Evaluation of two server prototypes for AXE O&M and the Alarm Management System

by Magnus Lewin

000302

Abstract

A communication network of today consists of many different platforms and techniques. New platforms and techniques are rapidly invented. Sometimes new platforms are integrated with older platforms. The evolution puts new requirements on the systems which are responsible for supervising the communication networks. In the telecom market these systems are referred to as telecom management systems or Operation and Maintenance systems. A node in the communication network is called a Network Element. Since a communication network may and in most cases does consist of different types of Network Elements, the management system must be able to supervise all of them. Preferably the Network Elements are looked at, as they were part of the one and same uniform system. The telecom management systems of today for example interact with both AXE exchanges and Radio base stations.

The first requirement this puts on the Network Element is that the interfaces used for O&M should be open. Open means that the interfaces use common protocols.

At Ericsson Utveckling AB two server prototypes have been developed that were meant to work as access points for Operation & Maintenance applications interacting with the AXE exchange. The first prototype named EMA (Element Management Access) exposes a Socket interface and the other called EMS (Element Management Server) exposes a Corba IDL interface.

My thesis project is mainly about evaluating the performance of EMA and EMS. First I studied the architecture and functionality of the servers. I also briefly got into what Operation and Maintenance of the AXE exchange is about. I studied Sockets and Corba. I developed a program in Java which I used for evaluating the performance of EMA and EMS. From the graphical user interface of the program it is possible to start any number of concurrent clients running at the same computer. Each client sends an optional number of requests in sequence to EMA or EMS. The performance of the servers is investigated by measuring the response time of each request and by registering unsuccessful requests.

The results of the experiments I performed showed that the response time of EMS was much lesser than the response time of EMA when they served just one client. When I increased the number of clients, EMS's response time rapidly increased while the response time of EMA almost was constant. The EMA become more and more unstable as the numbers of simultaneously requests grew. For example requests were lost.

The second task of the thesis project was to design a management system. The Alarm Management System which I have designed can be used by operators managing alarms from several Network Elements. **The system is not implemented**.

The Alarm Management System uses CORBA for communication and distribution. I have studied the Alarm IRP which is a specification that should be applied when a system for managing alarms is developed at Ericsson.

In the design of the system I have considered issues like availability and scalability.

The new Operation and Maintenance platform of the AXE exchange called APG40 will have an Alarm Server implemented in CORBA. My design of the Alarm Management System discusses how the Alarm Server could be implemented and connected to a system that manages alarms from any kind and any number of Network Elements.

I haven't got into any implementation details. The design covers the system logic and the IDL interfaces of the components involved in the system.

Table of contents

1 Introduction

During the spring of 1999 UAB/I/M (The department where I am doing my thesis project at) and LMF/T/F in cooperation started the project called Element Management Access (EMA). The purpose of the project was to prepare the AXE exchange for the next generation of telecom management system. The EMA was intended to be a gateway through which Operation and Maintenance applications would be able to interact with the AXE exchanges.

Important benefits with EMA were that it should provide enhanced MML functionality towards the AXE exchange, hide the low-level communication between the O&M applications and the exchanges and hide the type and version of the AXE exchanges from the O&M applications. In this way it should be possible to build general O&M applications managing any AXE.

The outcome of the EMA project was two server prototypes, the EMA prototype and the EMS prototype.

The EMA prototype supports MML communication and parsing of printouts. It has a graphical user interface for configuration and exposes a Socket interface for communication.

The EMS prototype only supports MML communication. It exposes a Corba interface for communication. The EMS is configured by editing a register file. The register file is loaded into the Windows NT registry.

The first task of the thesis project is to evaluate the Corba server prototype and the Socket server prototype in aspects of performance and functionality.

The second task is to design (not implement) a telecom management application towards the AXE exchange.

To be able to evaluate the servers mentioned above I briefly have got into what Operation and Maintenance of AXE is about. The functionality and architecture of EMA and EMS are found in their implementation proposals. I have studied Sockets and Corba that are used for communication between clients and the EMA server and the EMS server. Then I have developed a test program that can start several concurrent clients interacting with EMA and EMS. The test program measures the response time of the servers. The response time varies depending on:

- The number of clients that concurrently interact with the server.
- If the printouts are parsed or not.
- How many channels that are available between EMA or EMS and the Network Element.
- If the clients communicates with the same or different Network Elements.
- What MML command that is sent to the Network Element.

The telecom management application that I have designed is an alarm management system. The alarm management system let operators coordinate their alarm management towards multiple network elements. The system uses Corba for communication and distribution. The knowledge of managing alarms I have received from alarm managing in AXE, the alarm IRPs and Alarm Tool that is an application included in the network element management program WinFiol and Tools.

This report is structured as follows.

Section 2 briefly presents an overview of operation and maintenance of AXE and the architecture and functionality of EMA and EMS.

Section 3 discusses the Client/Server concept and the communication techniques Socket and Corba.

Section 4 presents how the EMA and EMS are evaluated and the results of that.

Section 5 describes the functionality of the Alarm Management System. Other issues that are related to the system as distribution, failure handling, reliability and scalability are also discussed in this section.

Section 6 Conclusions.

In the end of the report there are a glossary and a reference list.

2 AXE, EMA and EMS studies

2.1 AXE

The functionality in AXE can briefly be divided in two parts. One part includes the control system and the other part includes the traffic signaling system. These systems are in turn divided into several subsystems. When working with Operation & Maintenance against the AXE it is the control system that is mainly involved. The control system for example exposes an interface for downloading software to the exchange and an MML (Man Machine Language) interface. The MML interface is used to retrieve information about the status of the exchange and change the status if desirable. There are about 2000 different MML commands for these purposes. An MML command is sent in form of a text buffer and the response called a printout is also pure text. The text in the printout is structured in a way that it easily can be printed out and read by an operator. The MML language for different versions of AXE may distinguish a bit. New MML commands may have been added and printouts may have been changed.

The Operation part of O&M stands for detection, localization and correction of faults that appears in the system. The maintenance part is among other things about connecting and disconnecting subscribers, software and hardware updates, collecting charging information, collecting network statistics and traffic measurement data.

Telecom management systems communicate with the AXE through its I/O-system. Today existing I/O-systems are IOG11, IOG 20 and APG30. IOG11 and IOG20 run on operating systems developed by Ericsson. APG30 runs on a UNIX machine. A new I/O-system APG40 is under development. The APG40 runs on cluster architecture on the Windows NT platform. The I/O-systems supports different communication protocols like TCP/IP, RS 232(Serial port) and X.25. The I/O-system communicates with the different sub systems in AXE via the internal bus of AXE. There is today an ongoing research about using TCP/IP in the internal communication system of AXE.

2.2 EMA

2.2.1 Overview

The EMA was meant to be a server platform, which O&M applications should be built upon. It acts as a gateway for clients communicating with AXE exchanges. Several clients can connect to EMA and communicate with the exchanges reachable from that EMA server. This is demonstrated in Figure1.

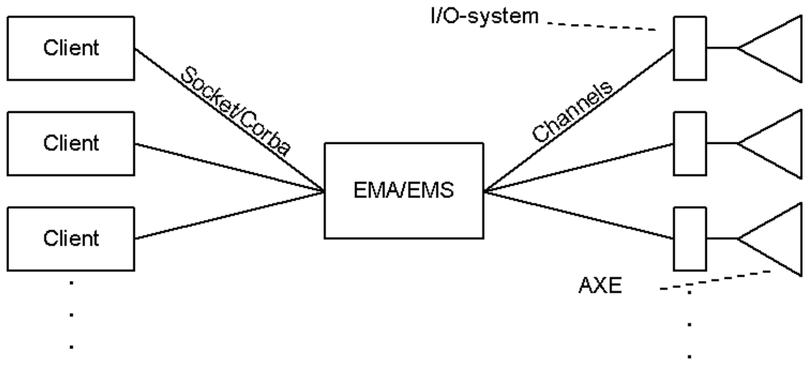


Figure1 EMA or EMS acting as a gateway between O&M clients and AXE.

The client uses Socket when it communicates with the EMA server and Corba when it communicates with the EMS server. The EMS server is handled in section 2.3. Both the

EMA and EMS establish channels for forwarding the client's requests to the Network Element, which consists of the I/O-system and the AXE exchange itself. The channel concept is explained in section 2.2.2.

EMA is programmed in C++ and aimed for the Windows NT platform. A reason for that the EMA project was closed was the idea of putting more functionality closer to the exchanges. This is what is happening in the APG 40, the future I/O-system of AXE. The APG 40 is not just an I/O-system it also includes a lot of business logic that before was managed somewhere else in the network. The APG 40 is a step towards distributed computing in the Operation and Maintenance of AXE. The EMA concept is also brought into APG 40 but there it is called MMLROS. The major difference with the MMLROS server is that it just manages one exchange not multiple as the EMA server does. This means that if one MMLROS server is not available then all other exchanges is still available through their MMLROS servers. The single point of failure that the EMA server could be is removed.

EMA exposes its services through protocol adaptations. The only protocol adaptation implemented in the prototype is the Socket adaptation described in section 2.2.4. The services supported are MML communication, parsing of printouts and file transfer. The file transfer service will not be investigated in this report. For more information about the file transfer API see the file transfer API implementation proposal [4].

2.2.2 Configuration

In EMA a **node** represents a physical AXE exchange on the communication network. A node is configured by associating the node with one or more **channels** and the **node type**. The configuration is done via a graphical user interface and stored in a relational database. The EMA prototype uses MS Access as database.

A channel represents the communication link between the EMA and the exchange. There can be multiple channels associated with one exchange. The channel is configured by specifying the **channel properties**. The channel properties are destination address, communication protocol and **target type**. The target type is the type of the exchange's I/O-system.

The node type in some way reflects the hardware and software the exchange consists of. When a new version of the AXE exchange is developed the hardware and software are changed and improved. New functionality may impact on the Operation and Maintenance management of the AXE. Specific functionality may also be added to improve the O&M management itself. Together with the release of a new control system or a new traffic signaling system in AXE a new ALEX book is delivered. The ALEX book contains among other things, documents and files that are related to Operation & Maintenance. There are for example operational instruction documents, command descriptions, printout descriptions and application information.

The syntax and structure of an MML command is stored in a PCM file. The same information related to a printout is stored in a PPM file. Every node type configured in EMA has its own file directory containing the PCM and PPM -files.

2.2.3 Architecture

EMA consists of the components showed in Figure2 in the end of this section.

The protocol adaptations are the external interfaces of EMA. The socket adaptation explained in section 2.2.4 is the only protocol adaptation that is implemented in the EMA prototype.

The FORM parser, MML & Event (M&E) and Naming & Configuration (N&C) components are part of the Element Management Resource Kit (EMRK) described in EM Resource Kit 1.1 [3]. M&E and N&C are implemented as DLL components and the FORM parser is a C++ class library in the C/C++ API of EMRK. EMRK also includes communication protocol drivers. EMRK is reused in EMA. The AXE element management program WinFiol and Tools uses EMRK for MML communication.

The N&C component exposes interfaces for creating and deleting channels.

The M&E component provides a high level interface to MML commands and printouts.

The FORM parser is used to parse printouts. The parsing process in EMA is explained in section 2.2.5.

The Resource Manager that is not part of EMRK is responsible for finding a free channel to the requested exchange and handling the sending and receiving of the command. It also serves as an interface to the configuration database.

To illustrate how the components in EMA cooperate the following scenario is arranged. EMA receives a request for MML communication. The request includes an MML command and the name of the node that the MML command will be sent to.

1. The request is received by the protocol adaptation
2. The parameters are forwarded to the Resource Manager that picks a free channel belonging to the requested node.
3. The channel identifier and command is passed to the MML & Event interface that opens the channel.
4. The command is sent and the printout is received.
5. The channel is closed and the channel identifier and printout are returned to the Resource Manager.
6. The Resource Manager returns the printout to the socket adaptation that sends the reply to the client which initiated the request.

This section is based on the EMA CORE implementation proposal [5].

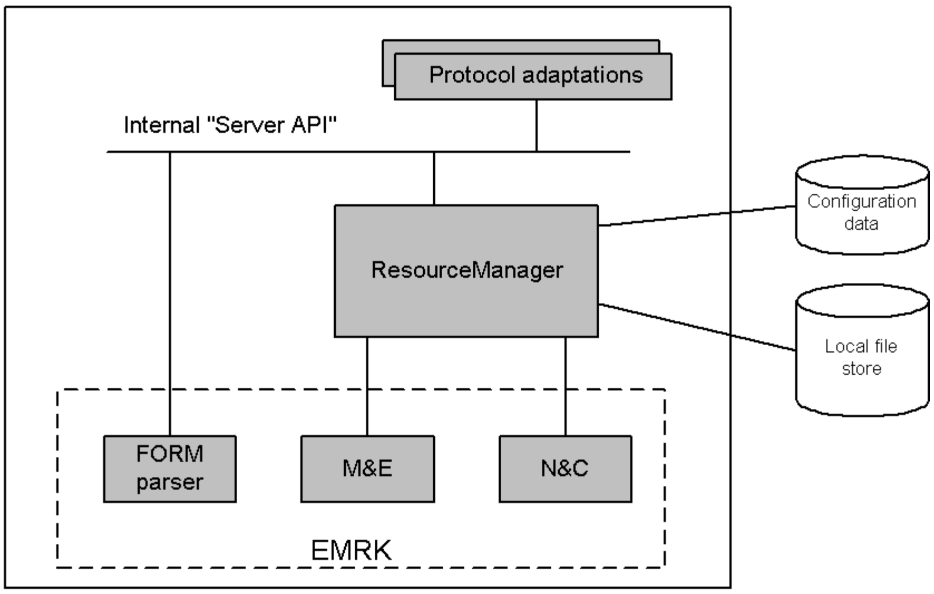


Figure2 EMA Architecture

2.2.4 *Socket adaptation*

Clients communicate with the EMA server via the Socket adaptation. The interface is text based, synchronous and stateless. A stateless protocol retains no memory of past requests. Every time the client sends a request a new connection is established with the EMA server. Clients connect to EMA using the IP address of the machine where the

EMA is located and the port number where the socket adaptation is listening for connections. The socket adaptation is multi-threaded which means that it can handle multiple requests from different clients concurrently. The well known protocol HTTP, used by web browsers request web pages from web servers behaves in the same way as the Socket adaptation in EMA. In section 3.1 there is a further discussion about stateless and concurrent servers.

The request sent to the socket adaptation is an ASCII text string. New line characters "\n" separate the parameters in the request. The end of the request is marked with the null character "\0". The parameters in the request are command, arguments, switches and switch arguments.

The valid commands are **send, list** and **file**. Send stands for MML communication with the requested node. List is used for listing the nodes that are reachable from that EMA. The file command is used for retrieving UPC file paths to PCM or PPM files.

An argument is additional information required for a specific command. If the send command is issued it is followed by an MML-command.

A switch item is additional information that sometimes is optional and sometimes not, depending on the command. A switch item always has the form "/X" where the x describes how the switch should be interpreted by the server.

A switch argument is additional information required for a specific switch. A switch argument always has the form "/xargument", where "/x" is the switch and "argument" is the switch argument.

An example of a request could be "send\ncaclp\n/nAXE1\n\0". This will result in that the EMA sends the MML-command "caclp" to the exchanged represented by the node-name AXE1. Here "send" is the command, "caclp" is the additional argument to the command "send", "/n" is the switch and AXE1 the additional switch argument.

The reply the client receives from EMA begins with an error code. If the request succeeded this value is "0000" and the result follows the error code. If the error code is not equal to "0000" an error text follows it. The result is structured in a way that corresponds to the request.

For more information about the socket adaptation see [6].

2.2.5 *Parsing process*

A printout may be very long and include a lot of information. The idea with parsing is that specific data can be selected from the printout. This help operators and applications to retrieve the information needed just for their purpose. The major advantage is for applications that shall take any specific action dependent of some specific information included in the printout. If the printout is parsed the result is structured and the application can easy access it. Using the parsing process in EMA will also decrease the amount of data sent on the network. This might be almost insignificant since the printout just includes text.

A request for MML communication where the client specifies what information of the printout it is interested in may look like this: "send\ncaclp\n/p date time\n/nAXE1\n\0". EMA manages the request in the following way.

1. The Resource Manager forwards the MML command to AXE1 and receives the whole printout.
2. The parse-switch "/p" informs EMA that it shall parse the printout with the parse-arguments "date" and "time".
3. The Resource Manager fetches the PPM file from the local file store. The PPM file belonging to the MML command "caclp" is located in a directory associated with the node-type of AXE1.

4. The FORM Parser is provided with the printout, the PPM file and the parse argument. The result is returned to the Socket adaptation.
5. The Socket adaptation sends the result to the client.
6. The text string received by the client looks like this: “0000\n990301\t10543\n\r\0”. First the error code 0000 is received which means that the request worked out successfully. Then the two parse-parameters date and time follows.

2.3 EMS

EMS is a prototype that from the beginning should include the same services as EMA was intended to do. It is developed by LMF/T/F at Ericsson in Finland. The only service implemented in the prototype is MML communication. File transfer and parsing of printouts is not supported.

2.3.1 Architecture

EMS is built upon EMRK in a similar way as EMA, section 2.2.3. It does not use the FORM parser since the parsing service is not available.

EMS requires EMRK 1.2 because this version of EMRK has additional server support that is lacking in EMRK 1.1.

The Resource Manager in the EMS prototype uses Windows NT registry for storing available nodes and channel files.

2.3.2 Corba adaptation

The Corba adaptation is implemented with Visibroker for C++ 3.3. It exposes three objects EmrkFactory, EmrkChannel and EmrkPrintout. The IDL interface for each object is presented in Appendix A.

A client sending an MML command to an exchange via the Corba adaptation of EMS:
1. The client connects to the EmrkFactory object
2. Invokes the method OpenChannel and receives an EmrkChannel reference.
3. Invokes the method SendCommand on the EmrkChannel object and receives an EmrkPrintout reference
4. Invokes the method GetAllLines on the EmrkPrintout object and receives a string sequence consisting all lines in the printout.
5. Invokes the method Release on the EmrkPrintout
6. Releases the EmrkChannel object by calling the method Release.

The client can use the same channel reference to send several requests in a row. This is not possible in the EMA socket adaptation where every request must be preceded by a new connection to EMA. The Corba adaptation therefor is a state-full protocol. The client can reserve system resources, which in this case are channels. If the client fails or if it for another reason never releases the channel, the channel will be locked and can’t be used by any other client. To prevent this a timeout can be used that allows the client to reserve the channel for just a predefined amount of time. This is not implemented in the EMS prototype.

If the client releases the channel, the channel is put into a channel pool managed by the Resource Manager where it stays for some predefined time. New clients requesting channels to the same exchange can reuse the channel. This is more efficient because the channel does not have to be opened and closed every time it is requested. The channel pool is not implemented in the EMA prototype discussed in section 2.2.

More information about the EMS server is find in [7] and [8].

3 Client/Server, Socket and Corba studies

3.1 Client/Server

3.1.1 Client Server Model

The client server model is used to describe the communication between two processes. The client requests services that are provided by the server. An example of a well-known client server system is the World Wide Web. Web clients like Netscape Navigator and Internet Explorer connect to Web servers and request for HTML pages to be downloaded and viewed in the web browser.

In the example above one side is the client and the other side is the server. This is not a must. In any interaction between two processes one is the client and the other is the server, but the same client and server may play the opposite role in another interaction.

3.1.2 Concurrent and non Concurrent Servers

There are concurrent and non-concurrent servers. Concurrent servers can handle several simultaneous requests from different clients. Every time the server receives a request a new thread or process is started to handle it. Non concurrent servers just manage one request at the time. The benefits with concurrent servers are:

- If a lot of requests arrive from different clients simultaneously, they all have to wait for the reply about the same time. The last one that arrives does not have to wait much longer than the first.
- Servers doing a lot of I/O for example reading from the disk or communicating with other servers can during that time process other requests. In this way the CPU is utilized more efficiently.
- If the server runs on a multi-processor system the different threads or processes can run on different processors.

Introducing a thread-pool can speed up a concurrent server. Instead of releasing the thread when the connection is closed the thread is inserted to a pool. Another client connecting to the server later reuses the thread. The time a thread is hold in the pool can be predefined. In this way the number of threads will increase and decrease dynamically and follow the present load on the server.

3.1.3 Stateful and Stateless Servers

As mentioned before a stateless server retains no memory of past requests. The client establishes a new connection every time it sends a new request. The big advantage with stateless servers is that clients can’t reserve system resources (CPU and memory - utilization) at the server no longer than it takes to handle one request. This suits client/server implementations over the Internet, which may have a lot of clients requesting the same services simultaneously. A disadvantage with stateless servers is that every request takes more time to perform, because a new connection must be established every time. It is also hard to implement more sophisticated servers without introducing states.

3.1.4 Failure handling

The thing that makes remote procedure calls different from local procedure calls is the possibilities of failures. First we have controlled failures that appears when a client sends an invalid request or when the server performs an invalid operation. The server informs the client of the failure by raising an exception.

Failures that are harder to deal with is when the client or server suddenly crashes. A stateless server can simply be rebooted while a stateful server must have some mechanism that makes the client and server agree on the state of the system. The mechanism to use depends on the logic and functionality of the system. States can be preserved in the system by storing them in persistent memory. Another problem that may appear is that the client or server crashes before the reply could be delivered to the client. Then the client does not know if the request was performed or not. If the request is idempotent meaning that it can be executed any number of times with same effect, the client can retransmit the request. If the server is not idempotent the client can attach a sequence number to every request. The processed requests are cached in the server and if the client sends duplicate requests the server will notice it. If the server crashes the cash of former requests will be lost. Then the server has to recover in some way when it reboots.

3.2 Sockets

3.2.1 Overview

The Socket Interface is an Application Programmers Interface (API) that allows a program to gain access to all the services provided by the communication protocols that the Socket implementation supports. Microsoft’s Socket implementation winsock2 today supports TCP/IP, UDP/IP ATM, IPX/SPX, and DECnet. The protocols may coexist within one application. Winsock2 is originally derived from Berkeley-Sockets which have became some sort of standard in the area. More information about winsock2 is find in [14].

The Berkeley-Socket API comprises eight system calls that are common to all Berkeley based Unix operating systems (SCO, Linux). Other Unix operating systems such as SunOS based on SVr4 support the socket interface as well. The Socket API is explained in section 3.2.3.

The name socket refers to a communication endpoint. A pair of sockets uniquely identifies a connection between two applications or processes located at the same host or at different hosts. If TCP/IP is used as communication protocol then the sockets pair is [(IP-address host A, TCP port application B), (IP-address host C, TCP port application D)].

For a client and server to be able to communicate with each other they must talk the same language. The language consists of a set of requests and replies that are predefined and forms the application protocol. An example is the get and put commands in FTP. Get is used to download files from an FTP server. Put is used for transferring files from an FTP client to an FTP server.

3.2.2 TCP/IP versus UDP/IP

Communication protocols are usually split into reliable or unreliable protocols. TCP/IP is a reliable protocol and UDP/IP is unreliable. TCP/IP is also a connection-oriented protocol meaning that a connection between the applications must be established before the exchange of data can begin. In UDP/IP the messages is sent without establishing a connection. Upon TCP/IP it is just possible to build synchronous application protocol while applications using UDP/IP can communicate asynchronous or synchronous.

The reliable properties of TCP/IP ensure that every packet is delivered non-faulty, in correct order and not duplicated to the application. TCP/IP also provide flow control. Both sides of the TCP/IP connection have a finite amount of buffer space. If the sender process is faster than the receiving process the buffer space of the receiving process will be full within some time. To prevent this a mechanism called sliding windows is used that helps the receiving process to slow down the sending process. A detailed explanation about the sliding window mechanism is find in Data and Computer Communication, William Stallings [13].

The UDP/IP protocol does only ensures that delivered packages are non-faulty. If they are lost or delivered in non-correct order the application have to deal with this. TCP/IP suits well for applications that don’t have great real-time requirements and where it is more important that the data is delivered correctly. The protocol for transferring files on the Internet File FTP is built upon TCP/IP. An application transferring video or audio signals in real time should not use TCP/IP because retransmission of lost or faulty -data is meaningless. The TCP/IP protocol suit is described in TCP/IP Illustrated Volume1 [12].

3.2.3 The Socket API

Socket:
Creates the socket which is associated with the protocol family for example AF_INET if IP is used, protocol type for example stream (TCP) or datagram (UDP) and the specific protocol to be used.

Bind:
Associates the socket with a local IP address and a port number.

Listen:
Is used by servers only which is put in a passive state listening for client connections on the specified port number.

Accept:
Accepts incoming client connections.

Connect:
Only used by clients to initiate a connection between the client socket and the server socket.

Read and Write:
A write call adds data to the socket’s outgoing queue. Lower level functions decide when enough data is added and form a packet that is sent across the net. The read call removes data from the incoming queue.

Close:
The close call breaks the connection and releases all local system resources allocated within the socket.

0 shows how a socket client and a multithreaded socket server are set up and the communication between them.

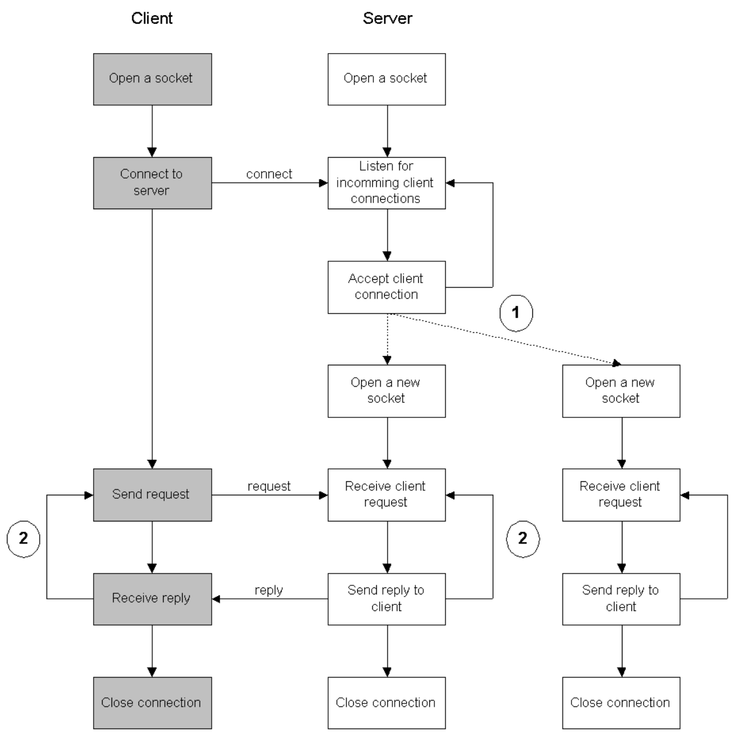


Figure3 Connection oriented (TCP/IP) communication between a socket client and a socket server

3.3 CORBA

3.3.1 Overview

CORBA makes it possible to develop applications consisting of objects implemented in different languages and that are located at different platforms, operating systems and computer networks without that the programmer has to concern about these issues.

The Client/Server model mentioned in section 3.1.1 can also be applied to CORBA. A client requests services from an object implementation. The services an object provides is the same as the operations it exposes through its IDL interface (Interface Definition Language). The communication between the client and the object implementation is explained in 3.3.2.

The OMG (Object Management Group) which is a consortium of computing-involved companies leads the evolution of CORBA. OMG does only produce specifications. The specifications are freely available for any company to implement. The activities performed by OMG are connected to the architectural framework OMA (Object Management Architecture) which was introduced 1990 by OMG. OMA consists of the following four main components.

1. Object Request Broker, which is responsible for the communication between clients and object implementations. This includes finding the object and invoking the requested operation with its parameters and returning the result.

2. Object Services, which provide services that are used by many distributed object programs. Examples of Object Services are:

The *Life Cycle Service* defines operations for creating, copying, moving and deleting objects.

The *Naming Service* allows clients to locate objects by name.

The *Event Service* allows clients and objects to dynamically register or unregister their interest in specific events.

The *Transaction Services* provides two-phase commit coordination among recoverable objects using either flat or nested transactions.

3. Common Facilities provide services that are needed by many end-user applications. They are divided into two categories:

Horizontal Common Facilities, which are shared by many or most systems.
There are four major sets of these facilities: User Interface, Information Management, Systems Management and Task Management. For example it could be services for mail and document exchange.

Vertical Market Facilities, which support the domain-specific tasks which are associated with vertical market segments. Examples of markets could be Telecommunication and Finance.

4. Application objects are not standardized by OMG. They can be referred to as objects, which may use all of the components mentioned above.

Details about OMA and CORBA are find at OMG's web site www.omg.org.

3.3.2 Communication at a high level

Before an object is accessible for any client it must be announced on the ORB or the object bus which the ORB also is called. The component responsible for that is the **Object Adapter**. The Object Adapter also assists the ORB with delivering requests to the object and with activating the object.

Before the client can access an object it first must receive the object's **object reference**. The object reference is received from either the Naming Service, another object or from a shared file. The object reference is not visible to the client. The client application just sees the **IDL interface** of the object.

The IDL interface could be known to the client in compile time and is then converted to a **stub routine** called the **Static Invocation Interface** (SII). The stub routine is the connection between the client and the ORB. When the client invokes a specific operation on the object, the object reference and information about the operation and its parameters is delivered to the orb via the stub. The ORB then forwards the request to the object which can be located locally or remotely. The object receives the request through its **skeleton routine** which like the client's stub is constructed at compile time. When the request is processed the result is returned to the client.

It is also possible for a client to access an object which type is not known at compile time. The client then uses the **Dynamic Invocation Interface** (DII).

How client and objects communicate at high levels and the lower levels is explained in the CORBA specification [2].

4 Evaluation of EMA and EMS

4.1 Introduction

The functionality of EMA and EMS are explained in section 2.2 and 2.3.

How is performance of a server measured? When we talk about a server computer the performance probably is measured in the number of instructions it can perform per second which also is called throughput. In this Master thesis I got the task to evaluate the performance of a server program. The performance can be measured by investigating some properties of a server program. First I would like to divide the properties in two groups. The first group includes the properties that impact on the performance of the server. The second group includes the properties that impact on the surroundings of the server. A property may impact on both the performance and the surroundings and therefore it could be included in both groups. The splitting in two groups is not necessary for my evaluation process of the servers. But it could perhaps help if the server should be adjusted. Adjusting one property of the server may impact on another property in the same or the other group. The adjustments that is made for making a server faster perhaps also lead to that the server uses more memory and more CPU than earlier. Perhaps is the server not allowed to use more than a specific amount of the system resources?

Example of properties that could be investigated:

Performance related properties

- Throughput, the number of requests a server can process per second. The throughput will be different for different kinds of requests. In a multithreaded server running on a single processor system simultaneous requests are processed concurrently. If a request leads to some kind of I/O management in the server like communication with another system, the free CPU time is used to process other requests. From an observer’s perspective it looks like some of the requests is processed in parallel. When the server processes enough requests simultaneously all the free CPU time associated with I/O management will be used. The server has reached its saturation level and the number of requests it processes per seconds will never exceed this limit.
- Response time, the time it takes to process a request. The response time is different for different kind of requests. For a multithreaded server the response time also depends on how many requests the server processes simultaneously. When the server reaches its saturation level (It uses all the CPU time it is capable of) the response time will increase linearly with the number of requests that the server processes simultaneously. This is illustrated in Figure4.

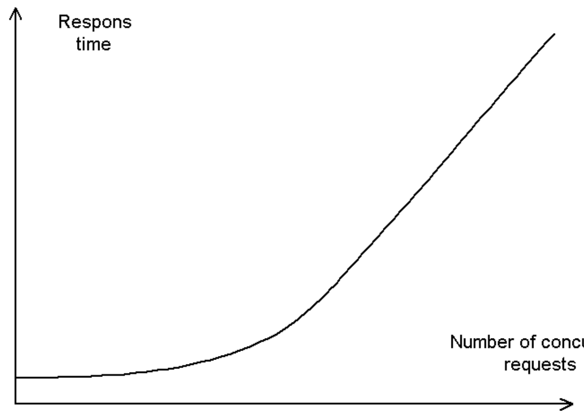


Figure4 The response time as a function of the number of simultaneous requests a server processes.

- The correctness of the server. Does the server lose requests? Are the requests successfully or not successfully processed.
- CPU usage. Used CPU time during a time interval divided by the same time interval. The CPU usage depends on how many other processes that are running on the same system. It is also possible to adjust the CPU usage among the processes by using priority scheduling.
- Memory usage. Is the amount of used memory almost constant or varies it a lot? It is possible to speed up the server by optimizing the memory management in the server program. What is the maximum and minimum amount of used memory during a server execution? Is there any memory leakage?
- Thread management. How changes the number of threads dependent on for example the number of simultaneous requests and/or time.

Properties that impact on the surroundings

- Memory usage. See performance related properties. The system resources are shared among all the programs that run on the same system.
- CPU usage. See performance related properties. The system resources are shared among all the programs that run on the same system.
- Traffic load on the communication network. The program logic and what communication technique that is used will probably impact on the traffic load.

Now I have discussed how the performance of a server can be measured by investigating some properties of the server. The second question is how is each property investigated? I have not done any kind of research in this area because my evaluation model presented in next section mostly is founded on response time measurements. It was also never required that I should investigate other properties. Probably there are several commercial programs which can be used to investigate Memory usage, CPU usage and thread management in a program. There are also programs for investigating the traffic on a communication network

4.2 Evaluation model

In my evaluation model I look at EMA and EMS as they were black boxes. I use my evaluation program (presented in next section) to investigate:

- The response time.
- The number of lost requests.
- The number of requests that were not successfully processed.
- The time it takes to perform an experiment.

In the experiments against EMA and EMS I don’t simulate any kind of user model. Each client involved in an experiment sends their next request as soon as it has received the reply from it last request. This types of experiments is called stress tests.

When I discuss the results of the experiments performed against EMA and EMS it in most cases are speculations. To really find the reason behind the results, the source code of each server must be investigated. The conclusions or guesses I make are based on the implementation proposals of EMA and EMS and the outcome of my experiments. Unfortunately I haven’t had the possibility to discuss the behavior of EMA and EMS with someone who was involved in the implementation of them.

4.3 Evaluation program

The program is developed in Java. In fact it is two programs which almost are similar, one for evaluating EMA and the other for evaluating EMS. The difference between the programs is the part which communicates with the server. The program for evaluating EMA of course uses Socket for communication and the program for evaluating EMS uses CORBA.

The graphical user interface of the program is used for setting up experiments against each server. In one experiment it is possible to run any number of clients requesting services from either EMA or EMS. All the clients run on the same machine and are simulated with threads.

If the response time of the server is much larger than the time slot each thread runs when it is scheduled plus the time it takes to propagate the request from the client to the server and the reply from the server to the client the requests almost are processed simultaneously at the server.

The response time measured by the evaluation program when it runs several clients will sometimes not be the true response time. Sometimes is the reply from the server returned when another client (Thread) is scheduled. The measured response time will be the real response time plus the time the client is blocked.

To produce real simultaneous requests the clients must run on different computers. This would be easy to do on a Unix system but it is much harder to achieve on a windows system of today.

The configuration of each client is about:

- Setting the number of requests it will send. The client will send the next request as soon as it has received the reply associated with the last request.
- Configuring the request. Inserting the MML command that will be sent to the server. Parse parameters may be added to the request. Only EMA supports parsing of printouts. The parsing process in EMA is explained in section 2.2.5.
- Deciding what node in EMA or EMS the client will interact with.
- Setting the timeout which decides when the client should give up trying to receive the response that is associated with the last request.

The results of an experiment are presented in one or more result windows.
Clients sending the same request can have a shared result window while clients sending different requests must have different result windows.

In the result window the following information is presented.

- Average response time, response-time(request1) + response-time(request2) + …+ response-time(requestN) / N.
- Max response time, max(response-time(r1), response-time(r2),..,response-time(rN)).
- Min response time, max(response-time(r1), response-time(r2),..,response-time(rN)).
- Number of successful requests.
- Number of non-successful requests.
- The response if the request carried out successfully. This one is optional.
- An error message if the request did not carry out successfully. This one is optional.

Since EMA and EMS are multithreaded servers I also investigated how they performed when several clients simultaneously send requests to them. In these Experiments the response time of a single request is not enough to make a statement of how the servers perform. Therefore I also look at the time it takes to carry out the experiment. I explain this by an example. If a single client sends 5 requests and the response time of each request is 1 second, then the time it takes to perform the experiment is approximately 5 * 1 seconds = 5 seconds. Then I let 5 clients send 1 request each. If the server can process each request in parallel the response time of each request is 1 second and the time it takes to perform the experiment is 1 second.

EMA and EMS run on a single processor systems and can't process any requests in parallel. But since EMA and EMS performs I/O operations, for example communicating with other systems it look like some of the requests are processed in parallel. At some level when enough requests are processed concurrently there is no more free CPU time to use. Then the response time of the server will increase linearly with the number of requests it simultaneously processes. This is illustrated in Figure4.

4.4 Evaluation platform

The evaluation program runs on a computer from Dell, 166 MHz Pentium processor and 128 Mbytes memory.

The EMA and EMS servers run on a computer from Dell, 450 MHz Pentium II processor and 128 Mbytes memory.

The AXE uses an APZ212/25 and the I/O-system is IOG20.

The computer network is a 10 Base T Ethernet

4.5 Results from experiments

All the experiments presented in this section are based on 5 detached experiments.

In the experiments where several clients communicate with the same node in EMA or EMS I have configured the node with the same number of channels (Telnet connections to the Network Element) as there are clients communicating with the node. This is illustrated in Figure5.

Unfortunately there was just one AXE exchange available in the laboratory when I performed my experiments. Therefore I couldn't perform any experiments where several clients simultaneously interacted with different nodes in EMA or EMS.

I did one experiment against EMA and EMS where I configured 5 "different" nodes that were connected to the same Network Element. The results of these experiments showed to be almost the same as the results from the experiments when the clients interacted with the same node. Figure5 and Figure6 show the difference in the configuration of the servers.

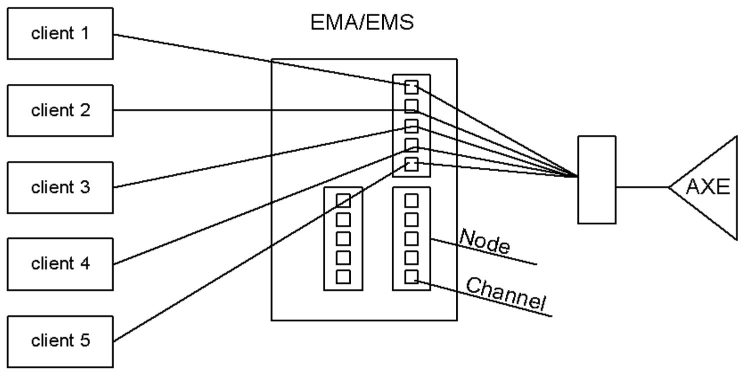


Figure5 5 clients interacting with the same node in EMA or EMS

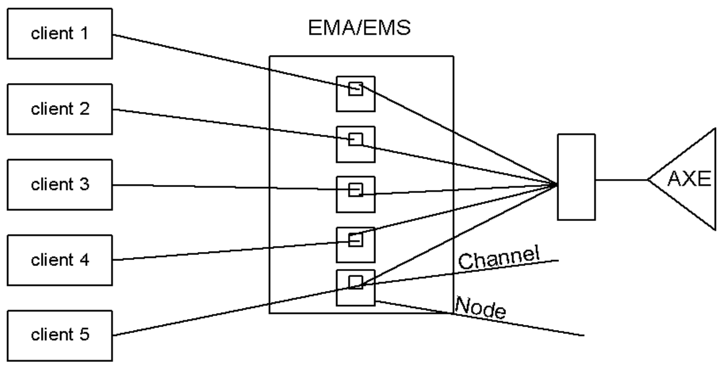


Figure6 5 clients interacting with "different" nodes in EMA or EMS

4.5.1 Experiments against EMA

1 client sends 10 requests to 1 node in EMA.

Experiment	Request	Average response time [ms]
1	caclp	2633
2	allip	2966
3	ioifp	3281
4	syfdp	2500
5	alexp	2389

Average response time (E1 – E5): 2754 ms
Average execution time: 28 s
Number of unsuccessful requests: 0
CPU usage: 0 – 30 percents

6	alacp	2420
7	alacp /p alcat	2529
8	caclp	2593
9	caclp /p date	3282
10	caclp /p date time	3302

5 clients send 10 requests each to the same node in EMA.
Experiment 11

	Request	Average response time [ms]
Client1	caclp	2601
Client2	allip	3019
Client3	ioifp	3266
Client4	syfdp	2454
Client5	alexp	2446

Average response time: 2757 ms
Execution time: 49 s
Number of unsuccessful requests: 1.6
CPU usage: 30 – 70 percents

4.5.2 Experiments against EMS

1 client sends 10 requests to 1 node in EMS.

Experiment	Request	Average response time [ms]
12	caclp	652
13	allip	577
14	ioifp	1163
15	syfdp	621
16	alexp	497

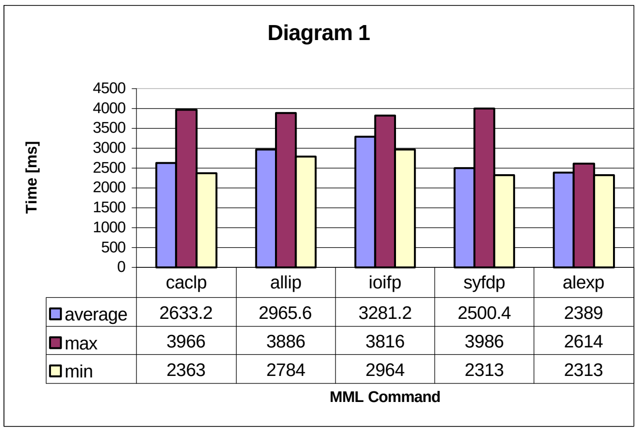
Average response time E12 – E16: 702 ms
Average execution time: 7 s
Number of unsuccessful requests: 0
CPU usage: 2 percents

5 clients send 10 requests each to the same node in EMS.
Experiment 17

	Request	Average response time [ms]
Client1	caclp	2403
Client2	allip	2918
Client3	ioifp	2716
Client4	syfdp	2380
Client5	alexp	2185

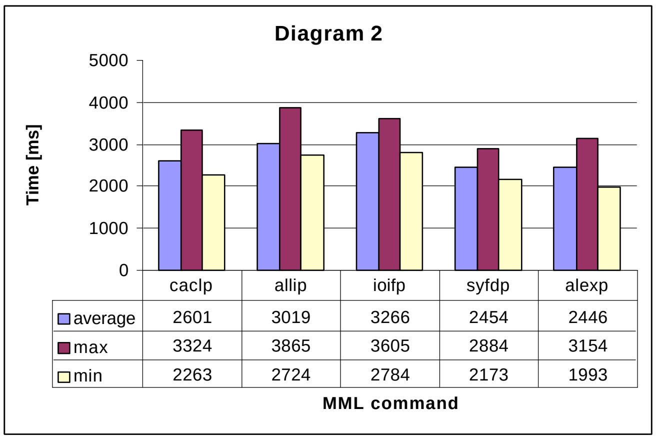
Average response time: 2520 ms
Execution time: 38 s
Number of unsuccessful requests: 0
CPU usage: 2 percents

4.6 Diagrams



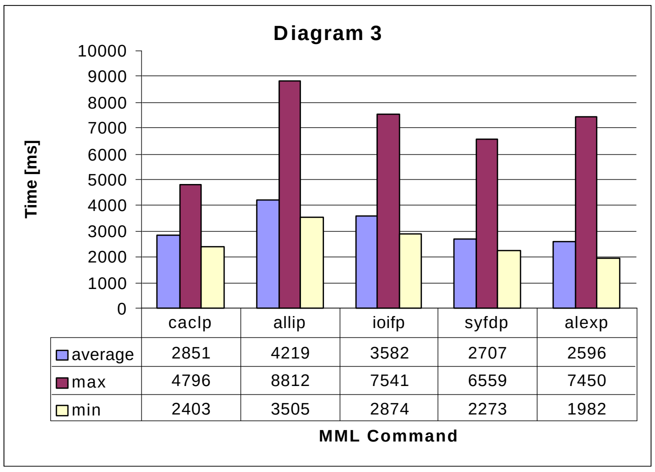
	caclp	allip	ioifp	syfdp	alexp
🟦 average	2633.2	2965.6	3281.2	2500.4	2389
🟥 max	3966	3886	3816	3986	2614
🟨 min	2363	2784	2964	2313	2313

1 client sends 10 requests to 1 node in EMA (Experiment 1 – 5).



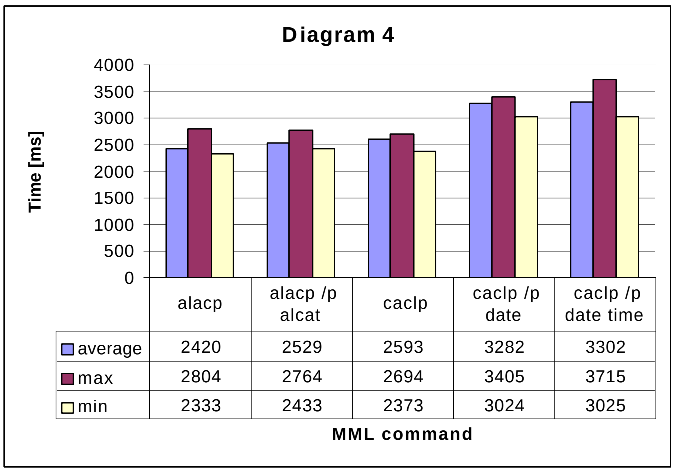
	caclp	allip	ioifp	syfdp	alexp
🟦 average	2601	3019	3266	2454	2446
🟥 max	3324	3865	3605	2884	3154
🟨 min	2263	2724	2784	2173	1993

5 clients send 10 requests each to the same node in EMA. (Experiment 11)



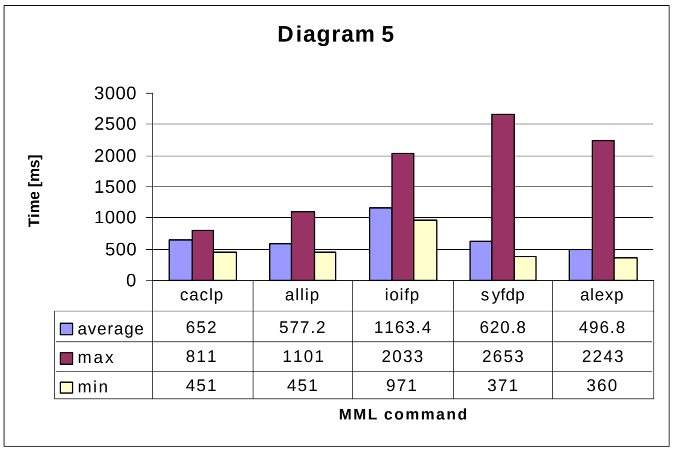
	caclp	allip	ioifp	syfdp	alexp
🟦 average	2851	4219	3582	2707	2596
🟥 max	4796	8812	7541	6559	7450
🟨 min	2403	3505	2874	2273	1982

5 clients send 10 requests each to “different” nodes in EMA



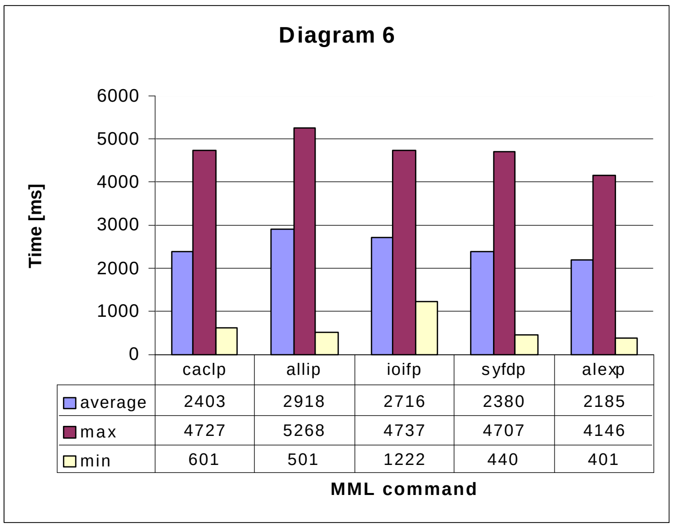
	alacp	alacp /p alcat	caclp	caclp /p date	caclp /p date time
🟦 average	2420	2529	2593	3282	3302
🟥 max	2804	2764	2694	3405	3715
🟨 min	2333	2433	2373	3024	3025

1 client sends 10 requests to 1 node in EMA. (Experiment 6 – 10)



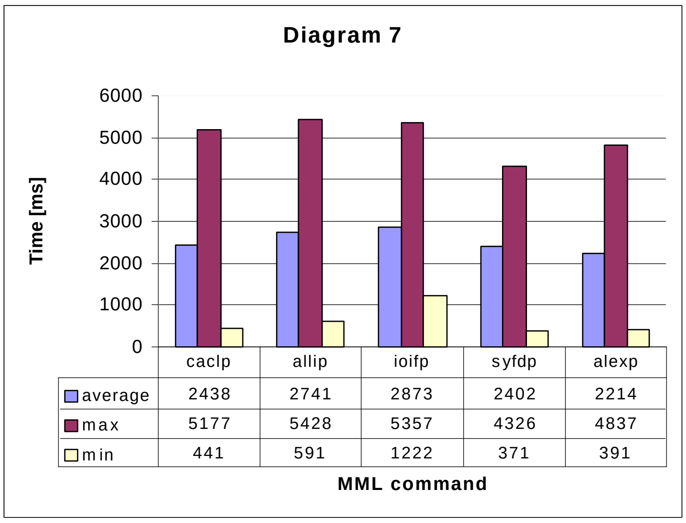
	caclp	allip	ioifp	syfdp	alexp
🟦 average	652	577.2	1163.4	620.8	496.8
🟥 max	811	1101	2033	2653	2243
🟨 min	451	451	971	371	360

1 client sends 10 requests to 1 node in EMS. (Experiment 12-16)



	caclp	allip	ioifp	syfdp	alexp
🟦 average	2403	2918	2716	2380	2185
🟥 max	4727	5268	4737	4707	4146
🟨 min	601	501	1222	440	401

5 clients send 10 requests each to the same node in EMS. (Experiment 17)



	caclp	allip	ioifp	syfdp	alexp
🟦 average	2438	2741	2873	2402	2214
🟥 max	5177	5428	5357	4326	4837
🟨 min	441	591	1222	371	391

5 clients send 10 requests each to “different” nodes in EMS.

4.7 Comments of results

1 client (EMA) versus 1 client (EMS)
The response time of EMA is about 4 times greater than the response time of EMS when 1 client runs against them. What could this depend on?

- Clients running against EMA must establish a new TCP/IP connection with EMA every time it sends a request. The ORB (Visibroker) probably keeps the connection between the client and the Channel object in EMS open some predefined time. Following requests will use the same connection. The time it takes to establish a TCP/IP connection is much smaller than the response time of the servers, so for that reason it shouldn't impact very much on the result.
- EMA opens a new channel (Telnet connection) to the Network Element at every request. EMS keeps the channel (Telenet connection) to the Network Element open until the client releases it. A Telnet connection is also fast to establish compared to the response time of the servers.
- The EMA server stores the node configurations in an ACCESS database (located at the same computer) and the EMS server stores them in the NT registry. Perhaps is the ACCESS database accessed every time the EMA server receives a request? The connection to the database is perhaps established for every request and closed when the server has processed the request?
- The EMA server logs every request in a log file. Perhaps the EMA opens and closes the log file for every request. Together with the request are also date, time and the host-name of the client stored.

1 client (EMA) versus 5 clients (EMA)
When 5 clients run against EMA the response time is almost the same as when 1 client runs against it. The execution time of the experiment with 5 clients sending 10 requests each was 49 seconds while the execution time of the experiment with 1 client was 28. If one client sends 50 requests it would take 5 * 28 = 140 seconds to perform the experiment. This probably means that some free CPU time associated with I/O management was used when the server processed several requests simultaneously.
The CPU usage of the EMA process according to the task-manager in Windows NT varied rapidly between 0 and 30 percents during the experiment with one client. The experiment with 5 clients showed a CPU usage between 30 and 70 percents and an experiment with 10 clients showed a CPU usage between 50 and 100 percents.

In the Experiment with 10 clients some of the clients were refused to establish a connection with the EMA server.

In the repeated experiments with 5 clients, 2 of the clients one time per experiment didn't receive the reply connected to one of the requests. The timeout was set to 25 seconds which means that 3 of the clients probably finished executing much earlier than the other two. In one request of 50 in the experiment with 5 clients the EMA server responded with the fault message 0004, which means that the EMA server could obtain a communication channel to the Network Element but that the request for some reason couldn't be successfully processed. Once the experiment carried out with no unsuccessful requests. The response time was about 0.4 s worse but the execution time of the experiment was about the same (49 s).

Parsed printouts versus non-parsed printouts (EMA)
The response time of a request just containing the MML command alacp is about 2.4 seconds. If the printout is parsed with the parse parameter alcat then the response time of the request containing alacp\n/p alcat is about 2.5 seconds. The parsing process had no major impact on the response time of alacp.

But if the EMA server parses the printout belonging to the MML command caclp there is a difference of about 0.7 seconds. This may depend on that the PPM file is larger for caclp (6.63 Kbytes) than for alacp (1.29 Kbytes) and that the "function" used for parsing the printout of caclp is perhaps more complex.

1 client (EMS) versus 5 clients (EMS)
In the experiment with 5 clients was the response time about 3.6 worse than the experiment with 1 client. The time for a single client to execute 50 requests is 7*5 = 35 seconds which is about the same time it took to perform the experiment with 5 clients (38 s). From this we can draw the conclusion that at least from an observers perspective the EMA server doesn't processes any requests simultaneously. The requests are in some way queued at the server and processed one at a time. Another explanation could be that the server has reached it saturation level and that the response time increases linearly with the number of requests the server processes simultaneously. This is probably not the cause since the response time increases dramatically also when just 2 clients run against EMS. Another interesting observation during the experiments was that the CPU usage according to the task manager in Windows NT never exceeded 2 percents irrespective of the number of clients.

The EMS server never failed to process an request.

5 clients (EMA) versus 5 clients (EMS)
The response time of the servers was about the same. The execution time of the experiment against EMA was 49 s and the execution time of the experiment against EMS was 38 s. Based on these values the performance of EMA has approached the performance of EMS. But we also how to consider that the EMA CPU usage is between 30 and 70 percents and the CPU usage of EMS is constantly 2 percents. EMA also became unstable when the number of clients running against was more than 3. It lost requests and other requests couldn't be successfully processed.

5 Alarm Management System

5.1 Introduction

The other task in my thesis project is to design a telecom management system. The purpose of this is to illustrate how Corba can be used for communication and distribution in such a system. The next generation of O&M platform for AXE, the APG 40, will have some kind of alarm management server which exposes its services through a CORBA interfaces. The Alarm Management System, which I have designed (not implemented) is a proposition of how alarms from any Network Element could be managed within one system.

Operators use the Alarm Management System to coordinate the management of alarms towards multiple Network Elements. Alarms emitted by Network Elements are stored in persistent storage. The management is about acknowledging and unacknowledging alarms, adding comments to individual alarms and adding comments to Network Elements.

By acknowledging an alarm the operator tells every other operator viewing the same Network Element that he is responsible for taking some action according to the information included in the alarm. The action could be managed from remote, for example first tracing the fault and then perhaps downloading new software into the exchange. If it is a hardware failure the exchange must be repaired at its location.

If an operator takes some action he thinks is valuable information for himself or other operators he adds a comment to the Network Element in question. The Alarm Management System does not include any applications for repairing failures. The interfaces of these applications could easily be added to the system and accessible from the Alarm Management Directory Server if desirable.

To avoid inventing the wheel again when a new telecom management application is to be developed there is a standardization committee at Ericsson that defines IRPs (Integration Reference Points). An IRP contains information about what functionality just that telecom management application should include and specific rules that should be followed if the application shall be able to coexist or interact with other similar systems.

The design of AMS does not stringently follow the general Alarm IRP [16] and the Corba Alarm IRP [17]. The basic rules of the IRPs will be followed but in the design of the Alarm Management System I will not go into any specific details of for example how an event or a specific alarm in the system should look like. If I think these details are necessary when explaining some functionality in the system I will get into them, but they may not follow the Alarm IRPs.

There have been no specific requirements from Ericsson on the system logic of the AMS. From the beginning of the design I have just assumed that the managed Network Element includes some mechanism for storing the active alarm list and some mechanism for emitting spontaneous alarms. I have also looked at Alarm Tool that is a part of WinFiol and Tools. Alarm Tool is used by one operator managing alarms towards one AXE exchange.

Figure7 shows at a high level how Networks Elements are supervised from “management centers”. The Network Elements are geographically scattered over wide areas. The network elements are managed over a computer network consisting of the Local Area Networks at the management centers and the LANs which the Network Elements are attached to. In between we have some kind of communication network that ties the Local Area Networks together.

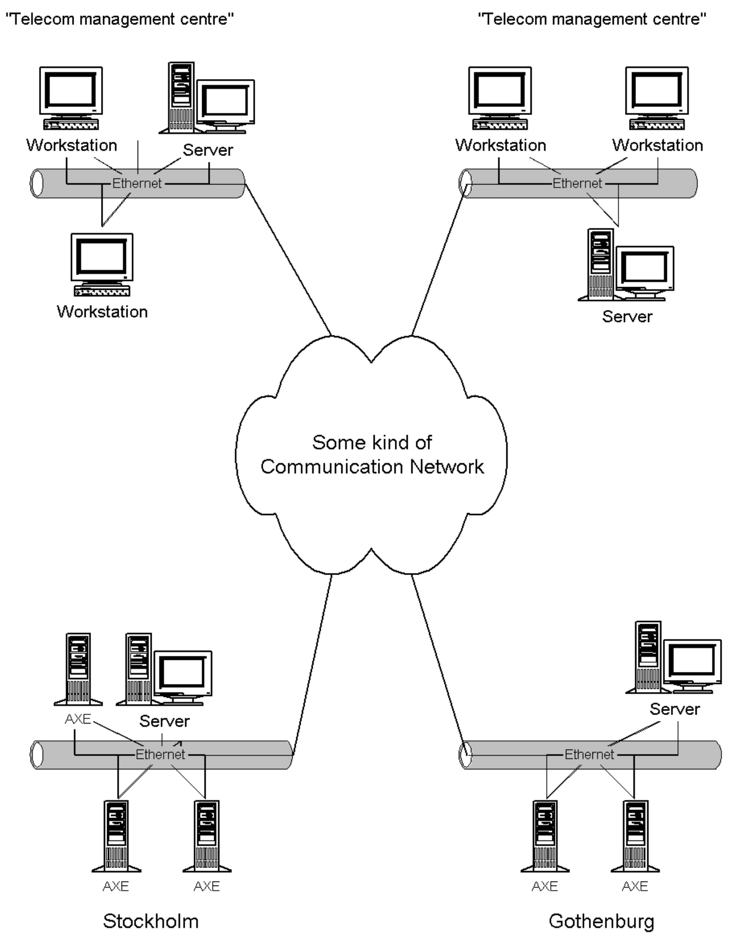


Figure7 Telecom management centers and Network Elements.

5.2 Limitations

Any security issues like authorization, authentication will not be included in the design of the Alarm Management System.

5.3 Alarms and Events

An Event is inserted to the Alarm Management System when an operator manage an alarm or when a Network Element emits an alarm. An event consists of an event record containing multiple attributes. Some of these attributes are mandatory and specified in the “general” Notification IRP [18] and the “general” Alarm IRP[16]. The structure of the event record in a corba environment is specified in the “Corba” Notification IRP [19] and the “Corba” Alarm IRP[17].

In section 4.1.3.1 in the “general” Alarm IRP, it is specified how an alarm list should behave. The alarm list contains all currently active alarms associated with a system, for example an AXE exchange. When a new alarm is emitted a new entry in the alarm list is created. An alarm has a certain severity level telling how urgent the alarm is. The severity level of the alarm can change. When the severity level of an alarm changes, a new entry is not created in the alarm list, the old one is used. When an alarm is cleared it is removed from the alarm list. All these changes are referred to as alarm Notifications. How to handle notifications is specified in the Notification IRP. The corresponding alarm notifications for the emitted events above are notifyNewAlarm, notifyChangedAlarm and notifyClearedAlarm. In the same section it is also specified how the system shall behave if the severity level is changed. Two rules should be applied:

1. If the new severity level is more urgent than the old level, the last acknowledge shall be removed. More exactly as specified in the Alarm IRP the system shall delete information in attributes AckUser and AckTime of alarm record (4.1.3.8). System updates the eventTime and PerceivedSeverity. System invokes notifyChangedAlarm notification.

2. If the new severity level is the same or less than the old level, System shall not delete information in attributes Ack User and Ack Time of Alarm Record. System updates the eventTime and PerceivedSeverity. System invokes notifyChangedAlarm notification.

Alarms have states. The possible lifecycle of an alarm is illustrated in the Alarm State Diagram in Figure8. The state diagram is an updated version of the alarm state diagram in section 4.2.1 in the "general" alarm IRP [16]. The state diagram in the Alarm IRP does not include rule 1 mentioned above. When I reported this to Edwin Tse, responsible of the Alarm IRP, he also informed me on a new feature that will be applied in the next version of the Alarm IRP. If an alarm is cleared without ever have been acknowledged it should not be removed from the alarm list. If an alarm is cleared its alarm number can be reused. Therefor must the application be able to handle this special case in some way.

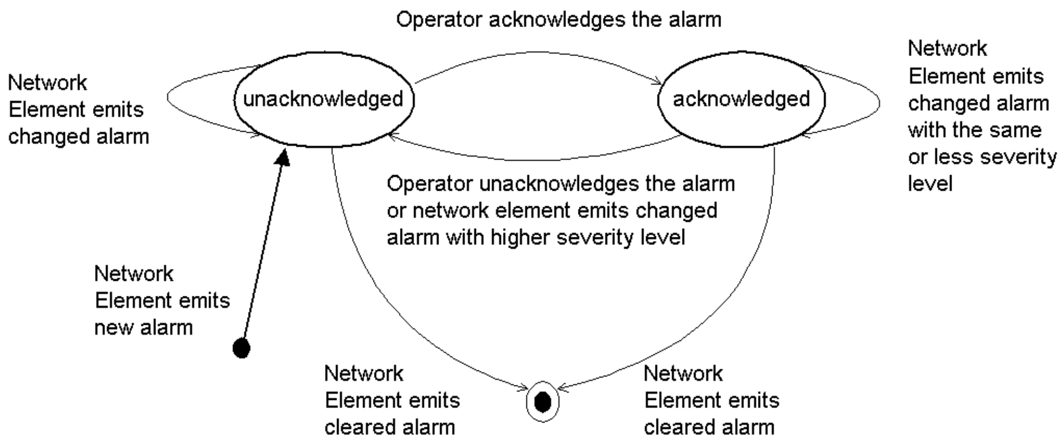

Operator acknowledges the alarm
Network Element emits changed alarm
unacknowledged
acknowledged
Network Element emits changed alarm with the same or less severity level
Operator unacknowledges the alarm or network element emits changed alarm with higher severity level
Network Element emits new alarm
Network Element emits cleared alarm
Network Element emits cleared alarm

Figure8 Alarm State Diagram

5.4 The components of AMS

The components involved in the Alarm Management System are presented in this section. The IDL interface of each component should be specified in the AMS module where also AMSException is specified.

```
module AMS{

    exception AMSException{
    enum exception {"Here are all possible exceptions listed"};
    };

    All interfaces ……………..
};
```

Last in this section some diagrams visually illustrate how the components are related to each other and how they communicate.

Before each component is presented we take a look at a possible running AMS which Figure9 shows. Two Network Elements are connected to AMS through their Network

Element Alarm Servers. The Event Log Server runs two Event Logs associated with each Network Element. The Event Handler runs two Event Channels associated with each Network Element. Two operators manage the AMS via Alarm Viewers. Operator 1 manage alarms from Network Element 1 and Network Element 2. Operator 2 just mange alarms from Network Element 2. To get an idea of how alarms from Network Elements are managed by AMS we follow alarm c1' that is emitted from a node in a communication network.

1. The Network Element Alarm Server (NEAS) receives the alarm from the single node it is managing.
2. Because the nodes in the communication network can be of different types the NEAS converts the type specific alarm to the alarm format of AMS. This is illustrated by c1' -> c1.
3. The alarm is logged in the Event Log belonging to the Network Element.
4. The alarm is distributed via the Event channel belonging to the Network Element to every operator interested in alarms from that Network Element.
5. Operator 1 has received and acknowledged alarm c1.
6. The acknowledge is logged together with the alarm in the Event Log.
7. The acknowledge is distributed via the Event Channel of the Network Element to every operator interested in alarms from that Network Element. Here it is just operator 1 that is interested.

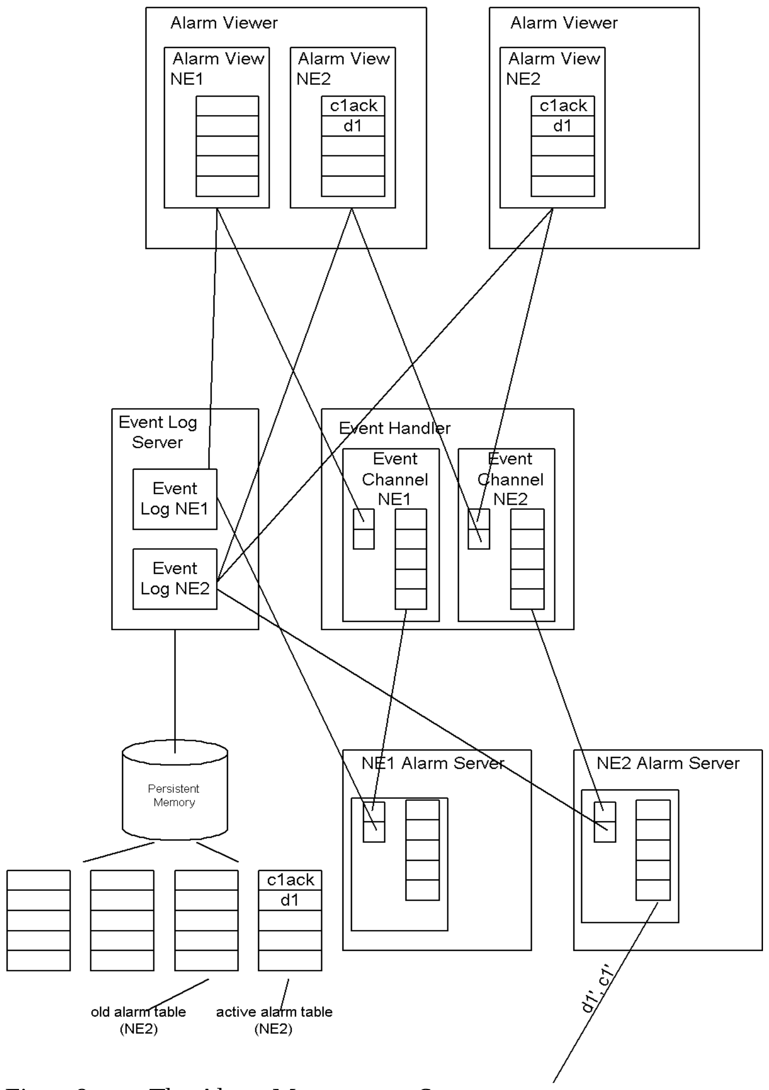


Figure9 The Alarm Management System

5.4.1 Event Producer

An Event Producer distributes events to every Event Consumer that have subscribed to the Event Producer. Events in the AMS could be alarms emitted by a Network Element or acknowledges of alarms emitted by Operators. If filtering of events is applied the Event Consumer specifies what filter that is to be used when it subscribes to the Event Producer.

IDL interface:

```
interface EventProducer{
      void subscribe(in string consumerId, in EventConsumer ref, in string filter); raises
      (AMSException);
      void unsubscribe(in string consumerId) raises (AMSException);
      void getSubscriptionStatus() raises (AMSException);
  };
```

5.4.2 Event Consumer

An Event Consumer receives events.

IDL interface:

```
interface EventConsumer{
      void push(in any event) raises (AMSException);
};
```

5.4.3 Alarm Storage

Alarm Storage provides operations for retrieving the active alarm list or the old alarm list associated with one Network Element. When a Network Element is connected to AMS it transfer its active alarm list by invoking the operation putAlarmList(in sequence<string>);

IDL interface:

```
interface AlarmStorage{
      sequence<string> getActiveAlarmList(); raises (AMSException);
      sequence<string> getOldAlarms(); raises (AMSException);
      void putAlarmList(in sequence<string>);
      AlarmSearch getAlarmSearch();
};
```

5.4.4 Alarm Search

This interface exposes operations for searching among active and old alarms associated with one Network Element.

IDL interface:

```
interface AlarmSearch{
    // Possible search functions on the active and the old alarm list are added here.
};
```

5.4.5 Administration

Event Log Servers, Event Handlers and Network Element Alarm Servers exposes this interface for administration. Any relevant operation that has to do with administration of these components should be added here.

The operation registerAtAmsDirectoryServer is used when the component connects to the Alarm Management System.

IDL interface:

```
interface Administration{
      void useAmsDirectoryServer(in string amsDirectoryServer) raises (AMSException);
      void dontUseAmsDirectoryServer(in string amsDirectoryServer) raises (AMSException);

      void registerAtAmsDirectoryServer(in sequence<string subSystem>) raises (AMSException);
      void unregisterAtAmsDirectoryServer(in sequence<string subSystem>) raises (AMSException);
};
```

5.4.6 Alarm Viewer

The Alarm Viewer is a visual component from where operators can subscribe to several network elements. The Alarm Viewer contains the alarm panels of all the Network Elements that the operator has subscribed to. By investigating the alarm panels the operator gets a quick overview of the alarm status of the Network Elements he is responsible for. If the alarm status of a Network Element changes and are required to be investigated more in detail, the operator opens the Alarm View window. In the window the whole active alarm list is presented.

5.4.7 Alarm View

The Alarm View is a visual component presenting the alarm status of a single network element. The Alarm view consists of three components. The alarm panel always shown in the Alarm Viewer, the Alarm View window that is opened on command from the operator and the Event Consumer object that receives events from the Network Element Alarm Server or from the Event Channel of the Network Element.

From the Alarm View the operator acknowledges alarms. The acknowledge is forwarded to the Event Log and Event Channel of the Network Element.

Since there probably are a lot of alarm views in one Alarm Viewer, in fact as many as the number of Network Elements the operator supervises, it perhaps would be better if the Alarm Viewer was responsible of receiving all events through one single Event Consumer interface. The events then are forwarded to the Alarm View through the internal interface between the Alarm Viewer and the Alarm View.

5.4.8 Event Log Server

The Event Log Server is responsible for starting up and removing Event Logs on commands from Network Elements.

When the Event Log Server is started it is provided with information about what subsystems in the AMS Directory it will serve. It registers itself by passing its object reference and the list of subsystems to the AMS Directory Server.

IDL interface:

```
interface EventLogServer{
      Administration getAdministration() raises (AMSException);
      EventLog newEventLog(in string neName) raises (AMSException);
      void removeEventLog(in string neName) raises (AMSException);
};
```

5.4.9 Event Log

An Event Log consumes events from one Network Element and several Alarm Views. Therefor it exposes the Event Consumer interface. When an event is received it is inserted to the active alarm table in the Database. If the event is an alarm that changes severity level the “new” alarm overrides the “old” alarm. The “old” alarm is inserted to the old alarm table.

The Event Log also exposes the Alarm Storage interface which is used by the Network Element Alarm Server for transferring the whole active alarm list when it for the first time connects to AMS. The Network Element Alarm Server also uses the Alarm Storage interface when it knows that the active alarm list in the Event Log is inconsistent. For example when the NEAS for some time haven't been able to store one or several alarms emitted by the Network Element.

The Event Log can be started by the Network Element itself or by an Event Log Server on command from the Network Element.

IDL interface:

```
interface EventLog{
     AlarmStorage getAlarmStorage() raises (AMSException);
     EventConsumer getEventConsumer() raises (AMSException);
     void release() raises (AMSException);
 };
```

5.4.10 Event Handler

The Event Handler is responsible for starting up and removing Event Channels on commands from Network Elements.

When the Event Handler is started it is provided with information about what subsystems in the AMS Directory it will serve. It registers itself by passing its object reference and the list of subsystems it shall serve to the AMS Directory Server.

The OMG Event Service is preferable used for realize the Event Handler. The Event Handler corresponds to the Event Channel Factory and The Event Channel explained in next section corresponds to the Event Channel in OMG Event Service. The Event Handler and Event Channel IDL interfaces presented here do not agree upon the IDL interfaces in the OMG Event Service, but they do include the same functionality.

The OMG Event Service does not include any mechanism for filtering events. If filtering of events is desirable, the OMG Notification Service is preferably used. Perhaps one operator is interested in alarms with a higher severity level and another operator takes care of alarms with less severity level.

IDL interface:

```
interface EventHandler{
     Administration getAdministration();
     EventChannel newEventChannel(in string neName);
     void removeEventChannel(in string neName);
};
```

5.4.11 Event Channel

An Event Channel consumes events from one Network Elements and several Alarm Views. The alarms are distributed to all Alarm Views that have subscribed to the Event Channel. The Event Channel exposes the Event Consumer interface and the Event Producer interface.

The Event Channel can be started by the Network Element itself or by an Event Handler on command from the Network Element.

IDL interface:

```
interface EventChannel{
     EventProducer getEventProducer();
     EventConsumer getEventConsumer();
     void release();
};
```

5.4.12 Network Element Alarm Server

The Network Element Alarm Server is the Network Element’s connection to the Alarm Management System. Before an alarm is emitted into AMS it must be transformed to the format supported by AMS.

An alarm is emitted into AMS by forwarding it to the Event Log and the Event Channel of the Network Element. The Event Log stores the alarm in stable storage and the Event Channel distributes the alarm to every interested Event Consumer (Alarm Views).

When an Alarm View is started it

The NEAS exposes an Event Log interface and an Alarm Storage interface. The interfaces are used by Alarm Views directly subscribing to the NEAS and by the Alarm Log that is started on command from the Network Element Alarm Server.

IDL interface:

```
interface NetworkElementAlarmServer{
     EventChannel getEventChannel() raises (AMSException);
     EventLog getEventLog() raises (AMSException);
     Administration getAdministration() raises (AMSException);
};
```

5.4.13 AMS Directory Server

The AMS Directory Server is the access point to the Alarm Management System. In the AMS Directory all Network Elements are sorted into Sub Systems. A Sub System is potentially container of several Network Elements, one Event Handler, one Event Log Server and any number of Sub Systems. The structure of the tree may image the graphical location of the Network Elements.

If there is no Event Handler or Event Log Server associated with a Sub System the first Event Handler or Event Log Server found above in the tree structure is used.

The OMG Naming Service is preferably used to realize the naming structure in AMS Directory Server. Every Sub System is a naming context. The Event Log Server and Event Handler in a subsystem may be bind to any Event Log Server reference and any Event Handler reference. The Network Element in a subsystem is a naming context containing name to reference bindings of at least the Network Element’s Network Element Alarm Server. Here could also other interfaces that the Network Element exposes be added. The tree structure of the Naming Service could look like in Figure10.

IDL interface:

```
interface AmsDirectoryServer{

     // Operations for building the tree structure of AMS Directory

     void addSubsystem(in string father, in string name) raises (AMSException);
     void removeSubsystem(in string name) raises (AMSException);
     void addNetworkElement(in string father, in string name) raises (AMSException);
     void removeNetworkElement(in string neName) raises (AMSException);

     //

     EventLogServer getEventLogServer(string neName) raises (AMSException);
     EventHandler getEventHandler(string neName) raises (AMSException);
     NetworkElementAlarmServer getNetworkElementAlarmServer(string neName) raises
     (AMSException);


     // Operations for connecting and disconnecting components to AMS
```

```
    void registerEventLogServer(in EventLogServer ref, in sequence<string subSystem>) raises
    (AMSException);
    void registerEventHandler(in EventHandler ref, in squence<string subSystem>) raises
    (AMSExcption);
    void registerNEAlarmServer(in NetworkElementAlarmServer ref, in string neName) raises
    (AMSException);
    void unregisterEventLogServer(in EventLogServer ref, in sequence<string subSystem>) raises
    (AMSException);
    void unregisterEventHandler(in EventHandler ref, in sequence<string subSystem>) raises
    (AMSExcption);
    void unregisterNEAlarmServer(in NetworkElementAlarmServer ref, in string neName) raises
    (AMSException);
};
```

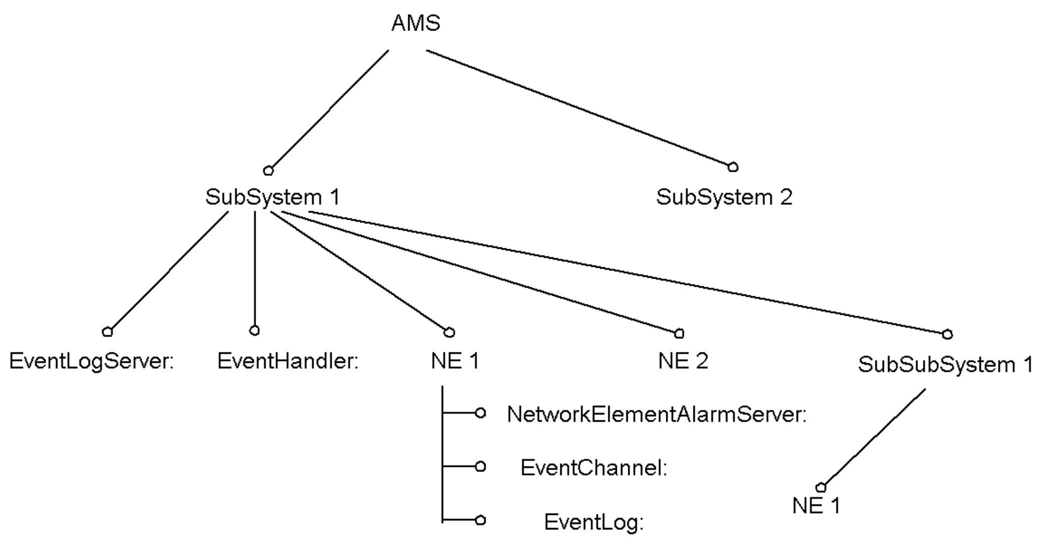


Figure10 Possible AMS Directory structure.

5.4.14 Communication in AMS

Figure11 shows the operations invoked on different objects when the Network Element Alarm Server is started. If the NEAS is able to start an own Event Log and Event Channel just operation 1 registerNeAlarmServer is required.

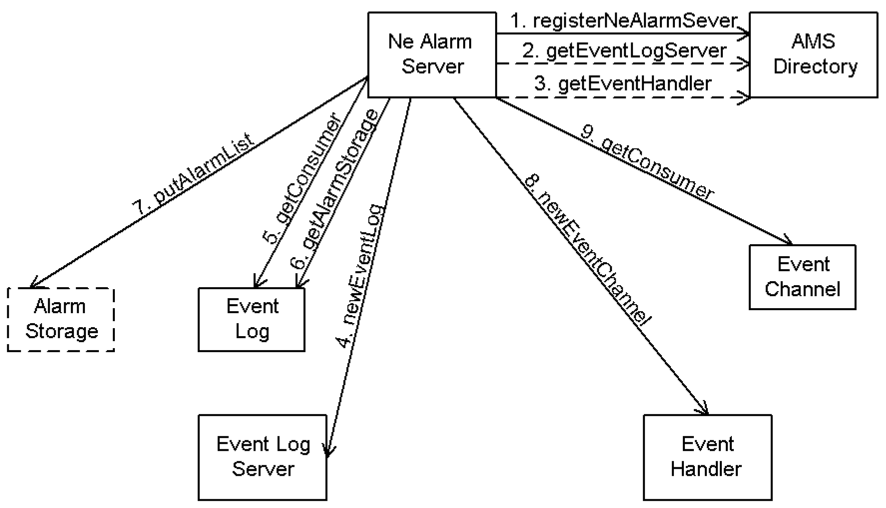


Figure11 A NEAS connects to AMS.

Figure12 shows the operations invoked on different objects in AMS when an Alarm Viewer subscribes to a Network Element.

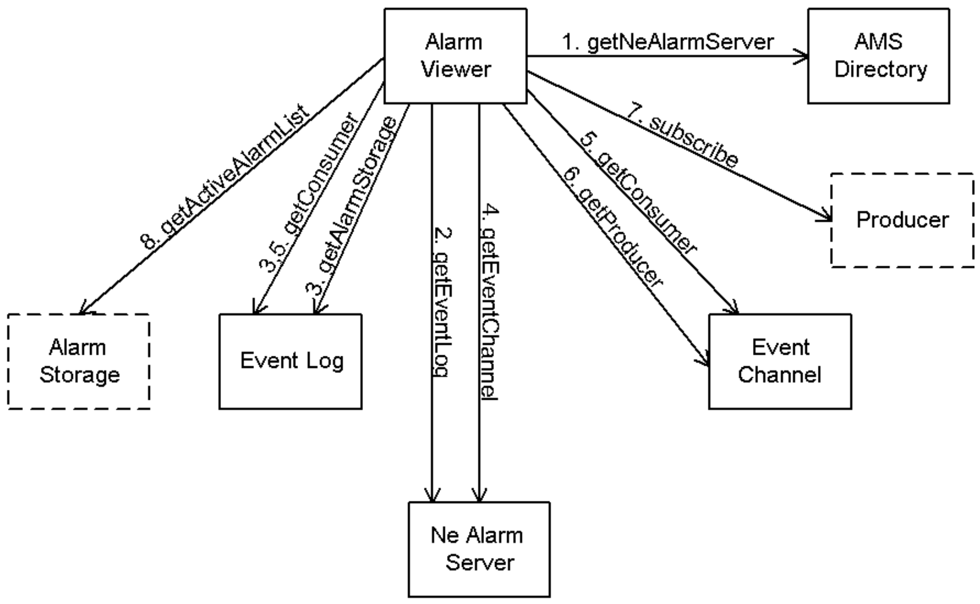


Figure12 Alarm Viewer subscribes to a Network Element

Figure13 shows a use-case of A Network Element Alarm Server emitting an alarm

1. The alarm is stored in the Network Element's Event Log.
2. The alarm is forwarded to the Event Channel of the Network Element.
3. The Event Channel distributes the alarm to every alarm view that has subscribed to the Event Channel.
4. The operator notices that the Network Element has emitted an alarm.
5. Same as 3.
6. Same as 4.

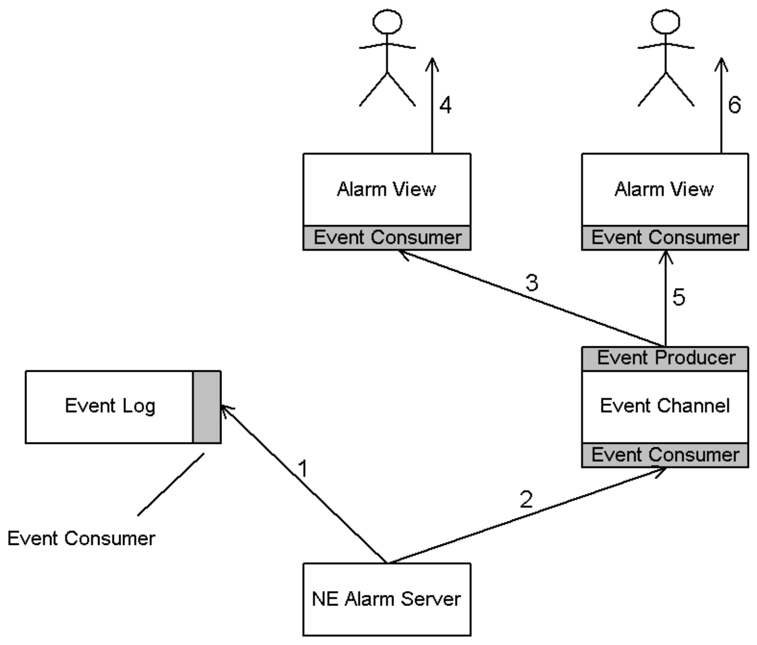


Figure13 The Network Element emits an alarm.

Figure14 shows a use-case of an operator acknowledging an Alarm

1. The operator acknowledges the alarm.
2. The acknowledgement is stored in the Event Log of the Network Element.
3. The acknowledgement is forwarded to the Event Channel of the Network Element.
4. The Event Channel distributes the alarm to every Alarm View that has subscribed to the Event Channel.
5. The operator notices the acknowledgement of the alarm.
6. Same as 3.
7. Same as 5.

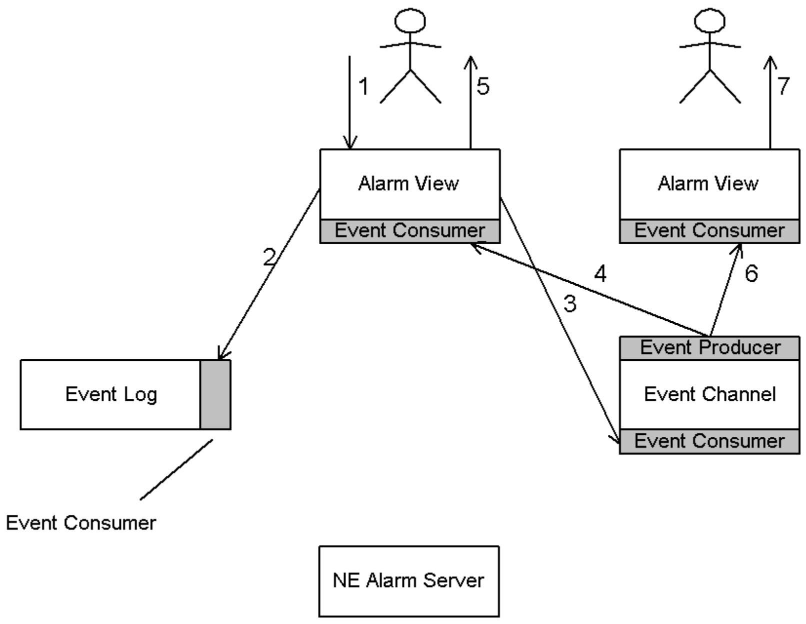


Figure14 An operator acknowledges an alarm.

5.5 Administration

The AMS administrators are responsible for:

- Starting the AMS Directory Server and build the Directory structure.
- Starting Event Log Servers and Event Handlers and provide them with information about what subsystems they will serve.
- Restarting the AMS Directory Servers, the Even Log Servers and the Event Handlers if they fail.

5.6 Failure Handling

If the Network Element doesn’t have any alarms to emit it emits a heartbeat periodically telling the surrounding system it is alive. This contradicts the Alarm IRP, which says that applications should include a function that periodically is invoked and issues the status of other components in the system.

If the Event Log or the Event Channel belonging to a Network Element Alarm server is unavailable the NEAS discards the object references and starts a new Event Log or Event Channel.

If an Alarm View is not available the Event Channel keeping its reference removes the Alarm View from the subscription list. The Alarm View realizes that something is wrong when it doesn’t receive any heartbeats from the Network Element.

If the Event Log or Event Channel is not available to an Alarm View it tries to subscribe to them again via the Network Element Alarm Server.

5.7 Availability

The most important thing in the system is that the alarms emitted from the Network Element Alarm Servers are logged.

When a Network Element Alarm Server discovers that its Event Log is not available anymore it just discards the Event Log reference and tries to start up a new one. If the present Event Log Server can't start a new Event Log the NEAS will contact another Event Log Server. There can be a problem with this solution. If new Event Logs are started and the old ones still are alive, the number of passive objects in the system could grow. The heartbeat sent out from the NEAS solves this. If an Event Log does not receive the heartbeat it will kill itself. The same procedure will be applied if the NEAS discover that the present Event Channel is not available. If the OMG Event Service is used to realize the Event Channel then it may be a problem to implement the heartbeat feature. I don't know if the Event Channel can investigate the events that it receives. If the OMG Notification Service is used there shouldn't be any problem, because it supports filtering.

5.8 Scalability

Because the number of Network Element in a Tele or Data -communication system can be very large and probably will increase in time it is important that the Alarm Management System can handle this. The computing capacity of the AMS is increased by adding new machines which run new Event Log Servers and Event Handlers. In this way it is theoretically possible for the AMS to manage any number of Network Elements. If the Event Log and Event Channel is managed by the Network Element itself the computing capacity of AMS naturally is increased when the new Network Element is connected to AMS.

A potential bottleneck in AMS when the number of Network Elements grow could be the AMS Directory Server. But this shouldn't be any problem because the AMS Directory Server is only used when a new Network Element is attached to the system, when an Event Log or Event Channel of an Network Element fails or when an operator becomes responsible of a "new" Network Element. These occasions shouldn't appear very frequently.

If AMS Directory Server becomes a bottleneck it could be replicated. Since a certain "management center" is responsible of a certain region of Network Elements which corresponds to a Sub System in the Directory tree of the Directory Server the Sub Systems can be replicated to the LANs where the "management centers" are located. This will also decrease the LAN to LAN traffic in AMS.

5.9 Flexibility

One purpose with flexibility is that the system should be adjustable to different customers' needs. One customer may run just a few network elements while another runs hundreds of network elements.

The flexibility may also help adjusting the system to different kinds of network elements, which may have different requirements of for example CPU load and memory use.

Here I discuss how the system could be deployed depending on different needs.

5.9.1 Event log location

If the network element have a database management system the alarms could be logged at the network element itself. All alarm management, acknowledging, adding comments and retrieving the active and old alarm list will load the network element.

One major advantage of logging alarms at the network element is that the alarm logs are naturally distributed. There is no single point of failure, which an external central DBMS could be.

The next generation of O&M platform for AXE the APG 40 which is a high available system will probably store the Alarm Log by itself.

One disadvantage with having the alarm logs distributed at the network elements is that searching of alarms among several network elements may be less efficient.

If the Event Log isn't located at the Network Element it is possible to place it at a management center's LAN or the LAN which the Network Element is attached to. Since it is very important that an alarm is logged the Event Log perhaps should be placed as close as possible to the Network Element i.e. the same LAN. The problem with this is the administration of the Event Log Server. Hopefully most or all of the administration can be done from remote.

5.9.2 Event Channel location

The Event Channel like the Event Log may also be located on the network element. If there are a lot of operators viewing the same network element the network element has to distribute all updates of the alarms to every operator. This may load the NE a lot.

If the event channel is located outside the NE the load of the network element will decrease. Another aspect a telecom operator may consider is that the network elements are scattered geographically over wide areas. Perhaps the communication network between the "management centers" and the Networks Elements has low capacity or for any other reason is slow. Then it could be better to place the Event Channel at the LAN where the "management center" is located.

Probably is also the "management centers" managing a certain region scattered geographically over wide areas. Since they manage different regions of the communication network they don't have to know about each other and the communication that arises due to alarm management will stay in their own local networks. Perhaps there is a need for overlapping the management of different regions. There are techniques that virtually build a single LAN of physical LANs that is interconnected via Internet. These techniques provide secure transfer of data over the Internet and transparent access of computers located on other physical LAN. The only difference the user will notice is that the communication perhaps is slower. One well-known technique that provides the features mentioned above is VPN (Virtual Private Network).

To summarize, the location of the components in the system may depend on how many network elements that are managed, the capacity and functionality of the network element itself, the geographical location of network elements and "management centers".

The most important thing to consider is that the location of the event channel or event log of a Network Element is completely transparent to the Alarm Management System and the operators using it.

6 Conclusions

6.1 EMA and EMS

The main task of the thesis project was to evaluate the EMA and the EMS servers.

The response time of EMA is about 4 times greater than the response time of EMS in the experiments with 1 client. EMA average response time: 2.8s, EMS average response time: 0.7. Possible reasons of the results are mentioned in section 4.7.

In the experiments with 5 clients the response time of EMA was about the same as the response time of EMS.

The response time of EMS was almost constant independent of the number of clients that ran against it. At some level the server must reach its saturation level which means that the response time will increase linearly with the number of clients. The saturation level was not possible to measure since the EMA server became unstable when the number of clients were more than 3. The server began loosing requests and other requests weren't successfully processed. In an experiment with 10 clients some of the clients couldn't establish a connection with EMA probably because the server had too much to do.

When several clients runs against EMS simultaneously it seems like the requests in some way are queued at the server. The response time of EMS increases linearly when the number of clients running against it grows. The execution time of the experiments also increased linearly when the number clients in the experiments were increased.

The CPU usage of EMA according to the task manager in Windows NT was between 0 and 30 percents in the experiments with 1 client. In the experiment with 5 clients the CPU usage was between 30 and 70 percents. An experiment with 10 clients showed a CPU usage between 50 and 100 percents.

The CPU usage of EMS was about 2 percents irrespective of the number of clients that ran against it.

The parsing process in EMA depends on what printout that is parsed. This was expected. When the printout from the MML command alacp was parsed the parsing process didn't impact on the response time very much. When the printout from caclp was parsed the average response time increased from 2.6 seconds (without parsing) to 3.3 seconds (with parsing).

The EMS server never lost a request and never refused a client to connect to it.

What is the future of EMA and EMS? The next O&M platform of AXE, the APG 40 will expose a Corba interface for MML communication with AXE. Perhaps the evaluation of EMA and EMS performed in this thesis project can provide the developers of the new interface at APG 40 with valuable information.

There are also further experiments left to perform on EMA and EMS. For example letting multiple clients communicate with different Network Elements. If there aren't enough Network Elements available a Network Element can be simulated with a Telnet server and programs that are executed when requests are received.

6.2 AMS

The Alarm Management System uses CORBA for communication and distribution. The choice of interfaces of the objects involved in AMS, the system logic and the fact that AMS uses CORBA have contributed to the following characteristics of the system.

Any Network Element can be part of the system by implementing a Network Element Alarm Server. The NEAS converts the specific alarms of the Network Elements to the format supported by AMS.

The logging of alarms is separated from the process that distributes alarms. In this way operators can receive alarms even if they can’t be logged. The Network Element can log alarms even if the distribution of alarms isn’t working.

The processes taking care of logging and distribution of alarms can be located at the Network Element itself or at any other machine that is part of the Alarm Management System.

Letting several Event Handlers and Event Log Servers having the possibility to distribute and log alarms of a Network Element increases the availability of the system. If an Event Log crashes the Network Element Alarm Server may start a new Event Log at another Event Log Server. The same is valid for Event Channels.

The AMS is a scalable system. If the number of Network Elements managed by the system is increased then the computing power of the system is raised by adding new machines. These machines run Event Log Servers and Event Handlers.

Appendix A   EMS IDL Interfaces

```
/*************************************************************************************/
/* Project:      EMA                                                                 */
/* File:         EMS.IDL  (Element Management Server CORBA IDL file)                 */
/* Created: 981130 - Jacco.Brok@lmf.ericsson.se                                      */
/* Changed: 990226 - Jacco.Brok@lmf.ericsson.se                                      */
/* Documentation: LMF/T/FI-98:00??  Rev. PA1                                         */
/* Notes:                                                                            */
/*************************************************************************************/


module EmrkServer
{
    exception EmrkException
    {
              string      errortext;
              long        errorcode;
    };

    typedef sequence<string> StringSeq;

    interface EmrkParser{};

    interface EmrkPrintout
    {
              void                  Release();
              boolean               GetResult(in long timeout) raises (EmrkException);
              StringSeq GetAllLines() raises (EmrkException);
              string                GetFirstLine() raises (EmrkException);
              string                GetNextLine() raises (EmrkException);
              string                GetLine(in long line) raises (EmrkException);
              long                  GetLineNumber();
              long                  GetTotalLines();
              long                  GetFlags();
              long                  GetFaultCode();
              long                  FindLine(in string token, in long start) raises (EmrkException);
              //EmrkParser getParser();
    };

    interface EmrkChannel
    {
              void        Release();
              EmrkPrintout SendCommand(in string mml_command, in long timeout)
                          raises (EmrkException);
              string      GetNodeName() raises (EmrkException);
              string      GetPortName() raises (EmrkException);
    };


    interface EmrkFactory
    {
              EmrkChannel OpenChannel(in string node, in string user)
                          raises (EmrkException);
    };

};  // module EmrkServer
```

References

[1] *AXE Survey. The platform and the applications*, Ericsson Telecom AB 1998
[2] *Corba Specification*, OMG, available at http://www.omg.org
[3] Element Management Resource Kit 1.1, Ericsson Utveckling AB 1999
[4] Gejdeman P (ÄS/UAB/I/MT), *File transfer API* Implementation proposal, internal document.
[5] Spong Anders, EMA CORE Implementation proposal, internal document.
[6] Spong Anders, Element management access, socket adaptation and script interface Implementation proposal, internal document.
[7] Jacco Brok, EMA Implementation Proposal, internal document.
[8] Jacco Brok, prototype Corba Element Management Server, internal document.
[9] RFC 2253 available at http://www.rfc-editor.org/cgi-bin/rfcsearch.pl
[10] Bedrock, Weldon, Goldberg, Belittled, *Programming with Visibroker*, Wiley, ISBN 0-471-23901-1.
[11] *Formalized MML*, internal document.
[12] W. Richard Stevens, TCP/IP Illustrated, Volume 1, Addison-Wesley, ISBN 0-201-63346-9.
[13] William Stallings, Data and Computer Communications fifth edition, Prentice Hall, ISBN 0-13-571274-2.
[14] Winsock2 Information, http://www.sockets.com/winsock2.htm.
[15] Creating a TCP Stream Socket Application. http://msdn.microsoft.com/library/wcedoc/wcecomm/winsock_25.htm
[16] LMC/FS Edwin Tse, Alarm Integration Reference Point (IRP) Specification: Informal Model, internal document.
[17] LMC/FS Edwin Tse, Alarm Integration Reference Point (IRP) Specification: Corba Solution Set Version1: 1, internal document.
[18] LMC/FS Edwin Tse, Notification Integration Reference Point (IRP) Specification: Information Model Version1, internal document.
[19] LMC/FS Edwin Tse, Notification Integration Reference Point (IRP) Specification: Corba Solution Set Version1: 1, internal document.

Glossary

ASCII	American Standard Code for Information Interchange
CORBA	Common Object Request Broker Architecture
DBMS	Database Management System
EMA	Element Management Access
EMRK	Element Management Resource Kit
EMS	Element Management Server
FTP	File Transfer Protocol
IP	Internet Protocol
IRP	Integration Reference Point
LAN	Local Area Network
LDAP	Lightweight Directory Access Protocol
MML	Man Machine Language
O&M	Operation and Maintenance
OMG	Object Management Group
PCM	Persistent Command Model
POS	Printout description
PPM	Persistent Printout Model
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VPN	Virtual Private Network