ is the qualitative state of x_j, which can be either -, 0 or +. The δ-function is 1 only if its argument equals 0 and is 0 otherwise. This implies that an ESDG arc derived from a CV will only be active when x_j is at its nominal steady state.

As an example of how to use the criteria above, we will consider the small process in Figure 2.2, consisting of two coupled tanks where the level in the lower tank is controlled by a PID-controller. The SDG is composed of two single tanks as in Figure 2.1 and a simple model for the controller. L_{sp} is the set-point of the level in the lower tank, E is the controller error, I is the integral part of the controller and U is the output signal, assumed to affect the inflow of the upper tank directly, since the pump is not modeled. This implies that no faults concerning the pump can ever be detected with reasonable accuracy and resolution.

The first step is to decompose the SDG into SCCs, and in this case the SDG consists of only one SCC containing all nodes in the graph except R_{s1}, R_{s2} and L_{sp}, which instead are disturbance nodes to the SCC. Each node in the SCC will now be tested with each disturbance node to see if the criteria will be satisfied. The first thing one discovers is that no positive cycles or self-cycles exist in the graph, and thus no inverse response can be found (see condition (2b)). Zero self-cycles and negative cycles exist though, so there might be nodes in the graph which can display compensatory response.

As a first example, we will look at F_1 and the disturbance node R_{s1}.

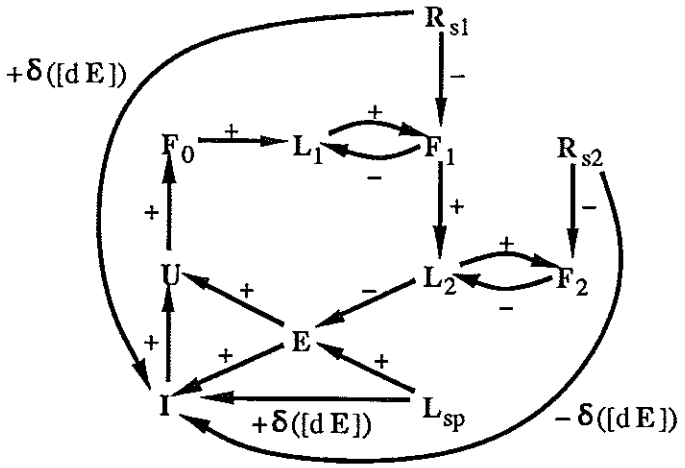

Figure 2.3 The ESDG of the process in Figure 2.2

Condition (1a) is satisfied twice, both the global control loop and the local loop with F_1 and L_1 are negative feedback loops. In condition (1b) all acyclic paths must be considered, and in this case there is only the direct path from R_{s1} to F_1. The complementary subsystem of this path consists of all nodes in the SCC except F_1. To get the acyclic subsystem of this complementary subsystem all nodes still involved in loops, in this case only L_2 and F_2, are removed. This leaves E, I, U, F_0 and L_1 as the sought acyclic subsystem, and this contains two zero self-cycles, making F_1 a compensatory variable with respect to R_{s1}.

Next, consider L_1 with the same disturbance node R_{s1}. Now there exists three acyclic paths from the disturbance to the examined node, among them is the one traversing the nodes R_{s1}, F_1, L_2, E, I, U, F_0 and L_1. This path leaves only F_2 as the complementary subsystem, which is in fact also its own acyclic subsystem. And since no integrator is left, the criterion (1b) gives that L_1 will not be a compensatory variable with respect to R_{s1}.

The results when this procedure has been repeated for all variables and all disturbances, are that F_1, L_2, F_2 and E are compensatory with respect to R_{s1}, L_2 and E are compensatory with respect to R_{s2}, and E only is compensatory with respect to L_{sp}. The corresponding ESDG arcs are added to the SDG and the new graph is shown in Figure 2.3. Note that all self-cycles are now removed, since they have been replaced with ESDG arcs in those cases they affect the process behavior. When using the ESDG, disturbance propagation can be determined through looking at acyclic paths only and this will make it easier to derive the Event Process Model described in the following section.

2.2.3 The Event Process Model

When MIDAS is working on-line it compares the detected process behavior to an event process model. This model is an event graph, and consists of events, root-causes and links connecting events with other events and with root causes. The events in the model correspond to transitions between the qualitative states of the measured variables. A qualitative state of a variable is either high, normal or low, compared to the nominal steady-state value. That is, for every measured variable four events are possible: a transition from normal to high, from high to normal, from normal to low and from low to normal. The events in the model bear a prefix "potential" to separate them from the

events that are actually detected. Since the potential events are members of the process model, they cannot be created or deleted, they just exist. Whenever the state transition associated with a specific potential event is detected on-line, a realization of the potential event is created. This implies that even though there exists only one potential event for each state transition, several realizations of this potential event may exist simultaneously. The same relation exists between potential root causes in the process model and realizations of these when creating hypotheses based on detected events. More about this in Section 2.3. Since this section only deals with the process model, the word "potential" may be omitted in the following.

It is possible to include other types of events in the event graph, such as operator actions, off-line test results, and changes in trend. MIDAS also allows the use of quantitative constraints based on mass balances, energy balances or other types of equations involving measured variables. These constraint equations are written in a residual form, and the residual is then treated as a new measured variable that may be either high, normal or low. These additional types of events are not discussed further in this report though, since MIDAS does not contain a structured way of building a model that can express relations between such events. After the model based solely on variable state changes has been constructed, other types of events may be added heuristically to enhance the diagnostic capability.

An event graph consisting of variable state changes is possible to derive directly from the ESDG, since this contains all variables in the system and the influence they have on each other. Briefly, the transformation of an ESDG into an event graph is done by removing all unmeasured nodes in the ESDG, but for each measured node create four nodes in the event graph, each representing one of the possible state changes for the variable. The arcs in the ESDG are translated into arcs in the event graph. There are two types of arcs, one type called local cause link, connecting a root cause with the event that should be the first symptom of the fault, and the other type called compiled link, connecting events to express possible fault propagation. The compiled links may have conditions attached to them telling when the arcs are valid and what diagnostic conclusions that may be drawn when two events are linked together.

An event graph for the single gravity flow tank in Figure 2.1 where the level and the outflow is measured is shown in Figure 2.4. The ESDG of the tank is almost the same as the SDG, only a $+\delta([dF_{out}])$ arc from R_s to L is inserted, and the self-cycle at L is removed. In addition, the set of root causes is somewhat modified, including sensor bias, downstream leak and high/low inflow instead of upstream leak. In Figure 2.4 the circles represent potential events, the dashed arrows are local cause links and the solid arrows are compiled links. When an event has been detected, e.g. the flow gets low, either of the root causes pointing at this event has probably occurred, in this case outlet blockage or flow sensor low bias. The :NOT-conditions attached to most of the compiled links will be interpreted as follows: when the events at both ends of the link are detected, the root causes in the :NOT-conditions of the link are no longer likely. Note especially that sensor bias is a member of the :NOT-conditions for every link, and this is so since a sensor fault will not cause any process changes unless it is a controlled variable. The :ONLY-IF-TRANSIENT means that the event at the termination of the link will not occur for any of the modeled root causes until the root cause has been corrected.

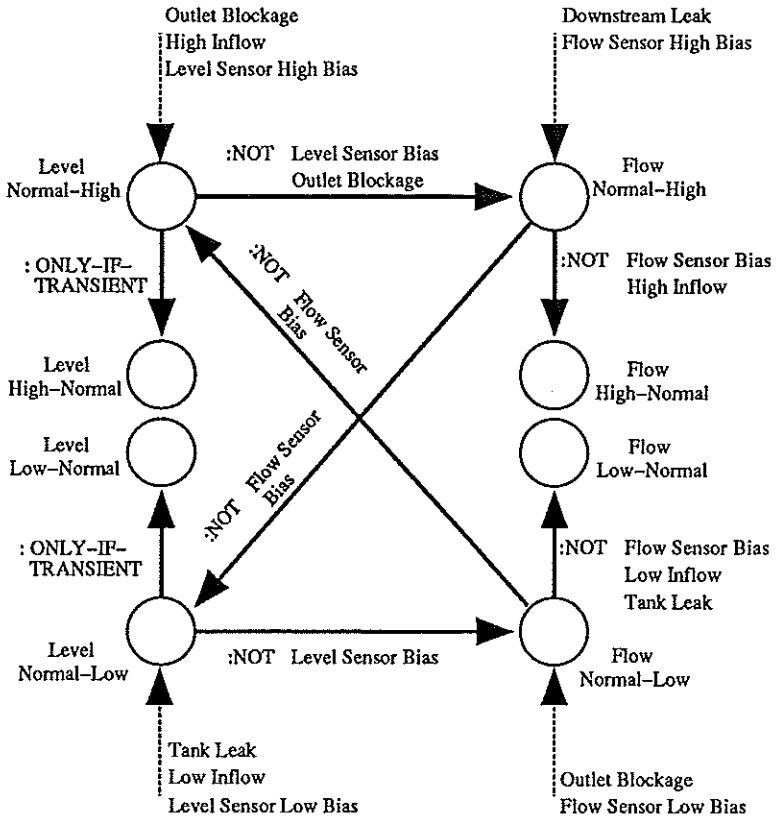

Figure 2.4 The event graph of a gravity flow tank

The different steps in deriving the event graph will be described below. A more detailed description is provided in Oyeleye (1989). The first step is to copy the root causes from the ESDG to the event graph and to create the four potential events for each measured node in the ESDG. Then the allowed state transitions of each measured variable for every plausible fault is listed. This is done with the sign products of the paths between measured nodes in the ESDG. Where paths with different sign products exist, quantitative information must be added to make clear what actually will happen.

Then, for each root cause, a primary deviation path in the ESDG is determined. This initiates at the root cause and terminates at the measured node which is likely to be the first symptom of the root cause. This primary deviation path in the ESDG is translated into a local cause link in the event graph. It starts at the root cause and ends at either the normal-to-high or normal-to-low event of the measured variable, depending on the sign product of the primary deviation path.

Next step is to create compiled links between events. This is done by finding successor paths in the ESDG. These paths are acyclic paths with minimum delay from one measured node to any other measured node. Minimum delay means that none of the other paths may have less time delay. Since the time delay is measured by 0 and 1 only it is sometimes hard to find out which paths have minimum delay. Deeper process knowledge sometimes can solve this problem. The sign product along the successor paths tells what transitions of the second node may follow transitions of the first node, and a compiled link is created for each such transition. This is compared to the list of allowable transitions to see if any of these are illegal for any of the root causes. If this is so, these root causes are members of the :NOT-conditions

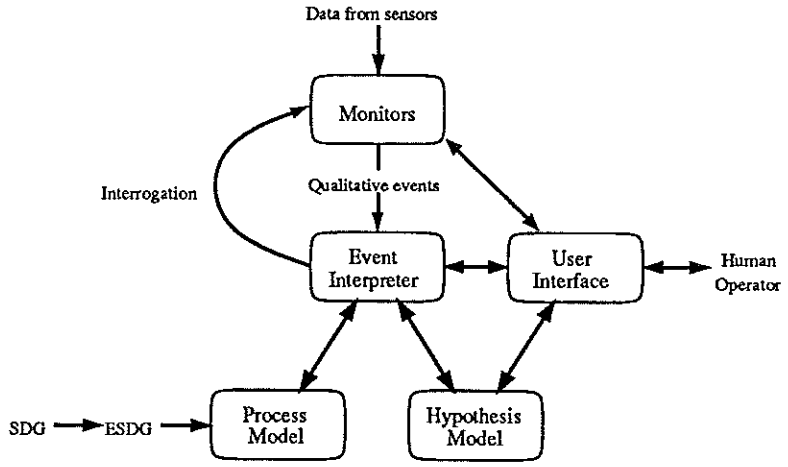

Figure 2.5 The structure of MIDAS

attached to the compiled link. If a transition is forbidden for all root causes, the compiled link is never created. When this is completed, all root causes will be blocked by :NOT-conditions along the links where the disturbances will not propagate.

In the last step :ONLY-IF-TRANSIENT links are created. These goes from normal-to-high (normal-to-low) of a variable to high-to-normal (low-to-normal) of the same variable for those variables that do not display the corresponding inverse or compensatory response for any process disturbance.

When all this is done the event process model, or the static knowledge-base, is ready. Depending on how the causality in a process is handled, a single ESDG may lead to different event graphs. These must of course create similar, preferably identical diagnoses, but they may differ in diagnostic resolution and efficiency. It is preferable to have a small, not highly connected event graph, since this makes the diagnosis faster and probably better. What remains now is the on-line part that actually performs the diagnosis. This is the subject of the following section.

2.3 Diagnosis

The previous section described a way to construct the event process model MIDAS uses for the diagnosis. This is done off-line either by hand or by using the original MIDAS program. In this section the on-line diagnosis will be described. The structure of the diagnostic function and the information flow is shown in Figure 2.5. The process model is a representation of the event graph derived from the SDG and the ESDG. It consists of all potential events, all potential root causes and the causal relationship between them. Each measured variable or constraint equation has an associated monitor which detects deviations from the steady state and creates the corresponding qualitative events. These events, named recorded events, can be seen as realizations of the potential events in the process model. The event interpreter is a set of rules and procedures that matches new recorded events against both the process model and the previous events in order to create clusters of related events. For each event cluster MIDAS creates a set of ranked hypothesized root causes that may be responsible for the detected events. All event clusters, occasionally called inferred malfunctions, are stored in the hypothesis model along with

their hypothesized root causes. The hypothesis model can be seen as the active, constantly changing knowledge-base, whereas the event graph is the static knowledge-base, derived from a physical model of the process. The different diagnostic functional units are described below.

2.3.1 Data collection and processing

The diagnosis unit in MIDAS is linked to the process via a type of objects called monitors. Every sensor variable has an associated monitor which is responsible for detecting state changes concerning the variable. This can be done at any level of complexity, from the simplest ones using raw measurements and thresholds between states only, to more complex schemes using advanced statistics and pattern matching. Since the primary task for MIDAS is to create chains of subsequent qualitative state changes, there is no need for a too advanced detection scheme. Though, it must of course be able to detect all true events, or at least as many as possible, and be robust against false detection.

This is achieved with reasonable performance using a quite simple implementation. The raw measurements are sampled at a specified sample rate and then smoothed with a first-order lag filter to reduce the risk of false alarm due to high frequent measurement noise and process noise. Then the filtered value is compared to the thresholds between the qualitative states to determine the current state of the variable. If the current state differs from the previously detected state, MIDAS creates a new recorded event which is taken care of by the event interpreter.

It is also possible to detect other types of events, not directly related to those in the process model. Example of such events could be that the signal is constantly increasing or decreasing, that the noise of the signal exceeds a given value, or that the measurement is out of range for the given process equipment. These events would make it easier to diagnose gross sensor failures.

A special type of monitors are the ones used for constraint equations. A constraint is a quantitative equation that must be satisfied if the process is working properly. The equation consists of measured variables and known constants and process parameters. The constraints can be either algebraic, differential or integral equations, and they are written in a residual form. The constraint monitor then treats the residual as a normal signal variable, and detects events when the residual becomes too large in either direction.

In Figure 2.5 there is an arrow labeled "Interrogation" from the event interpreter to the monitors. This indicates that the event interpreter may ask a monitor to predict a future event before this has actually occurred. This is done with polynomial, in its simplest case linear, prediction. If the monitor predicts an event to happen in the near future, a new predicted event is created and treated like a true recorded event, only with less probability of accurate detection. The cases where an interrogation may be done are described in the next section.

2.3.2 Event interpretation and hypothesizing

When a monitor has detected a new recorded event (RE), this will be handled by the event interpreter, which is a set of procedures and algorithms used to link new events with old ones and to make a diagnosis from the observations. The event interpreter must be implemented in a way that makes it

possible to use it with any process model, no matter what size and complexity this has.

The event interpretation consists of a procedure, called the inference cycle, that is run for every recorded event. When a new event occurs, the following steps are performed:

1) The event is created and catalogued in the hypothesis model.

2) The interpreter tries to link the new event together with clusters of old events. If the search for such a cluster, or inferred malfunction (IM), is successful, the event will be incorporated in this cluster, otherwise the event interpreter creates a new cluster containing the new event.

3) The cluster containing the new event is examined to see which events in the cluster may be source events, i.e. events that can explain all other events in the cluster.

4) The existing hypotheses are revised to include the information provided by the new event.

When the inference cycle has completed, the event interpreter will wait for next event to come.

The first step also contains a check whether the new event is a realization of a previously predicted event. If this is the case the predicted event is replaced in its event cluster and the second step may be omitted. Though, the source and evidence evaluation must still be performed, since MIDAS do not believe in a predicted event as much as in a recorded event.

The search and linkage phase will examine all existing event clusters to see if the new event is causally related to any of these. One of the following results may be the outcome of the search procedure:

- The new event cannot be linked to any existing events.
- The event can be explained by previous events.
- The event can explain previous events.
- The event can both explain existing events and be explained by the same or other events.

In the first case the event interpreter may interrogate monitors to see if any event that might link the new event to an existing cluster can be predicted. If this is so, a predicted event is created and is inserted in the existing cluster together with the new event. Otherwise a new inferred malfunction is created with the new recorded event as its only member.

The situation in the second case is shown in Figure 2.6a. The new event is incorporated in the cluster. When this situation occurs the diagnosis often changes very little, but hopefully the diagnostic resolution increases a little. If the new event can be explained by more than one cluster, it will be a member of each one of them, but the clusters will not merge into one.

The third case might look as in Figure 2.6b. In the figure the new event explains a source event, it is an out-of-order event and becomes a previously undetected source. The diagnosis will probably change drastically, since the current hypothesis cannot explain the new event. Though, if the new event does not explain a source event, e.g. the rightmost event in Figure 2.6b, the new event will not be a member of the existing cluster. Instead, a new cluster is created and the new event will be inserted into this along with the events

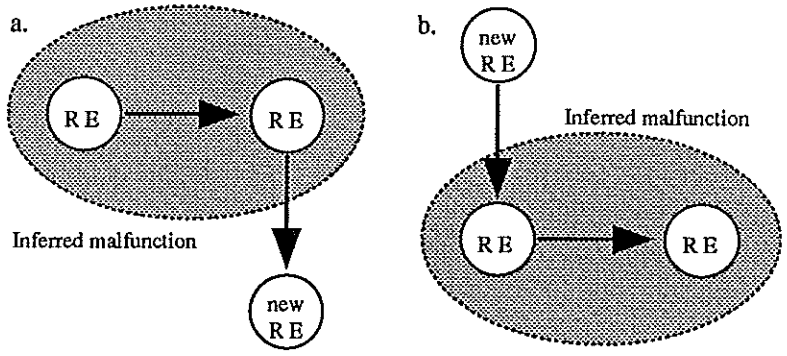

Figure 2.6 a. The new event is explained by others. b. The new event explains previous events

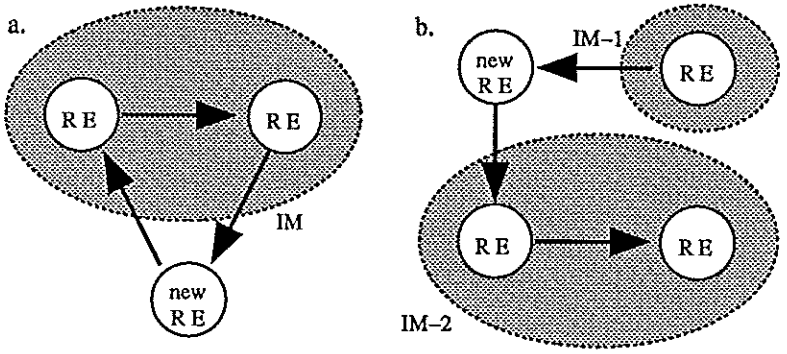

Figure 2.7 The new event both explains and is explained by previous events a. in the same cluster, b. in different clusters

explained by the new event. This is so since otherwise, there would not have been any source events in the cluster, i.e. no event that can explain all the others.

The last case, where the new event both explains and is explained, is shown in Figure 2.7. Here are two major possibilities. Either the new event can explain and be explained by events in the same cluster, or it can build a bridge between two clusters. In the later of these cases the two clusters will merge into one if the explained event in the second cluster is a source event. Otherwise the new event, along with the events it explains, is added to the first cluster, and the second cluster remains unchanged.

Now the second step in the inference cycle has completed and the source evaluation starts. This is done only for those inferred malfunctions that have been altered by the new event. As mentioned before, a source event is an event such that every other event in the cluster can be reached from this event using the causal links. Further, the diagnosis assumes that the source events in a cluster are primary symptoms of the present root cause. There must exist at least one source in every cluster. When causal loops occur, like in Figure 2.7a, and no event outside the loop explains any of the events in the loop, all events in the loop are considered the source events of the cluster. It is not recommended that the order of detection should determine which of the events in the loop is the true source, since out-of-order events often occur in real plants. When the source events of a cluster have been discovered, all root causes that have any of the source events as their primary symptoms, will be incorporated in the hypothesis model as hypothesized root causes of the cluster. These hypothesized root causes are copies of the associated potential root

causes in the process model, and for each event cluster one (1) hypothesized root cause is assumed to be the true cause of the cluster.

The last step in the inference cycle is to update the diagnosis to see if the new event somehow changed it. This is the procedure called the evidence evaluation, even if the word evidence might seem a far too strong word. The evidence is computed individually for each cluster, and is basically done by looking at all the events in the cluster to see which root causes they support and which they oppose. An event will support a hypothesized root cause either if the event is a primary symptom of the root cause or if a free causal pathway exists from a primary symptom of the root cause to the event. A free causal pathway is a chain of compiled links that is not broken or blocked by any conditions for the specific root cause. All events that are not supporting a root cause will be assumed to oppose it.

Now that the sets of supporting and opposing events have been determined for each hypothesized root cause, this information will be used one way or another to compute the most likely fault in the cluster. If each event has a probability of accurate detection and one of inaccurate detection, the likelihood of each hypothesis may be computed in a number of ways using statistics. Though, such probabilities must be guessed very roughly in most cases, but at least one must assume that a recorded event must have a higher probability of accurate detection than a predicted one. It is also possible to designate a prior probability to each potential-root-cause, indicating that some faults are more common than others. When the root causes have been ranked, the inference cycle has completed, the current diagnosis may be presented to the operator, and the event interpreter may wait for next event.

2.4 Operator interface

One of the most important things to do when writing a computer program is to design an operator interface. In MIDAS, the most important task for the operator interface is to display the diagnostic results in an appropriate way. But it will also be necessary to direct the diagnosis, e.g. by giving parameters such as threshold values, filter constants and sample rates to the monitors. It is also desirable that different types of users will have different operator interfaces. A plant operator should just see the fault presentation and be able to tell the system when a malfunction has been corrected. A process engineer should be able to follow the event interpretation, to look into the process and hypothesis models, and to tune the monitors in order to enhance the diagnostic performance.

This master thesis project is emphasized on the implementation of a diagnosis system and therefore the time spent on making a good operator interface has been minimized. How it actually was implemented in the G2 prototype is shown in Section 3.4.

2.5 Features and limitations

The differences between quantitative and qualitative diagnosis systems are mentioned in Section 1.2, and all of the attributes given to qualitative systems are valid for MIDAS too. In this section some of the more specific advantages and disadvantages of MIDAS will be discussed.

A causal model implies that the symptoms of a malfunction will appear in a certain order. Though, in many cases the symptoms will occur out-of-order depending on monitor tuning, sample rates or any unexpected process behavior. When MIDAS has detected a series of events, the diagnosis will be the same, regardless of the detection order. Thus, MIDAS is robust against out-of-order events, and this is mostly an advantage. An exception to this is the case where the detection order in itself bears valuable information, but these situations are considered to be more rare than the unintentional out-of-order events and they are therefore neglected.

Another feature is that MIDAS tries to minimize the hypothesis model, i.e. if a series of events can be explained by a small set of root causes, one of these is assumed to be the true root cause. Another solution would have been to include the possibility that some (or all) events are independent and thus enlarge the set of hypothesized root causes. If all possible cases would be included, the hypothesis model would grow combinatorially when the size of the clusters grow. And since simultaneous, independent faults can be assumed to be relatively rare, it is not considered a too serious limitation to exclude these possibilities. However, MIDAS can easily be changed so that any possible event interpretation could be chosen on request.

As mentioned above, multiple malfunctions will be interpreted as a single malfunction as long as any causal relationship between the symptoms of the faults exist. On the other hand, when a new event cannot be linked to any existing event a new event cluster is created, indicating that two faults exist simultaneously. This implies that multiple malfunctions can be detected if their symptoms do not overlap in the event process model.

One of the features of the (E)SDG is that it tells what *will* happen in the near future when a specific disturbance, e.g. a root cause, is present. The arcs in the event graph just tell what *might* happen when one or more events have been detected. If one wants to know what actually will happen, one also must know the present fault, i.e. the diagnosis must be done. This implies that when an event does not occur, MIDAS in its current version cannot use this piece of information to refine the diagnosis. If just one event occurs, e.g. an uncontrolled variable gets high, and nothing more happens during a very long time, a human operator will conclude that the sensor is broken. MIDAS though, assumes that all faults that primarily will cause the variable to get high are possible. This can be changed by associating a time-out attribute to each compiled link. When all links initiating at a detected event has timed out, all root causes not blocked by the link should be removed from the candidate set. This is not water-proof, but in most cases it would help to increase the diagnostic resolution. However, this feature is not included either in the original MIDAS concept or in the current G2 implementation.

3. An Implementation of MIDAS in G2

This chapter will deal with the generic parts of the G2 implementation, i.e. the representation of the different objects introduced in Chapter 2, and the rules and procedures that perform the diagnosis. For readers not familiar with G2 it is recommended to read Appendix A before reading this chapter.

3.1 The process model

The original MIDAS program both performs the diagnosis on-line and creates the process model used by the diagnosis module. In this project only the diagnostic function is implemented so the model has to be constructed manually. When the event graph has been constructed, its components must be given a representation in G2. From now on, the G2 classes, attributes etc. will be written in teletype font (`like this`). The event process model consists of `potential-events` connected together with `compiled-links`, and `potential-root-causes` connected to `potential-events` by `local-cause-links`. For structural reasons `potential-event` and `potential-root-cause` have the same superior class `pm-component`. The object-definitions and connection-definitions for these classes are found in Appendix B.

An event graph for the coupled tanks in Section 2.2.2 is shown in Figure 3.1. The names of the `potential-root-causes` can be chosen arbitrarily, but the names of the `potential-events` must be like `l1-n-h`, i.e. first a variable followed by the first letters of the states of the variable just before

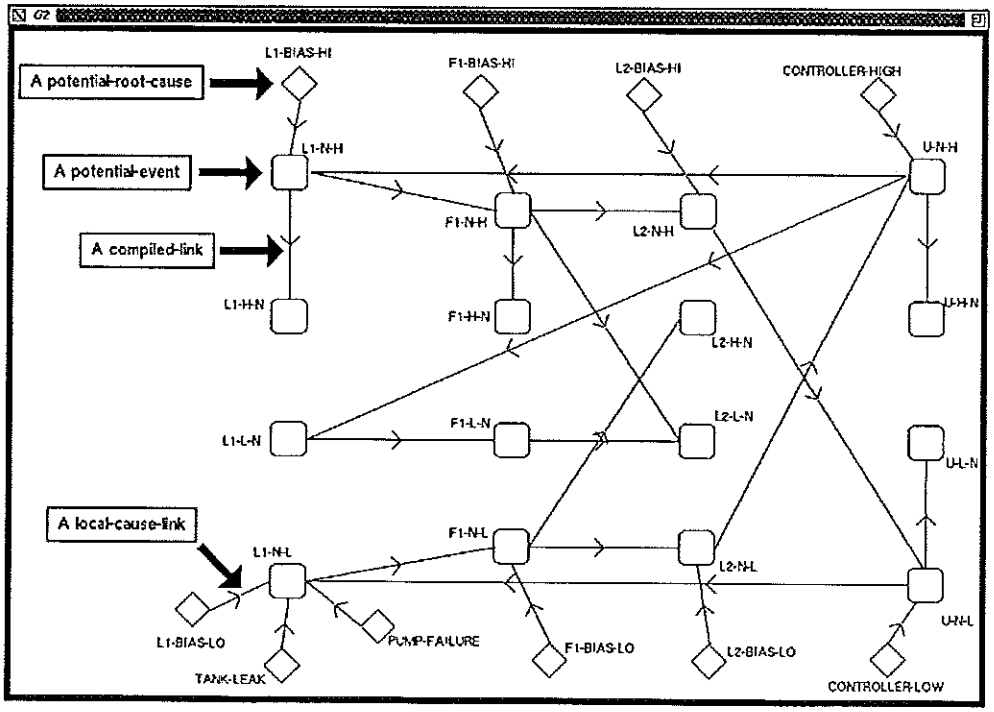

Figure 3.1 An event process model of the process in Figure 2.2

and after the transition. This must be so in order for all initializations to work properly. Each `compiled-link` has the attributes `not-conditions`, which is a list of `potential-root-causes` that are blocked by the `compiled-link`, and `only-if-transient` which, if set to `true`, indicates that the associated state transition only occurs if the fault has been corrected. These attributes are initialized by rules of either of the following forms:

```
initially insert l1-bias-lo at the beginning of the
   potential-root-cause list the not-conditions of
   compiled-link-1
```

or:

```
initially conclude that the only-if-transient of
   compiled-link-4 = true
```

When the `potential-events` and the `compiled-links` have been initiated, the process model is ready to be used by the diagnosis module.

3.2 The diagnosis module

The diagnosis module is the name for all the parts of MIDAS that use the process model to make the diagnosis. It consists of monitors, which are responsible for event detection, the hypothesis model, which contains previously detected events along with the existing diagnosis, and the event interpreter, which adds new events to the hypothesis model.

3.2.1 Monitors

The basic task of the monitors is to detect the events in the process model. The most usual, and in my implementation the only, type of event is the state changes of variables, either a sensor variable or a residual of a constraint equation. Detecting such state changes requires knowledge of the nominal steady-state value and the area around this that should be treated as the `normal` state. The class `monitor` is a fairly simple type of monitor, but it is sufficient in most cases. The object-definition for `monitor` can be found in Appendix B. The subworkspace of a `monitor`, as shown in Figure 3.2, can be used to explain most of its features.

The lower half of the subworkspace consists of type-in-boxes which make it possible to change parameters of the `monitor`. This way a `monitor` can be manually tuned to optimize the detection capability. Though, to determine the optimal values for the parameters may be very time-consuming. The parameters that may be altered are threshold levels, the sample rate, the time constant of the raw data filter, and the prediction parameters `trend-time` and `prediction-time`. As is indicated in the figure, threshold levels can be specified with hysteresis, which in many cases helps to avoid inaccurate detection without decreasing the probability of true event detection.

The `monitor` is sampled at a rate in seconds given by the attribute `sample-rate`. The value of the sampled variable is stored in the `raw-meas` of the `monitor`. To reduce the influence of process noise, measurement noise and other high frequent transients, the raw measurements are smoothed by a first-order lag filter. This filter is implemented with the generic simulation formula in Figure 3.3.

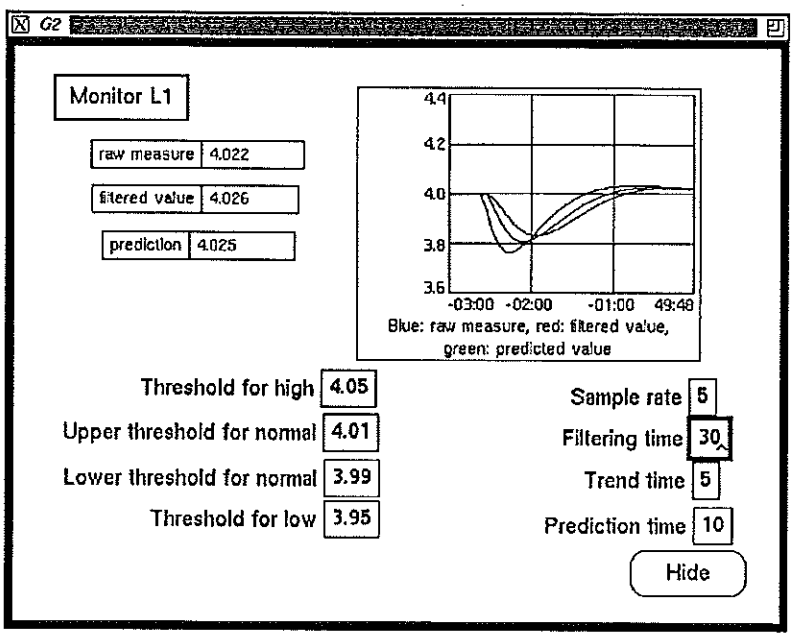

Figure 3.2 The subworkspace of the monitor M-L1

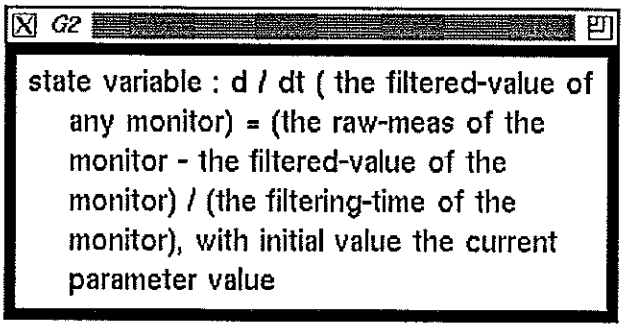

Figure 3.3 First-order lag filter for raw data

In order to support the interrogation, each monitor computes a predicted-value, by extrapolating the trend of the filtered value during the last trend-time seconds, looking prediction-time seconds into the future. When a monitor is interrogated it is asked if a given state of the monitor can be predicted in the near future. The monitor compares the predicted value to the threshold levels and then gives an answer according to this comparison.

When a monitor is sampled, the raw-meas of the monitor gets a value and a procedure UPDATE-MON is started. If the filtered value then has passed a threshold, the current and previous states of the monitor are changed according to this. If a monitor has detected a high state, a situation may occur where the monitor directly detects a low state without detecting the normal state in between. This situation is treated as two subsequent state transitions, one from high to normal and, when this is handled by the system, one from normal to low. If this is not done, the event graph must include the low to high and high to low events, and this would make the process model more difficult to develop and more complex.

As for potential-events, the name of a monitor cannot be chosen at random. The name must start with "M-" and then continue with the name of the monitored variable. This way the last part of a monitor's name and the first part of its associated potential-events' names are identical. This

is used in the initializations to determine which potential-events belongs to which monitors.

The state transitions of a monitor indicate that events have occurred, and these are treated by the event interpreter described in Section 3.2.3, but first the components and the structure of the hypothesis model will be defined.

3.2.2 The hypothesis model

If the process model is seen as the static knowledge-base in MIDAS, the dynamic knowledge-base would be the previously detected events and the fault hypotheses based on these. This dynamic knowledge-base is also called the hypothesis model, and for structural purposes a class hypothesis-model is defined and one instance, the-hypothesis-model, is a static member of the knowledge-base. The-hypothesis-model is managing the bookkeeping of events, clusters and so on.

The components of the hypothesis model are event clusters, hypothesized root causes and different types of events. The clusters and root causes are instances of the classes inferred-malfunction (IM) and hypothesized-root-cause (HRC) respectively. The event class has four subclasses:

- recorded-event (RE), which is instantiated when a monitor actually detects a new event,
- expected-event (EE) and
- latent-event (LE), which both represent predicted events, and finally
- copied-event (CE), which is created when an event is a member of more than one inferred-malfunction.

The object-definitions of all these classes and their superior class hm-component are listed in Appendix B.

Since G2 does not allow naming transient objects, each hm-component has a list of messages which can be used as the name(s) of the component. When a hm-component has been placed on a workspace, these messages are displayed near the icon. Every hm-component has a catalogue name consisting of a prefix determined by the type of component (IM-, HRC-, RE- etc.), followed by an index that is incremented each time a new IM etc. is created. Those components that have corresponding components in the process model, i.e. events and hypothesized-root-causes, may also have the name of the pm-component as a second name.

The objects at the highest level of the hypothesis model are the inferred-malfunctions. These IMs contain events that are linked together and hypothesized-root-causes that may have caused the events. IMs are created and altered as new events are detected and the diagnosis evolve. Each IM has an attribute status that depends on which events the IM contains. For this purpose the events in an IM can be classified in two ways. Firstly, they can be either source events or consequence events. As mentioned in Section 2.3.2, a source event is an event that can explain all other events in an IM, and thus are assumed to be the primary symptom of the actual fault. Secondly, an event can be either abnormal (leading away from the normal state) or normal (leading to the normal state). Basically, to determine the status of an IM is to check if the latest event detected by a monitor is abnormal or normal. The value of status can be either of the following symbols:

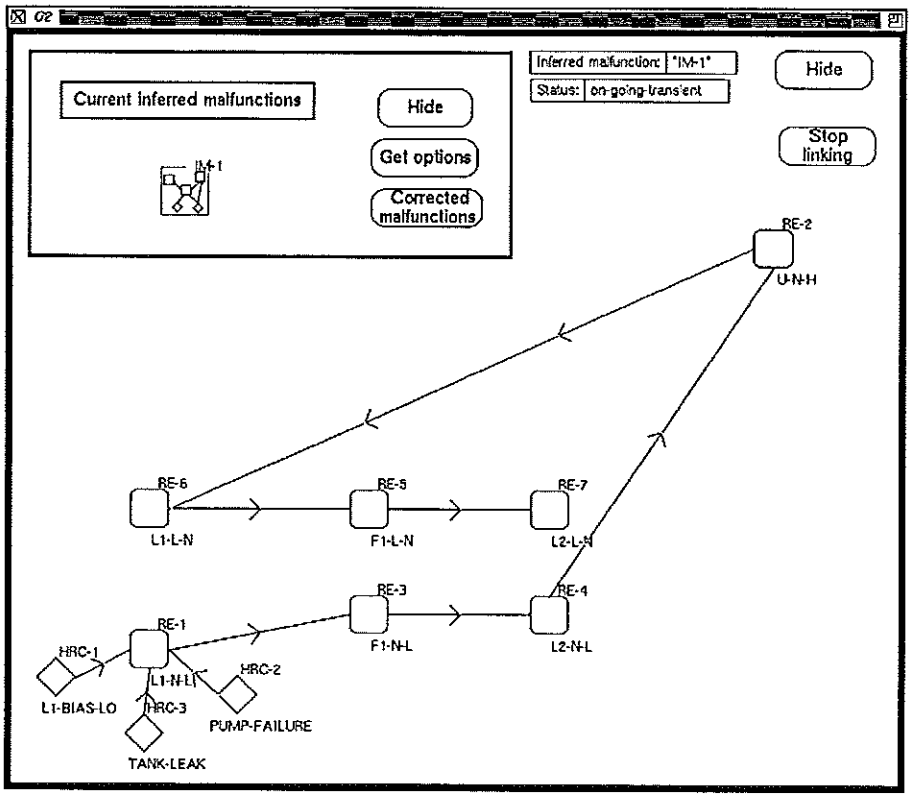

Figure 3.4 The `inferred-malfunction` IM-1

- `persistent`, if there exist source events that have not returned to normal state,
- `on-going-transient`, which means that all source events have been normalized, but at least one abnormal consequence event has not,
- `completed-transient`, indicating that all abnormal events in the IM are superseded by normal events,
- `spurious`, used instead of `completed-transient` if the IM consists of only one normalized event, probably indicating a false alarm, or
- `corrected`, used when an IM has been archived.

Figure 3.4 shows an example of an IM. The icon of the IM can be seen on the small workspace at the upper left corner of the figure. The rest of the G2 window displays the `recorded-events` and `hypothesized-root-causes` that are members of the `inferred-malfunction`. The links are of the same types as the links in the process model, i.e. `compiled-links` between `events` and `local-cause-links` between HRCs and `events`. The status of the IM is on-going-transient, since the source event (RE-1) has returned to normal state (through RE-6) but one non-source event (RE-2) has not. This IM has actually been created when MIDAS worked against a simulated tank process and a tank leak was simulated.

An `inferred-malfunction` can be seen as a subgraph of the process model, since the `recorded-events` and `hypothesized-root-causes` are copies of the corresponding `pm-components`. The IM is displayed in a way that makes it easy to see the relation with the process model. Compare Figure 3.4 to Figure 3.1!

Each IM has a number of item-lists that contain its members. Besides a list for all the events, there is one list containing the source events of the IM, and one containing the events in the IM that may be corrected. Each time an IM gets the status `completed-transient` or `spurious`, all current events in the IM are inserted into the list of correctable events. When a malfunction is corrected, the correctable part of the IM is archived, the rest of the events are treated as new ones. The different types of events are treated nearly the same way, the only exception to this is that LEs and EEs (and copies thereof) are considered less significant when the diagnosis is determined. The icons of the different types of events are also nearly the same, a CE has a dashed instead of a solid border, and a LE or EE is grey instead of black.

Until now no distinction has been made between EEs and LEs. Both are predicted events in contrast with REs, but in certain situations an EE is created, in others a LE is created. An EE is created if the prediction aims at linking a new event to an existing IM. If the prediction is not done in these situations a second IM will be created, and MIDAS will, probably wrongly, conclude that two faults exist instead of one. A LE though, indicates that an IM probably has a latent, i.e. not yet detected, source event, which will drastically change the diagnosis when it is actually detected. In this case the source events of the IM without the LE will first be determined, then the LE is added as an extra source event.

The current diagnosis is stored as a ranked and sorted list of HRCs for each IM. Each HRC contains a list of its supporting events, and based on this a relative likelihood can be computed for each HRC. When the HRCs in each IM have been ranked and sorted, it is assumed that one of the highest ranked HRCs from each IM is a current fault. How the faults are presented is described in Section 3.4.

3.2.3 The event interpreter

When a new event is detected it is incorporated in the existing hypothesis model. This is handled by the event interpreter, which is a chain of procedures that perform the inference cycle defined in Section 2.3.2. The event interpreter is triggered by the rule in Figure 3.5.

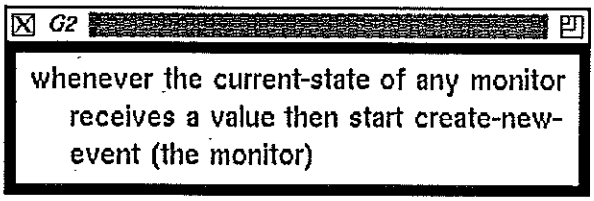

Figure 3.5 The rule that triggers the event interpreter

The procedure CREATE-NEW-EVENT has two major functions. First, as the name indicates, a new transient object, a `recorded-event`, is created and initialized (given names etc.), then a check is done to see if the new RE is a realization of a previously predicted LE or EE. If this is so, the predicted event is replaced with the new event, otherwise the search and linking phase takes place. This is done by the procedure NEW-RECORDED-EVENT, and the rules for the linking are the same as those given in Section 2.3.2, though they are carried out in another order.

If the new event can be linked to any existing IM, i.e. it is explained by any event in the IM or it explains a previous source event in the IM, the

linking is performed directly, without analyzing if the new event can be linked to any other IM as well. The `compiled-links` used for the linking are copied directly from the process model. When the linking has completed, the event interpreter tries to merge the IM containing the new event with all the other IMs in `the-hypothesis-model`. This is repeated until no more merging can be done.

If the event interpreter cannot directly link the new event to any existing IM, it searches for possible missing events and interrogates the `monitors` responsible for these events to see if any of the events can be predicted. A missing event is an event which, when it occurs, will link the new event to an existing IM. Missing events are determined by finding paths in the process model between the new event and all events in the IM. If such a path contains only one intermediate missing event, an interrogation takes place. If the interrogation is successful, i.e. the missing event can be predicted, an `expected-event` is created and, together with the new `recorded-event`, added to the IM. If the interrogation is never performed, or if it fails for all paths, a new IM is created and the new RE becomes its only member.

Interrogation will also take place as soon as a new RE has been added to an IM, to see if it is possible to predict any latent source events. If a latent source exist, but the interrogation is turned off, the risk of inaccurate initial diagnosis is quite high, since the root causes responsible for the latent source will probably not be members of the current hypothesis set for the IM. However, if an interrogation is performed successfully, a LE is created and added to the IM. Now, the root causes pointing at the LE become members of the current hypothesis. But since the LE is not actually detected, only predicted, the possible root causes without the LE in the IM will still be plausible.

To get an example of how the interrogation may work, reconsider the tank leak simulation in Figure 3.4. In this case the events are detected in the following order:

1. L1-N-L (RE-1). Since no other events exist, a new `inferred-malfunction`, IM-1, is created, containing RE-1 only.

2. U-N-H (RE-2). In the process model in Figure 3.1, no link exists between U-N-H and L1-N-L, thus RE-2 and RE-1 cannot be linked together. Further, there does not exist one single event in the process model that could build a bridge between U-N-H and L1-N-L, and therefore no interrogation takes place. Instead, RE-2 is inserted in a new `inferred-malfunction`, IM-2.

3. F1-N-L (RE-3). This event can be linked to L1-N-L, and is therefore inserted in IM-1.

4. L2-N-L (RE-4). This event can be linked both to F1-N-L and U-N-H, and since these events are in different IMs, IM-2 will merge into IM-1.

If the monitors are tuned somewhat differently, and the simulation is restarted, the detection order may be the following:

1. L1-N-L (RE-1). A new `inferred-malfunction`, IM-1, is created.

2. F1-N-L (RE-2). This event is inserted in IM-1.

3. U-N-H (RE-3). Still, U-N-H cannot be linked to any existing event. Though, if L2-N-L can be predicted, this event would build a bridge

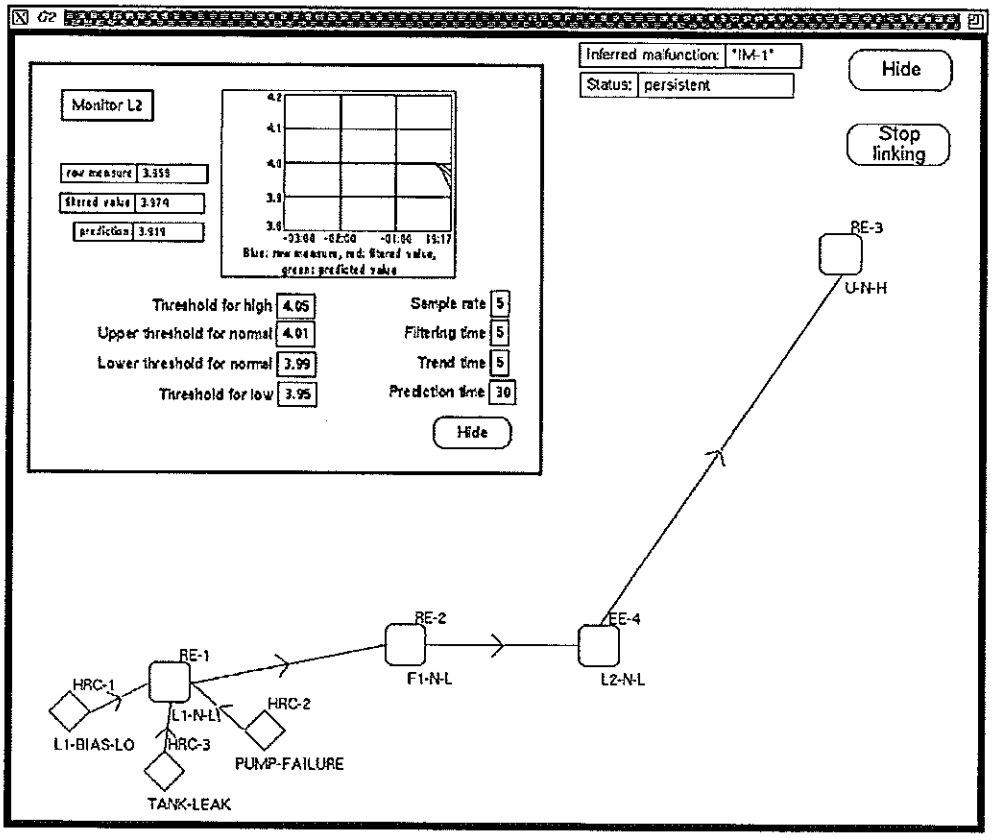

Figure 3.6 An example of successful interrogation

between U-N-H and F1-N-L. Thus, the `monitor` responsible for detecting L2-N-L is interrogated, and if the interrogation is successful, the situation in Figure 3.6 will occur. The subworkspace of the `monitor` M-L2 is shown in the upper left corner of the figure. Note that the filtered value is still normal, but the predicted value is low. This implies that an `expected-event` is created, which is indicated by the name EE-4, and all events are members of IM-1.

4. L2-N-L (RE-4). This event is the realization of the postulated event EE-4, and therefore EE-4 is replaced by RE-4. If L2-N-L is not detected after a long time, and it no longer can be predicted, the EE is removed and IM-1 will divide into two IMs.

If the `filtering-time` of the `monitor` M-L1 is increased it is possible to delay the occurrence of L1-N-L and thus achieve a third detection order. F1-N-L then becomes the first RE in the simulation and when it has been handled completely, the `monitor` M-L1 will be interrogated concerning the possible latent source L1-N-L. If the `prediction-time` of M-L1 is increased too, this event may be predicted, and then the resulting IM will be as in Figure 3.7. Note especially that the root causes of both the RE and the LE are incorporated in the candidate set, but when the LE is replaced with a RE, the root causes at F1-N-L will be removed.

Now the search and linkage phase has completed and the next step in the inference cycle is the source evaluation. This can easily be done directly after the linking with the following rules:

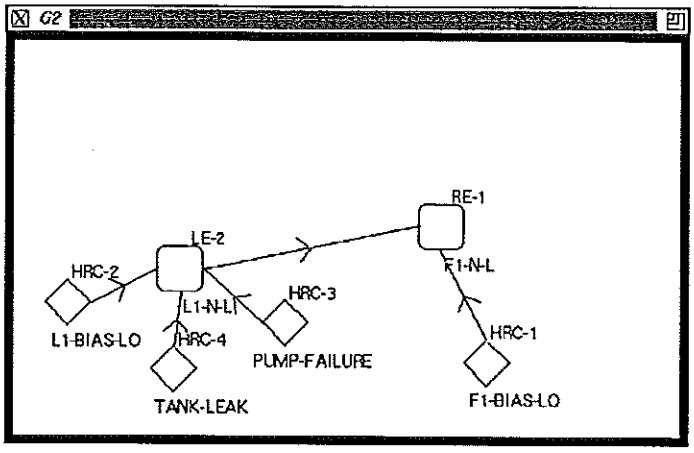

Figure 3.7 An `inferred-malfunction` with a latent source

- If a new `event` in an IM does not explain a previous source event in the IM, and if there exists at least one `event` that explains the new `event`, then the new `event` is a consequence and the other events remain unchanged.
- If a new `event` in an IM explains a previous source in the IM, or if there does not exist any `event` that explains the new event, then the new `event` is a source and the effects on the rest of the IM can be determined using the following recursive steps:
  1) Find all successors of the new `event` that are source events. For each one of these, remove the `event` from the `source-events` of the IM and repeat step 1) with this `event` as the new `event`.
  2) Insert the new `event` in the `source-events` of the IM, and find all its precursors. For each one of these, if the `event` is not a member of the `source-events` of the IM, repeat step 2) with this `event` as the new `event`.

This algorithm will cover all allowable configurations of IMs.

When the source events of an IM have been determined, the evidence evaluation will take place. Its aim is to compute which of the `hypothesized-root-causes` in the IM is the most likely. To do this the IM must be analyzed to see which `events` will be members of the list `supporting-events` of each HRC. This is done by a breadth first walk through the IM from each HRC. A HRC will naturally be supported by its primary symptoms, and these are therefore inserted in the `supporting-events` of the HRC. Next step is to look at all successors of the primary symptoms. If there exists a link between the primary symptom and a successor that is not blocked for the HRC, this successor will also be a member of the `supporting-events`. When all successors have been examined, the procedure is repeated for those successors that are supporting the HRC. This way the whole graph will be traversed through recursive procedure calls, and the events that are not supporting a HRC when the whole graph has been analyzed are assumed to oppose it.

When all HRCs have got their supporting and opposing `events`, some statistic calculation must be done to rank them. Finch (1989) suggests a number of methods to rank the HRCs in an IM, and I have chosen one of these called conditional probability. This method makes it possible to include knowledge of the prior probabilities of the possible faults. Each `potential-root-cause` (and `hypothesized-root-cause`) has an attribute `prior-probability` which has a default value 1.0, but it may be altered to any positive value. The

method also requires some knowledge or assumption of the probability of accurate event detection (β_i) and the probability of false event detection (α_i). For a RE, $\beta_i = 0.95$ and $\alpha_i = 0.05$ is a reasonable assumption. LEs and EEs are less reliable, Finch suggests $\beta_i = 0.60$ and $\alpha_i = 0.40$ for these events.

To compute the conditional probability CP of a hypothesis HRC_i, another probability, the probabilistic likelihood PL, must first be computed by the equation

$$PL(\text{HRC}_i) = \prod_{j}^{\text{SE}} \beta_j \cdot \prod_{j}^{\text{OE}} \alpha_j \tag{3.1}$$

where SE are the supporting events of HRC_i and OE are the opposing events of HRC_i. If the prior probability of HRC_i is given by $PP(\text{HRC}_i)$ and there exist n HRCs in an IM, the conditional probability is computed by the equation

$$CP(\text{HRC}_i) = \frac{PL(\text{HRC}_i) \cdot PP(\text{HRC}_i)}{\sum_{j=1}^{n} PL(\text{HRC}_j) \cdot PP(\text{HRC}_j)} \tag{3.2}$$

MIDAS then uses this conditional probability to rank and sort the HRCs. When this is done, the current diagnosis is made, and the event interpreter can wait for next event.

3.3 Operator interface

When all the functional parts of a computer program have been developed, you can spend how much time you like on making a good operator interface. If the program is meant to be run by others than the programmers, it is very important to make it easy to use. It is also important that different types of users can see and do different things. G2 supports this feature since it allows a knowledge base to be run in different, built-in or user defined, operator modes. In other senses too, G2 offers many features, e.g. nice graphics, that may aid the designers of operator interfaces. In this project, though, the operator interface is very simple.

In a diagnosis program the only necessary information to display to a plant operator is the current diagnosis, i.e. plausible current faults. This is currently done by just giving messages like the ones in Figure 3.8a to the operator. As new events are detected and the diagnosis evolve, a new message is displayed if the new events alter the current diagnosis. The last message typically occurs when a fault has been corrected and the process has gone back to the nominal steady state. The series of messages in Figure 3.8a will occur during the tank leak simulation if the detection order is as in Figure 3.4.

It not obvious that only the highest ranked faults should be displayed. Since the evidence evaluation includes assumptions of a priori probabilities for faults and probabilities of true event detection, unfortunate choices of these probabilities may cause an absent fault to be ranked slightly higher than a present one. The G2 implementation offers a full description of the relative probabilities of root causes as an option. An example of this is shown in the fourth message in Figure 3.8b.

A plant operator should also be able to switch the diagnosis on and off, and to tell MIDAS when all faults have been corrected. This can be done

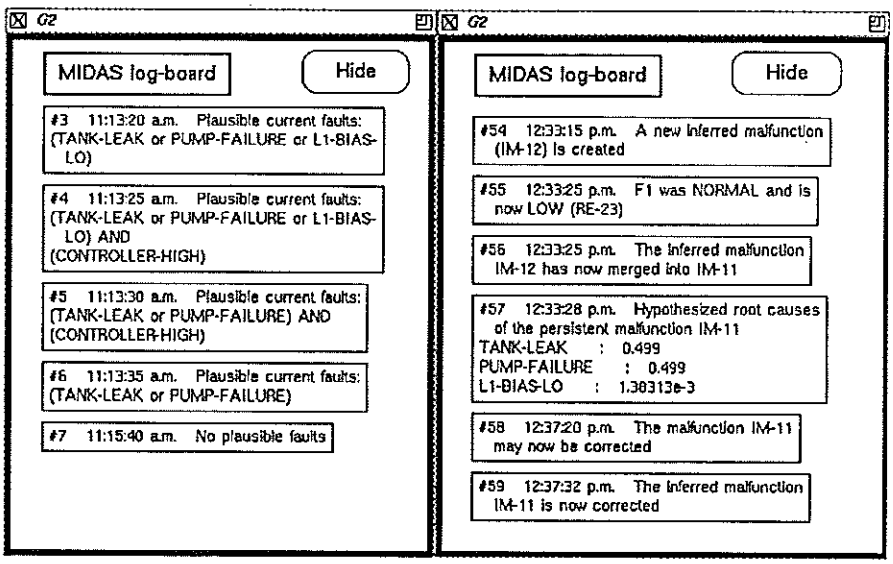

Figure 3.8 **a.** A series of fault messages. **b.** Various types of messages to e.g. a process engineer

by displaying buttons in suitable places. MIDAS is designed to work around a specified steady state, and when the process approaches this state it may display unexpected transient behavior, which may cause MIDAS to get confused. This behavior may occur during start-up and when certain faults are corrected, and to avoid irrelevant and false diagnostic conclusions, it is best to turn the diagnosis off until the process has reached the steady state.

Some users, e.g. process engineers, might like to get into the diagnostic parts in order to understand the system and change the diagnostic performance. A process engineer can choose to display other log messages than just fault presentation. Examples of such messages are shown in Figure 3.8b. Since the process engineer has access to both the process and the hypothesis model, he may find all information he wants from the events, root causes and so on. He can follow the event interpretation and the hypothesizing, and this may guide him when tuning the monitors. This is done by changing monitor parameters as was shown in Figure 3.2.

Now all of the generic parts of MIDAS have been developed and they have also been tested against a simulation of the coupled tank process. In the next chapter the Steritherm application, the second part of the master thesis project, will be discussed.

4. Fault Diagnosis in Steritherm

4.1 The process

During development, the MIDAS diagnosis module was tested against a number of small process models, e.g. the two coupled tanks described in Chapters 2 and 3. In order to see how MIDAS works in a fairly large industrial plant, a process model for Steritherm was developed. Steritherm is a full-scale process for indirect UHT (Ultra High Temperature) sterilization of liquid food products. Indirect heating means that the product is heated by heat exchangers. The product is subjected to high temperature for a short time, usually 135-140°C for a few seconds. This will kill all bacteria in the product. When the product has been packed, it can be stored at room temperature for months.

A simple block diagram for the Steritherm process is shown in Figure 4.1. The product supply consists of a balance tank, where the cold product is stored, and a feed pump. The product is first heated by warm product in the pre-heater. After this the temperature is approximately 75°C. Then the product is fed through a pump to the final heater. Here, the product is heated to the sterilization temperature, normally 137°C, by hot water. The holding tube is a long insulated pipe, where the product is kept warm for a couple of seconds. When the product has passed the holding tube, it is first cooled to about 75°C in the pre-cooler by the incoming cold product. Then it is cooled down to the filling temperature, 5-20°C, by cold water in the final cooler.

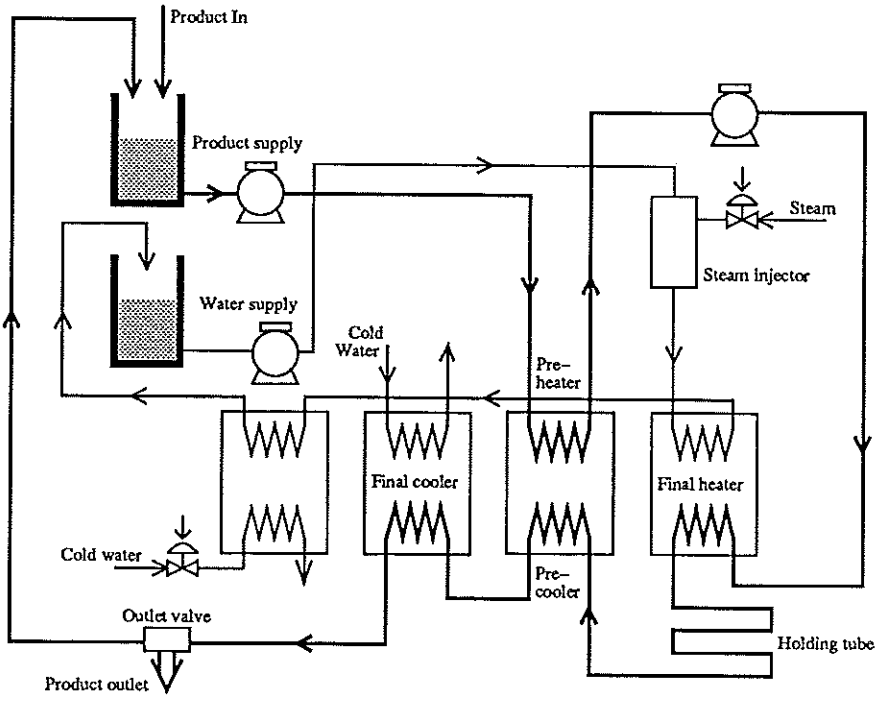

Figure 4.1 Steritherm block diagram

Under normal conditions the product will now be sent to a filling machine or an aseptic tank. A part of the product, though, is recycled back to the balance tank.

The hot water system is used to raise the temperature of the product in the final heater. Basically, it consists of a balance tank, a feed pump and a steam injector. In the steam injector, steam is added to the water in order to increase the temperature of the water. This hot water will then heat the product to the sterilization temperature. The temperature of the product just after the final heater is controlled by a PI-controller. The output of the controller affects the stem position of the control valve, which determines the steam flow into the steam injector. This way the temperature of the heating water, and thus the sterilization temperature, can be controlled. The hot water is cooled in a heat exchanger and then recycled into the water tank. The temperature of the recycled water is controlled by a PI-controller that controls the flow of cooling water through the heat exchanger.

There are four major phases in the normal Steritherm operation.

- **Plant sterilization** is done by circulating hot water through the product line in the plant for about half an hour. This will sterilize the process equipment and make the plant ready for production.
- **Production** can be started when the plant has been sterilized. The water in the product line is led to the drain, and the balance tank is instead filled with the product. The product pushes the water out of the system, and when it has reached the outlet valve, this is opened and the production starts. Normally, the plant will now stay in the production phase for 8-15 hours, depending on the product and the rate of disturbances. After the production phase, the plant must be cleaned, since hot product will cause burn-on in the heat-exchangers, thus decreasing their efficiency.
- **Intermediate cleaning** is done under sterile conditions and does not require a re-sterilization of the plant. This way the production period can be extended. Since an intermediate cleaning is not as efficient as normal one, it can only be performed a limited number of times between two normal cleanings.
- **Cleaning** will be done after a long production phase. The cleaning program depends on the product and it can vary with respect to time, flow, temperature and types of detergents. After a normal cleaning the plant must be re-sterilized.

A more detailed description of Steritherm and the simulation model in G2 can be found in IT4 (1990) and Christiansson & Ericsson (1989) respectively.

4.2 Development of the process model

Within this master thesis project only a part, approximately one half, of Steritherm was modeled for the on-line diagnosis. A schematic of the modeled part of Steritherm is shown in Figure 4.2. This schematic is only valid during the production phase, and thus MIDAS will not make an adequate diagnosis during the other phases. The chosen part consists of the "heart" of Steritherm, namely the heating and temperature control of the product. The measured variables are indicated in the figure as transmitters. In addition, the output signal, U, of the controller is measured.

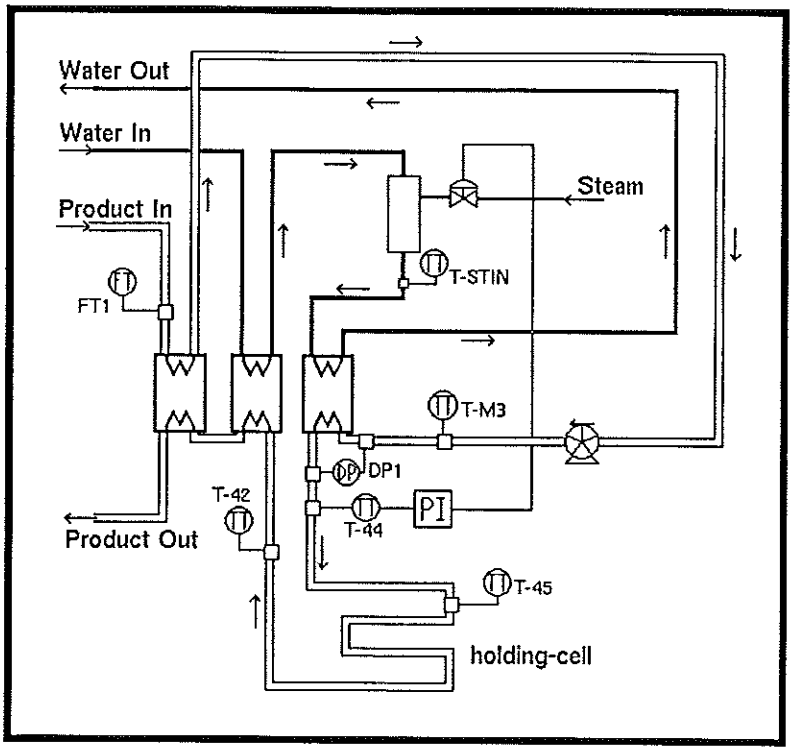

Figure 4.2 Schematic of the modeled part of the Steritherm process

4.2.1 The SDG

The process components that will be modeled are heat exchangers, a controller, a control valve, a steam injector, a centrifugal pump, and transmitters. Below, the sub-SDGs for these components will be displayed. For a more mathematically detailed derivation, see Oyeleye (1989), where most of the units are dealt with, though, in some cases, his models have been modified a little.

PI-controller

The model of a PI-controller (proportional and integral controller) is the same as the one used for the two coupled tanks in Section 2.2.2. X is the controlled variable, X_{sp} is the set point, E is the controller error, I is the integral part, and U is the controller output. Included faults are Controller Hi and Controller Lo. When these faults are present, the system will behave as if the set point of the controller is altered. The SDG is shown in Figure 4.3.

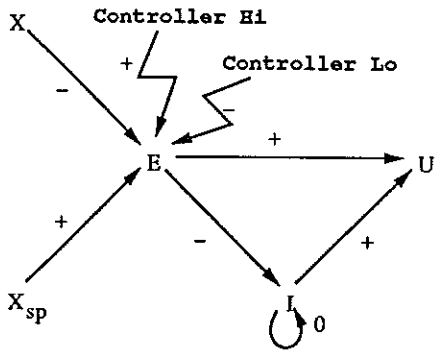

Figure 4.3 The SDG of a PI-controller

Control valve

A control valve can be described as a pipe with variable pipe resistance. This is modeled by the stem position, S_p, which is affected by the input signal, S_i from the associated controller. For the valve the only modeled fault is that the stem is stuck, either open or closed. When this happens, the input signal no longer will affect the stem position, and this is indicated by the possible zero sign at the corresponding arc in Figure 4.4. The pressure/flow configuration in the SDG can be found for all units acting as pipes.

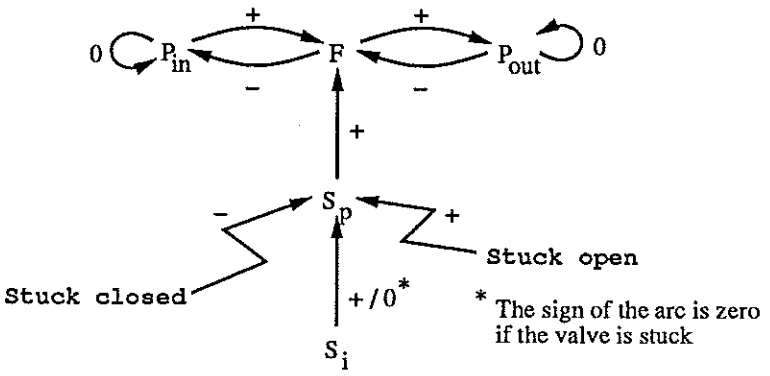

Figure 4.4 The SDG of a control valve

Heat exchanger (HTX)

The plate heat exchangers used in Steritherm can be described by the variables shown in Figure 4.5a. P stands for pressure, T for temperature and F for flow. The indexes w and c are for the warm and cold side respectively. An additional variable, Q, describes the rate of heat transfer from the warm to the cold side. The only malfunction considered is burn-on. Depending on what type of liquid that flows through the HTX, burn-on can occur either in one side, in both sides or in no side. For Steritherm, burn-on can occur only in the product line. It will affect the parameters U (the heat transfer coefficient) and R_s (the pipe resistance). The resulting SDG is shown in Figure 4.5b.

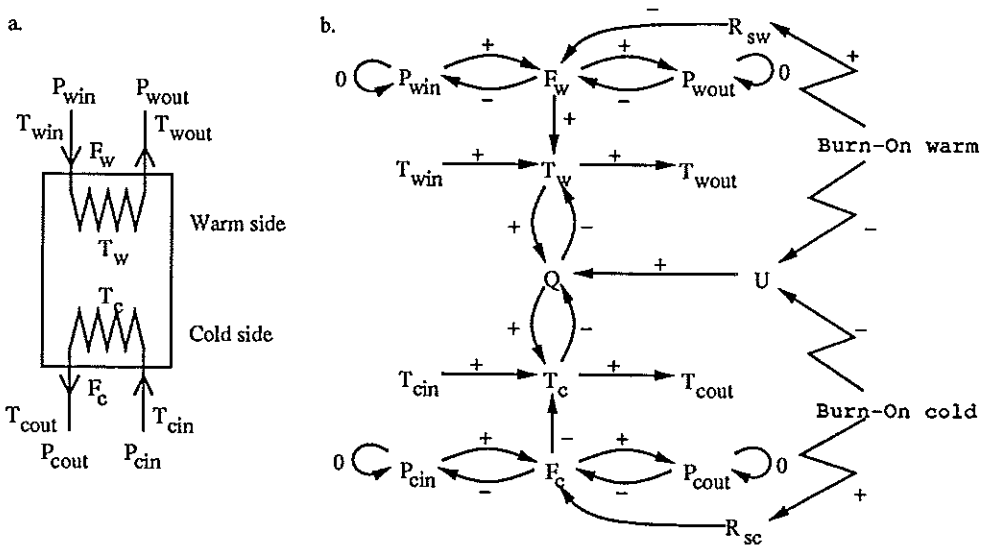

Figure 4.5 a. A schematic and b. the SDG of a HTX

Steam injector

In the steam injector hot steam is added to the incoming water to raise its temperature. The pressure/flow relation is analogous to the ones already described. Further, the temperature at the outlet of the steam injector will increase when either of the input temperatures or the steam inflow increases. If the water flow increases, though, the temperature of the output liquid will decrease, since then it consists of a greater amount of cold liquid. Since no faults are modeled for the steam injector, its SDG is as shown in Figure 4.6.

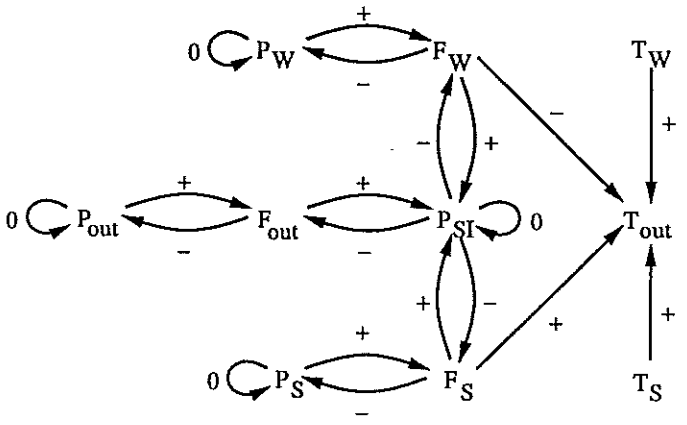

Figure 4.6 The SDG of the steam injector

Centrifugal pump

A simple model for the centrifugal pump is found in Figure 4.7. This is the last unit using the pressure/flow configuration. The flow is affected by the number of rotations per time unit, N. The only fault considered is a too fast or too slow rotation, leading to a high or low flow respectively.

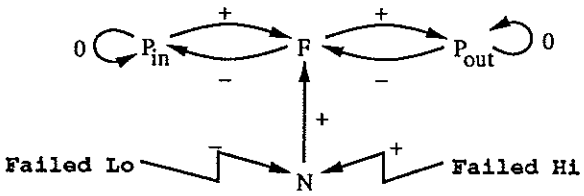

Figure 4.7 The SDG of a centrifugal pump

Transmitter

Sensors or transmitters are modeled since they are very likely to fail. For each sensor a positive or negative bias can occur. This leads to the SDG in Figure 4.8. X is the measured variable and X_m is the output signal from the transmitter.

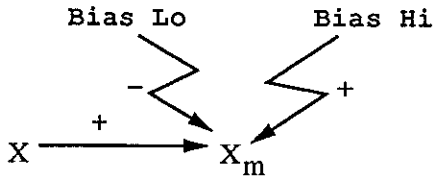

Figure 4.8 The SDG of a transmitter

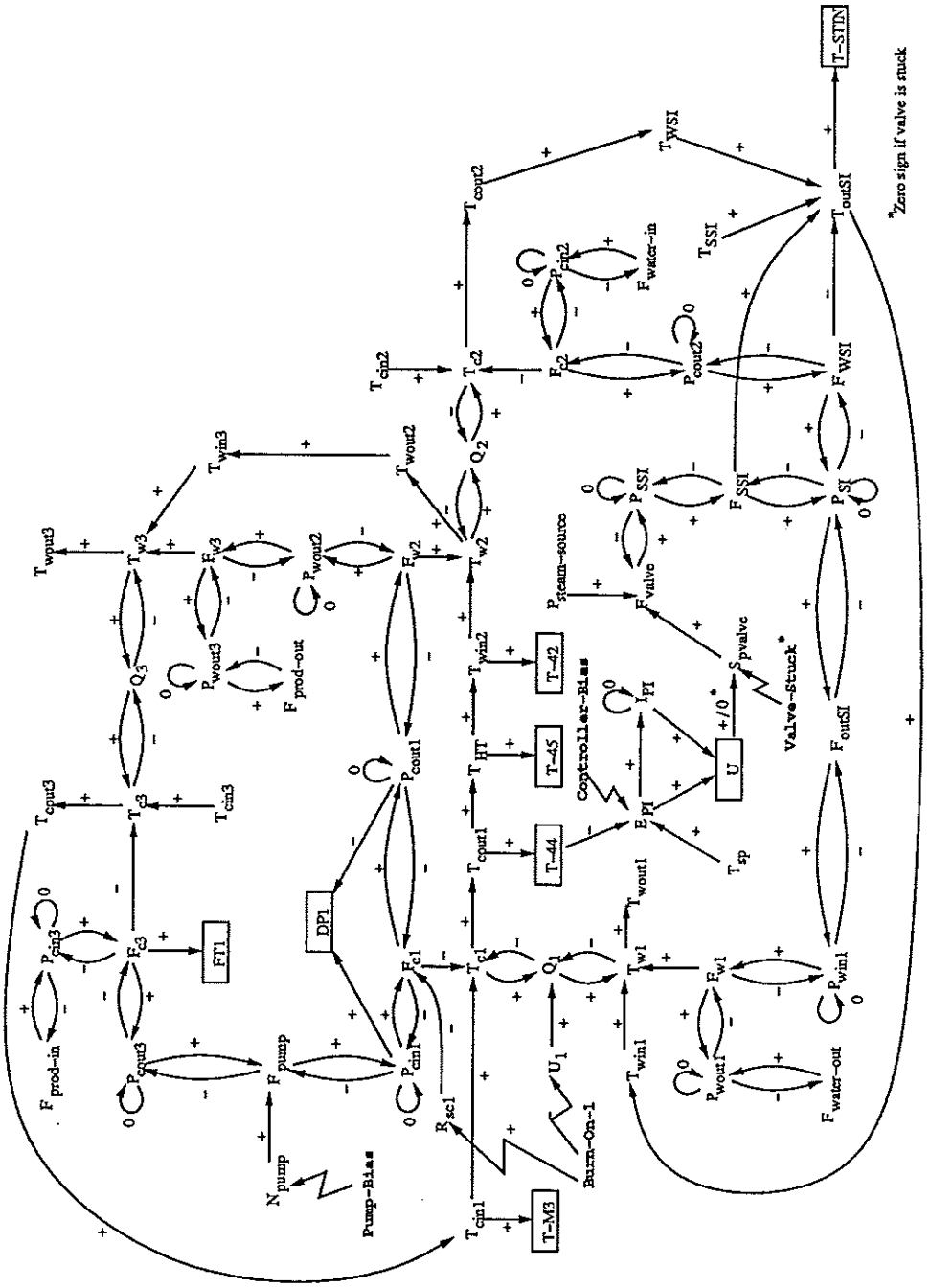

Figure 4.9 The SDG of the modeled part of Steritherm

Now that the SDGs for all the units have been developed, these will be put together to a SDG for the whole system. The variables of the components are chosen in a way so that adjacent units either share one variable, or have an obvious causal relationship. An example of shared variables is the connection of pipes, where the pressure of the adjacent pipes is the shared variable. Obvious causal relationship can be found where the temperature in one pipe affects the temperature in the pipe just downstream through a positive arc.

The resulting SDG for the process is shown in Figure 4.9. To avoid too much clutter, sensor bias faults are not included in the figure. Burn-on is just modeled for the final heater. To separate variables in different units, indexes

are added to the variable names. The HTXs have indexes 1, 2 and 3, counting from right to left in Figure 4.2. The other units have indexes PI, SI, HT (holding tube), valve and pump respectively. The sensor signals, displayed as boxes in the figure, are given names that are used for the measured variables in G2. In those cases where variables from different units overlap, one of the variable names has been chosen arbitrarily.

The SDG consists of several loops. Each pressure/flow circuit is in fact a local loop, as is the description of the heat transfer in the HTXs. There are also global loops, namely the controller loop and the loops coming from the thermal feedback in the liquid systems. Since the loops interfere, several cycles can be found. These are analyzed in the following section in order to find the compensatory and inverse variables.

4.2.2 The ESDG

When deriving the ESDG for the Steritherm process, the criteria and rules given in Section 2.2.2 were used. The SDG of the Steritherm model consists of only two strongly connected components (SCC). One SCC consists of the flows and pressures of the product line, and this contains no compensatory or inverse response. The other SCC, which is the major part of the system SDG, contains the controller, and this implies that at least some variables are compensatory. Since the thermal feedback loops are positive loops, and negative feedback loops also exist, variables that might display inverse response might be found. Though, when simulating the process, no inverse response was found for these variables.

The compensatory variables are found by looking at all the disturbance nodes that may be deviated by any modeled malfunction. There is no point in looking at other disturbance nodes, since MIDAS never can create an accurate diagnosis if these nodes are deviated. When the compensatory variables are found, the corresponding ESDG-arcs are added to the SDG. The resulting ESDG is shown in Figure 4.10. The ordinary SDG arcs from Figure 4.9 are drawn as dashed lines.

4.2.3 The event graph

After the ESDG has been developed it is possible to create the event graph that MIDAS uses when making the diagnosis. The four `potential-events` of each measured variable are created along with a `potential-root-cause` for every modeled malfunction. The fault propagation for all root causes is determined by finding primary deviation paths and successor paths in the ESDG, as is described in Section 2.2.3. According to this, the corresponding `local-cause-links` and `compiled-links` are created. When this is done for all root causes, :NOT-conditions are attached to the `compiled-links`. Each `compiled-link` has an attribute `not-conditions` which is an item-list. A fault is inserted in this list if the event just upstream the link is likely to happen when the fault is present, but the event just downstream the link is not. The last step is to create :ONLY-IF-TRANSIENT links. These will be created for all back-to-normal events that are not yet connected to any other event, thus indicating that these events will not occur until the fault responsible for the deviation has disappeared.

When these steps have been executed for the Steritherm ESDG in Figure 4.10, one resulting event graph can be seen in Figure 4.11. The figure

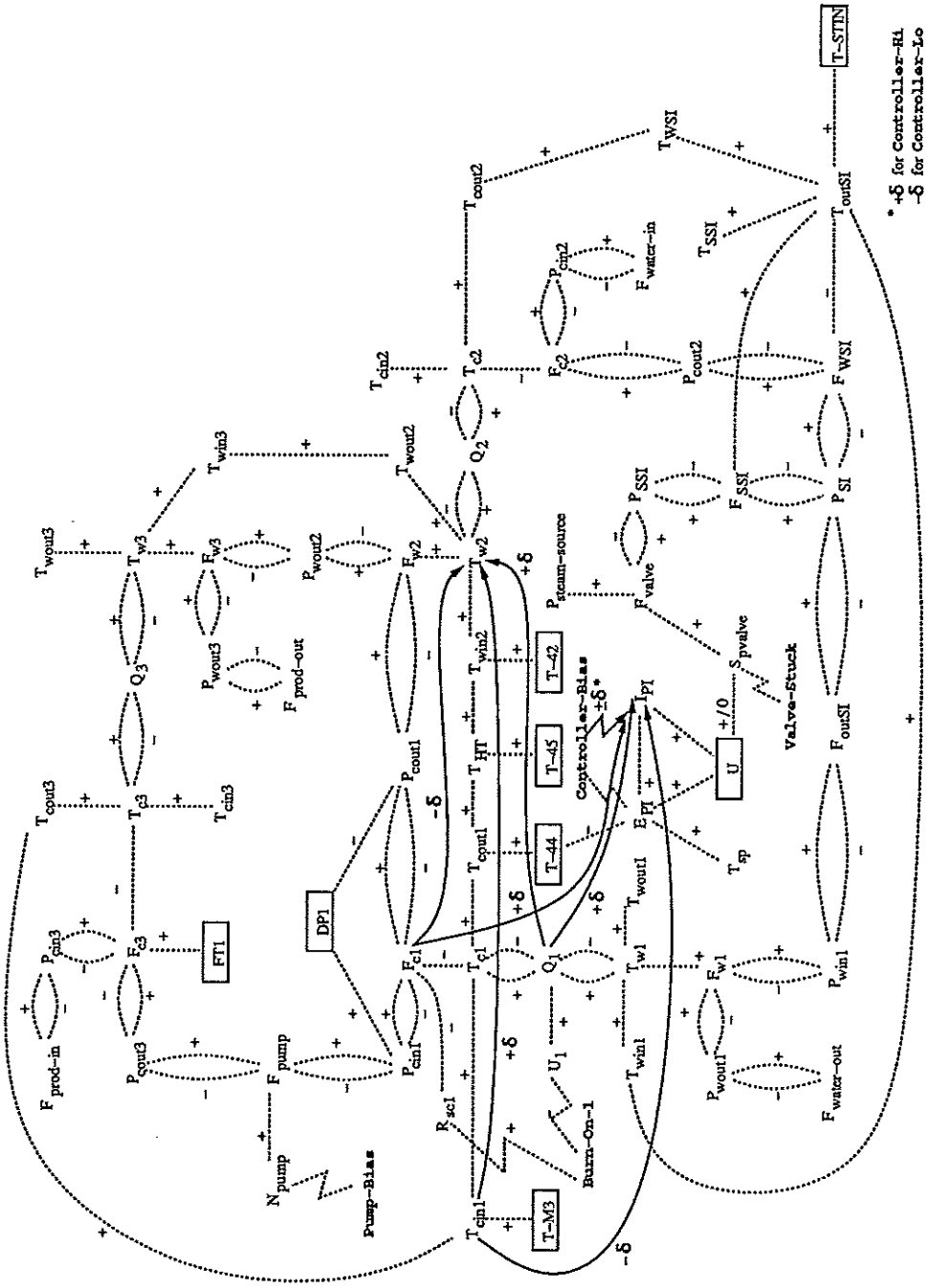

Figure 4.10 The ESDG of the process

is a hardcopy of the process model in the G2 implementation, and the numbers that are attached to most of the `compiled-links`, are the numbers of the `potential-root-causes` that are :NOT-conditions of the links. In this case all `compiled-links` pointing at a back-to-normal event, are :ONLY-IF-TRANSIENT links.

Note that this is not the only possible event graph describing the fault propagation in Steritherm. As mentioned in Section 2.2.3 it is possible to handle the causality in different ways, and this will lead to different, but equivalent, event graphs. It is desirable to have as few links as possible to improve efficiency during diagnosis. Further, the causal ordering in the ESDG

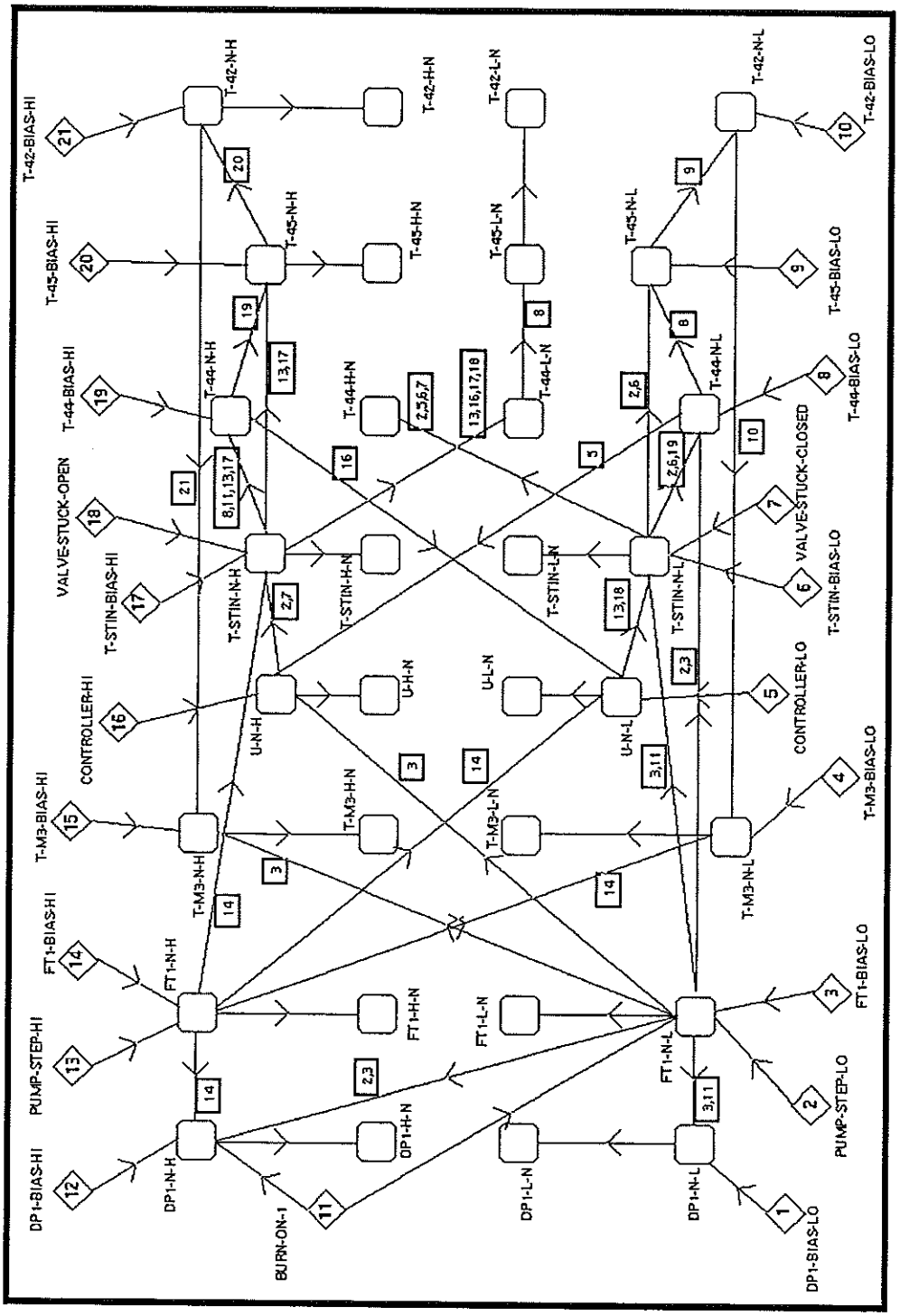

Figure 4.11 The event graph of the process

should as much as possible be retained, though it may be changed in order to improve the robustness to out of order events.

When the event graph has been implemented in G2 and the monitors have been tuned with sufficient accuracy, it is possible to start the on-line diagnosis. In the following section a number of fault simulations are performed when MIDAS is working against the simulated Steritherm process in G2.

4.3 Fault scenario simulations

Since one diagnostic methodology, the Diagnostic Model Processor, has previously been implemented in the G2 prototype, mechanisms for introducing faults in the simulation model already exist. The faults that are simulated include bias in all the sensors and the PI-controller, burn-on in the final heater, and a change in pump effect. All these faults, except burn-on, can be simulated as step faults or ramp faults. In addition, I have made it possible to simulate that the valve is stuck, either at an open or a closed position.

Before the fault simulation can be started, the process must reach its nominal steady state. Then the monitors must be tuned to detect events around this steady state. Depending on the process characteristics, it may be rather difficult to achieve optimal diagnostic behavior. MIDAS works at its best if the steady state is reached reasonably fast, and if the process is critically damped, i.e. it displays no unpleasant transient behavior. Unfortunately, Steritherm (or at least the simulation model) does not satisfy either of these conditions. When a process fault, for instance the valve is stuck open, is present, the variables deviate quite fast from the steady state, but when the fault is removed, some variables approach the steady state very slowly. The same thing happens during the start-up, and this implies that it can take quite a long time before the MIDAS diagnosis can be started. However, this might not be a too serious restriction, since in normal cases Steritherm will run for several hours in the production mode, and hopefully the start-up will take only a small portion of that time.

More important is the fact that when certain faults are present, some variables may display unexpected responses due to oscillatory system behavior. Different solutions to this problem can be found. Either one can try to tune the controller in a way that the underdamped responses disappear, or one could include the underdamped behavior in the process model. The first solution is hard, if at all possible, to achieve with a PI-controller, and the second solution will cause the event graph to contain a lot more links, which will drastically decrease the performance of MIDAS. Instead, I have chosen to tune the monitors in a way that they require rather large deviations to detect high and low states. Further, in the cases where the variables have high frequent transients, typically the controller output and the controlled temperatures, the lag times of the smoothing filters are set quite large. The disadvantage of this solution is that MIDAS may become sensitive to the fault magnitudes of some malfunctions. Though, when simulating faults with different magnitudes, the interval where MIDAS makes an accurate diagnosis proved to be fairly large.

The simplest type of fault that is simulated is sensor bias for uncontrolled variables. When, for example, T-STIN-BIAS-HI is simulated, and MIDAS is told to log the results of the evidence evaluation only, the displayed fault message is as shown in Figure 4.12.

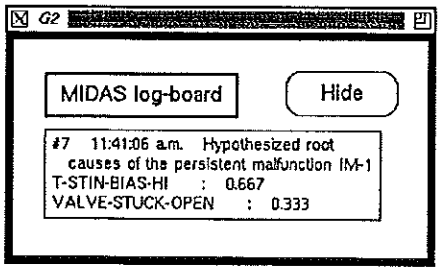

Figure 4.12 Log message when T-STIN-BIAS-HI is simulated

The reason why T-STIN-BIAS-HI is ranked higher than VALVE-STUCK-OPEN is that the a priori probability is set higher for sensor bias than for other faults. The created inferred malfunction, IM-1, only contains the event T-STIN-N-H, and since this sensor bias will not cause any process disturbance, no other events can be expected.

If T-STIN-BIAS-HI is not removed before an additional fault, T-45-BIAS-LO, is simulated, MIDAS will create a second IM since the two events T-STIN-N-H and T-45-N-L cannot be linked together. This can be verified by looking at the process model in Figure 4.11. The resulting fault messages are shown in Figure 4.13.

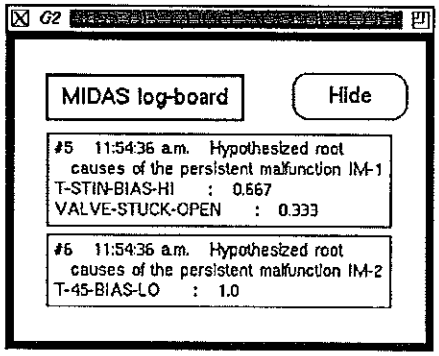

Figure 4.13 Log message when T-STIN-BIAS-HI and T-45-BIAS-LO are simulated

In this case MIDAS rightly concludes that the faults T-STIN-BIAS-HI and T-45-BIAS-LO exists simultaneously.

If the low sensor bias of T-45 is corrected and replaced with a high sensor bias, the created event, T-45-N-H, will be linked to IM-1 as is indicated in Figure 4.14. The fault message is displayed in the upper right corner in the figure.

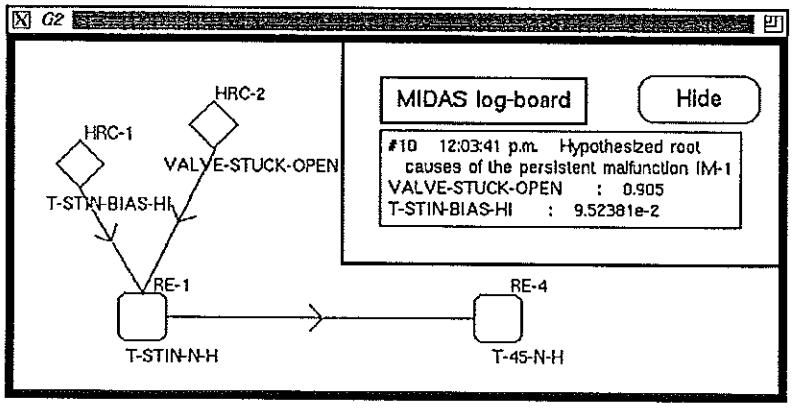

Figure 4.14 Log message when T-STIN-BIAS-HI and T-45-BIAS-HI are simulated

This is an example of a situation when MIDAS will create an inaccurate diagnosis. T-45-BIAS-HI is not even a possible candidate, and VALVE-STUCK-OPEN is ranked much higher than T-STIN-BIAS-HI, since the latter is blocked by the compiled link between the events (compare to Figure 4.11). MIDAS simply regards the two events as the start of the fault propagation for an open, stuck valve, and this is built upon the assumption that simultaneous, independent faults are very rare.

So far the fault scenarios have demonstrated how MIDAS deals with a priori probabilities and multiple malfunctions. Now some of the process faults will be simulated. These are faults that will lead to process state changes and not only deviations in the sensor signals, thus causing chains of events to occur. In the cases where a fault may happen in two directions, e.g. PUMP-STEP-LO and PUMP-STEP-HI, only one of the faults are simulated here. However, all modeled faults have been simulated and the created diagnoses have been verified. In the following examples only the most likely fault in every IM will be logged. For a better understanding, the reader may compare each fault simulation to the process model in Figure 4.11.

The only sensor bias fault that will cause process state changes, is the T-44 bias. Since T-44 is a controlled variable it will indeed affect the whole process. In Figure 4.15 a positive ramp fault is simulated for T-44.

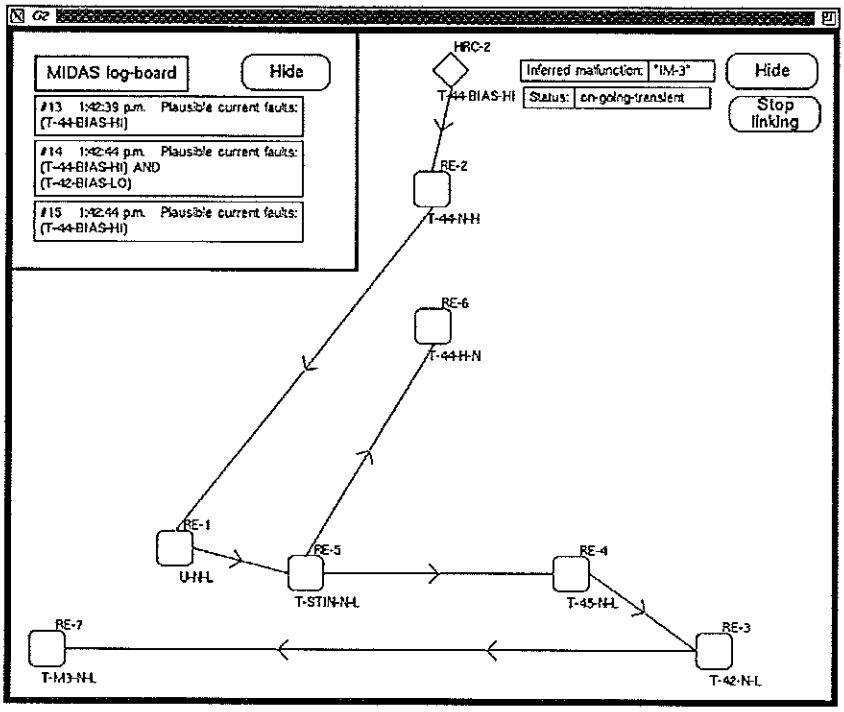

Figure 4.15 A simulation of a ramp fault in T-44

The detected events follow the expected behavior, even if they appear out of order as is indicated by the event numbering in the figure. However, this does not affect the diagnosis very much, since RE-2 can be predicted when RE-1 is detected. When RE-3 is detected, MIDAS draws the wrong conclusion that T-42 is biased low (see the second fault message). But in fact, RE-4 is detected within the same clock tick, and since RE-5 in this case can be predicted, all events will merge into the same cluster, and the inaccurate diagnosis is immediately corrected.

Next, a low pump effect is simulated in Figure 4.16. In this example DP1 and FT1 get low at the same time, and when both events have been handled, MIDAS concludes that PUMP-STEP-LO is present. When the other three events occur they all confirm the current hypothesis.

The simulations of controller faults and a stuck valve are not very exciting, since they only will produce a chain of low or high temperatures. Thus, the simulations of CONTROLLER-HI and VALVE-STUCK-CLOSED are presented in Figures 4.17 and 4.18 respectively, without any further comments.

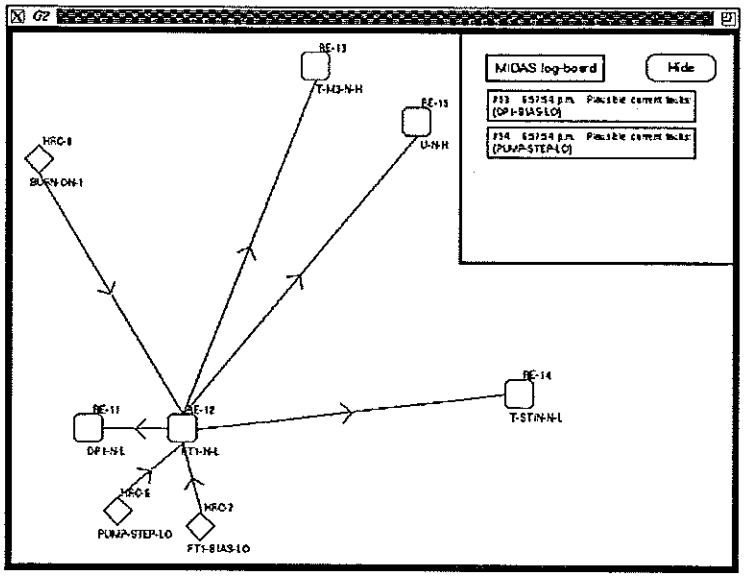

Figure 4.16 A simulation of a low pump effect

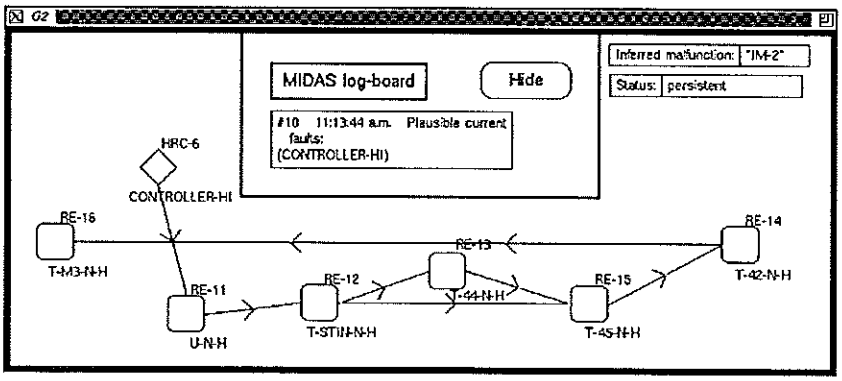

Figure 4.17 Results from a simulation of CONTROLLER-HI

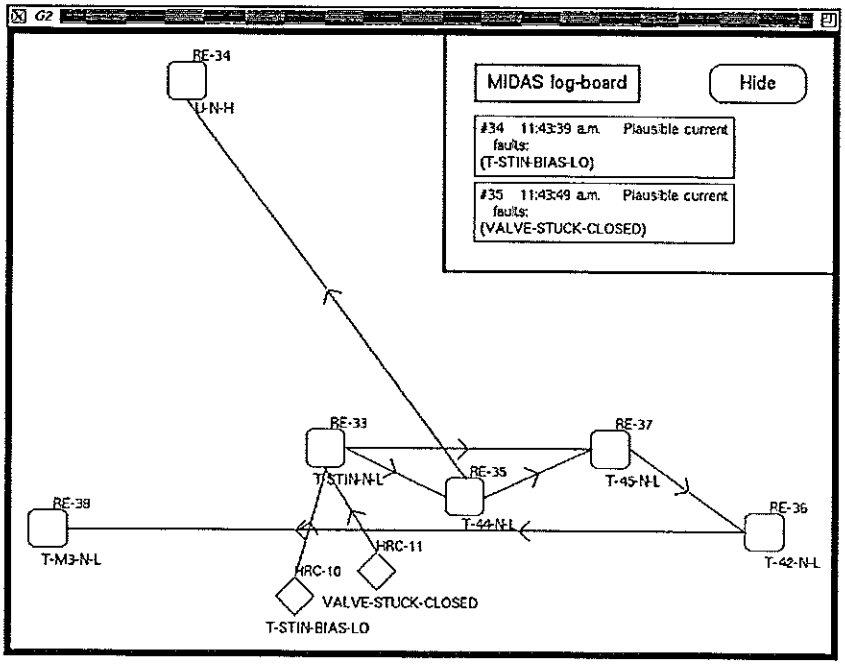

Figure 4.18 A simulation of VALVE-STUCK-CLOSED

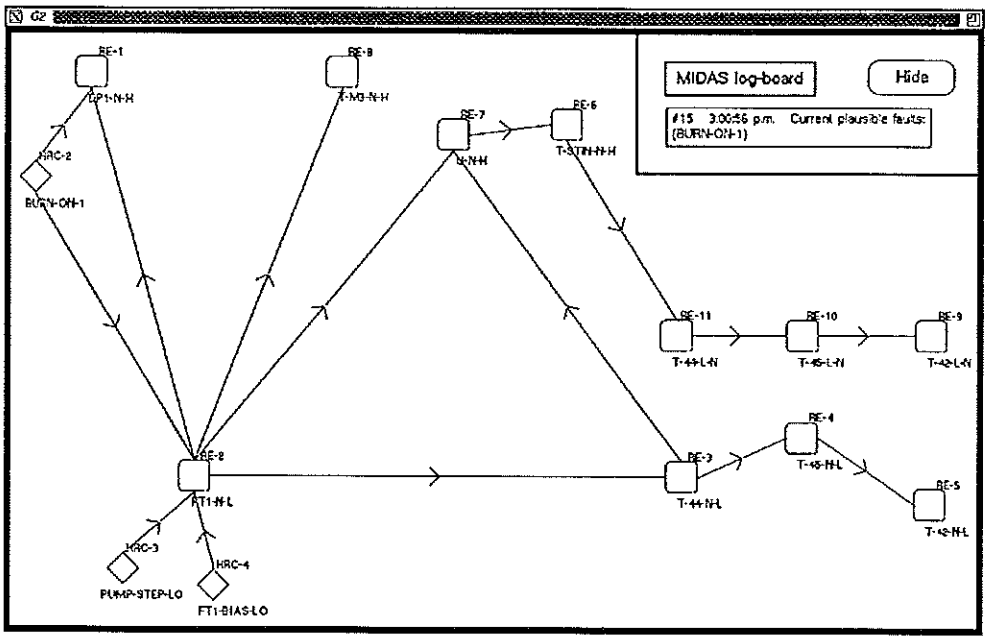

Figure 4.19 Burn-on in the final heater

The last fault to be simulated is the burn-on in the final heater. This is simulated both by decreasing the heat transfer coefficient and increasing the pipe resistance in the product line inside the HTX. The initial fault propagation is displayed in Figure 4.19. Thanks to successful interrogation, a correct diagnosis is created during the whole simulated period. After the snapshot in the figure, some unexpected events occur due to discrepances between the simulation model and the process model. However, during the initial phase, which in fact lasts for quite a long time, MIDAS will detect the burn-on and then the plant will be instantly cleaned. Then MIDAS must be turned off, and the unexpected events may never occur.

The results of the fault simulations are that MIDAS in all cases succeeds in creating an accurate diagnosis, if not two malfunctions with interfering symptoms appear simultaneously. However, a more thorough tuning of the monitors is required if the transient, inaccurate conclusions will not be drawn. Before the simulations above took place, not too much time was spent on the tuning.

5. Results and Conclusions

In this chapter some reflections on the thesis project in general, and the features of MIDAS in particular, will be made. The project has taken about 500-600 hours, far most of this time has been spent on the G2 implementation of MIDAS.

5.1 Reflections on MIDAS

The goal of this project has been to develop a qualitative fault diagnosis system for plant processes, and to examine the diagnostic performance of such a system. Mainly, one must say that the implementation of MIDAS works as one could expect. That is, for certain types of faults, where the fault propagation is unique and easy to determine, MIDAS will create an accurate and, as the fault propagates, more refined diagnosis. In Section 2.5 some features, limitations and possible improvements of MIDAS were mentioned. One of the suggested, and I think most valuable improvements to be done, is to include some information of time in the process model, thus telling when a certain event is likely to happen for different faults. Though, in many cases it is hard, or even impossible, to determine the delay along the causal arcs. It can of course be done by a series of independent simulations, but one of the strong points with MIDAS is that the event graph should be derived from a detailed, though qualitative, model of the process.

Like any other diagnosis system, MIDAS must have a correct process model to be able to create an accurate diagnosis. That is, the fault propagation must be unambiguous and preferably not too complex for any variable. For instance, it is hard to include oscillatory process behavior in the process model without decreasing the diagnostic accuracy and resolution.

One thing MIDAS is really good at, is to pick the most likely fault out of a set of candidates having the same event as their first symptom. This is achieved with the conditional links, the a priori probabilities of different faults, and the possibility to assign a probability of true detection to events. In the current version, these probabilities must be guessed quite roughly, but if the monitors responsible for event detection are further developed, they may actually compute the probability of true event detection through statistical knowledge.

During the fault simulations in Section 4.3, the time spent on event interpretation, evidence evaluation etc, was very short, normally shorter than one clock tick (1 second) in G2. That is, the performance with respect to time of the whole system, including simulator and control system, depend very little on the diagnosis. Though, when many events have been detected and the hypothesis model grow, the diagnosis will naturally take longer time. This is one reason why oscillatory responses are unsuitable, since these cause a lot of events, but, except for the first ones, these will not provide any useful information. The size of the process model, though, will not affect the time spent on diagnosis.

The perfect process to apply MIDAS to, would be a fairly big and critically damped process, with an average delay associated with the compiled links of

a few seconds. There may be a great number of root causes with different fault propagation patterns pointing at each event. For such a process I guess MIDAS will perform better than most diagnosis methodologies.

I guess that the best diagnostic performance in a diagnosis system is achieved by combining qualitative causal reasoning with quantitative information. When I have studied the performance of MIDAS, I have only used a pure qualitative process model. The built-in possibility of including constraints in the diagnosis will probably lead to more reliable conclusions.

5.2 Reflections on G2

Working with G2 is quite different from conventional programming. Much of the data structure is not created by traditional programming, but is instead built by creating objects, connections, buttons etc. using mouse interaction. This, combined with the graphical representation of most of the items, makes it very nice to work with G2.

For most mid-sized applications, the response time of G2, when running on a Sun Sparcstation 2, is not very disturbing. However, in large applications including simulation, e.g. the Steritherm prototype, G2 runs about 1.5-2 times slower than real time. This is of course very much depending on the used platform. The speed will probably increase when new versions of G2 are released.

Being an expert system shell, G2 is largely emphasized on the use of rules. In my implementation, though, I use rules basically to trigger the event interpreter, which consists of approximately 50 procedures. Another 15 procedures handle administration, like pure graphical routines etc. The size of the knowledge-base of MIDAS is about 270 kBytes (4500 lines). The process model of Steritherm occupies another 55 kBytes (850 lines).

One feature in G2 that I have made frequent use of, is the possibility to create and delete objects and connections dynamically. All objects inside the hypothesis model are in fact created during run-time. For structural purposes it would have been nice if workspaces and subworkspaces also could be created dynamically, but this is not supported in the current version of G2.

There are a number of things that would have made it even better to work with G2. For example, it is hard to understand the restriction that names cannot be given to dynamically created objects. Further, all procedures, data etc. are globally visible, since it is not possible to modularize the knowledge-base. Even though G2 has good ways of relating objects, primarily with connections and relations, a simple pointer, like the ones in other object oriented languages, would be convenient.

The overall experience of G2 is that it is easy to work with and that prototypes can be developed in a very short time. The nice graphics, including the animation possibility, and the built-in buttons, also make it easy to have a nice operator interface without too much effort.

Appendix A

G2 – a Tool for Real-Time Expert Systems

G2 (trademark of Gensym Corporation) is the most widely spread and technically most advanced real-time expert system tool available today. It is implemented in Common Lisp and runs on several platforms, in this project a Sun Sparcstation 2 has been used.

Technical Description

The main parts of G2 are: the knowledge-base, a real-time inference engine, a procedure language, a simulator, the development environment, the operator interface, and optional interfaces to external on-line data servers. The normal way of using G2 is to create a knowledge-base for a desired application off-line, and then run this knowledge-base in real-time.

Classes and objects

G2 is a strictly object oriented programming environment. This means that all components in G2, including rules, procedures, graphs, buttons, objects etc., are items. The items are organized into a hierarchy. All items have a graphical representation through which they are manipulated by mouse and menu operations. Operations exist for moving an item, cloning it, changing its size and color etc. The only type of item that the user has full control over, are the G2 objects. The user may define and manipulate objects in order to create the data structure for a certain application.

Objects are used to represent the different concepts of an application. They can represent arbitrary concepts, i.e. both physical concepts such as process components, and abstract ones. The objects are organized into a class hierarchy, i.e. only single inheritance is allowed. The class definition, or using G2 terminology, the object definition, defines the attributes that are specific to the class and the look of the icon. Icons can be created with an interactive icon editor. The attributes describe the properties of the object. The values of the attributes may be

- constants,
- variables,
- parameters,
- lists, or
- other objects.

Constants can be numbers, symbolic values, logical values (i.e. true or false), and text strings. During run-time, constants can only be changed explicitly by the user. Variables or parameters are used to represent entities whose values change during run-time. Variables are defined from four basic predefined classes:

- quantitative variables, i.e. integer- or real-valued variables,
- symbolical variables,
- logical variables, and
- text variables.

Parameters can be classified in the same way. The main difference between variables and parameters is that a parameter always has a value, whereas a variable must explicitly be assigned a value one way or another. A variable also has a validity interval, which specifies for how long a newly assigned value of the variable will be valid.

Lists may contain arbitrary values. The allowed values in a list can be specified. It is possible to have objects as the values of attributes in other objects. In that case, the attribute objects have no iconic representation.

Objects can be static, i.e., they are explicitly created by the developer, or dynamic, i.e., they are created dynamically during run-time. Dynamic objects can also be deleted during run-time. The G2 language contains actions to move, rotate, and change the color of an object. Using this, animations can be created.

Composite objects, i.e., objects that have an internal structure composed of other objects, can be created using objects as the value of attributes. It is, however, not possible to at the same time have a iconic representation for these objects. If such a representation is desired this has to be implemented using the subworkspace concept. In G2 each object and most items may have an associated subworkspace. In this (sub-)workspace arbitrary items may be positioned. The internal structure of an object can be represented on its subworkspace. It is, however, not possible to define that an object should have an internal structure of this type in the class definition.

Relating objects

G2 has different ways of defining relations between objects. One way is to let an object have attributes that are lists containing other objects. An object can be a member of any number of lists, thus very complex relations between objects can be defined this way.

A more direct way of relating two objects is to use connections. These are primarily used to represent physical connections, e.g., pipes or wires. It is, however, also possible to let connections represent abstract relations among objects. Connections have a graphical representation and may have attributes. They are defined in terms of a connection hierarchy. Both unidirectional and bidirectional connections are allowed. Connections can be used in G2 expressions for reasoning about interconnected objects in a variety of ways. A connection is attached to an object either at a pre-specified location, a port, or anywhere on the object. Connections can, like objects, be either static or dynamic.

A third way of relating objects is to use a special type of G2 item called relations. These can only be created at run-time and have no graphical representation. They have no corresponding relation hierarchy and cannot have attributes. Relations can be specified as being one-to-one, one-to-many, many-to-one, and many-to-many. They may actually relate any kind of items, not objects only. Relations can be used in G2 expressions in a similar way as connections.

The inference engine

G2 rules can be used to encapsulate an expert's heuristic knowledge of what to conclude from conditions and how to respond to them. Five different types of rules exist.

- **If** rules
- **When** rules
- **Initially** rules
- **Unconditionally** rules
- **Whenever** rules

When rules are a variant of ordinary **If** rules that may not be invoked through forward chaining or cause backward chaining. **Initially** rules are run when G2 is initialized. **Unconditionally** rules are equivalent to **If** rules with the rule conditions always being true. **Whenever** rules allow asynchronous rule firing as soon as a variable receives a new value, fails to receive a value within a specified time-out interval, when an object is moved, or when a relation is established or deleted.

The rule conditions contain references to objects and their attributes in a natural language style syntax. Objects can be referenced through connections with other objects. G2 supports generic rules that apply to all instances of a class. The G2 rule actions makes it possible to conclude new values for variables, send alert messages, hide and show workspaces, move, rotate, and change color of icons, create and delete objects, start procedures, explicitly invoke other rules, etc. G2 rules can be grouped together and associated with a specific object, a class of objects, or a user-defined category. This gives a flexible way of partitioning the rule-base. The following is an example of a G2 rule,

```
for any water-tank
 if the level of the water-tank < 5 feet and
 the level-sensor connected to the water-tank is working
 then conclude that the water-tank is empty
 and inform the operator that
 "[the name of the water-tank] is empty"
```

The real-time inference engine initiates activity based on the knowledge contained in the knowledge base, simulated values, and values received from sensors or other external sources. In addition to the usual backward and forward chaining rule invocation, rules can be invoked explicitly in several ways. First, a rule can be scanned regularly. Second, by a focus statement all rules associated with a certain focal object or focal class can be invoked. Third, by an invoke statement all rules belonging to a user defined category, like safety or startup, can be invoked.

Internally the G2 inference engine is based on an agenda of actions that should be performed by the system. The agenda is divided into time slots of 1 second's length. After execution, scanned rules are inserted into the agenda queue at the time slot of their next execution. Focus and invoke statements causes the invoked rules to be inserted in the agenda at the current time slot.

Procedures

G2 contains a Pascal-style procedural programming language. Procedures are started by rule actions. Procedures are reentrant and each procedure invocation executes as a separate task. Procedures can have input parameters and return one or several values. Local variables are allowed within a procedure.

The allowed procedure statements include all the rule actions, assignment of values to local variables, branching statements (**If-then-else** and **case**), iteration statements (**repeat** and **for**), **exit if** statements to exit loops, **go to** statements, and **call** statements to call another procedure and await its result. The **for** loops may be either numeric or generic for a class, i.e., they execute a statement or set of statements once for each instance of the class.

Procedures are executed by G2's procedure interpreter. The procedure interpreter cannot be interrupted by other G2 processing, i.e., the inference engine or the simulator. Other processing is only allowed when the procedure is in a wait state. A wait state is entered when a **wait** statement is executed, when the statement **allow other processing** is executed, and when G2 collects data from outside the procedure for assigning to a local variable.

Simulation

G2 has a built-in simulator which can provide simulated values for variables. The simulator is intended to be used both during development for testing the knowledge base, and in parallel during on-line operation. In the latter case, the simulator can be used, e.g., to implement filters for estimation of signals that are not measured.

The simulator allows for differential, difference, and algebraic equations. The equations can be specific to a certain variable of apply to all instances of a variable class. Each first-order differential equation is integrated individually with individual and user-defined step sizes. The numeric integration algorithms available are a simple forward Euler algorithm with constant step size and a fourth order Runge-Kutta algorithm, also with fixed step size. GSPAN, an interface between G2's simulator and external simulators is available as a separate product.

Development interface

G2 has a nice graphics-based development environment with windows (called workspaces), popup menus, and mouse interaction. Input of rules, procedures, and other textual information is performed through a structured grammar editor. The editor prompts all valid next input statements in a menu. Using this menu the majority of the text can be entered by mouse-clicking. It is, however, also possible to use the keyboard in an ordinary way. The editor has facilities for Macintosh style text selection, cut, paste, undo, redo, etc.

The Inspect facility allows the user to search through the knowledge base for some specified item. The user can go to the item, show all matching items on a temporary workspace, write them out on a report file, highlight them, and make global substitutions.

G2 has facilities for tracing, stepping, and adding breakpoints. The internal execution of G2 can be monitored using meters.

End-user Interface

G2 has facilities for building end-user interfaces. Colors and animation can be used. An object icon is defined as a set of layers whose colors can be changed independently during run-time. The meta-color transparent makes it possible to dynamically hide objects. Different user categories can be defined and the behavior with respect to which menu choices that are allowed can be set for each category. It is also possible to define new menu choices.

G2 contains a set of predefined displays such as readouts, graphs, meters, and dials that can be used to present dynamic data. G2 also has a set of predefined interaction objects that can be used for operator controls. Radio buttons and check boxes can be used to change the values of symbolical and logical variables by mouse clicking. An action button can be associated with an arbitrary rule action which is executed when the button is selected. Sliders can be used to change quantitative variables and type-in boxes are used to type in new variable values.

External interfaces

G2 can call external programs in four different ways: using foreign function calls, and using GFILE, GSPAN, and GSI. On some platforms, external C and Fortran functions may be called from within G2. GFILE is an interface to external data files that allows G2 to read sensor data from the files. GSPAN is the interface between G2 and external simulators. GSI is Gensym's standard interface. It consists of two parts; one part written in Lisp that is connected to G2 and one part written in C to which the user can attach his own functions for data access. On the same machine, the two parts communicate using interprocess communication media such as pipes or mailboxes. On different machines, TCP/IP – Ethernet is used.

Drawbacks

The main problems with G2 stem from the fact that G2 is a closed system. G2 can only be interfaced with other program modules through the predefined interfaces. The G2 environment in itself is also a quite closed world. It is impossible to modify the way that G2 operates internally. If what G2 provides in terms of, e.g., graphics, class – object structures, etc., is insufficient, nothing can be done about it.

G2 can not be modularized. Hence, it requires quite powerful computers even if only a small subset of the functionality is used within an application.

Although G2 is fast compared to many expert system tools, it can be too slow for certain applications. The smallest time unit is one second. For applications that require faster response, G2 is inadequate.

Appendix B

Object-definitions in the process model

PM-COMPONENT, an object-definition

Class	pm-component
Superior class	object
Attributes specific to class	none

POTENTIAL-EVENT, an object-definition

Class	potential-event
Superior class	pm-component
Attributes specific to class	associated-monitor is given by a symbolic-parameter; prior-state is given by a symbolic-parameter; consequent-state is given by a symbolic-parameter

POTENTIAL-ROOT-CAUSE, an object-definition

Class	potential-root-cause
Superior class	pm-component
Attributes specific to class	prior-probability is given by a probability

POTENTIAL-EVENT-LIST, an object-definition

Class	potential-event-list
Superior class	item-list
Attributes specific to class	none
Default settings	element type for item-list: potential-event; allow duplicate elements for g2-list: no

POTENTIAL-ROOT-CAUSE-LIST, an object-definition

Class	potential-root-cause-list
Superior class	item-list
Attributes specific to class	none
Default settings	element type for item-list: potential-root-cause; allow duplicate elements for g2-list: no

PROBABILITY, an object-definition

Class	probability
Superior class	quantitative-parameter
Attributes specific to class	none
Default settings	initial value for quantitative-parameter: 1.0

COMPILED-LINK, a connection-definition

Class	compiled-link
Superior class	connection
Attributes specific to class	not-conditions is an instance of a potential-root-cause-list; only-if-transient is given by a logical-parameter
Cross section pattern	1 foreground

LOCAL-CAUSE-LINK, a connection-definition

Class	local-cause-link
Superior class	connection
Attributes specific to class	none
Cross section pattern	1 blue

Object-definitions in the hypothesis model

HYPOTHESIS-MODEL, an object-definition

Class	hypothesis-model
Superior class	object
Attributes specific to class	inferred-malfunctions is an instance of an inferred-malfunction-list; old-inferred-malfunctions is an instance of an inferred-malfunction-list; no-of-recorded-events is given by a quantitative-parameter; no-of-hypothesized-root-causes is given by a quantitative-parameter; no-of-inferred-malfunctions is given by a quantitative-parameter; plausible-faults is an instance of a hypothesized-root-cause-list; free-workspaces is an instance of a symbol-list; unconnected-events is an instance of an event-list

HM-COMPONENT, an object-definition

Class	hm-component
Superior class	object
Attributes specific to class	id is an instance of an item-list

INFERRED-MALFUNCTION, an object-definition

Class	inferred-malfunction
Superior class	hm-component
Attributes specific to class	stop-linking is given by a logical-parameter; correctable-events is an instance of an event-list; events-explained is an instance of an event-list; source-events is an instance of an event-list; hypothesized-root-causes is an instance of a hypothesized-root-cause-list; existing-monitors is an instance of a monitor-list; status is given by a symbolic-parameter; shrink is 1.0; my-workspace is given by a symbolic-parameter

HYPOTHESIZED-ROOT-CAUSE, an object-definition

Class	hypothesized-root-cause
Superior class	hm-component
Attributes specific to class	associated-root-cause is given by a symbolic-parameter; primary-symptoms is an instance of a potential-event-list; opposing-events is an instance of an event-list; supporting-events is an instance of an event-list; prior-probability is given by a probability; likelihood is given by a probability

EVENT, an object-definition

Class	event
Superior class	hm-component
Attributes specific to class	associated-monitor is given by a symbolic-parameter; associated-event is given by a symbolic-parameter; state is given by a symbolic-parameter; copies is an instance of an event-list; symptom-of is an instance of an inferred-malfunction-list; explained-by is an instance of an event-list; explains is an instance of an event-list; probability-of-accurate-detection is given by a probability; probability-of-inaccurate-detection is given by a probability; superseded-by is an instance of an event-list; time-of-detection is given by a text-parameter; time-of-cessation is given by a text-parameter

RECORDED-EVENT, an object-definition

Class	recorded-event
Superior class	event
Attributes specific to class	none

EXPECTED-EVENT, an object-definition

Class	expected-event
Superior class	event
Attributes specific to class	none

LATENT-EVENT, an object-definition

Class	latent-event
Superior class	event
Attributes specific to class	none

COPIED-EVENT, an object-definition

Class	copied-event
Superior class	event
Attributes specific to class	original is an instance of an event-list

INFERRED-MALFUNCTION-LIST, an object-definition

Class	inferred-malfunction-list
Superior class	item-list
Attributes specific to class	none
Default settings	element type for item-list: inferred-malfunction; allow duplicate elements for g2-list: no

HYPOTHESIZED-ROOT-CAUSE-LIST, an object-definition

Class	hypothesized-root-cause-list
Superior class	item-list
Attributes specific to class	none
Default settings	element type for item-list: hypothesized-root-cause; allow duplicate elements for g2-list: no

EVENT-LIST, an object-definition

Class	event-list
Superior class	item-list
Attributes specific to class	none
Default settings	element type for item-list: event; allow duplicate elements for g2-list: no

ITEM-NAME, a message-definition

Class	item-name
Superior class	message
Attributes specific to class	none

MONITOR, an object-definition

Class	monitor
Superior class	object
Attributes specific to class	busy is given by a logical-parameter; raw-meas is given by a quantitative-parameter; next-sample is given by a quantitative-parameter; sample-rate is given by a quantitative-parameter; filtered-value is given by a quantitative-parameter; filtering-time is given by a quantitative-parameter; trend-time is given by a quantitative-parameter; prediction-time is given by a quantitative-parameter; predicted-value is given by a quantitative-parameter; upper-threshold-for-normal is given by a quantitative-parameter; lower-threshold-for-normal is given by a quantitative-parameter; threshold-for-high is given by a quantitative-parameter; threshold-for-low is given by a quantitative-parameter; current-state is given by a symbolic-parameter; previous-state is given by a symbolic-parameter; potential-events is an instance of a potential-event-list; recorded-events is an instance of an event-list; predicted-events is an instance of an event-list

MONITOR-LIST, an object-definition

Class	monitor-list
Superior class	item-list
Attributes specific to class	none
Default settings	element type for item-list: monitor; allow duplicate elements for g2-list: no

References

CHRISTIANSSON, M. and P. ERICSSON (1989): "Knowledge-based Control and modelling with G2," TFRT-5411, Department of Automatic Control, Lund Institute of Technology.

FINCH, F.E. (1989): "Automated Fault Diagnosis of Chemical Process Plants using Model-Based Reasoning," Sc.D. Thesis, Massachusetts Institute of Technology.

IT4 (1990): *Knowledge-Based Real-Time Control Systems.*

OYELEYE, O.O. (1989): "Qualitative Modeling of Continuous Chemical Processes and Applications to Fault Diagnosis," Sc.D. Thesis, Massachusetts Institute of Technology.

OYELEYE, O.O. and M.A. KRAMER (1988): "Qualitative Simulation of Chemical Process Systems: Steady-State Analysis," *AIChE J.*, **34**, 9, 1441.

PETTI, T.F., J. KLEIN, and P.S. DHURJATI (1990): "Diagnostic Model Processor: Using Deep Knowledge for Process Fault Diagnosis," *AIChE J.*, **36**, 565.

ROSE, P. (1990): "A Model-Based System for Fault Diagnosis of Chemical Process Plants," M.S. Thesis, Massachusetts Institute of Technology.