ISSN 0280–5316
ISRN LUTFD2/TFRT--7493--SE

An MFM Toolbox

Jan Eric Larsson

Department of Automatic Control
Lund Institute of Technology
December 1992

Department of Automatic Control **Lund Institute of Technology** P.O. Box 118 S-221 00 Lund Sweden	*Document name* Technical report *Date of issue* December 1992 *Document Number* ISRN LUTFD2/TFRT--7493--SE
Author(s) Jan Eric Larsson	*Supervisor* *Sponsoring organisation* STU, project no. 85–3042 IT4, project no. 3403 TFR, project no. 92–956

Title and subtitle

An MFM Toolbox

Abstract

This report describes a G2 toolbox for MFM modeling. The toolbox program contains definitions of MFM data structures and several rulebases, for performing syntax control, measurement validation, alarm analysis, and fault diagnosis. It also gives a short description of G2 and finally some hints and practical points to be considered when using the toolbox program. Two examples of processes and MFM models are also provided. Considering G2's strength as a tool for building control and supervisory systems, the toolbox program is probably the most efficient MFM construction and experimentation tool available.

Key words

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title 0280–5316		*ISBN*
Language English	*Number of pages* 12	*Recipient's notes*
Security classification		

1. *Introduction*

This report describes the algorithms and software developed during the doctor's project Larsson (1992). It contains descriptions of the data structures developed in G2 for constructing MFM models, of the rulebases which implement the diagnostic algorithms, and some details about the software.

The implementation consists of definitions of data structures and graphics for building MFM graphs, a set of rule bases that perform the diagnostic reasoning tasks, and a set of flow sheets and equations used for simulation of two example processes. In total, the program contains the following:

- Definitions of a class hierarchy of the objects in MFM models.
- Rule bases for syntax control of MFM models, measurement validation, alarm analysis, consequence propagation, and fault diagnosis.
- Definitions of class hierarchies of the components in the example processes.
- Generic simulation equations for the physical components.

The user may define his own components and construct a topological simulation model, which is the standard way of using G2. He then builds an MFM model using the toolbox definitions, and when that model is ready, he immediately has available the diagnostic algorithms described in Larsson (1992), instead of having to construct a rule base for the specific process. Thus, the general ideas is to replace the construction of a rule-based expert system with the building of an MFM model using the toolbox, see Figure 1. Of course, it is also possible to use the toolbox as a general tool for exploring MFM models.

MFM Model + Toolbox = Expert System

Figure 1. With the help of the toolbox, it is possible to replace the large effort of building a rule base for an expert system with the construction of an MFM model.

This report will describe the implemented toolbox, going through the data structures and rule bases, together with some practical advice on how to use them. To begin with, the expert system tool G2 will be overviewed.

2. *G2*

The MFM toolbox has been implemented in G2. A short description of G2 itself will now be given. The following part is based on Nilsson (1991); see also Larsson (1992).

The expert system tool G2 has been developed by Gensym Corporation, and is probably the most advanced real-time expert system tool currently available, see Moore *et al* (1987, 1991). It is implemented in Common Lisp, which is automatically translated to C, and it runs on many different computers. A Sun Sparcstation 2 was used in the doctor's project. G2 consists of several main parts:

- A knowledge database
- A real-time inference engine
- A procedure language interpreter
- A simulator
- A development environment
- An operator interface
- Optional interfaces to external on-line data servers

Classes and Objects

G2 is an object-oriented programming environment. All G2 components, including rules, procedures, graphs, buttons, objects, etc., are *items,* and organized into a class hierarchy with *single inheritance.* All items have a graphical representation through which they may be manipulated by mouse and menu operations. Operations exist for moving an item, cloning it, changing its size and color, etc. The user defined items are called *objects* in G2.

Objects are used to represent the different concepts needed in a specific application. The object definition defines the attributes specific to the class and the look of the icon. Attributes of many types are supported, such as constants, variables, parameters, lists, arrays, and G2 objects. The constants, variables, and parameters may be quantitative, (integer or real), symbolical, logical, or text strings.

Objects are either static, i.e., they are explicitly created by the developer, or dynamic, i.e., created and deleted dynamically during runtime. G2 contains operations for moving and rotating an object, and changing its color. With these facilities, simple animations can be created. Each G2 object may have an associated *subworkspace.* Here arbitrary items may be positioned, and thus the internal structure of an object may be represented on its subworkspace.

Relations Between Objects

G2 has different ways of defining relations between objects. One way is to have lists containing other objects as attributes. Another way is to use *connections,* which have a graphical representation and may have attributes. Connections can be used in G2 expressions for reasoning about interconnected objects in a variety of ways.

A third way of relating objects is to use *relations.* These may only be created at runtime and have no graphical representation. They have no corresponding relation hierarchy and cannot have attributes. Relations are used in G2 expressions in the same way as connections.

The Inference Engine

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

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

If rules my be invoked by forward and backward chaining, by scanning at a specified time interval, and by explicit invoking. *When* rules are a variant of *If* rules that may not be invoked through forward chaining or cause backward chaining. *Initially* rules are executed at initialization time only. *Unconditionally* rules are equivalent to *If* rules with a true premiss. *Whenever* rules trigger when a variable receives a new value, fails to receive a value within a specified time-out interval, when an object is moved, or when a relation is established or deleted, i.e., they acts as *demons.*

The rules contain references to objects and their attributes in a natural language style syntax. Objects may be referenced through connections with other objects, thus utilizing the connection structure of the objects instead of explicit names. G2 supports generic rules that apply to all instances of a class.

In addition to forward and backward chaining, rules may be invoked explicitly in several ways. A rule may be scanned at even time intervals. A *focus* statement invokes all rules associated with a certain focal class or object. An *invoke* statement triggers all rules belonging to a specified rule category.

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

Procedures

G2 contains a Pascal-style procedural programming language. The procedures are started by rule actions, they are reentrant and each invocation executes as a separate task, and they may have input parameters and return one or several values.

The set of procedure statements include all rule actions, assignment, branching, (*if-then-else* and *case*), iteration, (*repeat* and *for*), *exit if* to exit loops, the infamous *go to, call* to call a procedure and wait for its result, and *start* to call a procedure without waiting. The *for* loops may be either numeric or generic for a class, i.e., they execute a statement or set of statements once for each instance of the class.

Simulation

G2 has a built-in simulator which can provide simulated values for variables. The simulator is intended to be used both during development for testing the knowledge base, and in parallel during on-line operation. It allows for differential, difference, and algebraic equations on explicit form. These may be specific to a certain variable of apply to all instances of a variable class. Each first-order differential equation is integrated individually with an individual, user defined stepsize. The numeric integration algorithms available are a simple Forward Euler algorithm and a fourth order Runge-Kutta, both with fixed stepsize.

3. *The MFM Data Structures*

MFM models have a strong graphical nature and consist of objects connected together and collected in different networks. Diagnostic algorithms using MFM will use information about these objects and how they are interconnected. Thus, G2 is ideally suited for implementing the basic MFM data structures, as well as the diagnostic algorithms. The MFM concepts have the corresponding implementations shown in Table 1.

Goals and flow functions	*G2 objects*
Relations and links	*G2 connections*
The "inside" relation	*Subworkspaces*
Diagnostic methods	*Rules and procedures*

Table 1. MFM concepts and the corresponding G2 concepts used to implement them. The "inside" relation refers to the case when a path of flow functions resides in a network or manager function.

The animation facilities of G2 have been used to present results of the algorithm. For example, the primary and secondary failure states from the alarm analysis are shown in red and bright red, and a quick glance will give the operator a good idea of the total failure state of the process.

The time efficiency has not posed any problem during the project. As all the algorithms themselves are linear in effort, the main obstacle to overcome when scaling up the size of the knowledge database is the internal G2 representation. However, the G2 inference engine uses static links, and no decrease in efficiency has been observed when tackling somewhat larger processes as Steritherm.

All in all, it can be concluded that G2 is a very good programming tool, provided it is used to solve a fitting problem, and MFM is one such problem.

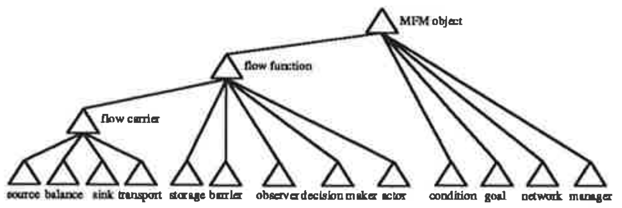


Figure 2. The MFM class hierarchy, which is a part of the G2 object hierarchy. Thus, the superior class of `MFM object` is the G2 `object`.

The Data Structure and Class Hierarchy of Objects

The MFM objects lend themselves to be put in a class hierarchy with single inheritance, see Lind (1990 b). This is easily done in G2, and the resulting classes are shown in Figure 2. Actually, it would be quite natural to use *multiple inheritance* to express that each flow function is either of type mass, energy, or information, but this is not supported in G2.

Rules

The different algorithms are easily implemented with G2 rules. As an example, consider once again, connected source and transport, see Figure 3.

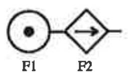


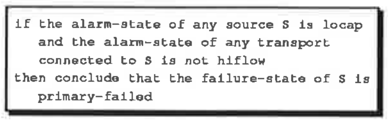
Figure 3. A connected source and transport. In the alarm analysis algorithm, this connection is treated with three rules.

In Larsson (1992) the following rules to find out the failure state of the source were presented:

- A transport *hiflow* alarm may cause a connected source to have a *locap* alarm.
- An alarm in a network will force a function depending on this network to fail.

Taking the normal situation into consideration also, three rules for the failure state of a source may be formulated.

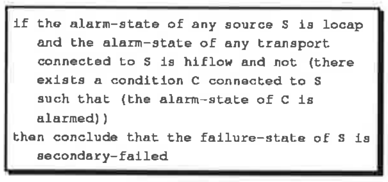
- If a source has a *locap* alarm and any connected transport does not have a *hiflow* alarm, then the alarm of the source is primary.
- If a source has a *locap* alarm and any connected transport has a *hiflow* alarm and there is no alarm in the network that the source may depend on, then the alarm of the source may be secondary.
- If a source is not alarmed, its failure state is working.



```
if the alarm-state of any source S is locap
   and the alarm-state of any transport
   connected to S is not hiflow
then conclude that the failure-state of S is
   primary-failed
```

Figure 4. A rule from the alarm analysis rule set, as it looks in the G2 implementation. It handles the case when there is a primary failure in the source, and it is quite similar to the text version of the same rule, presented earlier.

EXAMPLE 1
These rules are surprisingly easily implemented in G2. The possibility to reason about objects and their connections make the G2 rules look very much like the textual rules above. For example, the G2 rule corresponding to the first rule is shown in Figure 4. □

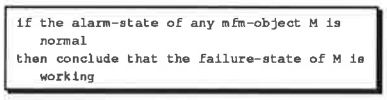


```
if the alarm-state of any source S is locap
   and the alarm-state of any transport
   connected to S is hiflow and not (there
   exists a condition C connected to S
   such that (the alarm-state of C is
   alarmed))
then conclude that the failure-state of S is
   secondary-failed
```

Figure 5. Another rule from the alarm analysis rule set. It corresponds to the second rule above, and treats the case when the source has a primary *or* secondary failure.

EXAMPLE 2
The G2 version of the second rule is shown in Figure 5. Note that the *locap* alarm is secondary only if there is no fault in the supporting network, i.e., a fact from the second rule. This test must be included in the G2 rule, as otherwise two rules could be in conflict and trigger each other infinitely. □

Example 3
If the source should be in a normal state, (i.e., not alarmed), the failure state must be set to working. This is handled by a more general rule, valid for all flow functions, see Figure 6. □

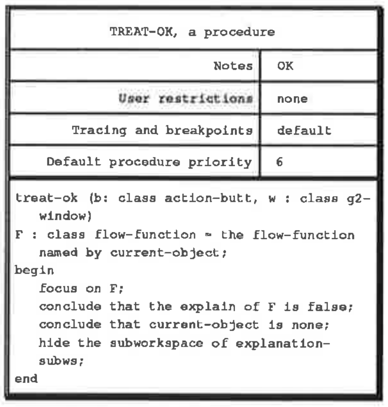


```
if the alarm-state of any mfm-object M is
   normal
then conclude that the failure-state of M is
   working
```

Figure 6. A third rule from the alarm analysis rule set. This rule is valid for any MFM object and shows the use of the "any" construction allowed in G2 rules. This enables the writing of rules that apply to whole classes of objects. The rule is used to set all MFM objects which are not alarmed to the working state.

Procedures

Parts of the algorithms are more easily expressed with procedures than rules. This is the case especially in the fault diagnosis.



TREAT-OK, a procedure

Notes	OK
User restrictions	none
Tracing and breakpoints	default
Default procedure priority	6

```
treat-ok (b: class action-butt, w : class g2-
   window)
F : class flow-function = the flow-function
   named by current-object;
begin
   focus on F;
   conclude that the explain of F is false;
   conclude that current-object is none;
   hide the subworkspace of explanation-
   subws;
end
```

Figure 7. A procedure from the fault diagnosis algorithm. This specific procedure is called when an explanation or remedy has been shown and the user clicks the OK button. Then the `explain` attribute is set to false, so that the explanation will not be activated again, the search variable `current object` is set to none so that the search may go on to look for more explanations, and the subworkspace upon which the explanation was given is hidden.

Example 4
A short procedure from the fault diagnosis algorithm is shown in Figure 7. This particular procedure treats the case when the user has been given an explanation or remedy and clicks the OK button. □

4. *The Rulebases*

The algorithms described in Larsson (1992) have been implemented in G2. The MFM graph structure is built with G2 graphical objects and connections, thus giving both a graphical presentation and an underlying data structure on which the G2 rules and procedures can operate. The construction of the flow models is simple and user friendly, as it uses G2's possibilities of graphical editing, creation, and cloning of objects, etc.

There are several groups of rules and procedures, each implementing one or a part of an algorithm. Thus, the main contribution in the implementation is to be found in these rule groups. It should be noted that G2's rules and procedures can be mixed and work well together. Thus, a rule set may really be a set of rules *and* procedures. The different rule sets will now be described.

Syntax Control

There are many rules of syntax that say how the MFM objects may or may not be connected. These have been stated in Larsson (1992). The G2 connections cannot be mixed, and therefore mixing of flow types is prohibited even from the initial design of the graphical structure. Within a flow path, however, flow functions may only be connected according to certain rules; for example, a source may only be connected to a transport. A small rule set is used to check this part of the syntax. These rules must be invoked by the user, however, and without doing so, it is possible to violate the MFM syntax. It should also be noted that the rules do not check the more unclear syntax rules that state that a node may not be filled or emptied only, see Section 3.5 of Larsson (1992).

Measurement Validation

The measurement validation algorithm is implemented with five rule sets. One handles the updating of group information, i.e., it keeps track of consistent and inconsistent subgroups. Another rule set controls the flow propagation. The flow values of the flow functions with no measurement connected to them must be guessed, and known flow values are propagated to them. Multiply supported flow values have precedence over singularly supported ones, and downstream propagation has precedence over upstream propagation.

A third rule set handles the recognition of subgroups and singular inconsistent subgroups, which should be specially marked. If the single group is surrounded by another, consistent group, its flow value is also changed. A fourth group handles the validated flow values, which can be different from the measured flows in case a single group is surrounded by a consistent group. The fifth group handles dynamic color coding of the information. Thus, the different inconsistent subgroups are shaded in different hues of gray, and the singular failures are marked with red.

Alarm Analysis

The alarm analysis rule set handles the recognition of the different alarm situations and sets the failure states accordingly. These rules only look at the flow functions two and two, or in some cases three and three. Thus, they work locally and efficiently.

Consequence Propagation

The consequence propagation rules handle the guessing of alarm states of flow functions which are not connected to physical alarm sensors. Similar causation rules to those in the alarm analysis are used. This rule set works locally and mixes with the alarm analysis rules.

Fault Diagnosis

The fault diagnosis algorithm performs a downward search in the MFM graphs and uses a dialog of questions and answers. The dialog part has had the consequence that most of the implementation is done with procedures. Thus, rules handle the search and decisions about which subtrees that need further investigation, while procedures handle the dialog and setting of fault values. The implemented search is not strictly local, (the diagnosis may skip parts of the graph altogether), but as the trees are graphically represented, the geographical coordinate values are used to make the search move in a left to right direction over the screen, and no additional reasoning over global paths is needed.

5. *How To Build an MFM Database*

The MFM toolbox program contains all the data structures necessary to build MFM models of a process, and once a model has been constructed, the rule bases included will make the three diagnostic methods available with no further work needed. The toolbox program is found in the file `toolbox.kb` and the steps of building an MFM database are the following:

- Make definitions for a topological model of the target process, including simulation equations and rules, if needed.
- Build a simulation model and flow sheet of the process.
- Use the MFM definitions and build an MFM model of the process.
- Connect the simulation of the process to the appropriate variables in the MFM model. This can be done via relations, rule bases, etc., but the toolbox does not provide any standard solution.
- Build a user interface for the diagnostic algorithms, and connect the result variables in the MFM model to this interface.

In the toolbox, two examples have been provided of how the MFM definitions can be used. In the first example, the tanks process, there is a flow sheet and a simulation, and a set of scanned rules are used to transfer data to the MFM model. The results of the alarm analysis are presented with the help of multicolored alarm icons in the flow sheet, while the results of the measurement validation are shown on a special operator's panel.

In the second example, the Steritherm process, no simulation has been implemented. Instead, the alarm analysis is performed so that the variables in the MFM model are set directly via buttons on a special alarm panel.

Input Parameters of the MFM Model

All input to the MFM model, and thereby to the diagnostic algorithms, goes via the symbolic and quantitative variables in the MFM objects. Thus, these variables must be set by some external action, e.g., scanned rules. Once the values have been set, the algorithms react automatically, and the results are made available in other variables. These must then be read out by some external actions. The following variables are concerned with the different algorithms:

- The measurement validation needs to know the measured flow values corresponding to every MFM object. If no such value is available, the attribute `measured` should be *false*, which is the default; otherwise it should be changed to *true*. This attribute controls whether the flow propagation should perform guessing or not. The attribute `flow` of any flow carrier should be set to the corresponding measured value. Storages instead have two such attributes, `inflow` and `outflow`, and two attributes to tell whether these are measured or not.

  The output of the measurement validation is primarily the `group` attribute, which is a cardinal telling which consistent subgroup the flow function in question belongs to. The `flow` attribute will contain the guessed value in case `measured` is *false*. Finally, the attribute `vflow` contains the validated flow value of the flow function, i.e., the flow value of a surrounding, consistent group in case of a single fault, or otherwise the same value as `flow`.

- The alarm analysis algorithm needs to know the alarm state of the flow functions. The attribute is named `alarm-state` and should be set to *normal, locap, loflow, hiflow, lovol, hivol, leak, fill* or *malfunction.* It must be set by external actions, for example a scanned rule group. The `alarmed` attribute tells whether there actually is a measurement connected to the flow function in question, or whether the consequence propagation should be allowed to guess alarm states.

  The results of the alarm analysis algorithm are made available in the `failure-state` attributes of each flow function. The values can be either *working, primary-failed,* or *secondary-failed,* where the latter is to be interpreted as primary *or* secondary.

- In order for the fault diagnosis to work, each flow function should be given a question text string as the `question` attribute, and the `ask` attribute should be set to *true.* Similarly, `explanation` should be set to a text string with an explanation of the fault or a remedy for it, and the `explain` attribute set to *true.* The questions should be phrased so that if the answer is "yes" the function is available, while a "no" means that it has failed.

  Currently, all input to the fault diagnosis must go via the questions in the flow functions, but it is also possible to use values set by other actions. In this case, the value of the attribute `alarm-state` should be set when the global variable `current-object` points to the flow function in question, and the `diagnose` if the flow functions is true. It is possible to use this conditions to give the fault diagnosis input from, e.g., the simulator. Note that the fault diagnosis uses the `alarm-state` attribute of the alarm analysis algorithm to store the status of the flow functions. Thus the two algorithms mix with each other.

6. *The Automatic Rule Generator*

Most of the alarm analysis rules can be automatically generated, from a set of general assumptions of flow function behavior, see Larsson (1992). A small G2 database performs this by looping through all possible connections of flow functions and testing whether the behavior combinations should give rise to an alarm analysis rule. This automatic rule generator is found in the file `autogen.kb` and it can be used to see how simple it is to produce most of the alarm analysis rule automatically. It should be noted that this program does not generate rules proper, but only messages. Turning these messages into rules is a simple task, however.

7. *Conclusions*

The MFM toolbox program hopefully provides a good environment for building and using MFM models. G2 gives the user ample opportunities to construct flow sheets, simulations, and other things needed in a control and supervisory system. This system may then easily be connected with the MFM data structures and algorithms. Once the MFM model has been built, then three algorithms described in Larsson (1992) are immediately available, with no extra modeling work needed. So far, the MFM toolbox is probably the most efficient MFM construction and experimentation tool available.

LARSSON, J. E. (1992): *Knowledge-Based Methods for Control Systems*, Doctor's thesis, TFRT–1040, Department of Automatic Control, Lund Institute of Technology, Lund.

LIND, M. (1990 a): "Representing Goals and Functions of Complex Systems—An Introduction to Multilevel Flow Modeling," Technical report, Institute of Automatic Control Systems, Technical University of Denmark, Lyngby, Denmark.

LIND, M. (1990 b): "Abstractions Version 1.0—Descriptions of Classes and Their Use," Technical report, Institute of Automatic Control Systems, Technical University of Denmark, Lyngby, Denmark.

MOORE, R. L., L. B. HAWKINSON, M. LEVIN, A. G. HOFFMANN, B. L. MATTHEWS, and M. H. DAVID (1987): "Expert System Methodology for Real-Time Process Control," *Proceedings of the 10th IFAC World Congress*, München, pp. 274–281.

MOORE, R. L., H. ROSENOF, and G. STANLEY (1991): "Process Control Using a Real-Time Expert System," *Proceedings of the 11th Triennial IFAC World Congress 1990*, Tallinn, Estonia, pp. 241–245.

NILSSON, A. (1991): *Qualitative Model-Based Diagnosis—MIDAS in G2*, Master's thesis TFRT–5443, Department of Automatic Control, Lund Institute of Technology, Lund.