15

# Design and Evaluation of a Distributed Event Recognition System

*Ingo Bürger, Bernd Meyer*
*RWTH Aachen, Computer Science Department,*
*Communication Systems, Ahornstr. 55, D-52056 Aachen, Germany*
*Telephone: +49/241/8021415, FAX: +49/241/8888220*
*e-mail: bernd@i4.informatik.rwth-aachen.de*

**Abstract**

Network and systems management environments, e.g. for telecommunications systems, have to deal with a huge amount of event notifications, i.e. hundreds of alarms per second. Conventional network management systems do not cope with this kind of workload due to their centralized nature. Therefore, distributed approaches are needed. We present a novel approach to event recognition, where not only event recording but also recognition of complex events is done in a distributed fashion. Often, exceptional system behavior cannot rely only on event notifications but has to become active to exactly determine the cause of an event. Therefore, a so-called active monitoring agent can either read additional data or trigger test operations. We describe the implementation of our distributed event recognition system on the ANSAware platform. The presentation of several measurement experiments in a MAN and WAN scenario will complete this work. Recognition time and buffer utilization is measured in different workload scenarios having deterministic arrivals, arrivals according to a Poisson Process and to a Markov-Modulated Poisson Process.

**Keywords**

Network and Systems Management, Monitoring, Event Recognition, Measurement, Performance Evaluation, Distributed Platforms,

## 1 INTRODUCTION

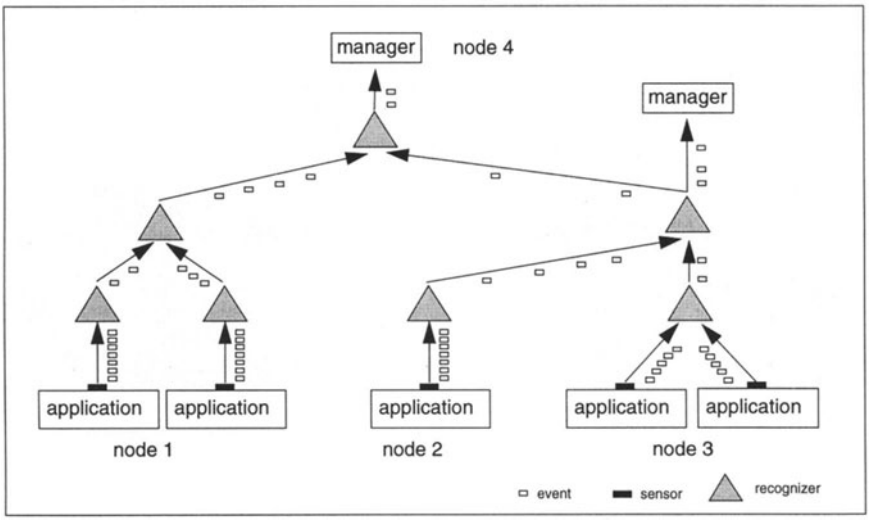
The on-going interconnection of isolated local networks to regional or even international intranets and internets enables applications to become distributed. Therefore, distributed platforms like the Common Object Request Broker Architecture (CORBA), ANSAware and the Distributed Computing Environment (DCE) are becoming increasingly popular because they free application programmers from certain aspects of distribution. Gaining distribution transpar-

J. Rolia et al. (eds.), *Open Distributed Processing and Distributed Platforms*

ency results in loosing control over the system's performance. Because state-of-the-art distributed platforms are lacking means for controlling application and platform performance, it is hard to run mission-critical distributed information systems with satisfactory performance.

Network and systems management tools are used to detect faulty system states or performance bottlenecks in a distributed system, see e.g. (Sloman, 1994) or (Hegering, 1995). Nevertheless, current management systems assume a human system operator to make decisions to improve system performance and availability. For managing a distributed platform, a reactive management system that detects bottlenecks or faults by itself and reconfigures it according to rule defined by the operator seems more promising.

On key element in network and systems management is monitoring. The process of monitoring can be further subdivided into event recording, recognition, analysis and presentation. Recording is done by sensors attached to a distributed system to be monitored forwarding event occurrences to the recognition system. The recognizer tries to match the incoming event stream with given event patterns. These patterns might rather be complex events requiring several different events to occur. The recognizer forwards a message to the analysis system once a certain pattern has been detected. Figure 1 shows a sample configuration of a distributed event recognition system. In contrast to recognition, analysis deals e.g. with gathering statistics on event occurrences. In addition, analysis is based on a model of the whole system under control whereas events generally only focus on parts of the system under control. Common distributed system models are e.g. graphs, Petri nets or process algebras. Presentation and graphical user interface allow system operators to control the inspected system. Several monitoring tools for parallel or distributed systems have been developed in the last years, e.g. ZM4 (Hofmann et al, 1994), JEWEL (Lange, 1992), Pablo (Reed et al, 1992) or DMS (Friedrich et al, 1996).




Chapter two introduces a new classification schema for event recognition systems used for comparing the most important event management systems. In the following chapter we present design and implementation of the distributed event recognition system (DERS) we have devel-

oped at RWTH Aachen. Besides defining an event definition notation we mainly on DERS configuration and distributed event recognition. DERS has been implemented on top of the ANSAware distributed platform. In chapter four we describe measurements we did based on this implementation. Measurements have been done for different workload scenarios and distributed systems of different geographical size including local, campus-wide and nation-wide scenario.

## 2 STATE OF THE ART

Event recognition systems recently gained much attention in the area of network and systems management, debugging tools and active database systems. They are part of advanced monitoring systems and consist of sensors attached to the application under control and one or more components analyzing the event stream coming from sensors, see also Figure 1. These recognizers are able to e.g. detect complex events or do event filtering. Finally, events are forwarded to managers that decide what to do next. Approaches developed can basically be distinguished by

- the kinds of events they are able to recognize, and
- the model they use for event recognition.

All recognition systems are processing basic events, i.e. events directly emitted by system sensors. Furthermore, event recognition systems allow

- the clustering of events using propositional logic operators, e.g. *and*, *or*, or *not*,
- the use operators specifying a causal ordering on events, e.g. *before*, or *after*,
- temporal or modal logic operators, e.g. *next*, *eventually* or *always*
- timed events, i.e. events occurring at a certain time or in a given time interval,
- event condensation, e.g. filtering or compression of events,

Often interval-based events are combined with a causal ordering operator, e.g. the occurrence of event *b* within *10 ms* after the occurrence of event *a*. Once an event has been specified, its description has to be transformed into an executable program that is able to recognize this event. These programs can be constructed on base of several (theoretical) models. Common models for the recognition of complex event structures are e.g.

- graphs,
- finite state automata,
- Petri Nets, or
- rule-based models.

In the sequel, we give a representative comparison of approaches to event recognition using the above defined classification, see Table 1. The GEM approach (Mansouri-Samani, 1995) offers a rich event model to be recognized. Eveent evaluation is performed by a graph model, that is

not realized in a distributed fashion since it requires incoming events to be totally ordered. The focus in the SEMS (Yemini et al, 1996) lies on efficient decoding of the event-cause relation. Therefore, they use elements from coding theory to reduce redundancy in event-correlation graphs and gain fault tolerance. This advantage has been gained by restricting the event model to clustering and system flexibility. Jordaan et al (Jordaan et al, 1995) use the information model of OSI Systems Management, namely the Guidelines for the Definition of Managed Object (GDMO) and the General Relationship Model (GRM). The resulting Management Information Base (MIB) is organized as a graph. This approach offers maximum flexibility, but only provides central analysis facilities. In contrast to the above approaches, IMPACT (Jakobson et al, 1995) is based on a rule-based approach to event recognition. It offers a rich event model, which is mapped onto the AR-TIM expert system. (Möller et al, 1995) also use a rule-based approach, but fall back onto a proprietary rule language. In essence, the main concept is an autonomous rule-based filter agent.

**Table 1: Comparing event recognition systems**

| *approach* | *event types* | *recognition model* | *special features* |
|---|---|---|---|
| GEM | clustering, timed events, causal ordering, time-triggered | graph (tree) | delay window |
| SEMS/DECS | clustering | graph ("codebook") | fast recognition |
| Jordaan et al | basic events, object relations | graph (MIB) | GDMO, GRM |
| IMPACT | clustering, causal ordering, time-based events | rule (expert system) | AR-TIM expert system |
| Möller et al | clustering, filtering | rule (proprietary) | autonomous filter agents |
| EBBA | clustering, filtering | graph (tree) | sharing of subtrees |
| Gehani et al | clustering, causal ordering, condensation | finite state automaton | |
| SAMOS | see Gehani et al | Petri Net | |

Whereas the above approaches originate from network and systems management, the EBBA system (Bates, 1989) has been developed for debugging distributed programs. Both next approaches have been developed for centralized active databases. They offer a rich event model but analysis is done in a centralized fashion. The only difference from our classification point of view is the recognition model. Whereas SAMOS (Gatziu et al, 1994) uses Petri Nets, Gehani et al (Gehani et al, 1992) falls back on finite state automata.

# 3 DESIGN AND IMPLEMENTATION

In this section we are going to describe the Distributed Event Recognition System (DERS) we have developed at RWTH Aachen. It enhances the ANSAmon distributed monitor that has been developed before, see (Meyer et al, 1995).

## 3.1 System components

The DERS consists of a configuration of system components like passive recognizes, active monitoring agents, buffers, and sensors. Sensors are attached to the distributed system or application to be monitored, see Figure 2. Sensors are used to record relevant application information and forward them to the monitoring system. They are implemented by inserting trace statement into the distributed systems' source code. If events are application-specific, instrumentation can be done either in the application source code or by creating an additional thread. Of course, memory addresses of the relevant data must be known to this thread. Non application-specific events can be inserted into the platform libraries or the platform run-time system. Since ANSAware sources are available, platform instrumentation is possible.

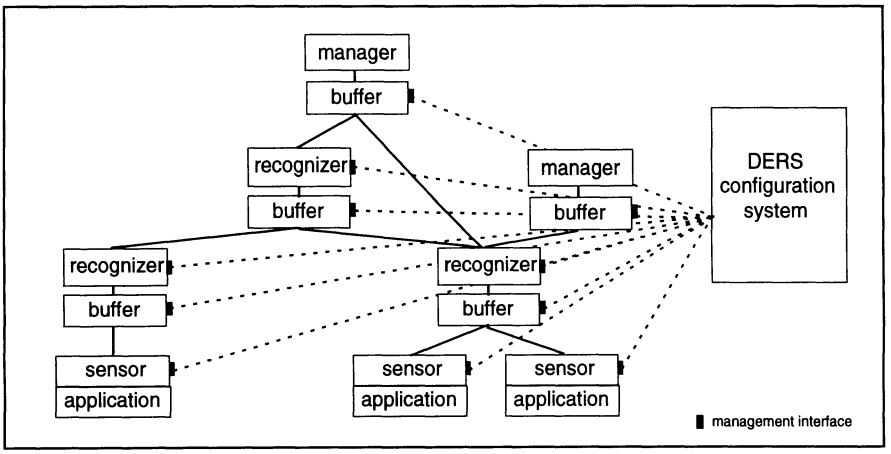



**Figure 2.** Sample configuration of a Distributed Event Recognition System (DERS)

Buffers can be inserted in order to decouple event-producing objects like sensors or recognizers and event-consuming objects like recognizers, agents or managers. Using no buffer would either block the producer until the consumer is ready to receive new events or would block the consumer until the producer is able to deliver new events. Therefore, we use an notification (or announcement in ODP terms) for forwarding events from the producer to the buffer. Event consumer can fetch events from a non-empty buffer by RPC (or interrogation in ODP terms). Because ANSAware announcements are lost when the receiver is busy, we have replaced it by an RPC, i.e. an ANSAware interrogation.

## 3.2 Event recognition

DERS expects each event to be described by its type, a timestamp set at occurrence time, identifiers for the application, the node, the capsule and the thread emitting this event. Finally, there may also be information (of fixed size) specific to a certain event type. A complex event extends a basic event by operands, i.e. links to other events, and an recognition interval, that will be explained later in this section. Every event expression defined by using our event definition notation (EDN), partially shown in Table 2, can be transformed into an equivalent tree representation named the event recognition tree (ERT). Defining a context-free grammar for the EDN enables automatic parser generation. The resulting tree contains events as leave nodes and operators as non-leave nodes. Each node (i.e. events and operators) representing a basic or complex event contains a flag indicating whether the associated event has already been detected or not. The values of all node flags determine the current state of an ERT. As soon as a recognition tree has been installed at a recognizer, incoming events are matched against the tree's leave nodes. Once a matching is detected, the state of the corresponding node becomes `true`. Afterwards, the complete tree will again be evaluated. If the complete event represented by the recognition tree has been detected, the tree's state has to be reset. We will discuss distributed event recognition later on.

**Table 2: DERS event operators**

| *event* | *meaning* | *event* | *meaning* |
|---|---|---|---|
| $E$ / $\neg E$ | event E occurs / does not occur | $E_1 \sqsubset E_2$ | event $E_1$ occurs before event $E_2$ |
| $E_1 \vee E_2$ | at least one event occurs | $E_1 \sqsupset E_2$ | event $E_1$occurs after event $E_2$ |
| $E_1 \wedge E_2$ | both events occur | $E + t$ ms | occurrence of event E is extended by $t$ ms |

*Remarks on event description semantics*
The interpretation of a negative event needs some clarification. We assume negation to be the non-occurrence within an certain time interval. This time interval is determined by the occurrence of the first and the last partial event of a complex event. As soon as the event occurs although it should not, the whole event becomes invalid and the corresponding recognition tree will be reset.

Whereas *and* and *or* operator semantics can be defined in a straightforward manner, recognizing causal orderings is more difficult. First of all, the event description does not determine, if the operator defines a total of a partial ordering, i.e. if all events are comparable or not. For the DERS we have decided to apply total ordering semantics. Another decision to take is, what to do with a second occurrence of an event that forms a part of a more complex event. One way would be to create a second incarnation of the same complex event and the alternative is to discard repetitions. We have decided to take the second alternative. The choice becomes clearer in the light of the interpretation causal operator expression. Therefore, we have chosen to apply the discarding-repetitions approach to the left side operand and the first occurrence on the right side operand. This makes sense, because evaluation of $a \sqsubset b$ does not change until $b$ occurs. We have found this interpretation to be the natural one in most cases.

*Distributed event recognition*

Everytime an event arrives at a recognizer, it checks, which subtree waits for the occurrence of an event of this type. It evaluates the complete subtree in case the occurred event matches. If this event occurrence completes the evaluation of the whole subtree, the recognizer forwards a notification to all objects that have registered to receive it. Figure 7 shows a sample event that has been distributed over three nodes. E.g. as soon as the recognizer on `nodes_2` has locally detected the event *c*, it forwards an event message to `node_1`. Once a subtree has been successfully evaluated, it has to be reset in order to avoid event occurrences to be part of a second evaluation of the same event tree. In addition, in case of a contradiction occurred while evaluating an event tree, see Figure 3, it also has to be reset.

Up to now, the recognition procedure has been straightforward. A problem occurs, when a causal relation between events on different nodes have to be detected. Because we decided to have a total ordering of events, we need to have a global system clock. We have developed an novel global time estimation algorithm that we will not discuss here, see (Bürger et al, 1997).

If the DERS is used in a larger distributed system containing communication links with different message delay or different link utilization, events happen to arrive in a different ordering as they have been send. As soon as the delayed event arrives, ordering can be established due to the global system time. But situations can occur where a delayed event cause an event not to be recognized although it has occurred, see Figure 3. The event $((a \wedge b) \sqsubset c)$ is to be recognized and all events occur on different nodes. Because event *b* arrives at the recognizer after event *c*, the recognition tree has already been reset when event *c* arrives.

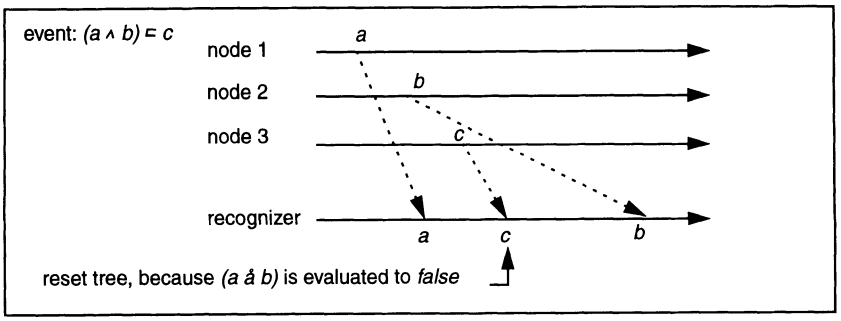



**Figure 3.** Faulty recognition caused by message delay of event *b*

To avoid these kinds of faulty recognitions, we delay decisions on the satisfaction of an event description until we are (almost) sure, that no delayed events will occur. The determination of the recognition delay puts up a conflict between having maximum certainty, i.e. making the interval long enough to detect delayed events and reducing recognition time of events aiming at a short delay. Therefore, a good trade-off has to be found. If one could determine average and standard deviation of the communication delay over a link, it would be possible to have tight recognition intervals, e.g. the recognition delay could be calculated by

recognition _delay = average + 2*standard_deviation. (1)

## 3.3 Active Monitoring Agents

In centralized systems, all state information is kept locally and event detection can be done without interactions with remote objects. This changes if event recognition is done in a distributed fashion. First, information is distributed all over the system and second, the communications subsystem causes additional failures. In conventional monitoring and management models, recognizers detect an event and forwards it to the manager, see Figure 4 a. The manager decides to retrieve additional information from a resource via RPC. All together, there are four interactions. Having an active monitoring agent, information can directly be retrieved at the resource and a precise event notification is sent to the manager, see Figure 4 b. Especially, if manager and monitoring system reside on physically distant computer nodes connected by a wide area network (WAN), WAN traffic is reduced from three to one event notification resulting in a better recognition time.

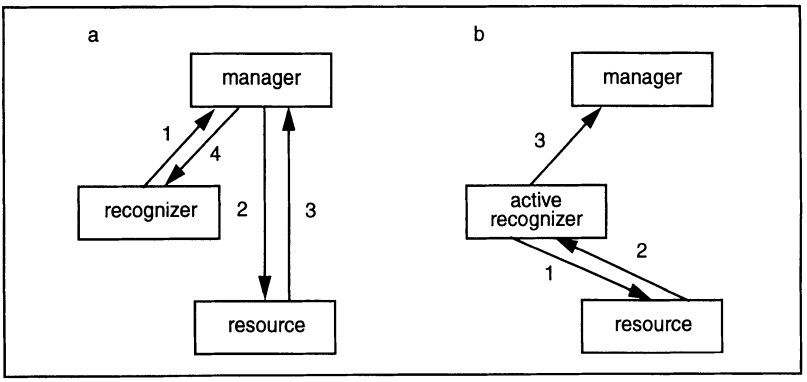



**Figure 4.** Monitor interactions in a (a) conventional management system and (b) using active monitoring agents

Active monitoring agents are either separate DERS components or are combined with passive recognizers. In the later case, active events are recognized by a separate component because recognition time depends on interaction durations with remote objects and would block the detection of other events. Therefore, active monitoring agents have to be able to

- receive event notifications from other DERS components,
- forward events notifications to other DERS components,
- read local and remote management attributes, and
- call remote operations.

As an example, a client process detects a binding error emitted by the middleware. This error can be caused by a breakdown of the called server, an RPC fault or a failure of the communication subsystem. Figure 5 describes this behavior as an active event rule. Once an active recognizer receives the `BindingError` event notification from the client, it calls the `ping` operation at the server node. This operation employs the Internet Control Message Protocol (ICMP) on the network layer of the Internet Protocol Suite. If the call fails (`on result`

`failure`, Figure 5), the agent infers a server node failure, an network laver communications fault or an communications device fault. The later two alternatives can be resolved, if a device driver level test tool is available, but server or communications device fault cannot be distinguished from the client node. In case of a successful ping operation, the failure must be caused on the transport layer or above it. To gain deeper insight, we assume a socket layer test operation (`socketTest`, Figure 5) to be available. If this operation fails, a transport layer fault has been detected, whereas otherwise an RPC failure is likely.

```
on event BindingError (server) from client
do
    call ping at server
    on result failure do
        forward event NodeFault+NetworkLayerFault+DeviceFault(client,server)
    on result success
    do
        call socketTest at server
        on result failure forward event TransportLayerFault(client,server)
        on result success forward event RPC_Fault(client,server)
```

**Figure 5.** Sample active event description for a binding error

## 3.4 The DERS configuration system

A management interface is required for every DERS component, see Figure 2, in order to enable dynamic reconfiguration of the DERS. First, every component offers operations to enable and disable itself and operations for users to register and deregister for an event, see Figure 6. In addition, sensors, recognizers or active monitoring agents offer operations to add and remove events, recognition trees resp. rules. The buffer is characterized by its maximum size, a current size limit, and the current buffer size. Whereas the limit can be modified at runtime, the maximum size is determined at creation time. Of course, the limit cannot be larger than the maximum size. In addition to configuring the installed DERS, each node should offer a factory enabling local and remote object creation. This allows the DERS configuration system to extend the current configuration.

Up to now, we have focused on describing complex but not distributed events. Therefore, we are extending our notation by the @ operator determining the node at which an event occurs. It can be applied to any event description, basic or complex ones, and is used as an expression postfix. If a new complex event is to be added to the DERS, its configuration system transforms the event description into a recognition tree, divides this tree into subtrees to be recognized by each node and forwards these subtrees to the corresponding recognizers. The event description is called static if all events have a location annotation. In order to achieve an optimal recognition system configuration it is necessary to decide where to place the event recognizers depending on the structure of a complex event and the current system configuration, e.g. the event message delay on communication links between two nodes. Therefore, a more dynamic event description that only determines the occurrence location basic events, i.e. events directly emitted by a sensor, is needed.

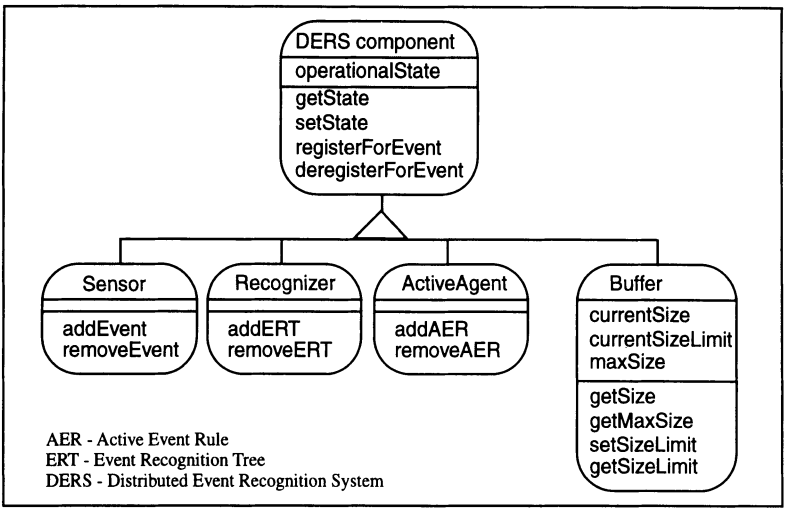



**Figure 6.** Information view on management interfaces of DERS components

In the sequel, we define rules for configuring the DERS with a complex events not taking performance and utilization information on nodes and links into account. Therefore, we need to compute the number of basic events for each node involved in the recognition of a complex event.

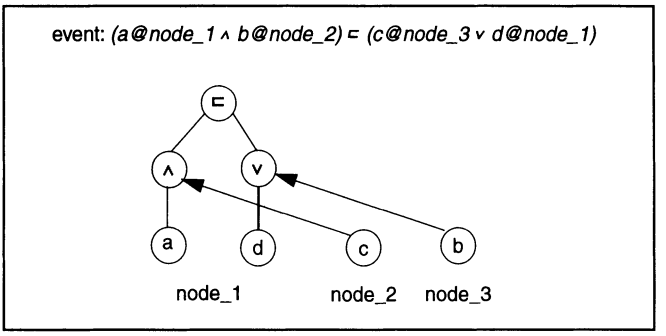



**Figure 7.** Sample distribute recognition tree

The DERS configuration is guided by the following rules,

- negation of an event should be recognized locally,
- *and* or *or* combined events should be recognized by the node that records most of the events of a composite event, and
- causal operators should be recognized at the node of the left operand.

The last rule derives itself from the operator's semantics using the last occurrence of the left side and the first occurrence of the right side. If the rule had been defined the other way, all events on the left side would have to be transferred although they would be discarded.

# 4 EVALUATION

## 4.1 Measurement scenario and workload characterization

Performance measurements of the DERS have been done in three geographical scenarios, a LAN-based scenario between `veilchen` and `primel`, a campus-size MAN scenario between `primel` and `lyra` at RWTH Aachen clinic (about 5 km distance) and a nation-size WAN scenario between `primel` and `tunix` at FernUniversität at Hagen (about 200 km distance), see Figure 8.

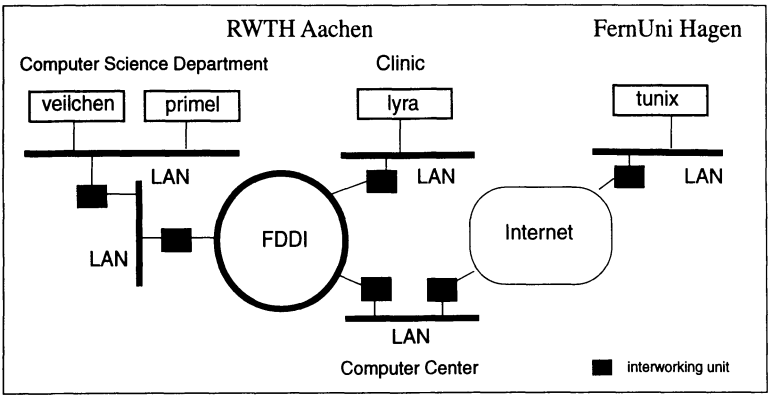



**Figure 8.** Measurement scenario

We used three different kinds of workload in order to investigate DERS' response behavior in more detail. The first scenario, called deterministic workload, assumes events to arrive with constant inter-arrival times. Time-triggered monitoring can modeled with this kind of workload. Nevertheless, constant interarrival time are not suitable for event-triggered monitoring, so we used Poisson workload in order to model independent event occurrences. Poisson workload means, that inter-arrival times are exponentially distributed. Even the independence assumption between event occurrences made for Poisson workload does not hold in realistic scenarios because it does not take into account, that the occurrence of one event in general will cause other events to occur, e.g. a communication link failure will cause every message transported over it to produce a fault indication. More realistic workload models should have bursty and non-bursty event occurrences. We have chosen Markov-modulated Poisson process (MMPP) to model this kind of behavior. MMPP models switch non-deterministically between different Poisson process with different rates using a Markov model. Which Poisson process (i.e. state) to switch to next only depends on the current Poisson process (i.e. state) and the transition probabilities between the different Poisson processes (i.e. states).

## 4.2 Recognition delay measurement for different workload scenarios

The most important performance metric for the integration of a event recognition system into a reactive management tool is its recognition delay. It determines responsiveness to event occurrences. We have measured the recognition delay, i.e. the time it takes for an event to get from the sensor to the manager, for all three workload scenarios in a MAN environment. Similar measurements have been done for LAN and WAN scenarios as well, see (Bürger, 1996). Figure 9 shows the average recognition delay of measurement series in the MAN scenario under deterministic and Poisson workload. A 95% confidence interval is given for all values.

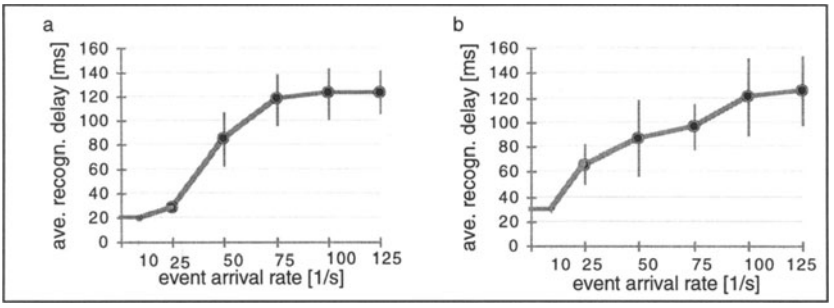



**Figure 9.** Average recognition delay (95% confidence intervals) in MAN scenario under (a) deterministic and (b) Poisson workload

A comparison of deterministic and Poisson workload shows, that DERS performs faster with Poisson workload than with deterministic workload when arrival rate increases over 50 events per second. Nevertheless, confidence intervals are smaller for deterministic load.

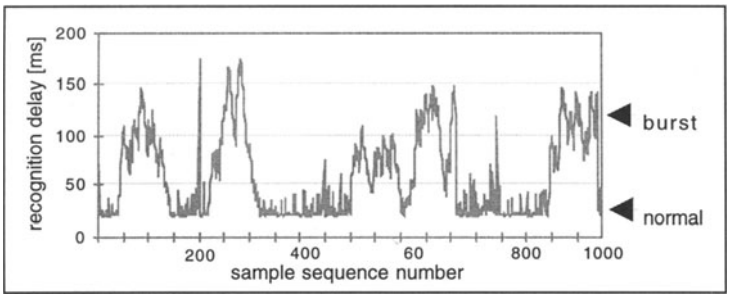



**Figure 10.** Recognition delay samples in MAN scenario for MMPP workload

Doing the same for MMPP workload would be pointless since confidence intervals are huge because the recognition delay switches between bursty and non-bursty intervals. In Figure 10 one notices, that DERS is fast enough in recognizing events, so that a burst with a rate of 125 events per second only has little influence on the succeeding interval with normal arrival rate, i.e. 10 events a second.

### 4.3 Buffer utilization for different distributed system size

Another important DERS performance metric is the buffer utilization. It is especially important for the DERS management system for finding a suitable buffer size. We have chosen a buffer size of 32 event records for the following measurements. Measurements presented in Figure 11 and Figure 12 show buffer utilization for a MAN and a WAN-based environment. Whereas a buffer size of 16 is sufficient for MAN (and of course for LAN) scenarios and arrival rates up to 125 events per second, events are lost for a arrival rate over 70 events per second in a WAN scenario.

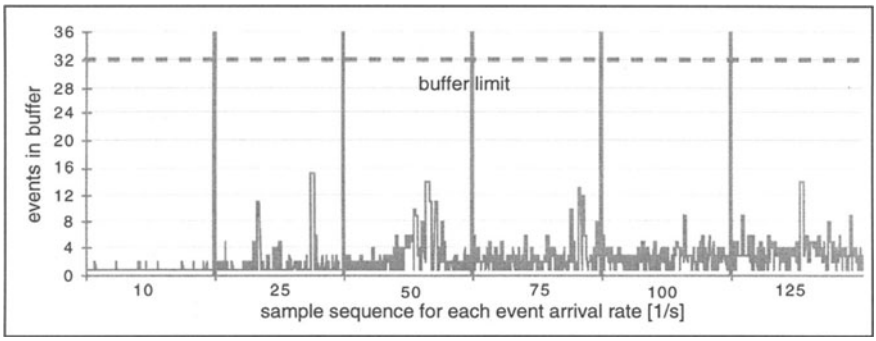



**Figure 11.** Buffer utilisation in campus-size MAN scenario

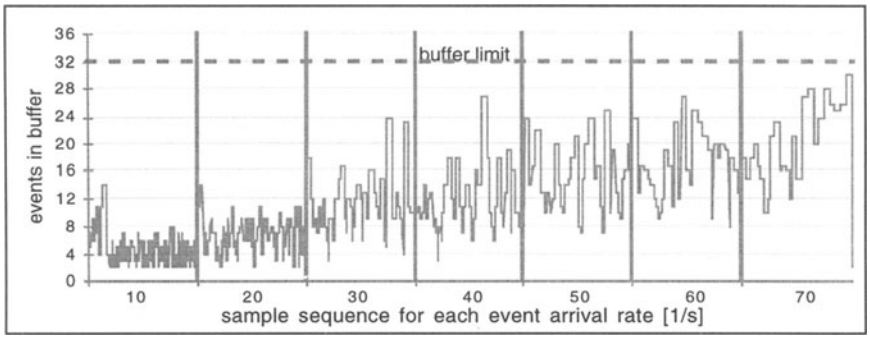



**Figure 12.** Buffer utilisation in nation-size WAN scenario

## 5 CONCLUSIONS

Event recognition is the most important part of monitoring systems. In this paper, we have presented a novel approach to event recognition in distributed system that also performs recognition and not only event recording in a distributed fashion. In addition, we proposed to use

active monitoring agents in order to reduce network traffic, that is especially important in WAN-based environments. Adding location annotations to events allows us to automatically configure a our recognition system for the detection of a certain complex event. We have presented measurements performed using three different workload scenarios, i.e. deterministic, Poisson and Markov-modulated Poisson workload, in a campus-size MAN scenario. We showed, that DERS is well suited for LAN and MAN-sized environments, where a buffer size of 16 are sufficient up to arrival rates of 125 events per second. Naturally, in a WAN scenario it only copes with lower rates, but in a WAN scenario we expect our active monitoring agent concept to be powerful enough to reduce WAN-based traffic to achieve reasonable recognition times. DERS has been implemented on ANSAware and is currently adapted to Orbix 2.0 MT and Orbix Talk. Future work will be done on an automatic instrumentation tool for DERS and its integration into a reactive management system prototype.

## 6 ACKNOWLEDGMENTS

This work is funded by the Deutsche Forschungsgemeinschaft under grant no. Sp 230/8-2.

## 7 REFERENCES

Bates, P. (1989) *Debugging Heterogeneous Distributed Systems Using Event-Based Models of Behavior. SIGPLAN Notices*, **24**,1, 11-22.

Bürger, I. (1996) *Development of active monitoring agents in consideration of clock synchronization in distributed system.*(in German), Diploma Thesis, RWTH Aachen.

Bürger, I.; Fasbender, A.; Meyer, B. and Rulands, I. (1997) *Distributed Event Recognition in Packet-Switched Networks.* (in German), accepted for: GI/ITG Conference on Communications in Distributed Systems, Braunschweig 1997.

Friedrich, R. and Rolia, J. (1996) *Performance evaluation of a distributed application performance monitor*, in Distributed Platforms (eds. Schill, A., Mittasch, C.; Spaniol, O. and Popien, C.), Chapman & Hall, pp. 259-71.

Gatziu, S. and Dittrich, K. (1994) *Detecting Composite Events in Active Database Systems using Petri Nets.* Proceedings of the 4th International Workshop on Research Issues in Data Engineering, pp. 2-9.

Gehani, N.; Jagadish, H. and Shmueli, O. (1992) *Composite Event Specification in Active Databases: Model & Implementation.* Proceedings of International Conference on Very Large Databases (VLDB'92), pp. 327-38.

Hegering, H. and Abeck, S. (1995) *Integrated Network and Systems Management.* Addison Wesley.

Hofmann, R.; Klar, R.; Mohr, B. et al (1994): Distributed Performance Monitoring: Methods, Tool and Applications. *IEEE Transactions on Parallel and Distributed Systems*, **5**, 6, pp. 585-98.

Jakobson, G. and Weissmann, M. (1995) *Real-Time Telecommunication Network Management: Extending Event Correlation with Temporal Constraints.* in Integrated Network Management IV (eds. Sethi, A.; Raynaud, Y. and Faure-Vincent, F.), Chapman & Hall, pp. 290-301.

Jordaan, J. and Paterok, M. (1993) *Event Correlation in Heterogeneous Networks Using the OSI Management Framework.* in Integrated Network Management III (eds. Hegering, H.; Yemini, Y.), North-Holland, pp. 683-95.

Lange, F.; Kröger, R. and Gergeleit, M. (1992) *JEWEL: Design and Implementation of a Distributed Measurement System.* IEEE Transactions on Parallel and Distributed Systems, **3**, 6, pp. 657-71.

Mansouri-Samani, M. and Sloman, M. (1995) *GEM: A Generalized Event Monitoring Language for Distributed Systems.* Imperial College Research Report No. DoC 95/8.

Meyer, B.; Heineken, M. and Popien, C. (1995) *Performance Analysis of Distributed Applications using ANSAmon.* in Open Distributed Processing - Experiences with distributed environments (eds. Raymond, K. and Armstrong, L.), Chapman & Hall, pp. 309-20.

Möller, M.; Tretter, S. and Fink, B. (1995) *Intelligent Filtering in Network Management Systems.* in Integrated Network Management IV (eds. Sethi, A.; Raynaud, Y. and Faure-Vincent, F.), Chapman & Hall, pp. 304-15.

Reed, D.; Aydt, R.; Madhyastha, T.; Noe, R.; Shields, K. and Schwartz, B. (1992) *An Overview of the Pablo Performance Analysis Environment,* Technical Report, University of Illinois, Urbana, Pablo Research Group.

Sloman, M. (ed.) (1994) *Network and Distributed System Management.* Addison Wesley.

Yemini, S.; Kliger, S.; Mozes, E.; Yemini, Y. and Ohsie, D. (1996) *High Speed and Robust Event Correlation.* IEEE Communications Magazine, **34**, 5, pp. 82-90.