27

Intelligent filtering in network management systems

M. Möller, S. Tretter, B. Fink
Philips Research Laboratories Aachen
Weißhausstr. 2, 52066 Aachen, Germany
Tel: +49 241 6003-{510, 552}, Fax: -519
{moeller, tretter}@pfa.philips.de

Abstract

Network management systems have to handle a huge volume of notifications reporting unprompted on events in the network. Filters that reduce this information flood on a per-notification basis fail to perform adequate information preprocessing required by management application software or human operators. Our concept of intelligent filtering allows for a highly flexible correlation of several notifications: Secondary notifications can be suppressed or a number of notifications can be aggregated. An intelligent filter was implemented using a rule-based language and was applied within SDH network management. Several modules, configurable while the filter is operating, support the user considerably and with excellent runtime performance. Further development is envisaged that provides for smooth integration into management application software.

Keywords

Network Management, Event Correlation, Filtering, Synchronous Digital Hierarchy

1 INTRODUCTION

1.1 The problem

Networked systems are growing in size and complexity, which means that a vast amount of information has to be handled by their management systems. Most of this information is pro-

A. S. Sethi et al. (eds.), *Integrated Network Management IV*

duced spontaneously: Notifications report on certain events within the network, e.g. a status change of a network element or an equipment malfunction. To make effective management possible - be it performed automatically by software components or carried out by the human operator - this message flood has to be preprocessed. Such preprocessing has to correlate information from different network resources and, based on these correlations, has to suppress superfluous notifications, generate lost notifications or aggregate notifications.

So far information preprocessing is mostly performed by filter modules that reduce the information flow in a context-free manner. This means that for a single notification it can be decided whether it will be suppressed or not, depending on the information it is carrying. Correlation of information from several notifications is still left to the management application or the human operator, e.g. to identify the primary message and neglect the secondary ones when a message burst is caused by a faulty component, or to condense several messages carrying superfluous details into one with more abstract information.

Intelligent filters are software components within the management system that perform this preprocessing task. They can be used to directly support the human operator as well as to separate tasks within a management application software.

1.2 Example: notifications in an SDH network

Within telecommunication networks using the new *Synchronous Digital Hierarchy* (SDH), correlation of notifications is very important. SDH has the ability to detect faults on its different capacity levels via embedded overhead information such as check sums and trail labels. In the standard information model (ITU-T G.774) the detection capabilities of the hardware (ITU-T G.783) manifest themselves as a set of termination points representing the multiplexing hierarchy and offering hooks for a management system. Within this model each termination point is able to send notifications concerning the transmission connection it is terminating.

The example in Figure 1 shows the alarm notifications sent in case of a failed transmission line with capacity STM-1 (155 Mbit/s), which is the basic transmission rate for SDH. In the example, one initial fault causes two primary alarm notifications: Two LOS (Loss Of Signal) notifications report on a loss of the carrier signal detected by the physical interfaces. But since the STM-1 carrier is able to transport up to 63 2-Mbit/s signals, up to 254 AIS notifications (Alarm Indication Signal, propagated via in-band signalling) are also sent by the termination points of the multiplexing hierarchy down to those of the affected 2-Mbit/s signals.

The example is based on the multiplexing structure for 2-Mbit/s transmission according to the ITU-T recommendation (ITU-T G.709) as it is used in Europe. The use of STM-16 (2.5 Gbit/s) transmission lines (the highest transmission capacity currently supported) would increase the number of notifications by a factor of 16.

1.3 Requirements on intelligent filters

By studying the functional capabilities of information preprocessing needed by automatic management software and human operators we identified the following main requirements:

Filtering Functionality: The filter should be able to perform notification suppression, compression and aggregation. This means it decides on the forwarding/non-forwarding of notifica-

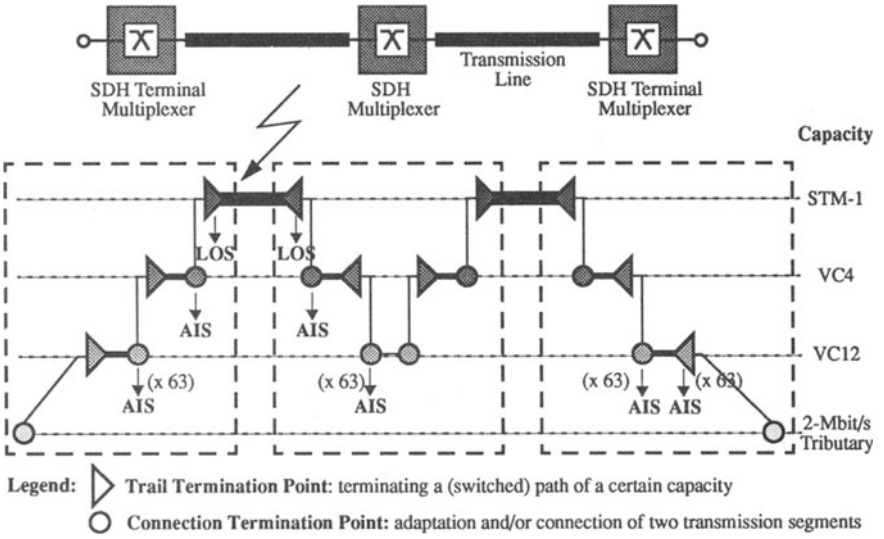

Figure 1 Example line failed on an SDH transmission line.

tions, it condenses multiple occurrence of the same notification to one and it generates notifications of higher semantical level. Suppression, aggregation and compression of notifications are based on their correlation over time and over different resources.

Ordering: The filter has to preserve the order of incoming events.
It may well be that events have overtaken each other, which can be detected by looking at their time-stamps. Nevertheless this reversed order may be of relevance for diagnosis and thus should not be corrected. The filter has to have an internal mechanism to correlate events arriving in an order deviating from the order in which they were generated.

From this and the functional requirements it follows that the filter has to have a notion of time.

Modification: A filter has to allow its modification concerning two aspects:
(1) The set of event types that shall be dealt with and the set of nodes they may come from may vary over time due to the dynamics of the networked system.
(2) Which events have to be suppressed, aggregated or generated at what time are filter parameters that may change.
The filter must allow these modifications to be made at runtime.

Performance: A filter has to guarantee a certain throughput: The time period necessary to process events (that is the decision to forward it, to suppress it or to send a generated event based on it) has to be limited.

Scalability: A filter must be applicable to networked systems of any scale and has to support the dynamic growth of the system.

So, what we call an 'intelligent filter' is a software component with three interfaces: Its *input* are notifications in a certain format, its *output* are notifications of the same format. An output notification is either generated by the filter or has been part of the input. Via the *modifi-*

cation interface the filter's behaviour can be adjusted. This definition is an extension of the discriminator object as introduced in the management standards (ITU-T X.734).

2 EXISTING APPROACHES

There are a number of significant publications on the topic of intelligent filtering (Boda, 1992, Brugnoni, 1993, Jacobson, 1993, Lewis, 1993, Pfau-Wagenbauer, 1993, Deters, 1994). As an application area all those use *fault diagnosis*. It should be noted here that of course this is the area where the most operator knowledge is available, but that in general filtering is required in other management areas as well. For example a certain network performance, deducible from several messages, is not necessarily a fault, but an item of information important for taking appropriate performance management measures. Thus we would like to apply filter components to any area where it is necessary to focus on relevant information and to discard unnecessary bits.

Most systems come as stand-alone or higher-level systems. That means that they work in addition to normal network management systems (Jacobson, 1993) or on top of them (Brugnoni, 1993). Our aim is to devise filter components that go into a management application in various quantities and at various places rather than having one big filter system. For this reason we see filters as passive components in the following sense: They work on the incoming notifications only, and do not retrieve additional information (such as attribute values) from the network. Thus a filter as such cannot perform diagnosis - the information carried by events does not in general suffice for this.

Correlation systems can have different underlying paradigms that are all taken from the area of Artificial Intelligence. Approaches that are based on Neural Networks or Case-based Reasoning rely on the fact that there is a large validated base of cases or training examples available (Lewis, 1993, Deters, 1994). However, when it comes to new technologies such as SDH, there is not enough operating experience and therefore no training set is available. Approaches that are based on models of either correct or faulty system behaviour (Pfau-Wagenbauer, 1993, Jacobson, 1993) seem easier to obtain, but have to allow for customisation at runtime in order to adapt the model to real system behaviour. We adopted the model-based, manipulative approach.

3 INTELLIGENT FILTERING

3.1 Design of the intelligent filter

The requirements for intelligent filtering led to the following design decisions:

- We chose a rule-based approach for the shallow model of notification dependences.
- Rules have to describe what shall be forwarded rather than what has to be suppressed.
- We divided the filter into modules, each of them responsible for a certain functionality and a part of the network.

- Each module is divided into three parts, describing the dependences, the topology of the network (or subnetwork) under consideration and rules for the filtering process as such. The dependences and the topology can be manipulated at runtime.
- The modules work jointly on the notification buffer, where notifications are stored for a certain time interval.

Figure 2 outlines this design. Notifications arriving from the network are preprocessed in order to obtain facts as used by the rule-based system. E.g. a message *1;27;mo.Hoern_Ac_040; mo.Hoern_Ac_040:TTP.0;comAl;191.2500;ATU.0;LossOfSignal;1* would result in the internal fact *occurred(mo.Hoern_Ac_040:ATU.0, LossOfSignal)*, stating that a loss of signal occurred at adaptation termination unit 0 of the managed object Hoern_Ac_040.

After preprocessing, the event is classified with respect to its relevance for filtering. This is done with reference to dependences and topology information contained in the filter modules or with reference to explicit discrimination constructs that are formulated on a per-event basis. This means: If a module contains rules that refer to this event, the event is stored in the notification buffer. If a discriminator construct determines that an event shall be forwarded, this event is directly passed to the postprocessing function. This case allows for context-free filtering as specified in the ITU-T standards (ITU-T X.734). If neither discriminator constructs nor modules refer to the event under consideration, the event is absorbed.

The notification buffer stores events for a predefined time-period. During this period, the different filter modules work on the buffer's content. Each module can mark events in the buffer as 'to be forwarded' or can put new events into the buffer. When the lifetime of an event in the buffer has elapsed, the event is forwarded only if it is marked accordingly. All unmarked events are absorbed.

The postprocessing function is the inverse of the preprocessing function, so that notifications leave the filter in the same format and the same order they had when they entered it.

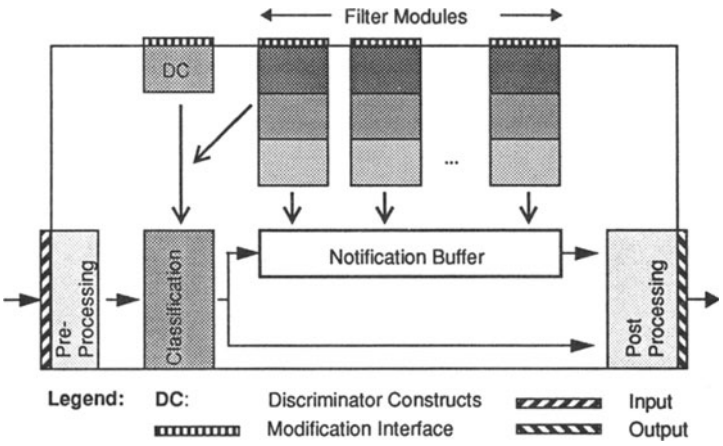

Figure 2 Intelligent filter design.

3.2 Design of a filter module

A filter module consists of 3 parts:

Topology information describe explicitly network elements and lines between them that are subject to the module's filtering functionality. Example 1 (Table 1) states that there are two adaptation termination units and that a line exists between them.

Dependences relate notifications from different nodes to each other. For instance a dependence may state that one event causes another event on the same managed object (MO, example 2 in Table 1) or that two events from different managed objects shall be aggregated to a third one (example 3).

Filtering function rules describe what the module actually performs based on topology and dependences. A causal relation as given in example 2 might lead to different actions: The *Loss of Signal* notification is forwarded and the *Alarm Indication Signal* notification is simply dropped, or the *Loss of Signal* notification is forwarded with the additional information that a certain *Alarm Indication Signal* that was a consequence of this has been dropped. This would allow the human operator to inspect the logfile later on if more detail is needed. Example 4 shows the former case, using variables for managed objects and events. Please note that all notifications that are not forwarded are absorbed by default - it is not necessary to express this.

Table 1 Examples of intelligent filter parts

Example 1: Topology Information
atu(mo.Hoern_Ac_040:ATU.0), atu(mo.City_Ac_04:ATU.3), line(mo.Hoern_Ac_040:ATU.0, mo.City_Ac_04:ATU.3)
Example 2: Causality between Events
causes(LossOfSignal,AlarmIndicationSignal)
Example 3: Aggregation of Events
aggregate(mo.Hoern_Ac_040:ATU.0, LossOfSignal, mo.City_Ac_04:ATU.3, LossOfSignal, Line_Ac_012, LineFailed)
Example 4: Filter Function Rules
causes(E_1, E_2) & occurred(MO, E_1) & occurred(MO, E_2) -> forward(MO, E_1)
Example 5: Rules with Time Annotations
(occurred(MO_1, E_1) & occurred(MO_2, E_2)/0.05)/0.01 -> forward(L, E_3)

3.3 Implementation of the intelligent filter

To implement the filter we used RTA, a rule-based language developed at Philips Research Laboratories (Graham, 1991). Within this language rules can contain time annotations that denote durations of facts or delays in firing. For example the rule given in example 5 (Table 1) states that if E_1 occurs at MO_1, and E_2 occurred at MO_2 no less than 50 ms earlier, then E_3 will be forwarded after another 10 ms.

The chosen language can be compiled. At runtime, rules can fire concurrently, and the RTA runtime system takes care of synchronisation among rules and with system time. Furthermore the language supports an interface to C so that function calls can be attached to facts. This allowed us to realize pre- and postprocessing functions as well as the classification function in C. RTA supports facts and rules being turned ON and OFF from outside. This way topology and dependences can be changed at runtime. However, 'compiled' rules mean that all possible values of variables have to be known at compilation time, so that changes at runtime can only be made within a range that is known in advance. If this range is too limited, modules have to be changed at source code level and recompiled. For this, RTA supports modules being loaded and unloaded at runtime so that a module can be exchanged while the others are still executing.

3.4 Modules for filtering in SDH networks

Each module can perform a certain filter functionality on a certain part of the network. It does so steered by dependences. In our implementation for the management of SDH networks we decided on three modules that cover the following functions:

- Simple Compression:
  All notifications of the same type from the same managed object within a given time interval are compressed into one notification.
- Causal Suppression:
  Notifications that are secondary ones are suppressed. Their identifiers are attached to the primary notification.
- Aggregation:
  Several notifications from one or more managed objects are aggregated to a new notification.

All three modules work on the entire network. However, the modules could have been configured in such a way that, for example, aggregation is only performed in one part and causal suppression in another.

3.5 The intelligent filter in use - a simple scenario

At runtime, the human operator or the management application software can configure the filter modules in various ways. A little scenario shall now demonstrate part of the overall functionality. For this, the filter is assumed to be used in a configuration as depicted in Figure 3: several managed objects are controlled by a manager (this can be a human manager or a software component). The notifications emitted by the managed objects are passed to the manager

via the filter. The manager can manipulate the filter (and of course the managed objects - this is, however, not depicted here).

Let us assume that at a certain point in time no modules are loaded within the filter, and let us assume that a line fails in the network. All managed objects concerned by this will now emit notifications; since no module is loaded, the filter will not suppress anything. All notifications will arrive at the manager that has to process this information flood. The manager can now load certain modules. It starts by loading the Compression module. By default, no dependences are switched on when this module is loaded, so the manager might decide that all operational state changes to '0' and all communication alarms with the severity 'major' from the same managed object shall be compressed into one notification with the semantics 'Multiple operational state changes to '0' occurred' or 'Multiple major communication alarms occurred' at the respective node. When another line fails in the network, the manager will now receive these two notifications from each of the two managed objects that are connected by the failed line.

This type of information is not the most appropriate to diagnose the failed line. Therefore, the manager unloads the Compression module and loads the Suppression module. Some of its dependences state that when a *Loss of Signal* is reported from an MO, the same MO will send out various other notifications caused by this. In fact, when a line fails, the two adjacent MOs both send a *Loss of Signal*. With the Causal Suppression module loaded, these two messages will now be forwarded with a list of identifiers attached to each of them. The lists denote the notifications suppressed as secondary and enable the manager to look those up in the logfile should this be necessary.

In some cases only two *Loss of Signal* messages might be too little information; e.g. it might be vital to be informed about major communication alarms, even if they are secondary. For this reason the manager can switch off the dependence causes(LossOfSignal, MajorCommunicationAlarm) within the Causal Suppression module. Now the filter sends two *Loss of Signal* notifications and all major communication alarms. Of course the latter are missing in the list attached to the former.

The manager can now in addition load the Aggregation module. This effects that two *Loss of Signal* notifications are aggregated to *Line L failed* if they are emitted by two MOs that have a common line L. However, due to the semantics of the Causal Suppression module, the two *Loss of Signal* notifications are forwarded as well. To suppress these, the Causal Suppression

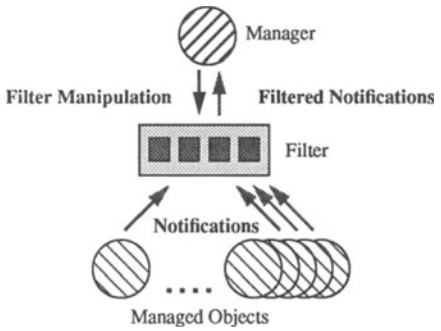

Figure 3 Filtering scenario

module has to be unloaded. The effect is that when a line fails only the message *Line L failed* is received by the manager.

This way the filter can be configured to cover various demands for information preprocessing. Should a situation occur where the filter's configuration is found to be not optimal and thus relevant information is not presented, the filter can be re-configured and re-run on the log-file off-line.

4 EVALUATION OF THE CHOSEN SOLUTION

The intelligent filter has been put to the test against an SDH network simulator and against a notification generator (Beyerlein, 1993). These experiments carried out on a SUN SPARCstation 10 under Sun OS 4.1.3 showed that the implementation is very fast: For a network with 13 network elements and 13 lines, 1000 notifications/s were sent to the filter for a period of one minute; during this minute the lag of the filter behind realtime rose linearly from 0 to 5 s. This means that during a heavy notification burst notifications left the filter not later than 5 s after they had entered it.

This excellent runtime behaviour can be attributed to the fact that the RTA language is compiled. This means the RTA compiler instantiates all rules that contain variables with possible combinations of their values. These variable-free rules can then effectively be executed at runtime. The memory space needed to do this is in the order of $(N_{MO}*T_N)^k$, where N_{MO} is the number of MOs, T_N is the number of notification types in the network under consideration and k is the maximal number of notifications that can occur in a correlation dependence. For the example mentioned above this leads to a memory consumption of 1.5 Mbyte. This means that only very few filters of this size are likely to be run at the same time.

The scenario in section 3.5 showed the high flexibility of the filter with respect to module loading and unloading as well as to setting topology facts and dependences ON or OFF at runtime. This is only possible, however, for facts and dependences that have been foreseen at compile time. For a dynamic network, though, where managed objects and relations between them are created dynamically this is not appropriate. Consider for example path objects in SDH networks: A path is a connection between two nodes that is switched via several other nodes; a path is provided to a client for a certain time period after which it does not exist any longer. Our filter would have to define beforehand as many path objects as could be present simultaneously. A more dynamic way of dealing with scalability is necessary.

Besides the fact that the filter's memory size limits the number of filters that can be integrated into management application software, the problem of how to integrate the filters at source code level has not yet been studied. So far, management application and filters are coded separately; no cross-checking can be performed before runtime. It is necessary to augment the language chosen for the development of management applications by filter constructs.

The filter is designed so as to perform multi-stage filtering (although this has not been applied in the implementation). Multi-stage filtering means that intelligent filtering is performed - again - on the filters output; e.g. aggregated messages could be aggregated another time. Two ways of implementing this can be envisaged. First, one can construct filter chains, which means directly coupling filters such that one filter's output is the next filter's input. Looking at the internal functionality (Figure 2), this would mean performing unnecessary post- and pre-

processing. The second approach is to add further modules that perform higher-level filtering: These modules would work on the same notification buffer, but would only consider notifications that are deemed to be forwarded by other modules or created by them.

5 FUTURE WORK

As stated before, the implemented intelligent filter is not integrated into the management application. This is a major drawback since management applications would want to influence the intelligent filter by:

- specifying or removing filter rules according to their special needs *and*
- supplying the filter with topology information during runtime to perform event correlation.

The management applications' notification handling is currently done by the installation of *event forwarding discriminators* (EFD) in an event distributor and specialised notification handlers in the applications themselves. The EFDs allow for filtering on single notifications only. Triggered by incoming notifications, the notification handlers perform arbitrary management actions with the application's state variables and one notification's additional information as parameters. This means that for context-sensitive filtering the context would have to be coded explicitly into the application's state and that the event correlation would be mixed up with the notification handling.

The approach we envisage now is to integrate intelligent filtering into the management language used for the application creation. Briefly, this language is an extension of GDMO (ITU-T X.722) in that it also allows managing objects to be specified and makes GDMO's behaviour clauses operational (DOMAINS, 1992).

A management application that wants to correlate notifications will have to implement a filter package with application specific filter rules. These will consist of boolean expressions over facts and relations on the left-hand side. A *fact* refers to the fields of one notification only; a *relation* refers to several notifications, it can for example contain topology information. Instance variables for notifications and managed objects can be used within rules. If a rule fires, a special action for the recognized situation (stated on the right-hand side) is called with all the necessary information from the left-hand side of the rule.

Example rule for the situation *Line Failed*:

```
[(N_1.type = comAlarm) & (N_1.probableCause = LOS)]          /* fact(N_1) */
[(N_2.type = comAlarm) & (N_2.probableCause = LOS)] &        /* fact(N_2) */
line(N_1.instanceName, N_2.instanceName, L)                  /* relation(N_1, N_2, L) */
-> lineFailedHandler(N_1.instanceName, N_1.time, N_2.instanceName, N_2.time, L)
```

where N_1 and N_2 are notification variables and L is a variable for a managed object.

Topology information as referred to by relations will not be hard-coded in the filter package but will be provided by the application during runtime. Thus the application is responsible for updating the topology information according to its knowledge about the network. For the new filtering scenario see Figure 4.

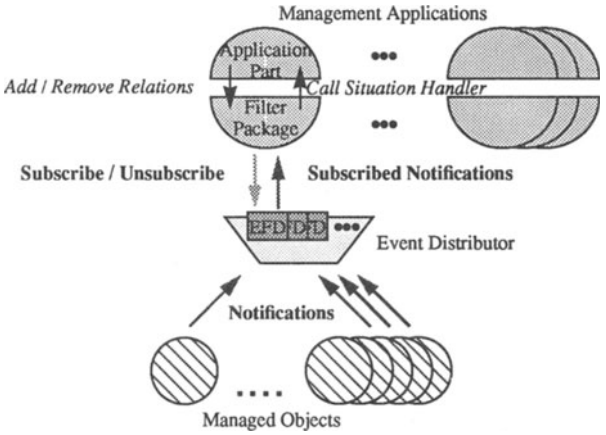

Figure 4 New filtering scenario

Consequently,

- the syntax of the application creation language will have to be extended by constructs for the specification of relations and for the filter package's rules *and*
- a runtime system has to be provided that handles a time window of incoming notifications, matches the filter rules and calls the specialised handlers.

A study should be carried out as to whether it is possible to automate the installation of EFDs in the event distributor based on the information given in the rules' facts. Then the application would be independent of the registration and only notifications that match at least one fact would be sent.

6 CONCLUSION

We have designed and implemented a powerful tool for intelligent filtering on notification streams. This has been evaluated by application to the network scenario of the Synchronous Digital Hierarchy. We have presented this application to network providers and found that there is a need for such tools and that our tool is suited for use by human operators. It can be used as a basis for professional tools enabling diagnosis and off-line logfile inspection. First concepts that allow for smooth integration of several smaller filters into our management system have been formulated. They are still to be implemented and tested.

7 REFERENCES

Beyerlein, R. (1993) Intelligent Filtering in Management Systems. Diploma Thesis Philips Research Laboratories Aachen / University of Dresden (in German).

Boda, M., Brandt, H., Gustafson, E. and Kling, L. (1992) Application of Neural Networks in Fault Diagnosis. Proc. XIV International Switching Symposium, Yokohama October 1992, pp 254-258.

Brugnoni, S., Bruno, G., Manione, R., Monatriolo, E., Paschetta, E. and Sisto, L. (1993) An Expert System for Real Time Fault Diagnosis of the Italian Telecommunications Network. Proc. IFIP 4th Int. Symp. on Integrated Network Management, San Francisco May 1993, pp 617-628.

Deters, R. (1994) Case-Based Event Correlation. Proc. 14th Int. Avignon Conference (AI 94), Paris May/June 1994.

DOMAINS (1992) DOMAINS Management Language. Deliverable D2c ESPRIT Project 5165 DOMAINS, May 1992.

Graham, M. and Wavish, P. (1991) Simulating and Implementing Agents and Multiple Agent Systems. Proc. European Simulation Multiconference, Copenhagen June 1991.

ITU-T G.709 Synchronous Multiplexing Structure. ITU-T Recommendation.

ITU-T X.722 OSI: Structure of Management Information: Guidelines for the Definition of Managed Objects. ITU-T Recommendation.

ITU-T X.734 OSI: Systems Management: Event Report Management Function. ITU-T Recommendation.

ITU-T G.774: Synchronous Digital Hierarchy (SDH) Management Information Model. ITU-T Recommendation.

ITU-T G.783: Characteristics of Synchronous Digital Hierarchy (SDH) multiplexing equipment functional blocks. ITU-T Recommendation.

Jacobson, G. and Weissman, M.D. (1993) Alarm Correlation. IEEE Network Nov. 1993, pp 52-59.

Lewis, L. (1993) A Case-Based Reasoning Approach to the Resolution of Faults in Communication Networks. Proc. IFIP 4th Int. Symp. on Integrated Network Management San Francisco, May 1993, pp 671-682.

Pfau-Wagenbauer, M. and Nejdl, W. (1993) Model/Heuristic-Based Alarm Processing for Power Systems. AI EDAM 1993 7(1), pp 65-78

The Authors

Marita Möller obtained her Diploma and Doctor's degree in Computer Science at the Technical University of Aachen, Germany. Her main areas of interest are Network Management and Artificial Intelligence.

Stefan Tretter graduated in Computer Science at the University of Kaiserslautern, Germany. He is a specialist in Telecommunications Network Management and Distributed Systems.

Barbara Fink received her Diploma in Electrical Engineering from the Technical University of Aachen, Germany, in 1967. Her key activities are architectures and computer languages.