presented at: DSOM'98

INTEGRATED EVENT MANAGEMENT: EVENT CORRELATION USING DEPENDENCY GRAPHS

Boris Gruschke*
Department of Computer Science, University of Munich
Oettingenstr. 67, 80538 Munich, Germany
Phone: ++49-89-2178-2170, Fax: -2262
E-Mail: gruschke@informatik.uni-muenchen.de
Web: http://wwwmnmteam.informatik.uni-muenchen.de/~gruschke

Abstract

Today's fault management requires a sophisticated event management to condense events to meaningful fault reports. This severe practical need is addressed by event correlation which is an area of intense research in the scientific community and the industry. This paper introduces an approach for event correlation that uses a dependency graph to represent correlation knowledge. The benefit over existing approaches that are briefly classified here is that this approach is specifically well suited to instrument an existing management system for event correlation. It thereby deals with the complexity, dynamics and distribution of real–life managed systems. That is why it is considered to provide integrated *event management. The basic idea is to connect the event correlator to a given management system and gain a dependency graph from it to model the functional dependencies within the managed system. The event correlator searches through the dependency graph to localize managed objects whose failure would explain a large number of management events received. The paper gives a short overview of existing approaches, introduces our approach and its application. It shows why dependency graphs are suitable, how they can be derived and finally presents the prototype developed so far.*

Keywords: Fault Management, Event Management, Event Correlation, Dependency Graph, Corba

1 Introduction

With the ongoing growth of distributed systems in size and complexity, the fault management of heterogeneous networks, end systems and applications is becoming more and more complex. It is widely accepted that efficient fault management requires an adequate management of events because of them triggering the whole fault management process. However, modern fault management is characterized by events that are inadequate to trigger further actions. The most unpleasant phenomenon is an *event storm* (aka event burst): Effects of faults are frequently detected by many managed objects simultaneously causing them to notify the management about some symptoms of the problem. The administrator gets flooded by events while being unable to retrieve the relevant information.

*The author's work is sponsored by Siemens Nixdorf Informationssysteme.

This severe practical problem is addressed by event correlation tools. Event correlators try to condense many events each with little information to few meaningful events. Classical filtering mechanisms that discard or forward events based on filter patterns are widely used but are not sophisticated enough. Unlike the related research area of *fault diagnosis* ([DR95], [KP97]), event correlators are not intended to perform in–depth analysis of particular fault scenarios by generating and analyzing sequences of diagnostic tests. Instead they passively investigate events to perform a broad first–level localization for a wide range of faults or, in the best case, for all kinds of faults.

Figure 1 shows the general design of an event correlator. Here a fault in a router causes other networking devices to complain about broken links and various systems and applications about lost connections. The event correlator receives these events, condenses and forwards them to the administrator responsible for the faulty routing device. The core part of an approach to event correlation is to design a repository from which the correlator gets information about whether and how incoming events are related, denoted as *knowledge base* in the figure. In practice, a hierarchy of event processing tools is used for scalability reasons.

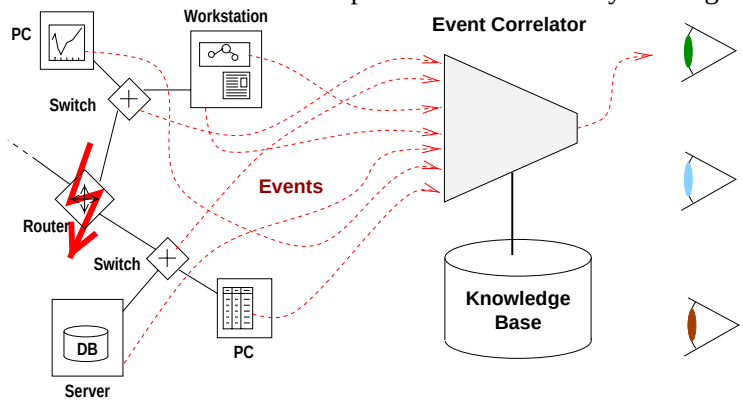

Figure 1: High–level Design of an Event Correlator

An event correlator has to address three problems: First, it has to reduce the overall number of events especially to eliminate event bursts. This directly reduces the workload of administrators and decreases the period of time between detection and repair of a fault. Rather than searching through excessive event lists, administrators can start with in–depth diagnosis and repair immediately. Second, events should indicate the cause of a problem instead of its symptoms. Deducing the cause of a fault from a set of symptoms puts high demand on expertise and skills of administrators. Meaningful events reduce the training and learning costs for administrative personnel. And third, today's events are frequently reported to the wrong persons which causes double work and confusing work flows. Events forwarded by an event correlator should contain the information needed by a trouble–ticket system to notify the right expert. These three problems are related: All of them may be solved if the event correlator is able to deduce the cause of the problem from the received symptom reports, i.e. do root cause analysis.

Figure 2 shows the correlation of connectivity problems in a simple scenario. Imagine components of two distributed applications complaining about lost connections to their servers. By knowing the network topology the correlator may find out that those events might be related and point to the router. Note that a fault might be a transient or partial problem. So the management events may reach the management system even if no alternative routes are available. This example looks simple but is challenging to state–of–the–practice event correlation products because they have to access a

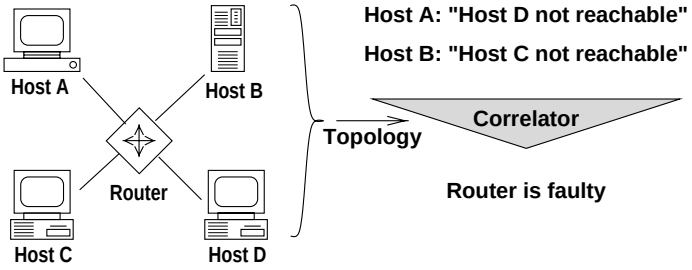

Figure 2: Correlation of Connectivity Problems

NMS's topology database. Nobody wants to enter the network topology manually. In general, retrieving information from the management system is just beginning even in the well–established area of network management. Still, it is far away for service management.

This paper is structured as follows: Section 2 gives an overview of the approaches developed in the research community and the industry so far and outlines still remaining problems of event correlation that are in the focus of the approach presented here. Section 3 introduces the key concept: dependency graphs. It shows how a dependency graph can be actually used to do event correlation. Section 4 presents what has to be done to apply the approach and section 5 gives an overview of the prototype developed so far. Finally, section 6 summarizes the paper and gives an outlook to further work.

2 Existing Approaches

2.1 Overview of Existing Approaches to Event Correlation

One group of approaches developed so far ([Lew93], [KYY+95], [HKH+96]) starts with a set of *cases* (codes, episodes) containing symptom–cause pairs and does some reasoning on them. When receiving events such a *case–based event correlator* regards them as a set of symptoms. It searches through its case database for cases with similar symptom sets and presents the stored solutions if available. Key points of approaches from this group are how to define the similarity of cases appropriately and how to acquire a suitable case database. For the latter issue trouble ticket archives storing a history of problems are investigated ([DR95]) for example. A recent proposal from [WTJ+97] is to use a neural network to define similarity. The outcome of case–based approaches obviously heavily depends on the quality of the case database.

The other group of approaches ([GH98], [WBE+98], [OMK+97], [MSS97] [Nyg95], [JW95] and [BBM+93]) and most products define some *programming* (or specification) language that enables an expert to enter his knowledge thereby building up the event correlator's knowledge base. A popular programming style is rule–based: "If event X and event Y arrive, then emit event Z". The other style is model–based (e.g. [OMK+97]): The managed system is modeled with respect to its event emission behavior (*event model*). The programmed knowledge somehow defines events to be related under some temporal or sequential constraints.

Products like Tivoli/EnterpriseConsole,[1] Siemens–Nixdorf/TransView and Micromuse/NetCool customize general purpose programming languages for event correlation requirements. Others such as HP Event Correlation Services (ECS), NerveCenter and Smarts/InCharge[2] invent own languages (*Correlation Circuits*, *Behaviour Models*, *MODEL*) specialized in event correlation. All approaches from the second group have in common that they presume a programmer with expert knowledge as primary source of information.

2.2 Open Issues

In spite of this intense research for the last years the demands for more comprehensive solutions remain. From the analysis of today's practice the following requirements not yet addressed were derived that should be kept in mind if developing an approach for event correlation:

[1] On `http://wwwmnmteam.informatik.uni-muenchen.de/proj/evcorr/public-htdocs` the author maintains quite an exhaustive list of existing event correlation products and links to online documentation and white papers.

[2] InCharge also incorporates codebook correlation belonging to the first group.

1. Event correlators are needed if the *complexity* of the managed system reaches an extent that renders fault management infeasible. But with existing approaches the amount of faults that are possible (i.e. the amount of cases), the amount of rules to program or the size of the model to design makes event correlators either infeasible as well or little used. One can not expect to reduce the complexity of an event correlator's knowledge base drastically compared to the managed system's complexity. The matter is to live with.

2. *Dynamics*: High change rates within the managed system fully hit the event correlator. New requirements are only one item. Whenever equipment respectively its configuration changes, new technologies are invented or other management tools get used, this has implications on to how the management events have to be interpreted by the correlator. In practice there is no abstraction layer that provides "generic" events.

3. *Distributed knowledge*: Most vendors presume to address the needs of distributed management by distributing event processing units. However, the knowledge required to set up an event correlator is highly distributed as well. In practice there is no single expert who has sufficient knowledge to feed an event correlator. Instead, the knowledge is distributed across many people at various organizational units.

4. And finally, the *reliability* of an event correlator is a critical factor for the whole process of fault management. Inconsistencies between the event correlator and the real system mislead the administrator instead of supporting him, turning fault localization sometimes worse than before.

So, from a software engineering point of view event correlation is the worst–case: Many people have to continuously maintain a complex program without making mistakes. The answer to the question how to live with the issues mentioned above is *integrated* event management, where an event correlator is used in conjunction with an existing management system.

The idea that initiated the approach presented here is that the existing management system could be instrumented as shown in figure 3 to ease the above problems. Instead of offering an user interface to enter knowledge, an interface is offered that connects the event correlator with other parts of the management system. The aim is to design an event correlator that retrieves its knowledge base from management data already present in the management system. While many approaches and products are currently extended in that direction (topology–based correlation) this approach is constructed with the philosophy of integrated event management in mind from ground up. When watching the evolvement of state–of–the–practice management models it becomes obvious that one can not presume a homogeneous, semantically rich overall management model for the near future. Therefore the challenges of this approach are to deal with the heterogeneity of the management itself and to find the "right" level of semantics for the knowledge base. When too much semantics is demanded building up the knowledge base takes too much effort. Too little semantics prevents a reasonable event correlation

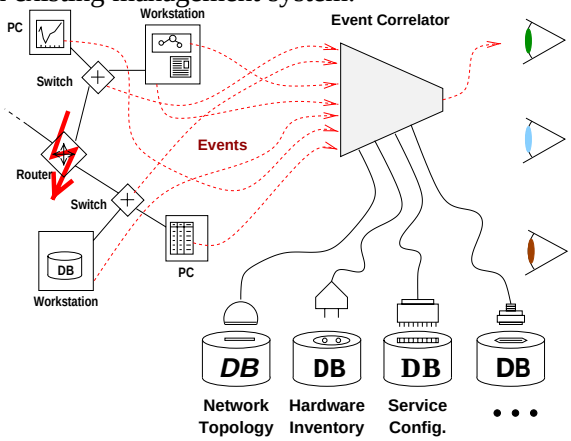

Figure 3: Connecting to the Management System(s)

algorithm on that knowledge base.

3 Concept of an Event Correlator with Dependency Graphs

3.1 Dependency Graphs

The event correlator designed within this approach uses a *dependency graph* to represent the knowledge about the correlation of events. A dependency graph models one aspect of the managed system: Its nodes (objects) reflect the managed objects (MOs) of the system. Its edges reflect the functional dependencies between MOs. "An object A depends on an object B" means that a failure in B *can* cause an failure in A. In contrast to object–oriented models there are no attributes or methods or other relations than dependency.

The managed system is dynamic, therefore the dependency graph is dynamic, too. Its edges and objects are added and deleted so that changes in the real system are approximated. The complexity of the managed system does not disappear here. It is reflected by a very large number of objects and dependencies. But a large graph can be better handled than a large program.

A graph seems to be a suitable data model for event correlation: First, graphs are quite easy to generate from whatever management models, especially from object–oriented system models with relations or associations between objects. Second, the operations permitted on graphs can be implemented in a robust manner, i.e. adding or deleting objects and dependencies can not cause the correlator to hang. Third, dependency graphs are naturally manageable in a distributed manner. Objects and dependencies can be added or deleted by different administrators quite independent of each other.

Dependency graphs are an attempt to answer the question "How much semantics is at least required for a system model sufficient for event correlation?". Focusing on the minimal requirements reduces the amount of work when applying the approach: building and maintaining the knowledge base. This design criterion is strongly guided by the environment in which event correlators are used: Event correlators are management applications extending existing management systems when the inefficiency of event management becomes obvious. Therefore the availability of plenty of management data can be assumed but most of it being stored in a proprietary way. Due to the lack of a generic representation of dependencies in current management systems generating a dependency graph requires code specific to a certain management tool. If one wants to generate the event correlator's model from existing data, it becomes clear why minimalism is advisable. From the mapping of management information models we know that information gatewaying is complicated and comes along with loss of semantics.

Many extensions to the concept of dependency graphs may be considered if corresponding management data is available. For example probalistic networks could be used to distinguish likely from unlikely dependencies [DLW93]. Various kinds of objects and relations could be distinguished and finally attributes and methods could be added to relations and objects to get a full–blown object–oriented model of the managed system which could be further extended with a behavior model.

The idea that functional dependencies between managed objects are required is not specific to event correlation. Many management applications need it or benefit from it, e.g. to compute service availability like in [DRK97], for change management or network planning. Among the extensions of dependency graphs are those that associate probabilities to the arcs of the graph ([KS95]). Thus the demand is not to invent a new model for event correlation but to formalize one that is needed anyway so that it is available to tools like an event correlator.

3.2 Doing Event Correlation with Dependency Graphs

Given a dependency graph, our event correlator works as depicted in figure 4: First, the raw events received by the event correlator are mapped to corresponding objects of the dependency graph. An input event is hereby interpreted as fault message and mapped to the object that is indicated to be faulty. Second, starting from these initial objects a search through the dependency graph is performed that looks for objects from which all (or many) initial objects depend on. These common dependent objects are forwarded as a condensed event. So the result of a correlation is a single event containing the analysis of the received set of events: A list of objects to be interpreted as possible explanations for the symptoms.

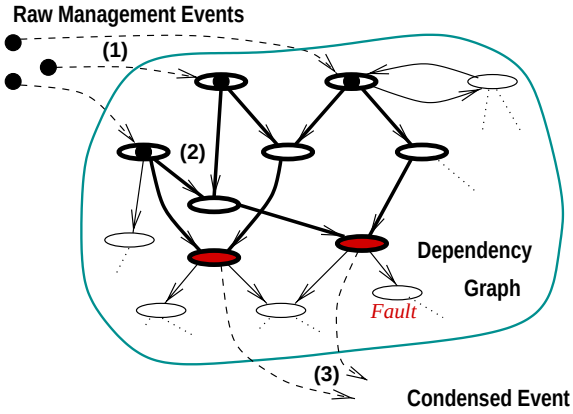

Figure 4: Working principle of the algorithm

If the correlation went well at least one of these objects points to the actual fault at least as good as the best event received. The metric for the quality of a fault indication is the length of the path from the indication (i.e. the reported object) to the cause (the actual fault) in the dependency graph because an administrator has to deduce a cause–symptom sequence similar to this path when localizing the fault.

The algorithm assigns one of two states to the objects of the dependency graph: faulty and correct. Therefore MOs as defined by the common understanding may have to be modeled in the dependency graph by multiple objects and MO–internal dependencies to properly reflect them. This simple state model relieves us of the necessity for a behavior model as an extension to the MOC hierarchy. Furthermore, the condensed events are made up of objects of the dependency graph. So defining the objects of the dependency graph also determines the error messages delivered to the administrator. These points lead to criteria for developing dependency graphs used below in section 4.

The algorithm that has been implemented is not presented in great detail here due to its complexity. However, the design criteria that are driven by the requirements of event correlation can be given:

- The real–time requirements encourage a breadth–first search strategy that finds “obvious” common dependent objects fast.
- To let the algorithm still work in event storms, a mechanism to avoid overloading is used. The simple way is to process a subset of the input events based on some criteria. A more sophisticated strategy is used here: The algorithm allows events to be continuously added to the search avoiding predefined subset definitions which may be too small or too big for the particular case. This strategy allows the event correlator to deliver a result even if it can not process all events in a timely manner.
- To limit the number of reported potential explanations the correlator dynamically adjusts the “hit level”. That is the threshold that defines how many initial objects must depend on an object to turn this object into a potential explanation (“hit”) that is reported.
- The length of the search path is limited to let the correlator finish work after some time.

As can be seen, the order of the events processed is important. "Good" events should be processed early. One criterion for the order of the events is of course the time of arrival at the event correlator. If there is more than one event in the input queue which is quite likely in event storms reordering the events is considered. A good generic strategy is based on the *depth* of an object within the managed system. Deep MOs are represented by objects with few dependent objects. Especially objects with no dependent objects at all can be identified easily.

An event correlator can not really prove whether events are correlated. The algorithm is based on the global assumption that it is unlikely for two independent faults to occur at about the same time. If there are multiple faults at the same time the algorithm will correlate to one of these faults only. In general, the results are as good as the underlying dependency graph. In practice it can not be expected that the dependency graph is complete, i.e. reflects *all* real dependencies. In these cases the algorithm finds a subset of the possible explanations only. The overall size of the dependency graph does not influence the algorithm.

There are certain techniques of event processing that can not be addressed by this approach. For example rules like "If event A arrives then wait thirty seconds for event B. If B also arrives then ignore both, else forward A." are difficult to model using dependency graphs. However, these kinds of in–depth event interpretation are usually wanted for events from the same source. They are a workaround for noisy agents and sufficiently addressed by existing approaches. Here only a global time correlation strategy is performed, i.e. the event correlator tries to correlate all events within a given time window. The length of the time window is based upon the average period of an event burst observed in the past.

The focus of this approach is *overall event correlation* of events from networks, systems and applications. It best fits for heterogeneous management systems.

4 Setting up an Event Correlator

4.1 Developing a Dependency Graph

The main work when developing an event correlator using this approach is the modeling and implementation of an adequate dependency graph.

4.1.1 Algorithmic Representation

As mentioned before, there is a large amount of objects and dependencies in complex systems with frequent changes. Therefore the dependency graph is not stored in a database but represented in an algorithmic way with parts of it being generated at runtime. It is important to note that the dependency graph is a *conceptual data structure.* It is represented at runtime by two operations:

- The getObject–Operation returns an object for a name that was extracted from an event. This operation is invoked when mapping events to objects.
- The getAdjacent–Operation returns the dependent objects for a given object. This operation is invoked when walking through the dependency graph.

When implementing these operations special attention is required to ensure that several invocations of getObject requesting the same object return the same object. The implementor should consider first, that these operations should closely approximate the actual dependencies in the managed

system in terms of completeness and timely reflection of changes, and second, that the period of execution is critical.

4.1.2 Design Guide

How to design an adequate dependency graph? As a guide through the process of design or updating an existing design the following should be considered:

- The design is driven by the requirements of the fault management process. By defining the objects of the graph one defines the condensed events forwarded to the administrator and therefore defines the triggers that initiate further fault localization and fault repair. Usually requirements are continuously added or refined. Thus there will be a customization of the event correlator by refining the dependency graph, i.e. introducing new objects with finer semantics and therefore more specific dependencies.

- The implementation is aimed at generating as many objects and as many dependencies as possible from the management system using general applicable expert knowledge. So the design of the dependency graph will be heavily influenced by the management model(s) used in the existing management system.

What remains open is the adequate granularity of the dependency graph. On the one hand a fine granularity gives detailed fault messages, while a coarse granularity is easy to implement. To resolve this trade–off the following question has to be answered: What management actions are triggered by certain fault messages? If one has the strategy to reboot a system whenever the operating system behaves weird, it makes no sense to deliver fault messages telling which subsystem of the operating system has problems. If one wants to avoid rebooting and tries to diagnose the problem in more detail first, modeling the system up to the level of system processes is worth being considered.

In general, the correlation is better when there are few dependencies only. Therefore the number of dependencies per object should be kept low. In the first place, this is a guideline for the design of the managed system itself. For example, replacing shared Ethernet with switched Ethernet, making collision domains smaller or using a central file server instead of cross–mounting workstation's file systems, reduces the number of dependencies. In the second place, the designer must avoid an "everything–depends–on–everything–graph". For example: Consider a model for IP connectivity using objects of the kind "IP–Service on Host X". What are the dependent objects? Well, the connectivity service has a malfunction if connectivity with *any* reachable host fails. So "IP–Service on Host X" depends on all hosts of the managed system. A better model is found if using objects of the kind "IP–Connectivity between Host A and Host B". These objects depend on the participating hosts and some networking equipment.

There is some structure in the dependency graph that is not regarded by the event correlator itself but useful to know for the designer. On the application/service level there are dependencies between the components of a distributed application and dependencies between services as shown in figure 5. Dependencies at this level are sometimes cyclic. One can resolve them by further refinement but they do not disturb the event correlator, because it just

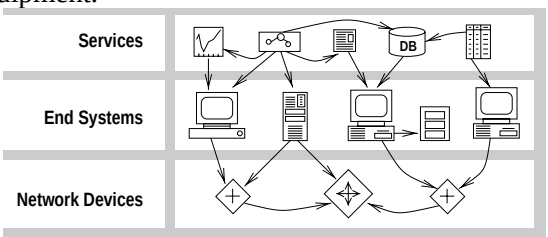

Figure 5: Structure of Dependencies

computes a transitive closure. Applications are instantiated on systems. Systems management is characterized by the layers of the operating system and by the hierarchy of hardware components. The transport service in addition depends on networking equipment. Dependencies at the network layer are given by the network topology.
Case studies in designing dependency graphs and gaining the necessary management data were made for IP connectivity and basic network services like a file service. They confirmed that dependency information at the network layer can be retrieved using well–known sources while service dependencies already mark or cross the edge line of what is possible with today's management systems. Main sources for dependency information are the network topology, system MIBs and inventory systems and application MIBs where available.

4.2 Deployment Methodology

To summarize the previous section the first task when setting up an event correlator is to design a dependency graph according to the requirements of fault management, i.e. the fault messages that are wanted, and according to the capabilities of the management system. The second step is to implement it by connecting the event correlator to the management system. This is the main part of the work for real–life management systems.

Note that up to now little attention was spent on the input events. This is the final step: All kinds of events that are seen by the management are mapped to the dependency graph now following a strategy of closest match. If this mapping turns out to be weird sometimes that says something about how the events that are currently forwarded meet the management requirements. Two things may happen: First, many event types are mapped to the same objects. This means that the events are too technical and too specific compared to the triggered management action. In these cases the approach helps defining generic events as a side effect. Second, there are some objects with no events mapping to them. Then a potential lack in the fault detection mechanisms is discovered. One should investigate the objects depending on the silent one to find out whether indirect events pointing to these objects may be sufficient or else consider an extension of the event stream possibly by additional management tools.

5 Prototypical Implementation

To evaluate the approach a prototype is under development that currently implements a simple but already real dependency graph from the scenario of the IP connectivity service. Its objects include hosts, IP interfaces and objects for the IP service. The implementation needs a list of hosts and MIB–II information to build up the dependency graph. The software components that retrieve the dependency graph are mobile agents ([Mou97], [MDR97]) and therefore implemented in Java. Due to the lack of adequate triggers the dependencies derived from MIB-II variables has to be updated by polling.

The architecture of the prototype is guided by a recent trend in management: It is embedded in a Corba–based management environment. The Corba Event Management Service as implemented by Visigenic is used to forward events between the components of the event correlator. Future versions may include the Corba Notification Service currently under standardization at OMG's TelecomDTF. Currently, Corba is being incorporated in our mobile agent infrastructure with special attention to the Corba Mobile Agents Facility.

The prototype processes events as depicted in figure 6:

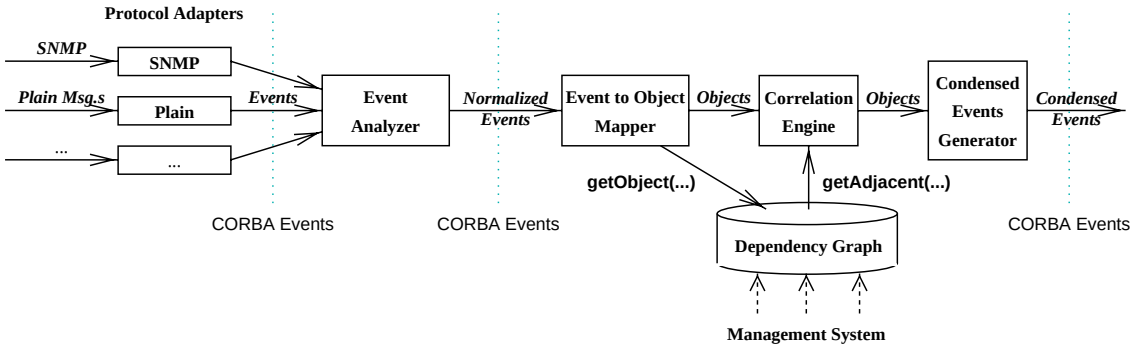

Figure 6: Architecture of the Prototype

1. Because modern management systems use various management protocols the first step is to map these protocols to a homogeneous representation, namely Corba events which are simply arbitrary IDL data structure. This is done by *protocol adapters*. SNMP traps are mapped to Corba events using the gatewaying algorithm specified by the Joint–Inter–Domain–Management group (JIDM). Another protocol adapter encapsulates a "plain" string messages in a Corba event.

2. The next step is to extract an identifier, called "normalized events" in the figure, from the Corba event that can be mapped to the dependency graph. Here the interpretation of an event as fault message takes place. This module, called *event analyzer* in the figure, requires event–specific customization that defines which object is indicated to be faulty by a particular type of events.

3. The next module, the *event–to–object–mapper*, is the first one that interacts with the dependency graph. It retrieves the corresponding objects for the received normalized events using the getObject–Operation. This is the place to implement a reordering strategy as mentioned in section 3.2.

4. The *correlation engine* is the core part of the event correlator. This module implements the algorithm outlined in section 3.2. It correlates the input objects by walking through the dependency graph using the getAdjacent–Operation searching for common dependent objects.

5. The last module in the event processing line, the *condensed event generator*, manages the output of the correlation engine. It generates an event containing the list of hits found by the correlation engine.

Ongoing work on the prototype includes the implementation of more complex scenarios especially from the area of systems and application management and extensions to deal with redundancy in management databases which is typical for heterogeneous management systems.

6 Conclusion and Outlook

Event correlation using dependency graphs is initiated by the idea to instrument an existing management system to gain the knowledge base of an event correlator. A dependency graph is con-

sidered to contain sufficient information for event correlation and to be simple enough to achieve automated generation to a great extent. When applying this approach the major part of customization is connecting the event correlator to provider–specific components of the management system by implementing corresponding parts of the dependency graph. The implementation of the dependency graph is driven by the requirements of fault management. The minor part is routing the management events through the event correlator and its dependency graph.

The strength of event correlators derived from this approach is to deal with dynamic, complex *managed* systems. They unconditionally require a working management system to be useful. One should not underestimate the efforts required to connect to real–life management systems. The approach is not designed to extend or improve existing system models. At most it can ease consistency checking between management databases. In a mid–term view, this approach profits from improvements in the area of management models much more than existing approaches do. A combination with case databases is advisable to profit from experience where available.

What makes the approach feasible in practice is the clear justification for putting efforts in implementing the dependency graph: First, it improves the management middle ware. The costs are shared between those management applications using it. Second, there is a direct feedback between dependency graph and day–to–day operations because the quality of the dependency graph determines the quality of fault messages.

Further work to be done includes the analysis of upcoming standards defining models for management information beyond Internet–SMI with respect to their suitability for gaining dependency graphs. Investigations include CIM and maybe WBEM, JMAPI and Corba services as well as telecommunications standards like IN.

Acknowledgements

The author wishes to thank the members of the Munich Network Management (MNM) Team for helpful discussions and valuable comments on previous versions of the paper. The MNM Team directed by Prof. Dr. Heinz-Gerd Hegering is a group of researchers of the University of Munich, the Munich University of Technology and the Leibniz Supercomputing Center of the Bavarian Academy of Sciences. Its web-server is `http://wwwmnmteam.informatik.uni-muenchen.de`.

References

[BBM+93] S. Brugnoni, G. Bruno, R. Manione, et al. An expert system for real time fault diagnosis of the italian telecommunications network. In Hegering and Yemini [HY93], pages 617–628.

[DLW93] R. H. Deng, A. A. Lazar, and W. Wang. A probalistic approach to fault diagnosis in linear lightwave networks. In Hegering and Yemini [HY93], pages 697–708.

[DR95] G. Dreo Rodosek. *A Framework for Supporting Fault Diagnosis in Integrated Network and Systems Management: Methodologies for the Correlation of Trouble Tickets and Access to Problem-Solving Expertise*. PhD thesis, Ludwig-Maximilians-Universität München, July 1995.

[DRK97] G. Dreo Rodosek and T. Kaiser. Determining the availability of distributed applications. In Lazar et al. [LSS97], pages 207–218.

[GH98] R. Gardner and D. Harle. Pattern discovery and specification translation for alarm correlation. In NOMS98 [NOM98], pages 713–722.

[HKH+96] K. Hätönen, M. Klemettinen, Mannila H., et al. Knowledge discovery from telecommunication network alarm databases. In *12th International Conference on Data Engineering (ICDE'96)*, pages 115–122, February 1996.

[HY93] H.-G. Hegering and Y. Yemini, editors. *Integrated Network Management III*, San Francisco, USA, April 1993. North–Holland.

[JW95] G. Jakobson and M. Weissman. Real–time telecommunication network management: extending event correlation with temporal constraints. In Sethi et al. [SRFV95], pages 290–301.

[KP97] S. Kaetker and M. Paterok. Fault Isolation and Event Correlation for Integrated Fault Management. In Lazar et al. [LSS97], pages 278–289.

[KS95] I. Katzela and M. Schwartz. Schemes for fault identification in communication networks. 3(6):753–764, December 1995.

[KYY+95] S. Kliger, S. Yemini, Y. Yemini, et al. A coding approach to event correlation. In Sethi et al. [SRFV95], pages 266–277.

[Lew93] L. Lewis. A case–based reasoning approach to the resolution of faults in communication networks. In Hegering and Yemini [HY93], pages 671–682.

[LSS97] A. Lazar, R. Saracco, and R. Stadler, editors. *Integrated Network Management V (IM'97)*, San Diego, USA, May 1997. Chapman & Hall.

[MDR97] M.-A. Mountzia and G. Dreo Rodosek. Using the Concept of Intelligent Agents in Fault Management of Distributed Services. *Journal of Network and Systems Management*, 1997.

[Mou97] M.-A. Mountzia. *Flexible Agents in Integrated Network and Systems Management*. PhD thesis, Technische Universität München, December 1997.

[MSS97] M. Mansouri-Samani and M. Sloman. An event service for open distributed systems. In J. Rolia, J. Slonim, and J. Botsford, editors, *International Conference on Open Distributed Processing and Distributed Platforms (ODP'97)*, number 14, pages 183–194, Toronto, Canada, May 1997. Chapman & Hall.

[NOM98] *Network Operations and Management Symposium (NOMS'98)*, New Orleans, USA, February 1998. IEEE.

[Nyg95] Y. A. Nygate. Event correlation using rule and object based techniques. In Sethi et al. [SRFV95], pages 278–289.

[OMK+97] D. Ohsie, A. Mayer, S. Kliger, et al. Event modeling with the model language. In Lazar et al. [LSS97], pages 625–637.

[SRFV95] A. S. Sethi, Y. Raynaud, and F. Faure-Vincent, editors. *Integrated Network Management IV (ISINM'95)*. Chapman & Hall, 1995.

[WBE+98] P. Wu, R. Bhatnagar, L. Epshtein, et al. Alarm correlation engine (ace). In NOMS98 [NOM98], pages 733–742.

[WTJ+97] H. Wietgrefe, K.-D. Tuchs, K. Jobmann, et al. Using neural networks for alarm correlation in cellular phone network. *International Workshop on Applications of Neural Networks in Telecommunications*, 1997.