Discrete Sensor Validation with Multilevel Flow Models

Bengt Öhman

GoalArt
Tunavägen 39C
SE-223 63 Lund, Sweden
E-mail: bengt@goalart.com

This is a *preprint* of an article published in *IEEE Intelligent Systems* 2002, and it is made available as an electronic reprint by permission of *IEEE Intelligent Systems*. The full reference to the published article is:

Öhman, B., "Discrete Sensor Validation with Multilevel Flow Models," *IEEE Intelligent Systems*, vol. 17, no. 3, pp. 55-61, May/June 2002.

Discrete Sensor Validation with Multilevel Flow Models

Bengt Öhman, *GoalArt*

This sensor validation algorithm monitors a process's active alarms, alerting operators and support personnel to potential false alarms.

Supervising and controlling an industrial plant often relies heavily on measurements from sensors throughout the plant. Having many sensors allows fine-grained control and supervision of the whole plant. Each sensor might have one or more associated alarms that warn the operators when something that has happened or is about to happen might require manual intervention.

Because the alarms are an important source of information for operators, they must be accurate. An alarm that goes off when it should not is a *false positive alarm*; one that does not go off when it should is a *false negative alarm*.

False positive and false negative alarms occur for several reasons. For example, the alarm limits might be badly tuned or might not apply to a certain operating mode. The sensors might break down or drift. Continual monitoring of alarm quality is necessary to detect these situations.

To automatically supervise a plant's alarm system during normal operations, GoalArt has developed an online algorithm called Discrete Sensor Validation with Multilevel Flow Models. Multilevel flow models, which describe processes in terms of a process's mass flows or energy flows, resolve the causal dependencies between the alarms in a process. The algorithm uses this information to detect possibly erroneous alarms.

Multilevel flow models

Morten Lind invented MFMs,[1] which model goals, functions, and their relations to technical systems. However, they also make excellent bases for diagnostic reasoning. Jan Eric Larsson,[2–4] Fredrik Dahlstrand,[5,6] and others have developed several diagnostic-reasoning algorithms using MFMs.

MFMs incorporate *goals*, *components*, *functions*, *networks*, and *relations*. They also incorporate a graphical modeling language, so symbols represent all objects and relations (see Figure 1).

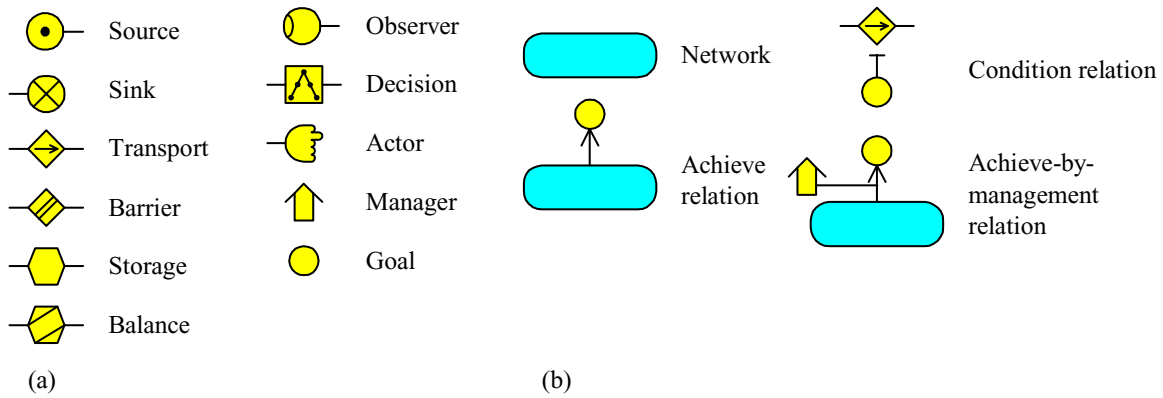

Figure 1. Multilevel flow model symbols: (a) goal and functions; (b) network and relations.

Goals

The goals describe a system's objectives. The goals might be expressed as a text string—for example, "keep the water level in tank T1 within an acceptable range," "produce electrical power," or "cool the feedwater pump." Identifying a system's goals is the first step in creating an MFM, because the system's purpose is to fulfill these goals. Figure 1a shows the symbol for a goal.

Components

The components are a system's physical structures. A component can be, for example, a pipe, a water tank, a pump, or an accumulator. Some MFMs show the components. However, ours do not because we derived our modeling method from Larsson's, in which the components are implicit.

Functions

The functions represent the capabilities of a system's individual parts, such as "transport water," "supply electricity," "store hydrogen," or "prevent radiation transport." The functions might or might not be associated with a physical component. A component might be associated with several functions. An electric pump, for example, might both transport water and act as a sink for electrical energy.

MFMs employ 10 basic functions:

- *Sources* provide unlimited mass or energy.
- *Sinks* drain mass or energy.
- *Transports* transport mass, energy, or information.
- *Barrier* prevents the transport of mass, energy, or information.
- *Storage* stores mass or energy.
- *Balances* connect one or more inflows to one or more outflows.
- *Observers* translate sensor values to information.
- *Decisions* represent control actions by humans or automatic controllers.
- *Actors* transform information into physical actions.
- *Managers* represent a system's management—for example, PID (proportional-integral-derivative) controllers or human operators.

Figure 1a displays the function symbols.

Networks

Functions are grouped into networks, each describing a flow of mass, energy, or information. Figure 1b shows the network symbol.

Relations

MFMs can involve four kinds of relations:

- *Achieve relations*—between networks and goals—indicate that all functions in the network must work as designed to achieve a goal.
- *Achieve-by-management relations*—between goals and managers— indicate that flow functions need management (automatic or by an operator) to function as designed.
- *Condition relations*—between goals and functions—indicate that the function is available only after the goal is achieved.
- *Realize relations*—between functions and components—indicate which component realizes a function.

Figure 1b shows the symbols for the first three relations. MFMs do not usually explicitly use the realize relation, so the figure does not show it.

An MFM example

Figure 2a depicts a simple system. The engine runs on gasoline and oxygen. The cooling system consists of two heat exchangers—one internal (connected to the engine) and one external (connected to the outside environment). A circulation pump moves water through the heat exchangers, transferring heat from the engine to the environment. The pump runs on electricity and must be lubricated to run.

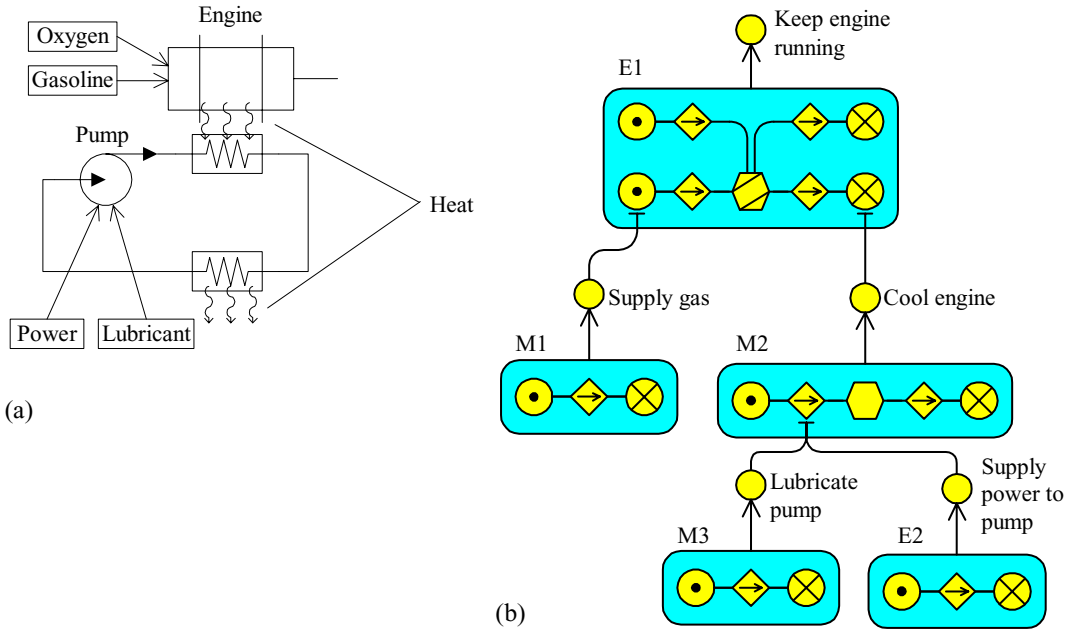

Figure 2. An engine connected to a cooling system: (a) a basic diagram; (b) the MFM. In the MFM, the networks marked "E" represent energy flows; the networks marked "M" represent mass flows.

To create an MFM of this system, we must identify the system's goals and then its functions. In this case, the top-level goal is "keep the engine running." The system also has several subgoals, such as "supply gasoline to the engine," "cool the engine," "lubricate the pump," and "supply power to the pump." The functions in this model include "gasolinestorage," "gasoline transport," "water transport," "lubricant source," and "heat sink."

To start building the MFM, consider the engine itself. The engine's desired output is kinetic energy, but it also generates heat. The gasoline and oxygen input chemical energy to the engine. Clearly, the engine transforms chemical energy to kinetic energy and heat. In the MFM in Figure 2b, Network E1 models the energy flow through the engine; its goal is to keep the engine running. E1 contains two sources, two sinks, four transports, and one balance. The sources represent the chemical energy from the oxygen and the gasoline, which are mixed and transformed in the balance. The sinks represent the kinetic energy and the heat. If all the functions in E1 work as intended, the goal "keep engine running" will be achieved.

Another goal is to provide gasoline to the engine. In Figure 2b, network M1 is a mass flow, describing the flow of gasoline from the tank into the engine. The goal "supply gasoline" is achieved when the functions in M1 work as designed. These functions—a source, a transport, and a sink—represent the gasoline tank, the transport of gasoline to the engine, and the engine. This network's goal connects to one of the source functions in E1 (the source for the gasoline's chemical energy) via a condition relation. This means that the gasoline's chemical energy is not available if the goal "supply gasoline" is not achieved.

The sink representing the removal of heat energy from the engine requires that the cooling system works. In Figure 2b, M3, E2, and M2 represent the cooling system. M3 and E2 model the transport of lubricant and electrical power to the pump; M2 models the water circulation. In M2, the left transport is a model of the pump; the storage in the middle is a model of the engine's heat exchanger. Both the source and the sink are models of the external heat exchanger. We could also express this by modeling the source and the sink as a single storage and creating a circular flow in the

network instead of a straight one.

This modeling method, which is used to create all MFMs, is an iterative, top-down process, where the MFM gradually describes more functionality. The iterations stop when the level of detail is enough to describe the desired functionality. For diagnosis, model refinement stops when all input signals (alarms, and so on) are connected to an object in the model. A process with 200 measurement points will result in an MFM with approximately 500 MFM flow functions organized in roughly 40 networks.

Alarm analysis with MFMs

Larsson's alarm analysis algorithm[4] is closely related to our discrete sensor validation algorithm, so I present it here. The alarm analysis algorithm sorts alarms into two types. *Primary alarms* are associated with the root cause of a process upset; *secondary alarms* are symptoms of consequential faults in a process.

The algorithm associates discrete alarm states, such as "low flow" and "high volume," with each flow function in the MFM. Through semantic interpretation of the flow functions, we can conclude that certain faults might cause consequential (secondary) faults in connected flow functions. The algorithm uses a set of *causality rules* to separate primary faults from those that might be the primary fault's consequences. However, this method does not guarantee that the faults marked as secondary are not actually primary faults. Hidden primary faults might look as if another primary fault caused them, so this method could classify them as secondary. Therefore, the algorithm categorizes faults only as "primary" or "secondary or hidden primary." Larsson invented the causality rules for the alarm analysis algorithm, and the discrete sensor validation uses the same rules. The main difference between the algorithms is that the alarm analysis algorithm always assumes that the input data is correct, whereas the discrete sensor validation algorithm can also give an indication if the data is to be trusted or not.

Let's examine how the algorithm works. Figure 3a shows a closed tank connected to a pump; Figure 3b shows this system's MFM.

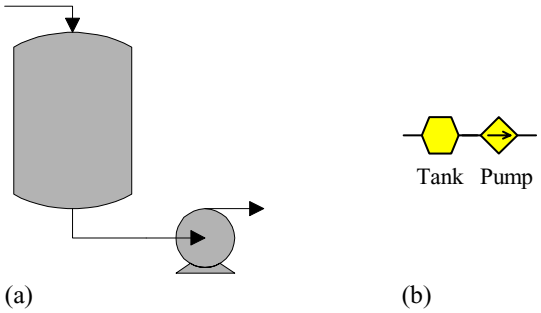

Figure 3. A closed tank with an inflow and an outflow that connects to a pump: (a) a basic diagram; (b) the MFM.

In this simple system, four causality rules describe how one fault might cause another:

- A too-low volume in the tank might cause a too-low flow through the pump.
- A too-low flow through the pump might cause a too-high volume in the tank.
- A too-high volume in the tank might cause a too-high flow through the pump
- A too-high flow through the pump might cause a too-low volume in the tank.

Assume that the tank becomes empty, activating a level alarm. Because of the tank's low level, the pump's flow becomes too low, activating another alarm. The first rule lets us conclude that the low volume caused both alarms—it is the primary fault—and that the low flow is a secondary fault. The algorithm uses a set of rules like these for every possible connection between two MFM functions to distinguish primary from secondary alarms.

MFM functions have associated alarm states. Storages have three alarm states: normal, high volume (hivol), or low volume (lovol). Transports also have three states: normal, high flow (hiflow), or low flow (loflow). The following type of formula sets the states: "If the flow through the transport is less than a threshold value f_{lo}, set the transport's alarm

state to loflow." Process engineers can set the threshold value in advance, or the control system can update it dynamically—for example, when the process's state changes.

Now we can rewrite the four causality rules using only MFM functions and their associated alarm states:

- A storage lovol might cause a downstream transport to have a loflow.
- A transport loflow might cause an upstream storage to have a hivol.
- A storage hivol might cause a downstream transport to have a hiflow.
- A transport hiflow might cause an upstream storage to have a lovol.

All possible MFM connections have similar rules, which Larsson describes in detail.[4] Because the number of possible MFM connections and states is limited, a table can effectively express the causality rules. The algorithm looks at each pair of MFM functions. If both functions are in an alarmed state, the algorithm uses the table to conclude which function has a primary alarm and which has a secondary alarm. If only one function is alarmed, that alarm is primary.

A special case occurs when a function is not connected to a sensor (the function is *unmeasured*). Without a connected sensor, knowing the function's real state is impossible. So, the algorithm consults the table to perform *consequence propagation*; that is, it deduces the unmeasured functions' states from the connected functions' states. For example, if the flow through the pump is unmeasured and the tank's volume is low, the consequence propagation rules will assume that the flow through the pump is low. This means that alarm analysis does not require sensors connected to all MFM functions.

Because of the MFMs' structure, looking at only two connected functions at a time is always sufficient. Also, since the consequence propagation will always stop when the algorithm finds a function with a sensor, it is usually only necessary to look at a very small part of the MFM whenever a new alarm is received. This lets the algorithm complexity remain linear, or even sublinear, with regard to the model's size.

Other Sensor Validation Methods

Sensor validation methods either operate on single sensors or combine information from several sensors. Single-sensor methods often employ simple algorithms such as limit checking. Information about operating mode or time dependencies can enhance these algorithms.

Most research, however, has focused on the more interesting multisensor methods. One approach is to install several sensors measuring the same process variable and then directly compare the measurements. This method is simple but might be prohibitively expensive, especially in retrofit cases.

A less-expensive approach is to compare the values of sensors measuring related process variables using a model. The model can take the form of, for example, simple relationships, a neural network, a set of expert system rules, or a dynamic process simulation. One modeling method that is similar to *multilevel flow models* (described in the main article) is *causal graphs*, such as the *signed directed graphs* that Olayiwola O. Oyeleye describes.[1] However, SDGs lack explicit goals and part–whole model breakups. During construction of an MFM, its consequence rules automatically supply the causal dependencies, whereas in an SDG they must be explicitly stated.

Reference

1. O.O. Oyeleye, *Qualitative Modeling of Continuous Chemical Processes and Applications to Fault Diagnosis*, doctoral thesis, Department of Chemical Engineering, Massachusetts Inst. of Technology, Cambridge, Mass., 1989.

Discrete sensor validation

Our algorithm has three parts:

- Condition checking
- Checking for normal/failed situations
- Full consequence irregularity analysis

The first two are fast and are invoked automatically whenever the process state changes. They give simple, easily

presentable results. The third part is a bit more complex. It can yield many plausible explanations when the alarms disagree with the MFM consequence propagation rules. This makes full consequence irregularity analysis more powerful than the algorithm's other two parts, but it also means that you must use it differently. Condition checking and checking of normal/failed situations should always be active, continuously giving the operators information. But operators should choose when to invoke the full consequence irregularity analysis so that they have more time to study the results.

Condition checking

In an MFM model for the alarm analysis algorithm, each condition relation has an associated property that indicates how the connected goal's failure will affect the connected function. This property can tell if a goal's failure will make the function go to a *fail high* state (such as high flow or high volume) or a *fail low* state, (such as low flow or low volume). The alarm analysis algorithm uses this information to decide whether an alarm on the connected function should be classified as primary or secondary.

For example, Figure 4 shows an MFM with two networks. The lower network describes electrical energy from a power supply (function F4) via a cord (F5) to a pump (F6). Together, these functions achieve goal G2, "Supply electrical power to the pump." The upper network describes water flowing from a source (F1) via a pump (F2) to a sink (F3). Together, these functions achieve goal G1: "Supply water to the sink." To transport water, F2 needs electrical power; the condition C1 between G2 and F2 indicates this. C1's properties illustrate that if G2 is not achieved, F2 will go to a low-flow state.

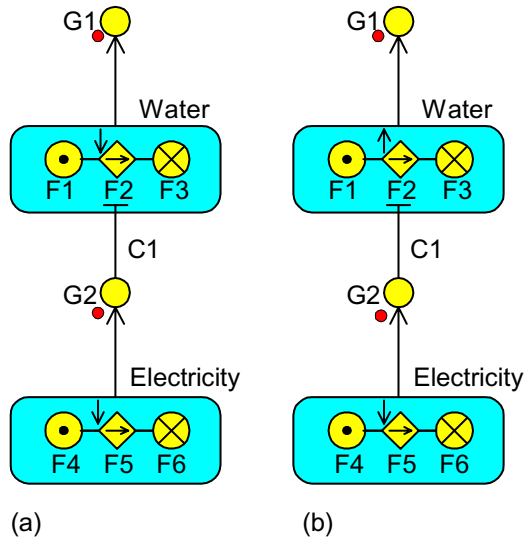

Figure 4. Alarm situations in an MFM where the lower network describes electrical flow and the upper network describes water flow (arrows indicate alarms, and the dots indicate failed goals): (a) functions F5 (a cord) and F2 (a pump) both have a low-flow alarm; (b) F5 has a low-flow alarm, and F2 has a high-flow alarm.

Now consider the alarm situation in Figure 4a. The low-flow alarm on F5 indicates a low electrical flow to the pump, and the low-flow alarm on F2 indicates a low water flow through the pump. F2's alarm state matches the information from C1—that is, that F2 should go to a low-flow state if G2 is not achieved.

Figure 4b also shows a low-flow alarm on F5 but a high-flow alarm on F2. In this case, F2's alarm state does not match the expected value, based on G2's failure and C1's properties. So, the upper network has a suspect alarm.

This condition-checking method can identify one other situation. If F2 in Figure 4a did not have an alarm—that is, if it were in a normal state—condition checking would recognize and indicate this.

In short, this part of the algorithm works as follows:

> *For each condition in the MFM, where the connected goal is not achieved, check whether the connected function's alarm state can be explained with the consequence propagation rules and the condition's properties. If not, indicate that the network*

containing the connected function might have a faulty alarm.

Checking for normal/failed situations

As we've seen, condition checking deals with inconsistencies between networks. However, another kind of simple inconsistency in an MFM network might indicate suspicious alarms.

For example, Figure 5 shows two situations. In Figure 5a, the MFM consequence propagation rules explain the alarm situation, and no conflicts exist (a low-flow alarm in a transport will not cause a low-capacity alarm in a connected source). But in Figure 5b, the consequence propagation rules do not explain the situation. Two rules come into question here: "A low flow in a transport to the left of a storage will cause a low volume in the storage," and "A normal situation in a storage to the right of a transport will cause a normal situation in the transport." The situation does not match either of these rules, so our algorithm will indicate that the network containing these functions has a suspicious alarm.

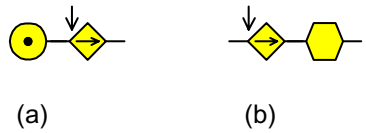

Figure 5. Two alarm situations: (a) one with no suspicious alarms. (b) one with no explanation under the MFM consequence rules.

To keep this method simple, the algorithm only inspects situations where a function with an alarm is next to a function without an alarm. This strategy allows the algorithm to detect situations with one false negative alarm or one false positive alarm. For example, an operator might guess that a false negative alarm exists in Figure 6a and that a false positive alarm exists in Figure 6b. In Figure 6a, there is a low volume in the two storages, but there is no alarm on the transport between them. In Figure 6b, there is a low flow alarm on the transport, but there is no alarm on the connected storages. Both of these conclusions require that this situation persist, to rule out the possibility that the situations are only transitional. The algorithm has no notion of time, so it cannot safely conclude that an effect is transitional. So, it only indicates suspect alarms; a human must decide whether they are false.

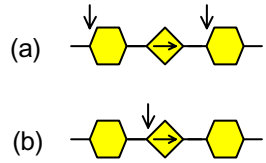

Figure 6. Two suspicious alarm situations: (a) a possible false negative alarm; (b) a possible false positive alarm.

This part of the algorithm works as follows:

> *For each situation in the MFM where a function with an active alarm is next to a function in a normal state, check whether a consequence propagation rule can explain this. If not, indicate that the network containing the two functions might have a faulty alarm.*

Full consequence irregularity analysis

The previous two parts of the algorithm target simple inconsistencies that can tell that "something might be wrong here." The third part, on the other hand, identifies situations in which assuming that one alarm is false makes the most sense under the consequence propagation rules.

Full consequence irregularity analysis relies on Larsson's alarm analysis algorithm. If one alarm cannot explain another's existence, the unexplained alarm is primary; if it can, the explained alarm is secondary.

When assuming the existence of a single faulty alarm yields a simpler alarm situation, the number of primary alarms decreases (according to the alarm analysis algorithm). This rule determines which MFM networks the algorithm will

inspect. Figure 6b shows the simplest case: an MFM network with a single failed function. In this case, checking for normal/failed situations will identify any possible sensor fault, so I will not discuss this situation in this section.

Instead, consider a case in which an MFM network has two primary alarms (according to the alarm analysis algorithm). Our algorithm might find a solution with only one primary alarm by assuming that one of the two alarms is erroneous. The solution with only one primary alarm is simpler than the two-alarm situation and therefore more probable (assuming that single faults occur more frequently in the process than multiple faults do). Our algorithm reports that the MFM network in question has a suspicious alarm situation, and it presents the simplest solution or solutions together with the function that it assumed to have an incorrect alarm.

Figure 7 illustrates how this part of our algorithm works. The figure assumes that every MFM function has an associated measurement or alarm (*measured functions*). This assumption does not change anything in practice, but it simplifies demonstrating the algorithm. The figure shows an MFM flow where six of the seven functions have active alarms and F7 is in a normal state. The dark dots indicate alarms that the alarm analysis algorithm has classified as primary; the lighter dots indicate alarms classified as secondary. Our algorithm will check each function to see whether the number of primary alarms decreases if it assumes that function has an inaccurate alarm state. If so, it can simplify the situation.

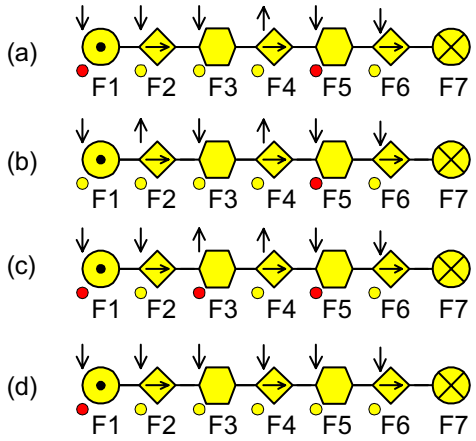

Figure 7. Full consequence irregularity analysis on an MFM flow. (a) Larsson's alarm analysis algorithm[4] indicates two primary alarms (marked with dark dots). Assuming that one alarm is incorrect can simplify this situation. (b) Assuming that F2 is in a high-flow state instead of a low-flow state simplifies this situation by producing one primary alarm. (c) Assuming that that F3 is in a high-volume state instead of a low-volume state produces three primary alarms, which does not simplify things. (d) Assuming that F4 is in a low-flow state instead of a high-flow state also produces one primary alarm.

First, consider the source function, F1. A source has only two alarm states: normal and low capacity. If the measurement for F1 is incorrect, the function must be normal. If we assume that F1 is in a normal state, the number of primary alarms does not decrease—F2 and F5 become primary. In the original case, the low capacity alarm on F1 explains the low flow in F2, but after changing F1 to normal, there is no cause for F2 to be low (the low volume in F3 does not explain a low flow through F2). We gain nothing by assuming that the F1 measurement is inaccurate, so our algorithm does not report F1's alarm as suspect.

Now consider the transport F2. If we assume this function is in a normal instead of a low-flow state, we gain nothing; F1 and F5 are still primary. But if we assume F2 is in a high-flow state, only one primary alarm exists—F5. All other alarms become secondary, so the situation has been simplified (see Figure 7a). Our algorithm presents the operator with the simplified situation and indicates that F2's alarm is suspect. The operator must decide whether to further investigate F2's sensor value.

Assuming that F3 is really in a normal state does not simplify the situation. F1 and F5 are still primary. Assuming instead that F3 is in a high-volume state does not simplify the situation either; in fact, the situation becomes worse. F1, F3, and F5 then become primary, thereby increasing the number of primary alarms from two to three (see Figure 7c).

If we assume that F4 is in a normal instead of a high-flow state, the situation does not improve. But, if we assume it

is in a low-flow state, only one primary alarm exists, and the situation becomes simpler (see Figure 7d).

Performing the same analysis for F5, F6, and F7 does not yield any situations with only one primary alarm. So, we can simplify the situation in Figure 7a by making one of these assumptions:

- F2's measurement is wrong and should have been a high flow instead of a low flow.
- F4's measurement is wrong and should have been a low flow instead of a high flow.

The algorithm will present both of these possible explanations to the operator, who will have to decide (based on previous experience or other available information) which solution is the correct one.

Now, let's examine another situation. Figure 8 is an MFM flow with three primary alarms. In this case, assuming that one single measurement is incorrect will not reduce the number of primary alarms. At least two measurements must change to reduce this number, because in this network, all transports and balances must have the same alarm state in order to have only one primary alarm, that is, either all "low" alarms, or all "high" alarms.

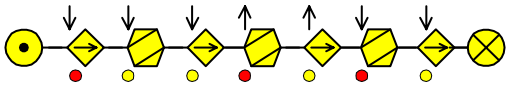

Figure 8. In this situation, assuming that one measurement is incorrect will not reduce the number of primary alarms. A low flow through a transport will not cause a high flow in a connected balance, and vice versa.

We could extend our algorithm to assume that two or more measurements in a network might be false. However, in practice such analysis could quickly degenerate. If the algorithm was allowed to change two measurements in a network with four sensored functions, which is not an unusual number, it would assume that half of the measurements were wrong. With even more freedoms for the algorithm, this would quickly lead to a situation where almost none of the real measurements was trusted. Therefore, the algorithm uses a single-failure assumption.

So, this part of our algorithm works as follows:

> *For each network in which the alarm analysis algorithm classifies at least two alarms as primary, perform the following:*
>
> 1. *For each flow function F with an associated sensor value, store the original state of F, and change F to another state.*
> 2. *Perform a new alarm analysis on the network.*
> 3. *If the number of alarms classified as primary decreases but is still greater than zero, report the sensor connected to F as suspicious.*
> 4. *If all alarm states of F have been tested, restore F's original state and continue from step 1. Otherwise, change F to a new state, and continue from step 2.*

An operator can either manually activate this part of the algorithm or set it to run automatically every time a new alarm arrives in the system. With manual activation, this part must check every network in the MFM. With automatic activation, it must check only the MFM's affected parts (usually one or a few networks), which assures that the algorithm runs quickly.

A real-world example

Our algorithm is useful in many situations; for instance, it might have helped prevent the Three Mile Island incident in 1979. One cause of that accident was that the *pilot-operated relief valve* was incorrectly indicated as being closed[7] (Figure 9a depicts this part of the reactor). Because the PORV was open, the reactor quickly lost a lot of water, and the drain tank started to fill. Because the operators believed that the PORV was closed, they did not observe this for some time.

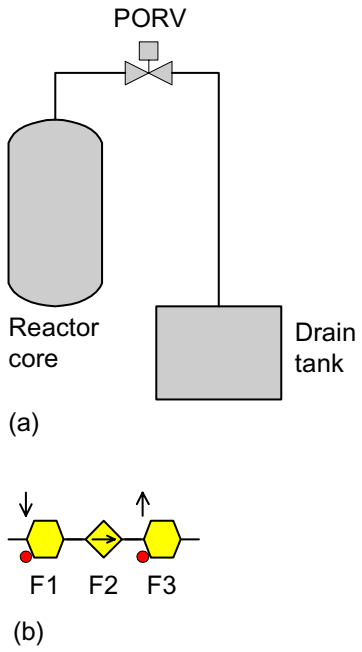

Figure 9. The Three Mile Island 2 nuclear reactor: (a) the core, the pilot-operated relief valve, and the drain tank; (b) an MFM model including two primary alarms.

Figure 9b expresses this situation in an MFM. Here, checking for normal/failed situations would alert the operator that F2's alarm state, describing the PORV, might be wrong. Furthermore, full consequence irregularity analysis would indicate that F2's alarm state should be high instead of normal, thereby decreasing the alarm situation's complexity.

Wrongly tuned alarms and malfunctioning sensors occur regularly in industrial processes, although not usually with such drastic consequences. Such a situation makes the operator's job harder; automatically monitoring the alarms with the discrete sensor validation algorithm makes detecting and correcting problems easier.

GoalArt constructed the discrete sensor validation algorithm for online use with other algorithms (for example, alarm analysis), to supervise the process and to help the operator make decisions. Because all MFM algorithms operate on the same model, the cost of adding our algorithm to an alarm analysis system is small.

Because this algorithm operates on discrete alarm values and needs to inspect only a small part of the MFM, it does not perform heavy computation. So, every time the system receives a new alarm it can activate the algorithm without adding much computational load.

First tests of this algorithm on a real process will take place this year. The details on this are not clear yet, but most probably the algorithm will first be evaluated using simulated process data, and then the algorithm will be allowed to use actual process data in real time for some months. After this step, the benefit of using the discrete sensor validation algorithm can hopefully be measured.

References

1. M. Lind, *Representing Goals and Functions of Complex Systems: An Introduction to Multilevel Flow Modeling*, tech. report 90–D–38, Inst. Automatic Control Systems, Tech. Univ. of Denmark, Lyngby, Denmark, 1990.
2. J.E. Larsson, *Knowledge-Based Methods for Control Systems*, doctoral thesis, Dept. of Automatic Control, Lund Inst. of Technology, Lund, Sweden, 1992.
3. J.E. Larsson, "Diagnostic Reasoning Strategies for Means-End Models," *Automatica*, vol. 30, no. 5, May 1994, pp. 775–787.
4. J.E. Larsson, "Diagnosis Based on Explicit Means-End Models," *Artificial Intelligence*, vol. 80, no. 1, Jan 1996, pp. 29–93.
5. F. Dahlstrand, *Methods for Alarm Reduction with Multilevel Flow Models of Industrial Processes*, licentiate thesis, Dept. of

Information Technology, Lund Inst. of Technology, Lund, Sweden, 2000.

6. F. Dahlstrand, "Consequence Analysis Theory for Alarm Analysis," *Knowledge-Based Systems*, vol. 15, nos. 1–2, 2002, pp. 27–36.
7. C. Perrow, *Normal Accidents: Living with High-Risk Technologies*, Princeton Univ. Press, Princeton, N.J., 1999.

Bengt Öhman is a senior knowledge engineer at GoalArt, a spin-off company started by the AI-group at the department of Information Technology, Lund University. He has a Master of Science in Computer Engineering and Technology from Lund University in 1997. After that, he continued with Ph.D. studies at the department of Information Technology, Lund University. During this time, he worked with the MFM modeling method and algorithms for MFM, and received the degree of Licentiate of Engineering. In 2000, Bengt and his colleagues founded GoalArt, where he has been employed since 2001. His main research interests are model-based diagnosis, MFM algorithms, and game-playing programs. Contact him at GoalArt, Tunavägen 39C, SE-223 63 Lund, Sweden; bengt@goalart.com.