Project Number:	AC112
Project Title:	TRUMPET
	Inter Domain Management with Integrity
Deliverable Type:	I
CEC Deliverable Number:	AC112/GMD/WP3/DS/I/011/C
Contractual Date of Delivery:	September 30th 1998
Actual Date of Delivery:	October 13, 1998
Date of This Version:	October 13, 1998
Title of Deliverable:	Implementation (Final Demonstration)
Workpackage contributing:	WP3
Nature of the Deliverable:	T
Document Location:	Trumpet/deliverable/del11c/d11c_v08.zip
Authors:	Editors: Marcus Wittig, Damien Artiges

Abstract:

This deliverable describes the final stage of the software implementation in TRUMPET, which was directed at the third trial, in October 1998 and the final demonstration of project results. It has served as a baseline document for both, the software developers in TRUMPET, and the system's administrators which needed to install and run the software as part of the trials.

Keyword list:

TMN, inter-domain management, security, integrity, availability, confidentiality, security measures, security profiles, security policies.

This page intentionally left blank

Executive Summary

This deliverable describes the final stage of the software implementation in TRUMPET, which was directed at the third trial, in October 1998 and the final demonstration of project results. It is based on a updated version of D11B, and it includes new sections for the new software components and extensions, which have been developed in last year of the project.

This document has served as a baseline document for both, the software developers in TRUMPET, and the system's administrators which needed to install and run the software as part of the trials. Accordingly this deliverable has been organised as a set of handbooks and manuals which describe the various aspects. While the configuration and operation manuals describe how to install and run software, the developers handbooks address particularly the details of implementation design. Moreover, the security components have been described in a separate set of handbooks since the security software and documentation will be distributed to other projects such as the ACTS MISA project.

As part of the implementation design the developer handbooks describe the details of component engineering models, APIs and communication interfaces. Thus, these handbooks present a refined view on the detailed component design presented in deliverables D8 and D9 [TRUMPET-D8, TRUMPET-D9] and describe how the individual components have been mapped onto the selected implementation technologies. While these handbooks are in particular addressed to the software developers, the installation and operation manuals should help the systems administrator with the installation and execution of the software. The latter also includes a description of hardware and software prerequisites for each component so that all the information is given which is needed to set-up a site to run the TRUMPET system.

List of Contributors

Damien Artiges	**Ascom Monetel (P01)** CICA, 229 rte des Cretes F-06560 Sophia Antipolis France	tel.: +33.92.94.22.04 fax: +33.92.94.20.20 email: artiges@ascom.eurecom.fr
Cyril Autant **Francois Letort**	**Alcatel ISR (P02)** 3, rue Ampère F-91349 Massy Cedex France	tel.: +33 1 69 76 24 76 fax: +33 1 69 76 25 50 email: Cyril.Autant@isr.alcatel-alsthom.fr Francois.Letort@isr.alcatel-alsthom.fr
Dominique Maillot **Christiane Pace**	**Sema (P03)** Departement Projects Européens 3-9 rue Helene Boucher F-78280 Guyancourt France	tel.: +33.1.30.96.42.12 fax.: +33.1.30.96.44.72 e-mail: Maillot@sqy.sema.fr pace@sqy.sema.fr
Shahrzade Mazaher **Jonn Sketting**	**Norwegian Computing Center (P08)** P.O. Box 114 Blindern N-0314 Oslo Norway	tel.: +47.2285.2500 fax: +47.2269.7660 e-mail: Shahrzade.Mazaher@nr.no, Jonn.Skretting@nr.no
Marcus Wittig **Oliver Schittko**	**GMD FOKUS (P09)** Kaiserin-Augusta-Allee 31 D-10589 Berlin Germany	tel.: +49.30.3463.7218 fax: +49.30. 3463.8218 e-mail: Wittig@fokus.gmd.de Schittko@fokus.gmd.de,
Lionel Sacks	**University College London (P10)** Torrington Place London WC1E7JE England	tel.: +44.1.71.419.3976 fax: +44.1.71.388.9307 e-mail: l.sacks@eleceng.ucl.ac.uk
George Andrianopoulos	**ATC Bull (P12)** Aharnon 438 str, GR 11143 Athens Greece	tel.: +30.1.2182008-9 fax: +30.1.2182010 e-mail: andriano@01p.gr
Robert Croke	**Paris Data** Technology House, Lissadel Street, Salford M6 6AP, England	tel.: +44 161 278 2555 fax: +44 161 278 2506 email: rc@parisdata.co.uk

Table of Contents

Table of Figures

Table of Tables

1 INTRODUCTION

The TRUMPET secure inter-domain service management systems have been developed as part of the activities in Work Package 3, which is responsible for the architectural design, the definition of test scenarios, and the implementation of the TRUMPET system. To validate the results of the design and implementation work, the TRUMPET system has been executed and tested as part of trials in which real users work in a TMN multi-domain environment established by National Hosts. This deliverable describes the final stage of the software implementation, which was directed at the third TRUMPET trial, in October 1998 and the final demonstration of project results. It is based on a updated version of D11B [TRUMPET-D11B], and it includes new sections for the new software components and extensions, which have been developed in last year of the project. These achievements can be summarised as follows.

- The Secure Management Package, which comprises the security components developed by TRUMPET, was further developed. Support for access control to managed objects, data Integrity, and confidentiality were implemented and evaluated throughout the UK trial.
- To secure the management interfaces between the Java-based management systems a Secure Socket Layer implementation was integrated with Java communications infrastructure. For the interaction between the TRUMPET CPN and VASP management system this solution provides support for strong mutual peer-entity authentication, data integrity and confidentiality.
- A Java-based Security Administration Tool has been developed which allows for the configuration of the Secure Management Package and the monitoring of Security Alarms generated by a secured management system. The tool provides a graphical user interface to manipulate security profiles and access control rules, and it provides an event display for security events and alarms.
- To improve the security solutions suggested by TRUMPET with respect to external access of customers to the service provider management system a mobile Personal Security Environment has been integrated. This technology allows to authenticate customers using smart card technology independently from the terminal location or the end user application.
- To be able to visualise the complex system interactions and to monitor internal events, a graphical Event Trace System (Messaging System) was developed. This tool was used successfully as part of the public demonstration at IS&N'98, and it also turned out to be of added value for the operation of the UK trial.
- To meet the requirements of the UK and French trials additional management functions were implemented for the configuration management of the end-to-end broadband connections. This included the release of virtual path connections as well as the provisioning of status information and event reporting on VP connections. Accordingly the CPN user interface has been extended to visualise connection status information and to display event reports messages received. Besides this, the CPN user interface has been extended to manipulate site data which is kept as part of the network topology map display. This allowed to adapt the CPN user interface more easily to the network topology used for the UK & French trials.
- Significant efforts have been undertaken to stabilise all software components and to simplify the installation and configuration of the software. To detect and fix bugs advanced debugging tools have been used as well as the feedback gained from the testing activities as part of the trial operations. For the installation of the TRUMPET software on the UK trial sites a runtime package distribution has been compiled which included a system setup script and the start-up scripts for applications and services. As compared to the preparation of the Swiss Trial this helped a lot to reduce the efforts required to install and configure the TRUMPET software on the trial sites for the Scottish & French trial.

This document is intended to serve as a baseline document for both, the software developers in TRUMPET, and the system's administrators which need to install and run the software as part of the TRUMPET trials. Accordingly this deliverable has been organised as a set of handbooks and manuals which describe the various aspects. As part of the implementation design the developer handbooks describe the details of component engineering models, APIs and communication interfaces. Thus, these handbooks present a refined view on the detailed component design presented in deliverables D8 and D9 [TRUMPET-D8, TRUMPET-D9] and describe how the individual components have been mapped onto the selected implementation technologies. While these handbooks are in particular addressed to the software developers, the installation and operation manuals should help the systems administrator with the

installation and execution of the software. The latter also includes a description of hardware and software prerequisites for each component so that all the information is given which is needed to set-up a site to run the TRUMPET system.

The structure of the document is as follows: The following chapter gives an overview on the TRUMPET management architecture, the security architecture, and the technology environment. Chapter 3 describes the installation and configuration of the software distribution packages which have been compiled by TRUMPET. Following this, chapter 4 is directed at, both system administrators and operators. It describes the configuration and operation of the various applications and components of the TRUMPET system. Chapters 5 and 6 contain the developers handbooks for the service management system and security components. Finally, the list of references can be found in Chapter 7. A list of acronyms is included in Annex A.

2 ARCHITECTURE

2.1 Management System Architecture

The TRUMPET project investigates security of management communications and operational interfaces within the context of inter-operation between several players involved in broadband provisioning. A model service management scenario has been developed in which to explore these issues. The scenario shown in Figure 1 is of a number of public network operators (PNO) co-operating and competing in the provisioning of semi-permanent Virtual Path Connections (VPC) between Customer Private Networks (CPN) across international - and within national - domains [TRUMPET-D6]. The network operators are considered to be owners of networking systems including transmission connections and switching equipment. In addition we consider the role of an organisation who doesn't own any network systems but works with the public network operators to sell on network services and who is licensed for value added resale (VASP). This player presents a 'one stop shopping' facility for the corporate VPN use.

Customers see an end-to-end connection and are not necessarily aware of which PNOs are contributing to establish the connection. In contrast to this a VASP sees a end-to-end connection as a set of *segments*, each supported by a different PNO, but does not know how each segment has been set up within the corresponding PNO (*i.e.*, what ATM switches are used). The essential interactions between the three players is considered to be facilitated by TMN based management information exchange. Thus the management systems of the players mentioned above form a service provisioning system for management and provision of broadband (ATM) network connections between two customers/end users. CPN is a dedicated service in the customer organisation, which already has a contract with the VASP. The VASP management system provides network connectivity to customers by utilising the resources of one or more PNOs. VASP allows customers to create, modify and delete connections, thus effectively providing the Virtual Private Network (VPN) service to the customers. PNOs provide the physical infrastructure, i.e. the network, and the adequate management interface to interact with it.

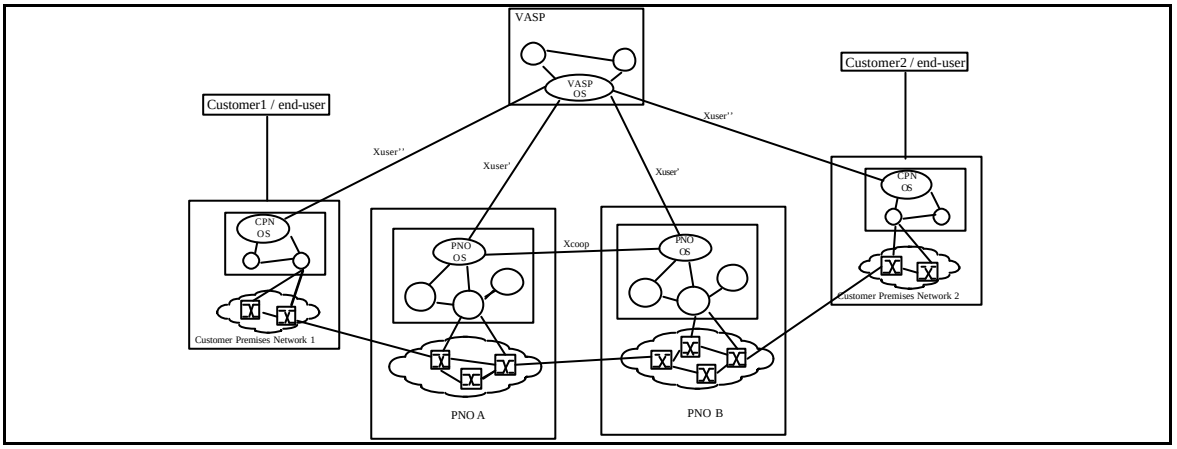

Figure 1: The TRUMPET Reference Architecture

This scenario thus presents a reasonable view of emerging service provisioning and market players. Within this context the needs for high integrity and secure management information exchange between the players can also be clearly seen. That each player must have publicly accessible data network interconnections there is a clear exposure of both the data in transit and the accessibility of operational interfaces. Further, the management platforms employed by each player cannot be restricted to that of any individual vendor. Thus there is also a demand for understanding the robustness impacts of the inter-working of different technologies. The exposure of data and interfaces points to the need for data encryption and integrity requirements for data in transit; the exposure of the interfaces requires access control. Both these security requirements imply that the organisations can exchange security information such as public keys, private keys and access rights. These concerns are the principle focus of the implementation of the security elements of TRUMPET. These security functions are, moreover, explored in the kinds of technology – both apparent and emerging – which can be used of open service provisioning. The functional requirements on the reference architecture are:

- VASP receives requests for services from a customer across a TMN like interface Xuser''. These requests concern the establishment of inter-domain connections between two customers. The requirement placed on the VASP consists in finding the best solutions to connect the two customers, taking into account the requirements concerning the connection (Quality of Service, bandwidth...), the physical resources available in the PNO domains, and optional criteria related to financial costs.
- PNO provides the physical infrastructure, i.e. the networks. It has a contract with the VASP, which knows the entry points of the PNO. During the establishment of the connection, a negotiation takes place between the PNO and the VASP (*via* the Xuser' interface) to reach an agreement for an offer from the PNO which corresponds to the VASP requests.
- Customer is the end-point of the connection. Essentially, two kinds of customer will be considered, although here they have been merged in a single entity. The customer/end-user is the user of the application requiring the connection. The customer/network (CPN) is the organisation which will send the connection request to the VASP. It usually is a dedicated service in the customer organisation, who has already subscribed a contract with the VASP.

2.2 Security Architecture

The security architecture consists of a set of security components, which can be used by TMN platforms with open or closed protocol stacks. The distinction is described as:

- For a research management platform, the internals of the protocol stack can be accessed for additions and modifications. In this case, TRUMPET suggests use of the Generic Upper Layer Security (GULS) specifications and the Transport Layer Security Protocol (TLSP).
- For a commercial management platform, security features can only be added on top of the interface to the protocol stack, i.e. over CMISE or ACSE. Data integrity, confidentiality and non-repudiation are especially difficult to implement if the protocol stack is not accessible.

Since TRUMPET has selected a commercial TMN platform for implementation, only the commercial platform architecture has been further developed into component specifications, and is presented here. However, the security architecture is designed to be as generic as possible by insulating the security-relevant code from the actual environment through an integration layer. Therefore, the applicability of the TRUMPET security architecture to other management environments (open protocol stack, WBEM) should be straightforward.

TRUMPET's security policies are based on the use of public key mechanisms for authentication and prescribes the use of Trusted Third Parties (TTP) in the role of Certification Authorities (CA). Symmetric encryption is used for confidentiality and data integrity protection.

The architecture is shown in Figure 2, with dashed lines indicating the components added by TRUMPET, and solid lines indicating existing components.

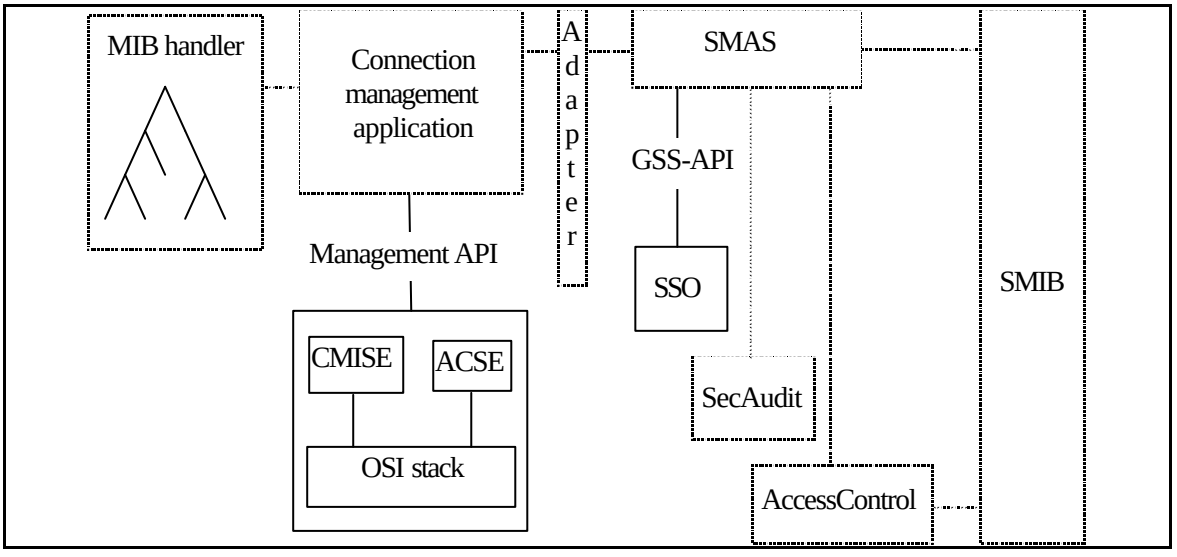

Figure 2: Architecture for a Commercial Management Platform.

This security architecture aims at securing communications between a Management Application Entity (MAE) belonging to a domain and another MAE belonging to another domain. Different MAEs within a domain may use different security profiles, and the choice of security profile is made during the initialisation of the security context. Selection of a security profile may be constrained by the mechanisms supported by an implementation, by internal policies, or by target OS policies. The architecture must support all security profiles because it is not a priori possible to determine which security policies that will be applied to a particular X interface. However, some policy decisions inherently affect the architecture, like the decision to use public key cryptography.

When a MAE belonging to the initiator OS performs inter-domain management operations, it may be working on behalf of another entity (a human user or another MAE) or on its own behalf. To preserve privacy of users, and to facilitate management of access privileges (authorisation), the MAE will always use its own identity and associated set of privileges to perform the management operations on the target OS. This implies that proper internal security measures must be enforced before human operators or MAEs are given access to the management capabilities of MAEs that perform inter-domain management.

To be as independent as possible from the management application and its environments, the security components are accessed through an *Adapter* component. Ideally, the interfaces provided by the Adapter to the application should be identical to the interfaces the application uses for access to the communication services. Alternatively, a defined interface (like the Generic Security Service API (GSS-API) [RFC 1508]) could be offered from the security components, in which case the applications must incorporate this interface, and perform necessary transformations on the data passed over the interface.

To achieve a high degree of flexibility to particular security mechanisms, the architecture is based on the GSS-API. The GSS-API interfaces to a Security Support Object (SSO) used to establish a security context between the communication parties and to perform security transformations on the application data.

With respect to a commercial management platform, there is some minimal support required for the transfer of security data between communicating parties. The specific requirements that must be satisfied are:

- The authentication field of ACSE must be supported to establish the security context and for authentication;
- The access control field of CMIS management operations must be supported to transfer security related information.

The security architecture also requires that agents have control over accesses to the MIB. This is necessary to enforce access control to MOs. Although TRUMPET is responsible for the implementation of agents, their design may be restricted by tools provided by the platform provider. For example, the code generated by a GDMO compiler may not be compatible with the introduction of access control mechanisms.

The architecture shown in Figure 2 is able to support most of the security services required by the TRUMPET policies. The security services that cannot be fully supported are integrity, confidentiality, non-repudiation and security negotiations.

When encryption for confidentiality is performed above CMISE, the encrypted data must be inserted into one of the fields of the particular CMIS operation being requested. However, most of the fields have specific pre-defined types that cannot accommodate an encrypted data type. In general, the only exception, and the only fields that can be encrypted, are the fields used to carry the attribute values to and from the target MIB.

2.3 High-Level Technology Viewpoint

This section presents the choices for implementation technologies used in the system as described in the previous sections, at a global or high level. These choices have been made so as to cover a number of requirements derived from the project goals and demands from the trials scenarios. As shown in the figure below, the system has used a mixture of techniques. The CPN and most of the VASP have been implemented using Java. The motivation for this are two fold. The CPN site was specified to be very versatile. Thus using Java would allow the CPN to interface the VASP interface with almost any in house tools required. The second motivation was to be able to extend the TRUMPET security architecture to other underlying technologies - particularly important in view of the attitude of NM Forum that TMN must be supported by more than only CMIP. To allow these things to be fully supported the VASP supports a managed object model close to that required for TMN. This is achieved by using a reliable object communications and object persistence technology (the Voyager package) and by locating the objects with in a X.500 naming structure supported by the Lightweight Directory Access Protocol (LDAP). The communications between the VASP

and the PNOs are supported by a CORBA based Xuser gateway. This innovation has three important aspects.

The PNO itself supports the Xuser' interface compatible with that defined by the MISA consortium and derived from that developed in EURESCOM P408. Thus this interface should be fully compatible with any TRUMPET / MISA trials foreseen. Finally, the network and network element support is achieved by an underlying implementation of the ATM Forum M4 interface.

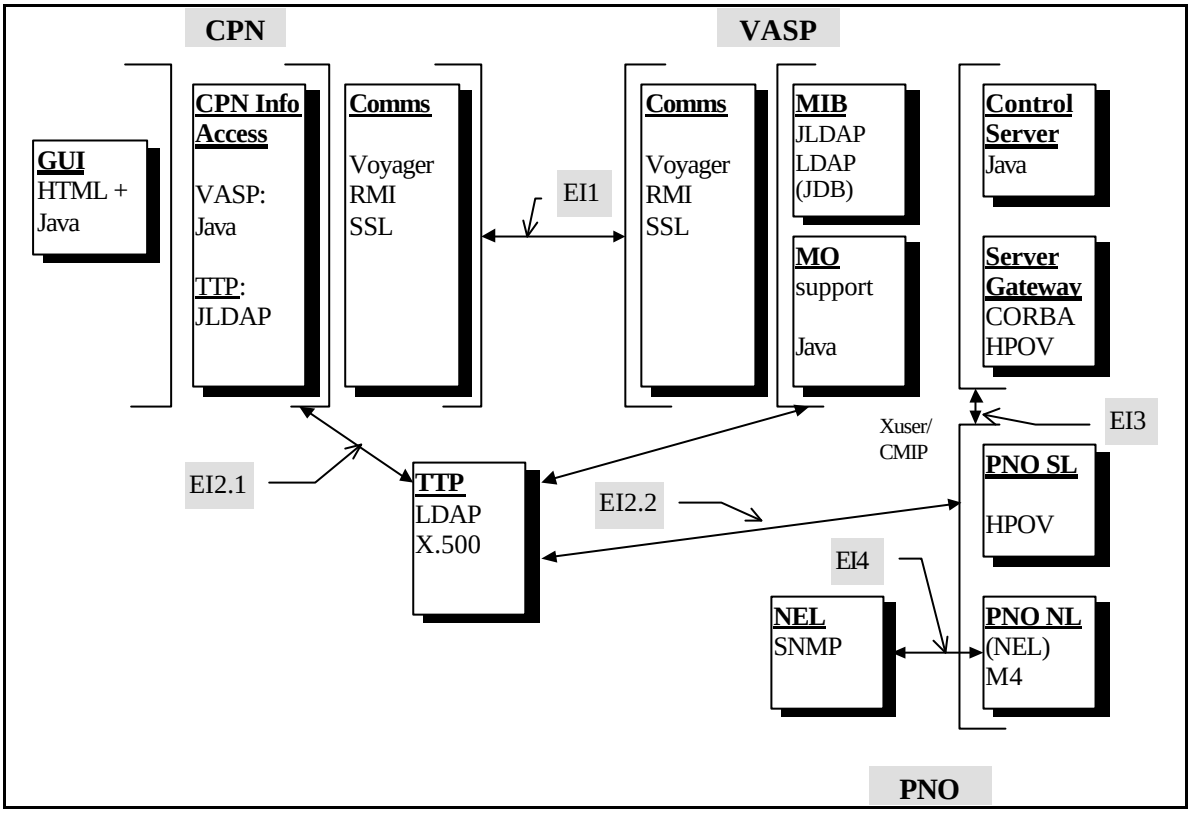

Figure 3: Technologies and Interfaces used in the TRUMPET System

There are four major sites, each with specific communication needs the; CPN, VASP, PNOs, TTP. These are communication interfaces *external* to each domain and are labelled EI1-4. Although the components within each site also have inter process communications, these are not highly problematic and need not be considered in detail here. The communications channels considered are those which concern inter-domain security.

Key to Figure 3:

CORBA: Common Object Request Broker. An Object Oriented RPC like system suitable for building distributed systems over heterogeneous systems.

LDAP: Light weight Directory Access Protocol. A slimmed down implementation of X.500.

JLDAP: A Java API to LDAP

HTML: Hyper-Text Mark-up Language. A specialisation of SGML, adding functionality for linking material together in a networked environment (although loosing much of the type setting and document data base functionality of SGML).

Java: A trendy programming language plus a set of libraries suitable for building distributed systems and (light weight) user interfaces. Cross platform portability is provided by defining a Virtual Machine operating environment, rather than by forcing cross platform development in all environments and the utilisation of platform independent communications (*e.g.* CORBA or CMISE)

SNMP: Simple Network Management Protocol. For internet based network elements.

RMI: Remote Method Invocation. The low-level functionality for Java communications, in addition to pipes and sockets.

Voyager: A product which extends / completes Java communications to cover full functionality of such things as synchronous & asynchronous communications.

SSL: Secure Sockets Layer. A facility in Java for adding confidentiality, integrity and authentication to the classic socket and pipe communication model.

M4: A OSI management interface at the network and network element level defined by the ATMForum for ATM network management.

TTP: Is the trusted third party.

Summary of External Interfaces:

EI1: Between the Customer Premises and the VASP (customer) server. Using Java facilities. Low-level over IP (backbone or over ISDN) , High level protocol to support object reference model as supported by LDAP.

EI2: Between clients and TTP site. There are alternatives here supported by SecuDE; LDAP access over IP or (full) X.500.

EI3: Between VASP Control server and PNO sites. Xuser over CMIP. Requires Java to CMIP gateway to be supported *via* CORBA.

EI4: From Network Elements Adapters to network elements. SNMP over IP.

3 CONFIGURATION MANUAL

This manual gives general information on how the TRUMPET software is going to be installed on a computer system. Two distribution packages have been compiled which include all the applications and software components developed by TRUMPET:

- The runtime package distribution only contains the executable code such as binary code, byte code for Java applications, and executable shell scripts to start applications and communication services. This package has been used by TRUMPET to install the software on the trial sites as part of the UK and French trials.
- The developers package distribution contains both executables and the full source code of TRUMPET development including supplementary documentation and configuration files to build the software.

The following Section 3.1 describes the hardware and software prerequisites which must be met to install and run the TRUMPET software. Following this, Section 3.2 describes the runtime package distribution and Section 3.3 describes the development package distribution.

3.1 Hardware and Software Prerequisites

The TRUMPET software has been developed and tested on the following system environments:

- Hewlett-Packard, Inc. PA workstations with HP-UX 10.x operating system
- Sun Microsystems, Inc. SPARC workstations with Sun Solaris 2.x

Additionally, the components of the Secure Management Package excluding the XMP adapter have been ported to Microsoft Windows NT version 4.0. As the TRUMPET software has been engineered for portability, it is expected that adaptation to other operating systems with POSIX compliant APIs can be achieved easily. Beyond this, some parts of the TRUMPET software have been developed with the Java Developers Kit (JDK, Sun Microsystems, Inc.) which provides an abstraction from the underlying hardware and operating system. It is expected that Java applications can run on any system environment which is supported by JDK.

The following table gives an overview on the software packages which are required to run the TRUMPET system. For each software package, the TRUMPET applications and subsystems which rely on it are indicated. Moreover, the availability of package is highlighted for system environments used by TRUMPET and the dependencies on other software packages is indicated.

Vendor/Package	Required for	Availability and Dependencies	Description/Comments
Sun Microsystems, Inc. JRE/JDK 1.1.4 (or higher)	CPN User App., CPN Server, VASP Customer Server, VASP Control Server, Security Admin. Tool, and CMA Messaging System Adapter	Solaris 2.x Windows 95/NT HP/UX 10.x	Java Development (JDK) and Runtime (JRE) toolkit, JRE is a subset of JDK including the JAVA libraries and the virtual machine to run Java Applications. *NOTE: JDK for HP/UX is provided by Hewlett-Packard.*
Netscape Navigator 4 or Netscape Communicator 4 or Microsoft Internet Explorer 4.0	CPN User App., Security Admin. Tool, and for online documentation	Solaris 2.x Windows 95/NT HP/UX 10.x (Internet Explorer is not available for HP/UX 10.x, for Solaris 2.5 only a beta version is available to date)	A web browser which provides the execution environment to download and run Java Applets. *NOTE: This is optional since the applet can also be executed with the appletviewer which is included in the JRE/JDK pakage*
Objectspace Voyager 1.0.0	CPN Server, and VASP Customer Server	100% pure JAVA (requires JRE/JDK 1.1.x)	Voyager is a Java-centric distributed computing platform supporting transparent access to

			remote objects and facilitates object mobility.
Objectspace JGL 2.0.2	CPN Server, and VASP Customer Server	100% pure JAVA (requires JRE/JDK 1.1.x)	JGL includes 11 optimized data structures including sequential containers, sets, maps, and queues. Both ordered and hashing versions of sets and maps are available. *NOTE: Support for object persistence is only provided using JRE/JDK 1.1.*
IAIK-JCE 2.0 and IAIK iSaSiLk 2.0	CPN Server, and VASP Customer Server	100% JAVA based (requires JRE/JDK 1.1.x)	Java Cryptography Extension and SSL Implementation
Netscape LDAP Java SDK 1.0	CPN Server, and VASP Customer Server	100% JAVA based (requires JRE/JDK 1.1.x)	Java toolkit to build applications that access networked directory data through the Internet standard Lightweight Directory Access Protocol (LDAP, RFC 1777). Constitutes a subset of the Netscape Directory SDK.
IONA Orbix 2.3 /C++ (Multi Threaded version)	VASP CORBA/TMN Gateway, and CMA Messaging System Adapter	Solaris 2.x (requires SunWSpro Compiler 4.x for to build executables) Hewlett-Packard HP/UX 10.x (reqs HP aC++ Compiler)	OMG CORBA 2 compliant C++ ORB, required for CORBA-based client/server applications, Runtime & Developers Licenses are available.
IONA OrbixWeb for Java 2.x	VASP Control Server & Event Handler of the control server)	Solaris 2.x Windows 95/NT HP/UX 10.x (requires. JRE/JDK 1.0.2 or higher)	OMG CORBA 2 compliant Java ORB, Required for generation of the stubs used by the VASP Control Server to access the VASP CORBA/TMN gateway, Runtime & Developers Licenses are available. *NOTE: Not required for a TRUMPET runtime installation, as the OrbixWeb class library is included in the TRUMPET distribution packages.*
Netscape Directory Services 3.0	CPN Server, and VASP Customer Server	Solaris 2.x Windows 95/NT HP/UX 10.x	Directory Service implementation (X.500 based) supporting LDAP version 2 and 3. *NOTE: This package includes the Netscape Directory SDK which in turn includes LDAP Java SDK 1.0beta 2.*
University of Michigan Michigan LDAP library	Certification Authority, SMP	Solaris 2.x Windows 95/NT HP/UX 10.x	Various LDAP tools which have been developed at UMich. Needs to be in place to operate an CA. Pointers to LDAP-related sources can also be found at reference.com.
Rogue Wave tools.h++ 7.x	potentially all components developed	Solaris 2.x Windows 95/NT	C++ foundation class library contains over 120 classes,

	with C++, i.e., SMP components, and VASP CORBA/TMN gateway	HP/UX 10.x	including dates, times and strings, sets, bags, B-Trees, sorted collections, linked lists, queues, stacks, and more This library is bundled with SunWSpro C++ and the HP aC++ compilers.
Sun Microsystems, Inc. Workshop pro C/C++ compiler 4.2	all components developed with C/C++, i.e., SMP components, VASP CORBA/TMN gateway, and PNO Xuser Agent,	Solaris 2.x	SUN ANSI C/C++ Compiler, Linker and Libraries (includes Rogue Wave tools.h++ 7.0), Contact and price info. can be obtained through SunExpress
Hewlett-Packard HP aC++ for HP/UX-10	all components developed with C/C++, i.e., SMP components, VASP CORBA/TMN gateway, and PNO Xuser Agent,	HP/UX 10.x	HP ANSI C/C++ Compiler, Linker, and Libraries (includes C++ standard lib and Rogue Wave tools.h++ 7.0), *NOTE: HP offers another Compiler (Cfront) called C++ 3.0 or CSET which is NOT suitable since there is no proper support for Orbix, and limitations for library support such as tools.h++!!!*
Hewlett-Packard OpenView Distributed Management version 4.21 or 5.01 (Manager/Agent platform)	VASP CORBA/TMN Gateway, PNO Xuser Agent, and PNO NMS	Solaris 2.x, HP/UX 10.x	Distributed management platform providing the infrastructure to develop and operate TMN management applications.
Hewlett-Packard Cumulative Consolidated Patch PSOV_01730 for HPOV-DM 4.21 on Solaris 2.x	VASP CORBA/TMN Gateway, PNO Xuser Agent, and PNO NMS	Solaris 2.x	Cumulative Consolidated Patch for HPOV-DM 4.21 on HP/UX 10.x. The patch is required to run the VASP CORBA/TMN gateway and the PNO Xuser Agent. The patch can be obtained from HP Support web site.
Hewlett-Packard Cumulative Consolidated Patch PSOV_12211 for HPOV-DM 4.21 on HP/UX 10.x	VASP CORBA/TMN Gateway, PNO Xuser Agent, and PNO NMS	HP/UX 10.x	Cumulative Consolidated Patch for HPOV-DM 4.21 on HP/UX 10.x. The patch is required to run the VASP CORBA/TMN gateway and the PNO Xuser Agent. The patch can be obtained from HP Support web site.
Fore Systems ForeThought 4.02	PNO NMS	FORE ASX200 (embedded Sun Solaris system)	ATM Networking Software which also provides the SNMP management interfaces (MIBs) required for the NMS

Table 1: List of required platforms and packages

To simplify the configuration of the TRUMPET software, all packages should be installed in a common directory called "Pkgs". This directory needs to be created subordinate to the directory where the TRUMPET software is going to be installed. If some of the packages have been installed already in different directories it is sufficient to set symbolic links from the "Pkgs" directory to the installation directories. To run the TRUMPET software without major changes of the configuration files, the "Pkgs" directory should contain the following files (directories or symbolic links):


```
LDAP                    University of Michigan LDAP library
ORBXns1.03              IONA Naming Service version 1.03 for Orbix
Orbix_2.3MT             IONA Orbix verion 2.3MT
SSL                     IAIK iSaSiLk 2.0a and IAIK-JCE 2.0
jdk1.1.4                SUN Java JDK version 1.1.4
jgl_2_0                 Objectspace JGL 2.0.2
ldapjdk-unix            Netscape LDAP Java SDK 1.0
rw7                     RogueWave Tools.h++ version 7.00
sec51c                  Secude GmbH, Secude version 5.1
voyager1.0.1            Objectspace Voyager 1.0
OV                      Hewlet Packard OpenView DM version 4.21 or 5.01
```

3.2 Runtime Package Distribution

The runtime package distribution only contains the executable code such as binary code, byte code for Java applications, and executable shell scripts to start applications and communication services. It is distributed as a single file called "trumpet_RT.tar" which has been created using the UNIX tape archive format. This archive needs to be unpacked in the directory where the TRUMPET software is going to be installed. To extract files from the archive apply the tar user command provided on UNIX systems:

- `tar -xvf trumpet_RT.tar`

As a result a new directory called "Trumpet.RT" has been created in the current directory. The detailed directory structure is described below.

```
Trumpet.RT       Top-level diretory
     /bin             Executable shell scripts to run TRUMPET
                      applications and services
     /bin_hp          Binary executables for HP-UX 9
     /bin_sun5        Binary executables for SUN/SPARC Solaris 2.5
     /classes         Class files (bytecode) for JAVA applications
     /data            Initialization and data files for TRUMPET
                      applications
     /etc             Configuration files for TRUMPET application
     /lib             Shared objects for HP-UX 10 and
                      SUN/SPARC Solaris 2.5
```

To complete the installation three system environment variables need to be set:

- `HOME`

  The absolute path name of the home directory for the user account on which the TRUMPET software is being installed. Usually this variable is set already for the system environment.

- `TRUMPET_HOME`

  The absolute path name of the directory where the "Trumpet.RT" and the "Pkgs" directories can be found.

- `SYSTEM`

  The system identification. Possible values are "sun5" or "hp".

3.3 Development Package Distribution

The development package distribution provides the full set of implementation files including the source and binary code files for the various applications developed by TRUMPET. It is distributed as a single file called "trumpet_dev.tar" which has been created using the UNIX tape archive format. This archive needs to be unpacked in the directory where the TRUMPET software is going to be installed. To extract files from the archive use the tar user command provided on UNIX systems:

- `tar -xvf trumpet_dev.tar`

As a result a new directory called "Trumpet.dev" has been created in the current directory. The detailed directory structure is described below.

```
Trumpet.dev            Top-level directory
      /bin                   Executables
            /scripts               Executable shell scripts
      /doc                   Online-documentation for TRUMPET software
      /etc                   Configuration files
      /javacode              Java class files
      /src                   Source code directory
            /cpn                   CPN applications directory
                  /cpn                   CPN Server application
                  /gui                   CPN GUI applet
            /corbaGateway          CORBA/TMN static gateway
            /vasp                  VASP applications directory
                  /controlServer         VASP Control Server
                  /customerServer        VASP Customer Server
                  /vaspTypes             Common files
            /securityPackage       SMP components directory
                  /acControl             Access control component
                  /manager               Audit & Alarm manager
                  /self                  Audit & Alarm agent
                  /sso                   SSO component
                  /smasc                 SMASC component
                  /xmpV7_adapter         XMP-Adapter implementation

            /xuser                 Source code of the Xuser-Agent
```

To complete the installation three system environment variables need to be set:

- `HOME`

  The absolute path name of the home directory for the user account on which the TRUMPET software is being installed. Usually this variable is set already for the system environment.

- `TRUMPET_HOME`

  The absolute path name of the directory where the "Trumpet.dev" and the "Pkgs" directories can be found.

- `SYSTEM`

  The system identification. Possible values are "sun5" or "hp".

3.3.1 Makefile Configuration

To simplify the use of makefiles to build the TRUMPET software, a set of rules and macros have been defined. These definitions can be found in the following files which are located in the "Trumpet.dev/src" directory:

- `Config.mk`

  Contains common macros like DEBUG or OPT to control code generation and machine-independent setting like classpath settings.

- `<system>.mk` (where "<system>" is "sun5" or "hp")

  Contains machine-dependent definitions like compiler and linker flags, system libraries and system commands.

- `Rules.mk`

  Contains rules and targets to be used in the makefiles.

Depending on the location of required software packages (see also Section 3.1) it may be required to modify the "Config.mk" or "<system>.mk" files. Usually there is no need to modify the "Rule.mk" file or the specific makefiles. Each specific makefile includes the "Config.mk" and "Rule.mk" files, while the definitions set in "<system.mk>" are obtained implicitly through the "Config.mk" files. The relevant macros which may used throughout the makefiles are listed below.

```
QUIET=quiet     supress commandline output
DEBUG=debug     generate debugger code
OPT=opt         optimize code
PIC=pic         generate psosition independent code
PROF=prof       generate profiling code
STATIC=static   generate static executables

INCS            C/C++ include directories
CPPFLAGS        C/C++ preprocessor flags
CFLAGS          C compiler flags
CXXFLAGS        C++ compiler flags
JFLAGS          JAVA compiler flags
LDFLAGS         linker flags
LDLIBS          libraries to link with
```


4 OPERATION MANUAL

4.1 Service Management

4.1.1 CPN User Application

4.1.1.1 Environment Settings

The CustomerGUI.html file contains the parameters used to configure the CPN User Application. Four parameters define host access details and the rest define the scope of the map to be displayed. The four parameters that define access the CPN Server and the VASP can be broken into two groups of two parameters. Two of the parameters define the host on which the CPN Server is running and the port on which it listens. The other two parameters define the distinguished name for directory root on the VASP LDAP directory server and the password to access it.

In the example below, the CPN server host is set to 'localhost' communicating on port 3001. The VASP LDAP distinguished name is set to: 'cn= Directory Manager, o=vasp' and the password set to 'Trumpet1'

```
<param name="CPN_HOST" value="localhost">
<param name="CPN_PORT" value="3001">
<param name="VASP_DN" value="cn= Directory Manager, o=vasp">
<param name="VASP_PASSWORD" value="Trumpet1">
```

Other parameters are also specified which relate to the background map used by the GUI and which is read from the file map_nsp.gif. The following is an example of the parameters used to define the map.

```
<param name="MAP_WIDTH" value="601">
<param name="MAP_HEIGHT" value="739">
<param name="MAP_LONG_TL" value="-11.2">
<param name="MAP_LONG_TR" value="3.55">
<param name="MAP_LONG_BL" value="-9.7">
<param name="MAP_LONG_BR" value="2.4">
<param name="MAP_LAT_TL" value="59.48">
<param name="MAP_LAT_TM" value="59.48">
<param name="MAP_LAT_TR" value="59.48">
<param name="MAP_LAT_BL" value="49.84">
<param name="MAP_LAT_BM" value="49.84">
<param name="MAP_LAT_BR" value="49.84">
<param name="MAP_TALLEST" value="59.48">
<param name="MAP_MIDDLE" value="4">
<param name="MAP_LONGTOP" value="59.48">
<param name="MAP_LONGBOT" value="49.84">
```

MAP_WIDTH is the width of the map in pixels.

MAP_HEIGHT is the height of the map in pixels.

MAP_LONG_TL is the longitude reference for the top left-hand corner of the map.

MAP_LONG_TM is the longitude reference for the top mid point of the map.

MAP_LONG_TR is the longitude reference for the top right-hand corner of the map.

MAP_LONG_BL is the longitude reference for the bottom left-hand corner of the map.

MAP_LONG_BM is the longitude reference for the bottom mid point of the map.

MAP_LONG_BR is the longitude reference for the bottom right-hand corner of the map.

MAP_LAT_TL is the latitude reference for the top left-hand corner of the map.

MAP_LAT_TM is the latitude reference for the top mid point of the map.

MAP_LAT_TR is the latitude reference for the top right-hand corner of the map.

MAP_LAT_BL is the latitude reference for the bottom left-hand corner of the map.

MAP_LAT_BM is the latitude reference for the bottom mid point of the map.

MAP_LAT_BR is the latitude reference for the bottom right-hand corner of the map.

MAP_TALLEST is the longitude reference for the widest part of the map (normally the mid point)

MAP_MIDDLE is the latitude reference for the middle point of the map. **Note**: This latitude reference should be the same at the top of the map and the bottom of the map i.e. the latitude at the middle of the map should be vertical.

MAP_LONGTOP is the longitude reference at the MAP_MIDDLE latitude at the top of the map.

MAP_LONGBOT is the longitude reference at the MAP_MIDDLE latitude at the bottom of the map

These values should only be changed if the map background image (map_nsp.gif) is changed.

4.1.1.2 Starting the Application

The GUI can be executed using the Appletviewer that comes with the JDK or any Java enabled web-browser.

Note: For the GUI to successfully communicate with the CPN Server, the CPN Server must already be running.

4.1.1.3 Operation

The CPN User Application or GUI (Graphical User Interface) is a Java applet. As an applet it can be executed using the Appletviewer program, which comes with the Java Development Kit or executed in a Java enabled web-browser which supports the required JDK (currently 1.1).

The main display of the GUI is shown below:

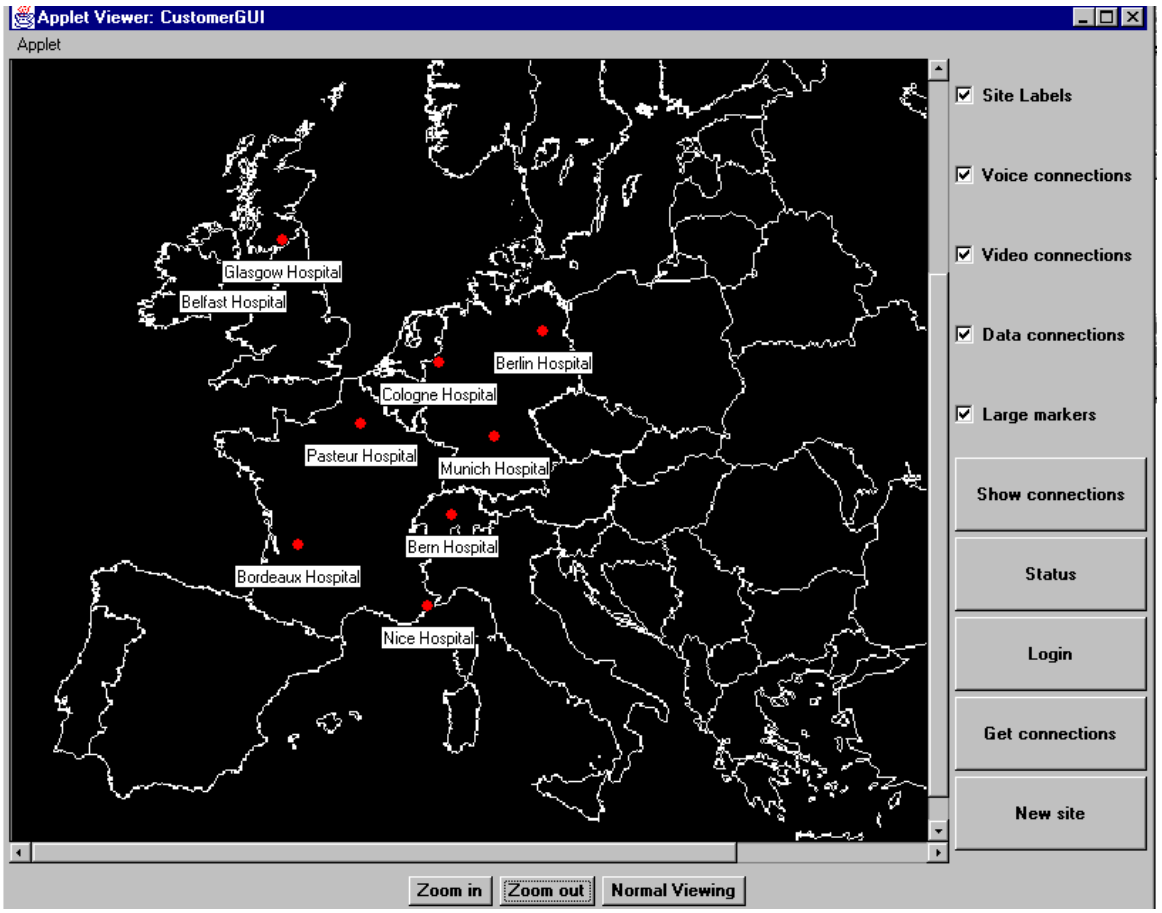

Figure 4: Main Window of the CPN User Application

Note: The map displayed by the GUI may be changed to best suit the requirements of the Customer Premises Network that is being managed by the GUI. The functionality of the program is not changed just the backdrop.

The display is constructed from four areas, the map; the checkboxes; the controls on the right hand side (checkboxes and large buttons) and the smaller buttons at the bottom of the map.

The map is a display area on which sites and connections are displayed and can be scrolled using the scrollbars as required.

The checkboxes control the information that is displayed on the map area. The “Site Labels” checkbox when checked causes the labels containing the site name to be displayed close to the site marker, and when unchecked these labels are removed. The three “connection” checkboxes (Voice, Video and Data) control the display of which types of connections are displayed. Checking the box will cause connections of that type to be displayed in the map area. The final checkbox changes the size of the site marker. Checking the box causes larger markers to be displayed; removing the checking means that smaller site markers are used.

The five larger buttons have the following actions:

Button Label	Button Action
Show connections	Displays a scrolling list box containing a text description of the known connections
Status	Displays a scrolling list box containing status messages
Login	Displays a login dialog
Get connections	Sends a request to the CPN Server for all know connections
New site	Displays a dialog for creating a new site

The buttons underneath the map allows the map to be displayed larger (Zoom in) or smaller (Zoom out) or to be returned to its default size (Normal Viewing).

4.1.1.3.1 Interacting with the Map

In order to make use of the GUI it is necessary to interact with the map. This section describes the functions that can be carried out using the mouse to initiate actions on the map.

Displaying Site Details

The details of a site can be displayed in a dialog box by moving the mouse over the required site marker and pressing a mouse button. The following image shows an example of possible site details:

Figure 5: Display of Site Details

Creating Connections

To create a new connection the following actions should be taken:

1. Click on the first site of the pair to be connected (site marker should change to blue)
2. Keeping the mouse button depressed move the mouse to the second site (a line should be displayed between the mouse and the first site as shown below)

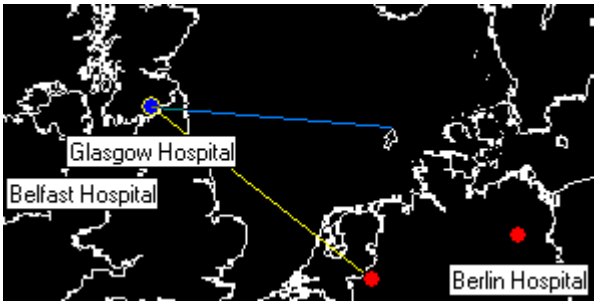

Figure 6: Creating Connections

3. When the second site marker changes colour the mouse button can be released, a line connecting site one with site two should be displayed

4. When the two sites have been connected, a dialog box is displayed which allows the user to specify the connection requirements. The dialog is shown in the diagram below (Note: later versions of the GUI also allow bandwidth to be selected):

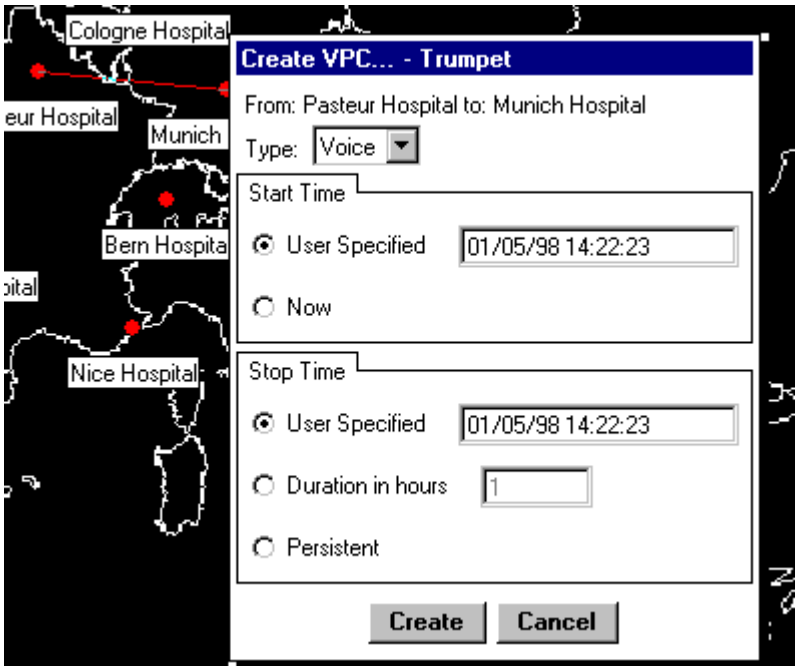

Figure 7: Dialog Box "Create VPC"

Type: This dropdown list allows the connection type to be set, possible choices are Voice, Video and Data. Later versions of the GUI allow the required bandwidth for the connection to be selected from another dropdown list, which has values suitable for the connection type selected.

Start Time: This section of the dialog allows the user selection of the required activation time for the connection.

Stop Time: This section of the dialog allows the user selection of the required deactivation time for the connection. It can be set to a specified date and time, duration in hours or marked as a permanent connection, which means the user must delete the connection to release the resources used.

Displaying Connection Details/Status

The status of a connection is indicated by the colour of the connection line on the map as shown in the table below:

Colour	Indicated Connection Status
Yellow	Connection has been requested
Orange	Connection has been reserved by the VASP
Green	Connection active
Grey	Connection deactivated
Red	Connection is in an error state.

However, if there are multiple connections between sites or the details of the connection need to be displayed then pressing down the mouse button on the connection line will display an information dialog. For a single connection between sites the following dialog is displayed:

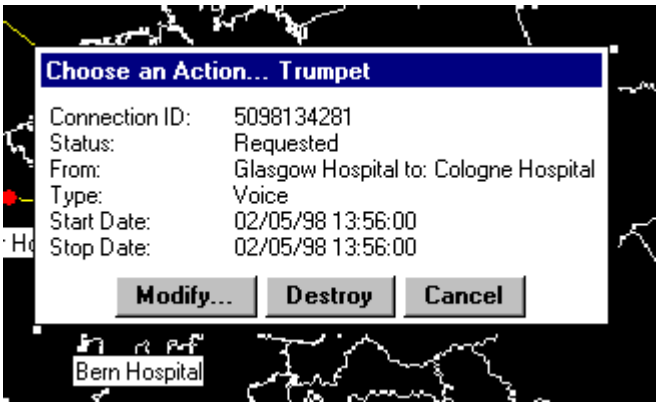

Figure 8: Dialog Box "Choose an Action"

Later versions of the GUI also display the bandwidth of the connection is this dialog.

If there are multiple connections between a site then the following dialog is displayed:

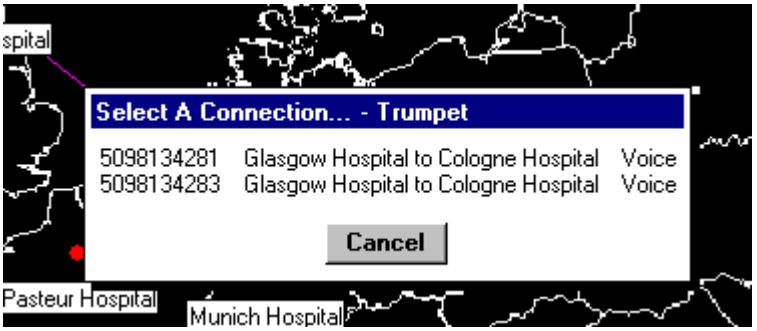

Figure 9: Dialog Box "Select a Connection"

As the mouse moves over each connection, its colour is changed indicating the selected connection. If the mouse button is then pressed, the connection details are displayed for that connection in a dialog box as shown.

Modifying Connections

To modify a connection's settings, display the connection details dialog as previously discussed and press the Modify button. When selected, a dialog box similar to the create connection one is displayed, the difference being some of the fields cannot be changed. At present only the stop time and bandwidth of a connection can be modified.

Deleting Connections

To delete a connection, display the required connection details dialog and select delete. This will remove the connection from the map and send the required request to the CPN Server.

4.1.2 CPN Server

4.1.2.1 Environment Settings

CLASSPATH Configuration

The paths to the following class directories must be set in the CLASSPATH environment variable.

```
../voyager/lib/voyager1.0.0.jar
../jgl/jgl_2_0
```

```
../netscape/ldap/classes
../iaik_jce/lib/iaik_jce_full.jar
../iaik_jce/lib/jdk11x_update.jar
../iSaSiLk/lib/iaik_ssl.jar
../iSaSiLk/lib/jdk11x_update.jar
../
```

The path to voyager/bin and to the Java/bin directory must be included in the PATH environment variable.

You must also ensure that you have the latest versions of the VASP classes used by the CPN Server installed in the directory or the CLASSPATH (i.e. VAssociationServer, VCustomerService, EntrySet, Entry, Attribute, AttributeList, CMISException).

CPN Server Configuration

The CPN Server can now be configured using its INI file (CPNServer.ini). This is used to store default information, which is required to run the CPN Server and provide values for the CPN GUI.

The following sections are defined

[**CPN Server**]	This section is used to define information required for the execution of the CPN Server
port=3001	This key specifies the default port number on which the CPN Server listens for connections from CPN GUI's.
loglevel=0	This key specified the level of logging to be output by the CPN Server, 0 indicates that no output should be generated apart from exception causes.
[**VASP Server**]	This section is used to define information about the VASP Customer Server.
port=8000	This key specifies the default port number on which the VASP Customer Server listens for connections from the CPN Server.
hostname=localhost	This key specifies the default host name on which the VASP Customer Server is running.
[**CMA Event Manager**]	This section is used to define information about the CMA Event Management Server.
port=50096	This key specifies the default port number on which the CMA Event Management Server listens for events from the CPN Server.
hostname=localhost	This key specifies the default host name on which the CMA Event Management Server is running.
[**Site Details**]	This section defines the CPN Server default site details. The sites specified in this section are only used if the file SITES.STO does not exist when the CPN Server starts. If this file does not exist then it is created using the site details from this section. This section can contain multiple examples of the following key: site='Sitename A','Location of site A',Latitude,Longitude
[**VoiceBandwidths**]	This section contains the bandwidths, which should be displayed by the CPN GUI when Voice connections are selected.
Converter=64	This key specifies the conversion factor required to change the selected bandwidth to the units required by the VASP. This key must only occur once within this section.
b1='1 x 64K'	This key specifies the first bandwidth that should be displayed by the GUI. The second bandwidth should have the key b2, the third b3 and so on.
[**DataBandwidths**]	This section contains the bandwidths, which should be displayed by the CPN GUI when Data connections are selected.

Converter=1024	This key specifies the conversion factor required to change the selected bandwidth to the units required by the VASP. This key is only used by the GUI when the unit letter in the display string (see next key) contains 'M', as the default units are KBits per second. This key must only occur once within this section.
b1='1K'	This key specifies the first bandwidth that should be displayed by the UI. The second bandwidth should have the key b2, the third b3 and so on.
b2='1M'	This key specifies the second bandwidth to be displayed.
[**VideoBandwidths**]	This section contains the bandwidths, which should be displayed by the CPN GUI when Video connections are selected.
Converter=1024	This key specifies the conversion factor required to change the selected bandwidth to the units required by the VASP. This key is only used by the GUI when the unit letter in the display string (see next key) contains 'M', as the default units are kilobits per second. This key must only occur once within this section.
b1='1M'	This key specifies the first bandwidth that should be displayed by the GUI. The second bandwidth should have the key b2, the third b3 and so on.

Running the CPN Server as a Windows NT Service

The CPN Server can be run under Windows NT as a service using the SRVANY.EXE software, which comes with the Windows NT Resource Kit. Follow the instructions in the file SRVANY.WRI to install SRVANY.EXE as a Windows NT service. When “Interact with Desktop” is selected, a command window is displayed by the system where output messages are displayed as if the CPN Server was run from a Command Prompt window. Also displayed is the Connection Status Window.

When specifying the parameters to the service in the registry the following are required:

Application value = <installed jdk path>\bin\java.exe

AppDirectory value = <CPN Server classes path>

AppParameters value = CPN

It is important to ensure that the CLASSPATH environment variable is set as discussed above at the System level not user login level.

Note: When running the CPN as a background service it is advisable to reduce the amount of information output by the CPN server to its log file by setting the loglevel key in the INI file to 0 (zero)

4.1.2.2 Starting the Application

On Solaris or HP OpenView platforms execute the runServer script file.

On Windows 95 or NT configure the CLASSPATH and PATH environment variables as described above, start a command prompt session and execute the command “java CPN”.

Alternatively, the CPN Server can be run as a Service under Windows NT, which has a web server installed and the SRVANY.EXE (from the Windows NT Resource Kit) program installed. As such the execution can be started automatically by the system or manually by the user through the control panel (see SRVANY.WRI for additional information).

4.1.2.3 Operation

The CPN Server runs inside a Java Virtual Machine and can therefore run on any platform which supports a suitable Java VM. For users on Solaris or HP OV platforms a script file, runServer, can be used to start the CPN Server using predefined hosts and ports which override those in the CPNServer.Ini file (see Environment Settings above). For users of Windows 95 or NT then the following command can be used providing the environment has been correctly set as previously discussed.

java CPN

This command would start the CPN Server using the host and port values defined in the CPNServer.Ini file. If the values from the INI file need to be overridden then the following command line can be used:

java CPN <listening port> <VASP host> <VASP port> <CMA host> <CMA port>

Where <listening port> is the port number on which the CPN Server will listen for connections from user applications. <VASP host> is the host name on which the VASP Association Server is running. <VASP port> is the port number on which the VASP Association Server is listening for CPN Server connections. <CMA host> is the host name on which the CMA Event Manager is running. <CMA port> is the port number on which the CMA Event Manager listens for connections.

Note: If any parameters need to be specified, all of the parameters must be present otherwise an error will be generated and the Server execution terminated.

4.1.3 VASP Customer Server

4.1.3.1 Environment Settings

CLASSPATH configuration

The Customer Server is a Java code program, and the environment variable CLASSPATH needs to lead to:

- The Vasp directory (which contains the customerServer, controlServer and vaspTypes directories),
- The virtual classes of the used CPN classes,
- The Objectspace Voyager 1.0 package,
- The Objectspace JGL 2.0.2 package,
- The Netscape LDAP SDK 1.0 package and filter package,
- IAIK-JCE 2.0 package,
- IAIK iSaSiLk 2.0a packages,
- The JDK 1.1.4 package.

Communication with the CMA Event Manager

The parameters of the communication can be configured using the following properties:

- *CUSTOMER_CLASS* which specifies the Managed Object Class of the Customer Server (default value =2),
- *CUSTOMER_ID* which specifies the Managed Object Identity of the Customer Server (default value = 5),
- *CMA_SERVER_HOST* which specifies the hostname on which the CMA Event Manager Server is running (default name = localhost),
- *CMA_SERVER_HOST* which specifies the port on which the CMA Event Manager Server is listening (default value = 50096).

Connection to the LDAP Directory Server

The parameters of the connection can be configured using the following properties:

? *LDAP_SERVER_HOST* which specifies the hostname on which the LDAP Directory Server is running (default name = localhost),

? *LDAP_SERVER_HOST* which specifies the port to connect to the LDAP Directory Server (default value = 1389).

4.1.3.2 Starting the Application

For users on Solaris or HP OpenView platforms, the runCustServer script file sets up the CLASSPATH and runs the application with specified values. An edition can be necessary to adapt it to the specific environment.

Otherwise the user has to configure the CLASSPATH environment variable, and to run the application with the following command :

Java -DLDAP_SERVER_HOST=*host name* -DLDAP_SERVER_PORT=*value* -DCMA_SERVER_HOST=*host name* -DCMA_SERVER_PORT=*value* -DCUSTOMER_CLASS=*value* -DCUSTOMER_ID=*value* AssociationServer

Note: the -D option permit to specify a Java property value. And the non-specification of one property lead to the use of its default value.

4.1.4 VASP Control Server

The following will be required in order to run the classes that comprise the ControlServer properly:

- JDK 1.1.3 or later installed and available.
- IONA OrbixWeb 2.1 RunRime

This module is started by running the *vaspVpnManager* class. This class's *main* method creates the one instance of the *vaspVpnManager* class. The main method expects one parameter which is the Id of the VASP used in communication with the PNOs. Moreover, the controlServer module uses a routing table containing static routing information. The routing information should be described in a file, the routing table, prior to starting up the VASP.

The full name of this table file, i.e., the full path-name plus the file name, is made known to the controlServer by means of Java's *property* mechanism. The property name chosen for the route table is **ROUTETABLE_PATH**. The value of this property is set by mean of the **-D** option of the **java** command. For example, assuming that the routing information is in the file *myRoutingTable* with a full path *myRoutingTablePath*, and that the Id of the VASP is *TRUMPETVasp*, one should start up the VASP in the following way:

java -DROUTETABLE_PATH=/myRoutingTablePath/myRoutingTable

controlServer.vaspVpnManager *TRUMPETVasp*

Note that if the ROUTETABLE_PATH is not set, no connection can be managed/set up.

The format of this file is given below in the form of an example that illustrates its content:

```
cpnId: NR
accesspoint: NRAtmSW1

pnoId: pnoNorway
accesspoint: nAccessAddress1
accesspoint: nAccessAddress2

pnoId: pnoSwiss
accesspoint: swAccessAddress1
accesspoint: swAccessAddress2

pnoId: pnoScotland
accesspoint: scAccessAddress1
accesspoint: scAccessAddress2

cpnId: EPFL
accesspoint: EPFLAtmSW5
```

The semantic of this table is that the access point of the first customer (NR) connects to the first access point of the following PNO (pnoNorway), and the second access point of that PNO connects to the first

access point of the next PNO (pnoSwiss) in the list, and so on. Finally, the second access point of the last PNO in the list (pnoScotland) connects to the access point of the other customer (EPFL).

Multiple route entries in the file are separated by a line having an "&" as its first character (the rest of the line is ignored). Matching is made on the basis of the Ids of the source and target end-users (cpnId). The first entry whose first (starting) cpnId matches the given source cpnId and whose last (terminating) cpnId matches the given target cpnId is returned.

In the controlServer directory, the shell script *runControlServer* is made to start running the controlServer module (vaspVpnManager class). This shell script in addition to what was explained above sets the java environment variable *CLASSPATH* and checks whether the RMI Registry and the Orbix demon on which the controlServer depends are running.

Note that in case that the VASP Id and/or the routeTable path are different from those specified in the *runControlServer* script, one must edit the script to put the correct values.

4.1.5 VASP CORBA/TMN Gateway

4.1.5.1 Environment Settings

The following environment variables are normally set in the script files provided with both the runtime and development packages.

`LD_LIBRARY_PATH`: this variable should contain the path to the SECUDE library as well as to the Orbix 2.2 C++ MT libraries.

- `TMS_CMA_HOME`: this variable should be set if the CMA management event reporting system is used. It should give the path to the configuration file `cma_tms.cfg`, which specifies CMA host and port number. Examples of entries in this file are:

```
CMA_ME_HOSTNAME          medoc.eurecom.fr
CMA_PORT_NUMBER          50096
CMA_MAX_PENDING          5
```

  Other variables in this file define the codes for object classes and instances in the specifications of events in the TRUMPET messaging system.

- `XUSER_NMS_COFIG_FILE`: this variable is used for TMN association between the gateway and the Xuser agents. It should give the complete path to the configuration file of the Xuser manager (recall that the gateway acts as an Xuser manager). In this configuration file, the AP Title and the Presentation Address for the manager entity should be defined, as well for each agent that it may establish an association with. In the presentation address, the last 12 digits represent the IP address of the host running the process (in the example below, IP address = 193.55.114.166).

```
MANAGER_AP_TITLE = C=fr; O=TRUMPET Project; OU=Ascom; CN=XuManager
MANAGER_PRESENTATION_ADDRESS = OVDM,ses0,tp0,0x540072872203193055114166


AGENT_NUMBER        = 1


AGENT0_NAME = pno1
AGENT0_AP_TITLE = C=fr; O=TRUMPET Project; OU=Ascom; CN=XuAgent1
AGENT0_PRESENTATION_ADDRESS = OVDM,ses0,tp0,0x540072872203193055114166
```

4.1.5.2 Starting the Application

The CORBA/TMN Gateway must be started only when the Xuser agents in the PNO domains are already running. This is because in the current implementation, the gateway (i.e. the Xuser manager) associates to

the agents when it starts, but does not attempt any further association while it is running. It should be started before the Control Server.

The following options are supported:

```
-d : level 1 debug output
-D : level 2 debug output
-S : enables use of security package (default OFF)
-s : specifies name of CORBA server
-C : enables emission of management events for CMA (default OFF)
```

4.1.5.3 Utilisation

Once the CORBA/TMN Gateway has been launched, its functioning is automatic with no possible human interaction. It simply translates CORBA requests received from the Control Server into CMIP requests for the Xuser agents, and translates CMIP notifications into CORBA notifications in the opposite direction.

From the trace output given by the process, the user can monitor the following operations:

- Association with the Xuser agents
- Connection establishment with the Control Server through CORBA
- Reception of CORBA requests and emission of CMIP requests
- Reception of CMIP notifications and emission of CORBA notifications

4.1.6 VASP CMA Management Event Reporting

The CMA based TMS (Trumpet Messaging System) is composed of three different entities:

- ME (Mediation Entity)
- MOSE (Managed Object Server Entity)
- AE (Application Entity)

The entities communicate through CORBA (Orbix 2.3) and use the naming service to connect to each other. There are two different AEs: a "text-mode" application that is here mainly for testing, and a graphical application in java. For a demo, the graphical java AE should be used.

The package for the complete CMA messaging system includes the following:

- Executable files for Solaris 2.5.x:
  - `TMS_ME`
  - `TMS_MOSE`
  - `TMS_AE_TEXT`
- Scripts:
  - `tms_me` : script for the ME
  - `tms_mose` : script for the MOSE
  - `tms_ae_java` : script for the java graphical AE
  - `tms_ae_text` : script for the text-mode AE
- Data files: the following files are used by the CMA entities to profile the object model and should be located in the ${TRUMPET_RT}/data/cma-v41 directory of the runtime distribution.
  - `hosts.pro`
  - `m3100.pro`
  - `tms.pro`

- Configuration
  - `cma.csh` : do a "source" of this file (located in the ${TRUMPET_RT}/data/cma-v41 directory) to have the naming service bin directory in your path, and ensure that the TMS_CMA_HOME variable is set
  - `cma_tms.cfg` : config file used by the ME

4.1.6.1 Environment Settings

Environment variables. The variables required for this application are set in the configuration file cma.csh: check this file and do a source of it to get a correct environment. In particular, the variable TMS_CMA_HOME should define the directory where the configuration file cma_tms.cfg can be found by the application.

Orbix daemon. The CMA entities use Orbix to communicate, so the Orbix daemon must run. The daemon should be started with the options -t and -u: the -t option is not strictly required but provides more trace output, the option -u is absolutely necessary for the naming service.

Orbix Naming service. Check that the name server has been registered: with "lsit", the server name "NS" should be listed, and you should have launch and invoke rights on it. If the name server is not registered, do:

```
putit -l NS <NAMING SEVICE BIN PATH>/ns
```

Java GUI Application. The IOR for the name service should be specified in the file OrbixWeb.properties in the java subdirectory. To get this IOR, use the program "iorns", which is located in the naming service bin directory. Check also the other variables set in this file:

`OrbixWeb.IT_LOCAL_DOMAIN`	local domain (e.g. fokus.gmd.de)
`OrbixWeb.IT_NAMES_SERVER`	name server : should be NS
`OrbixWeb.IT_NS_HOSTNAME`	name server host (can be IP address)
`OrbixWeb.IT_LOCAL_HOSTNAME`	local host name for the java AE
`OrbixWeb.IT_ORBIXD_PORT`	port number of orbix daemon

4.1.6.2 Starting the Application

The CMA system should normally be started before the Trumpet processes that send management events: CPN server, customer server, CORBA/TMN gateway. To start the CMA servers, simply use the provided scripts in the following order:

```
./tms_me
./tms_mose
./tms_ae_java
```

Use separate windows in order to monitor some trace output by each process, look in the scripts for the "-trace" option that produces the traces.

4.1.6.3 Utilisation

In order to get in the GUI panel the management events generated by the various processes of the TRUMPET system, the following sequence of actions should be observed, starting from the initial dialog box.

- subscribe to "trumpet"
- click event browser: this should create event record tree window
- double click top icon "event record" in event tree window this should create event panel window
- select "trumpet:cpnServer" in initial window
- click event filter: this should create event filter window

- select "PerceivedSeverity" in toggle menu: this should make the word "PerceivedSeverity" appear in the middle field
- Click "Add", which should add a filter line at the bottom of the window, and close the window.
- Repeat A, B, C, D for customer server and Xuser manager.
- At this point, events can be received and displayed.
- When events are received, they are automatically displayed in the event panels as new lines. When double-clicking on any line, you get more detailed information on the event.

As an alternative to the "one event panel does it all" scenario, it is also possible to create separate event panels for each managed entity: for example, select "trumpet:cpnServer" in the initial window, click on the browser button, and double-click on the top icon in the event record tree panel. This will create a event panel specific to the CPN server. The same applies to Customer server and Xuser manager.

In the event tree panel, if the icon "alarm records" is selected (in the subtree below the root "event records"), then a panel specific to the management events is created, that does not contain the object

creation notifications. Note that in this case, the panel will not be displayed until a first event is received (the application needs this first event to perform the dynammic configuration of the panel columns).

Gateway events. As a reminder: to enable the emission of managemenet events by the CORBA/TMN gateway, it should be started with the "C" option ("runGateway -C"). Otherwise, no events are generated.

Troubleshooting:

- Check Orbix version (orbixd -v): should be 2.3
- Check Orbix daemon port number (also with orbixd -v) it should match the one in OrbixWeb.properties
- Check Orbix daemon option -u (ps -cafe | grep orbix)
- Make sure that not more than 1 NS process is running
- Make sure that not more than 1 ME and MOSE process are running
- If problem with naming service, check the existing naming contexts (with command "lsns" in naming server path) and remove the tms context (with command: "rmns tms").

4.1.7 PNO Xuser-Agent

The Xuser-Agent software realises the service layer management system of a PNO which provides the Xuser interface to the VASP for the management of VP segment connections. As part of the TRUMPET run-time package distribution the software has been tailored to simplify the software installation according to the TRUMPET scenario involving two PNO domains.

4.1.7.1 Hardware and software prerequisites

The Xuser-Agent software as part of the current TRUMPET run-time package distribution has been built for the following environment:

- SUN/SPARC Solaris 2.5 or higher;
- HP OpenView DM version 5.01.

If the TRUMPET development package distribution is available and recompilation of the Xuser-Agent software is required the following tools are required additionally:

- GNU C, C++ compiler version 2.6 or higher;
- GNU make utility version 3.7 or higher.

4.1.7.2 Installation and configuration instructions

The PNO Xuser-Agent is installed using configuration files. The TRUMPET run-time package distribution contains two sets of configuration files for the two PNO domains. The only exception constitutes the local registration file "Xuser.lrf" which only exists once as it does not include system specific information. All the configuration files are located in the directory "~/Trumpet.RT/etc". To differentiate the sets of configurations files, the name of the PNO is included in the filenames. For the sake of generality the names PNO1 and PNO2 are used in the run-time package distribution which may be replaced by specific names. Note, however, that if the filenames are changed, the shell scripts "runAgent<PNO Name>" and "runSched<PNO Name>" need to modified (see below) to allow the Xuser-Agent software to access the configuration files.

For the development package distribution the configuration files are located in the directory "~/Trumpet.dev/src/xuser/etc". Note, that the naming convention are slightly different, as the name of the system host is added to filenames.

The Xuser-Agent configuration consists of the following files:

- Xuser.<PNO Nane>

  This files contains a set of variables to define the Xuser-Agent application entity title and addressing information. These variables need to be set if the Xuser-Agent shall use the explicit ACSE facility (default). Note, that explicit ACSE facility is required to activate TRUMPET security.

  Below the relevant variables which need to be set for the explicit ACSE facility according to system configuration are described below. There are more variables included which are solely used for internal or debugging purposes.

  AGENT_AP_TITLE= C = de; O = GMD GmbH; OU = FOKUS; CN = PNO1

  Defines the application title of the Xuser Agent application entity. The title can be given as an object identifier or distinguished name, however, if TRUMPET security is being used a distinguished name must be provided.

  AGENT_AE_QUALIFIER= 2

  Defines the application entity qualifier which is a numerical value. This value is not used if the application title is given as a distinguished name.

  AGENT_PRESENTATION_ADDRESS= OVDM,ses0,tp0,0x540072872203193175132212

  The Xuser-Agent PSAP address according to the RFC1006 specification. The last twelve digits contain the internet address of the system host where the Xuser-Agent is installed.

- sched. <PNO Nane>

  This file may contain internal data of the Xuser-Scheduler. There is no need to edit this file.

- XU_EDB.initial.<PNO Name> and XU_EDB.<PNO Name>

  These two files contain the persistent data of the Xuser-Agent management information base. The first file contains the initial database which is intended to serve as a backup copy so that the management information base can be set back easily to its initial status. The latter file contains the actual database file which may change during the operation of the Xuser-Agent. The contents of the two files are identically before the Xuser-Agent has been started for the first time. Usually there is no need to modify these files. However, if the name of the ServiceProvider Managed Object needs to be modified, edit both files and search for the value of the variable "SYS" in the first line. Then replace this value globally with the new name of the ServiceProvider Managed Object. Note, that it is essential to replace all occurrences of the name value in both files to keep the consistency of the management information base.

- Xuser.lrf

  If the explicit ACSE facility is not used the Xuser-Agent needs to be registered with Object Registration Service (ORS) of the HP OpenView DM run-time system. Note, that the explicit ACSE facility is required to use TRUMPET security.

To register the Xuser-Agent with the ORS the "ovaddobj" command is used: "ovaddobj Xuser.lrf"

4.1.7.3 Runtime

The Xuser-Agent software includes two programs: The Xuser-Agent and the Xuser-Scheduler. The agent handles the management requests while the scheduler is responsible for the timely activation and de-activation of scheduled virtual paths. Both programs can be run using shell scripts, which are located in the directory "Trumpet.RT/scripts" of the run-time package distribution. Within the development package distribution the shell scripts can be found in the directory "~/Trumpet.dev/src/xuser". Similar to the configuration files, each shell script exists twice according to the TRUMPET scenario involving two PNO domains, namely PNO1 and PNO2. Note, that the naming convention used with the development package distribution is slightly different, as the name of the system host is added to filenames.

To run the Xuser-Agent software the Xuser-Scheduler program needs to be started first: "./runSched<PNO Name>". Then start Xuser-Agent program : "./runAgent<PNO Name>". For the Xuser-Agent the following option apply:

- -i: Re-initialize the management information base.
- -A: Turn on explicit ACSE facility. Default, if -S option is given.
- -S: Turn on TRUMPET Security

4.1.8 PNO CMA-Based NMS

4.1.8.1 Environment Settings

The CMA based network management system for the FORE switch is composed of an API used by the Xuser agent, and two processes: the NMS_MOSE and the NMS_ME. The environment requirements are as follows:

- The CMA processes communicate through CORBA: the orbix daemon must be running (version Orbix 2.3), and the Orbix Service Naming should be installed.
- The NMS_ME uses the HP-OV APIs (HP OpenView DM 4.21 or higher) for SNMP management.
- Configuration files: the files m3100.pro, hosts.pro, xuser.pro, community.txt should be present in the ${TRUMPET_RT}/data/cma-v41 directory.

  The file hosts.pro describes the object instances in the CMA model: for the objects of class “ATM Equipment”, the IP address of the equipment (switch) should be given as “AgentId”. The corresponding community names for SNMP management should be given in the file community.txt. Other files need not be changed.
- Environment variable (they are normally set in the provided scripts):

NMS_CMA_HOME	directory ${TRUMPET_RT}/data/cma-v41
Xuser_Def_VPI	default VPI
Xuser_VPI_Base	Connections will be established with VPIs in
Xuser_VPI_Range	the interval [Base, Base + Range].
CMA_SNMP_COMMUNITY	community name for SNMP management

4.1.8.2 Starting the Applications

Use the scripts provided in the runtime distribution to start the NMS_ME and the NMS_MOSE, and then start the Xuser agent.

The ATM switches to manage should be known before starting the system: their IP addresses should be given in the configuration file hosts.pro and the community names in the file community.txt as described above. In the TRUMPET demo environment, there is only one FORE switch to manage per Xuser agent; in

this case, there is only one IP address to fill in the hosts.pro file, and the community name of the switch should be given in the community.txt file and as an environment variable.

4.2 Security Management

4.2.1 Security Admin Tool

This window allows to start the management session. The administrator can select the security service in the administration menu : Security Profile Management, Access Control or Audit. The Security Status menu displays the number of security alarms for each severity level. Two buttons allow to display more information about the alarms and management events.

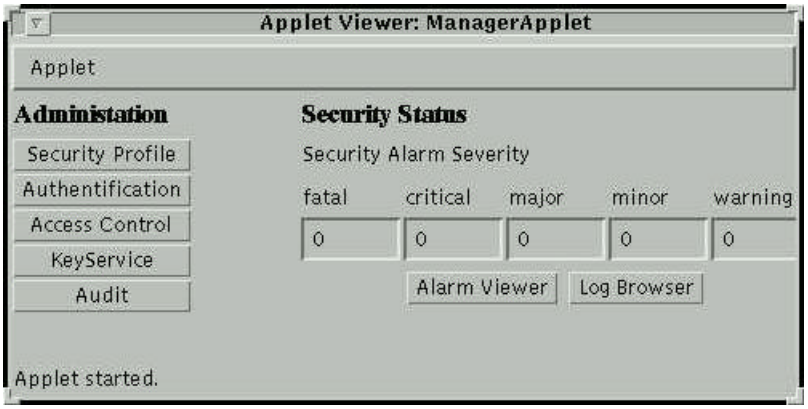

Figure 10: Main Window of the Security Admin. Tool

4.2.1.1 Management of the Security Profiles

4.2.1.1.1 What are Security Profiles?

Security profiles are consistent sets of security services and mechanisms specified to meet requirements for varying Quality of Protection (QoP).

According to the security requirements for the X interfaces of a service management system, three levels of QoP are defined by Trumpet : minimal, basic, and advanced. Additionally a nil-security profile is defined to be able to interact with existing system having no security implemented.

0. The nil-security profile comprises no security mechanisms. It can typically be used for temporarily removing security in interactions with a particular destination.

1. A minimal security profile, stressing the correctness of stored data and a minimal accountability for all management activities. It comprises the following security services : identification and authentication of the initiator (managing) entity, management association access control and access control to resources, and security audit trail and alarm.

2. A basic security profile, adding protection of transferred data against disclosure, modification or insertion. In addition to – or in replacement of – the security services of the minimal profile, strong authentication[1] of the initiator (managing) entity, data origin authentication, and transferred data integrity and confidentiality are present.

3. An advanced security profile, for areas such as accounting and security management, where accountability has to be stressed, confidentiality of transfers is needed, as well as high availability. In addition to, or replacement of, the services for the basic security profile, strong mutual authentication of the management entities (both manager and agent), management notification access control, connection integrity and confidentiality, non-repudiation of origin and non-repudiation of delivery are needed.

The protection provided by the security profiles defined by TRUMPET is summarised in the following table.

[1] Strong authentication refers to an authentication mechanism making use of cryptographic means as opposed to simple or protected passwords.

Table 2. Trumpet Security Profiles

SP0	SP1	SP2	SP3
No security	Emphasis on the integrity and confidentiality of stored managed resources	**SP1** plus integrity of transferred data	**SP2** plus strong accountability of management operations
	• Authentication of initiating management entity • Management association access control • Managed resource access control • Security alarm, audit and recovery	• Authentication of initiating management entity • Management association access control • Management notification access control • Managed resource access control • Data origin authentication • Connection integrity • Security alarm, audit and recovery	• Mutual authentication of peer management entity • Management association access control • Management notification access control • Managed resource access control • Data origin authentication • Connection integrity • Non-repudiation of origin • Non-repudiation of delivery • Security alarm, audit and recovery
		SP2A	**SP3A**
		SP2 plus confidentiality of selected communicated data	**SP3** plus connection confidentiality and confidentiality of selected communicated data

The Advanced Security Profile cannot be implemented in a satisfying way on the commercial management platforms, such as HP OpenView DM .

In TRUMPET, SP0 and SP2A are implemented for validation in the TRUMPET trials.

4.2.1.1.2 How to Select a Security Profile?

The entity to which a security policy applies is a couple of two Management Application Entities (MAE), known by their Distinguished Name (DN), located in two different TMN domains. This allows the security administrators to select the most appropriate profile depending on such parameters as the level of trust between the two domains and the sensitivity of the application. The Distinguished Names of the corresponding applications should be exchanged between the security administrators of the two management systems by out-of-band means prior to configuring the security profile.

Security Policy rules are set up as a table with entries

(initiatorTitle, initiatorRole, responderTitle, accessControlDirectory, securityProfile).

A java GUI allows to fill each of these entries :

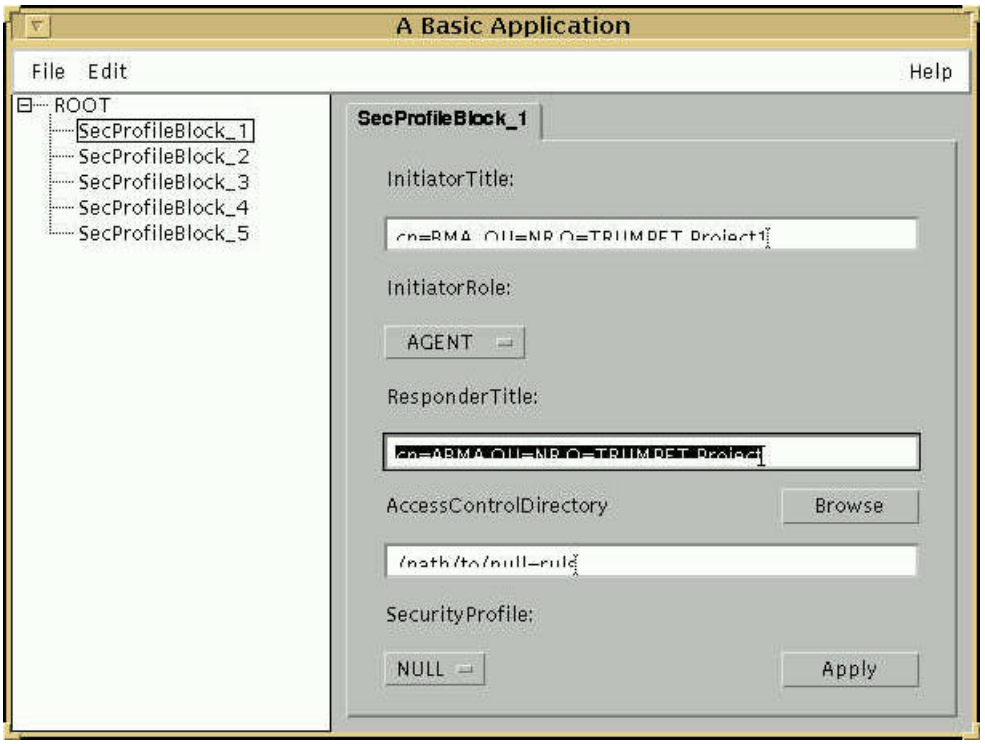

Figure 11: Window of the Security Profile Manager

The table is expected to be found on a file which name is retrieved from the environment variable SECURITY_PROFILES. In case this variable is not set, a file "*secprofs.txt*" is looked for, and in case this file neither exists, no security profiles will be set up.

The file must have entries like this :

```
iT: cn=BMA,OU=NR,O=TRUMPET Project
iR: [AGENT|MANAGER]
rT: cn=ABMA,OU=NR,O=TRUMPET Project
aC: /path/to/access/control/directory
sP: [NULL|MIN|BASIC|ADV]
```

The meaning of the *sP* values are :

NULL:	SP0
MIN:	SP1
BASIC:	SP2
ADV:	SP3

Table entries must be separated with lines starting with a space (or just a new line). If multiple (iT, iR, rT) tuples are present in the file, the last one is used. In case of a bad entry specification, the entry is skipped.

A sample *secprofs.txt* file :


```
iT: cn=BMA, OU=NR,O=TRUMPET Project
iR: AGENT
rT: cn=ABMA,OU=NR,O=TRUMPET Project
aC: /path/to/null-rules
sP: NULL

iT: cn=mv,OU=UCL,O=TRUMPET Project
iR: MANAGER
rT: cn=amv,OU=UCL,O=TRUMPET Project
aC: /path/to/min-rules
sP: MIN

iT: cn=BMA,OU=NR,O=TRUMPET Project
iR: AGENT
rT: cn=mv,OU=UCL,O=TRUMPET Project
aC: /path/to/basic-rules
sP: BASIC

iT: cn=BMA,OU=NR,O=TRUMPET Project
iR: AGENT
rT: cn=m.v,OU=UCL,O=TRUMPET Project
aC: /path/to/adv-rules
```

sP: ADV

4.2.1.2 Audit and Alarm Administration Tool

EFD Management

The window allows the management of the EFD. A scrolled list allows the selection of a host. The create and delete buttons allow the creation on the selected host of an EFD and the deletion on the selected host of the selected EFD. A list displays all the EFD that have been created for a selected host.

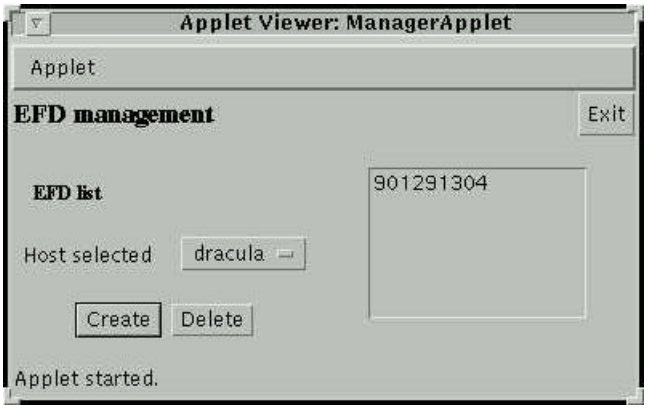

Figure 12: Dialog Box "EFD Management"

Alarm Viewer

This window shows the collected events. For the alarms the severity is displayed and for the non security events, the severity is empty.

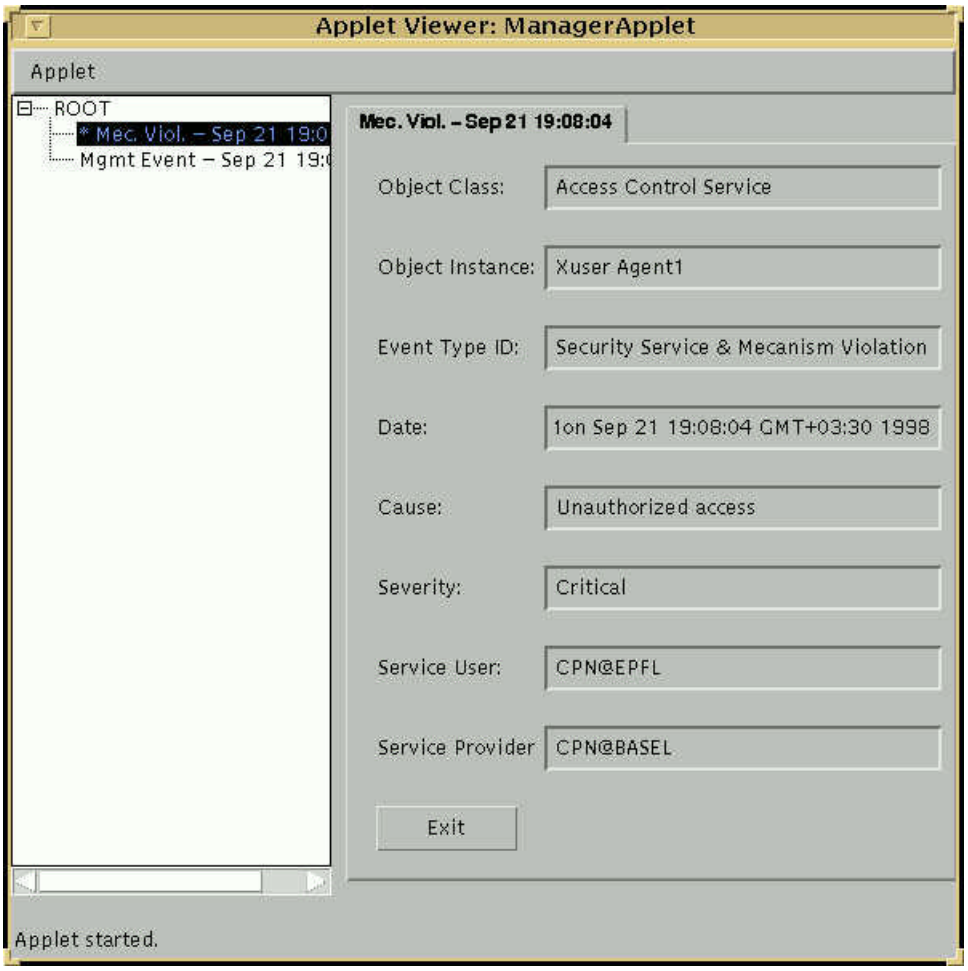

Figure 13: Window of the Alarm Viewer

Log Management

This window enables the display of the logged events with the Edit button and the deletion of the log with the button Delete.

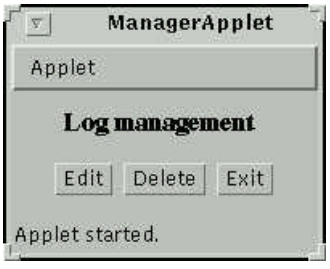

Figure 14: Dialog Box "Log Management"

4.2.2 Management of the Security Profiles

4.2.2.1 What are Security Profiles?

Security profiles are consistent sets of security services and mechanisms specified to meet requirements for varying Quality of Protection (QoP).

According to the security requirements for the X interfaces of a service management system, three levels of QoP are defined by Trumpet : minimal, basic, and advanced. Additionally a nil-security profile is defined to be able to interact with existing system having no security implemented.

0. The nil-security profile comprises no security mechanisms. It can typically be used for temporarily removing security in interactions with a particular destination.

1. A minimal security profile, stressing the correctness of stored data and a minimal accountability for all management activities. It comprises the following security services : identification and authentication of the initiator (managing) entity, management association access control and access control to resources, and security audit trail and alarm.

2. A basic security profile, adding protection of transferred data against disclosure, modification or insertion. In addition to – or in replacement of – the security services of the minimal profile, strong authentication[2] of the initiator (managing) entity, data origin authentication, and transferred data integrity and confidentiality are present.

3. An advanced security profile, for areas such as accounting and security management, where accountability has to be stressed, confidentiality of transfers is needed, as well as high availability. In addition to, or replacement of, the services for the basic security profile, strong mutual authentication of the management entities (both manager and agent), management notification access control, connection integrity and confidentiality, non-repudiation of origin and non-repudiation of delivery are needed.

The protection provided by the security profiles defined by TRUMPET is summarised in the following table.

Table 3. Trumpet Security Profiles

SP0	SP1	SP2	SP3
No security	Emphasis on the integrity and confidentiality of stored managed resources	**SP1** plus integrity of transferred data	**SP2** plus strong accountability of management operations
	• Authentication of initiating management entity • Management association access control • Managed resource access control • Security alarm, audit and recovery	• Authentication of initiating management entity • Management association access control • Management notification access control • Managed resource access control • Data origin authentication • Connection integrity • Security alarm, audit and recovery	• Mutual authentication of peer management entity • Management association access control • Management notification access control • Managed resource access control • Data origin authentication • Connection integrity • Non-repudiation of origin • Non-repudiation of delivery • Security alarm, audit and recovery
		SP2A	**SP3A**
		SP2 plus confidentiality of selected communicated data	**SP3** plus connection confidentiality and confidentiality of selected communicated data

The Advanced Security Profile cannot be implemented in a satisfying way on the commercial management platforms, such as HP OpenView DM .

[2] Strong authentication refers to an authentication mechanism making use of cryptographic means as opposed to simple or protected passwords.

In TRUMPET, SP0 and SP2A are implemented for validation in the TRUMPET trials.

4.2.2.2 How to Select a Security Profile?

The entity to which a security policy applies is a couple of two Management Application Entities (MAE), known by their Distinguished Name (DN), located in two different TMN domains. This allows the security administrators to select the most appropriate profile depending on such parameters as the level of trust between the two domains and the sensitivity of the application. The Distinguished Names of the corresponding applications should be exchanged between the security administrators of the two management systems by out-of-band means prior to configuring the security profile.

Security Policy rules are set up as a table with entries

(initiatorTitle, initiatorRole, responderTitle, accessControlDirectory, securityProfile).

The table is expected to be found on a file which name is retrieved from the environment variable SECURITY_PROFILES. In case this variable is not set, a file "*secprofs.txt*" is looked for, and in case this file neither exists, no security profiles will be set up.

The file must have entries like this :

```
iT: cn=BMA,OU=NR,O=TRUMPET Project
iR: [AGENT|MANAGER]
rT: cn=ABMA,OU=NR,O=TRUMPET Project
aC: /path/to/access/control/directory
sP: [NULL|MIN|BASIC|ADV]
```

The meaning of the *sP* values are :

NULL:	SP0
MIN:	SP1
BASIC:	SP2
ADV:	SP3

Table entries must be separated with lines starting with a space (or just a new line). If multiple (iT, iR, rT) tuples are present in the file, the last one is used. In case of a bad entry specification, the entry is skipped.

A sample *secprofs.txt* file :


```
iT: cn=BMA, OU=NR,O=TRUMPET Project
iR: AGENT
rT: cn=ABMA,OU=NR,O=TRUMPET Project
aC: /path/to/null-rules
sP: NULL

iT: cn=mv,OU=UCL,O=TRUMPET Project
iR: MANAGER
rT: cn=amv,OU=UCL,O=TRUMPET Project
aC: /path/to/min-rules
sP: MIN

iT: cn=BMA,OU=NR,O=TRUMPET Project
iR: AGENT
rT: cn=mv,OU=UCL,O=TRUMPET Project
aC: /path/to/basic-rules
sP: BASIC

iT: cn=BMA,OU=NR,O=TRUMPET Project
iR: AGENT
rT: cn=m.v,OU=UCL,O=TRUMPET Project
aC: /path/to/adv-rules
sP: ADV
```

4.2.3 Management of the Authentication Keys

4.2.3.1 What are Authentication Keys?

There are two types of keys used in the TRUMPET SMP : asymmetric and symmetric keys. Asymmetric keys are used for :

- peer-entity authentication based on public key algorithms;
- management of symmetric keys;
- non-repudiation.

Symmetric keys (the term secret key is also used) are used for data integrity, data authentication, and data confidentiality.

In TRUMPET, the symmetric keys are session keys whose life duration is limited to a management association; they are managed (i.e. generated) by the SMP without any involvement of the security administrator; hence this section is only related to the management of asymmetric keys for authentication.

Keys for security algorithms have to be generated, distributed, stored, updated, withdrawn, destroyed, and possibly notarised at various stages in their existence. Key management includes all these operations, and has as its central purposes :

- ensure that keys are of sufficient quality;
- ensure that private keys are kept confidential;
- ensure that public keys are certified and distributed in a reliable manner.

Asymmetric keys in TRUMPET (private / public key pairs) are generated, installed, certified, and withdrawn by off-line means, and the keys remain valid for a long period of time. On-line management of asymmetric keys, i.e. activities that have to be performed in the course of establishment or use of a secure management association, are restricted to :

- access to private keys;
- access to public keys and key certificates.

Each entity in TRUMPET that needs to authenticate or otherwise use public key encryption, must have a private key that is accessible only to the entity itself, and a public key that is distributed in the form of an X.509 certificate issued by a Certification Authority (CA). The certificate binds the public key to the identity of the entity, given as an X.500 Distinguished Name (DN), as specified by the security policies.

Note that for TRUMPET, the owner of the keys are the MAEs as opposed to human operators. Human operators may also have asymmetric keys for authentication towards the management system, but this is related to intra-domain security and is out of the scope of TRUMPET.

4.2.3.1.1 Storage of Private Keys

In the TRUMPET experiments, the private keys are stored on disc, in encrypted form. It must be decrypted and transferred to memory before it can be used. For TRUMPET, this decryption / transfer will take place when the MAE is started. The private key will be kept in memory by the SSO object, indexed by the MAE identity.

4.2.3.1.2 Storage/Caching of Public Keys and Certificates

The public keys of local entities (MAEs) are stored in the same way as their private keys. Storage needs not be in the form of certificates, since the local storage in any case has to be trusted for the private keys. The public key is indexed by the MAE identity, and the key is available to the SSO.

The public key of the root-CA can be stored in the same way as the public keys of local entities. Alternatively, this key can be stored in the form of a self-signed certificate issued by the root-CA. In any case the key must be available to the SSO object.

The public key of a peer entity is accessed by obtaining the X.509 certificate of the entity. This is the task of the Certificate Handling object. A certificate may be fetched from the issuer CA (or from other external sources) by use of the LDAP protocol. There is no need to secure this operation, as it is a pure read operation of signed information (certificate). Depending on local policy anything from only the peer's certificate to all certificates in the forward certification path may be fetched. If a certificate is fetched from the issuer CA, it can always be assumed to be valid, since a CA should never give away a certificate that has been revoked.

Certificates may be cached locally according to local policy, and kept in the cache as long as permitted by local policy. If a locally cached certificate is used, a check against CRLs should be done. This implies that CRLs must be fetched regularly from the relevant CAs. Note that CRLs are also signed by the CA. Each time a certificate is fetched from a CA, it is also stored in the cache, unless this feature is explicitly disabled. Since inter-domain operation using a hierarchically structured certification system may involve retrieval of a lot of certificates belonging to intermediate CAs, it is especially important to cache the certificates of CAs.

Caching of certificates and CRLs opens an important issue : availability of the system versus security. Availability in this context means not only that the information must be accessible (which without caching must rely on on-line services provided by CAs), but also that the real-time requirements of the management applications are met. On the other hand use of cached information implies trust that this information is correct. The simple caching procedure outlined above may be enhanced by :

- Read-forward : each time a cached certificate (or CRL) is used, the version number is verified with the CA. If it matches, the cached information is used, else the cache is discarded and the certificate (or CRL) is fetched from the CA. This is particularly useful for large amounts of information, like a CRL or a complete forward certification path, and probably meaningless for one certificate. This policy has the benefit of being initiated by the user.
- Write-back : each time a CA modifies revokes a certificate, it sends a warning to its clients. This policy has the benefit of being efficient in terms of speed, but it depends on the CA being aware of who its clients are and how they can be reached. Also, a breach in the connectivity between a user and the CA will pass unnoticed.

The standard way of storing certificates is by use of an X.500 directory. This is not used by TRUMPET because maintenance of the information in the directory is considered too complex. Use of the LDAP protocol enables any kind of certificate storage to be used by a CA, even an X.500 directory.

4.2.3.2 Obtaining Certificates for your Management System

For your management system MAEs to be authenticated by the remote management systems, they must have public-private key pairs; in the TRUMPET architecture, the public key must be certified by a Certification Authority (see Section 4.2.5.1). This sub-section describes how to have public keys certified and other related management operations.

Below there is one section for each SECUDE command needed to manage the public keys of a management system.

If SECUDE is not started, start it by typing[3] :

```
Secude
```

4.2.3.2.1 Create a User PSE

For each MAE, a PSE must be created. A user PSE is created by :

```
secude> psecrt -v -k 1024 [-p <pse_name>] “cn=MAE,ou=NR,o=TRUMPET
Project”
```

You find a suitable DN for your purpose and replace the one above in the command. When there is only one MAE to be certified the –p option can be omitted : as no -p option is given, the PSE will be put on `$HOME/pse` (the default PSE).

http://www.darmstadt.gmd.de/secude/Doc/htm/utilusgr.htm describes this command in more detail.

4.2.3.2.2 Make a Certification Request

Having created a user PSE, you will need to make a certification request for it. You do this by doing :

```
secude> pem CERT-REQ [-p <pse_name>] -o cert.pto
```

The certificate request will be put on the file *cert.pto*. You then should send the *cert.pto* file by mail to the certifying CA (i.e. to trumpet@nr.no).

http://www.darmstadt.gmd.de/secude/Doc/htm/utilpegr.htm gives the parameter description to PEM.

4.2.3.2.3 Install / Caching a Certificate

A certificate issued as the result of a certification request, is accepted by the requester by doing :

```
secude> pem -v -I [-p <pse_name>] -i cert.issued -u yes
```

The file *cert.issued* contains the issued certificate. The -I option tells that scanned issuer certificates shall be stored as trusted keys in the local `PKList`. The `PKList` contains public keys to be trusted in the validation of signatures and message encryption.

PEM will detect which certificate you are trying to install, and ask you if you want to do this.

To cache a certificate you can also do a

[3] When SECUDE is run on Solaris 2.5, the following export commands should be done in an appropriate shell prior to issuing SECUDE commands :

```
export HOME=/path/to/secude/pses/Home
export SECUDE_ETC=/path/to/secude/etc
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/path/to/secude/lib
export PATH=/path/to/secude/bin:$PATH
```



```
secude> psemaint addpk <certificate>
```

see http://www.darmstadt.gmd.de/secude/Doc/htm/utilpegr.htm

4.2.3.2.4 Show Trusted Certificates

To investigate your user PSE, do a :

```
secude> psemaint show [ PKList | PKRoot | FCPath | Cert]
```

If you type `PKList`, your set of trusted certificates will be displayed. `PKRoot` will show the certificate belonging to the root in your PKI, `FCPath` will show your Forward Certification Path and `Cert` will show your own certificate.

4.2.3.2.5 Uninstalling / decaching a certificate

To delete a public key from your PKList do a

```
secude> psemaint delpk <issuer> <serial number>
```

or

```
secude> psemaint delpk <owner>
```

see http://www.darmstadt.gmd.de/secude/Doc/htm/utilpegr.htm

4.2.3.2.6 Creating the Credentials

The credentials file *cred* is the file where the SMP reads the PSE associated with each MAE distinguished name. To make the credentials of all the secured MAE known to the SMP:

```
secude> seclogin –p <pse_name1>
secude> seclogin –p <pse_name2>
…
```

Check the contents of the credentials file *cred* with :

```
secude> seclogin –l

CN=MAE_name1, OU=Sema, o=TRUMPET Project
            /usr/home/~/name1.pse
CN=MAE_name2, OU=Sema, o=TRUMPET Project
            /usr/home/~/name2.pse
…
```

4.2.3.2.7 User Commands

User commands for signing / encrypting / checking of messages are not discussed in detail, as these operations are performed by the SSO via the GSS-API in the TRUMPET SMP. But the commands are described in the standard help system. Here are a few commands just to test your installation :

```
secude> pem MIC-CLEAR -i <inp.file> -o <outp.file>
secude> pem ENCRYPTED -i <inp.file> -o <outp.file>
secude> pem -i <inp.file> -o <outp.file>
```

The first two commands will sign and encrypt, respectively, the contents of a file *inp.file* and then leave the output on the file *outp.file*. The third command will take a signed / encrypted file *inp.file* as input and then

leave the decrypted version on a file *outp.file*. You can use these three commands to sign / encrypt some text, and then check the signature, or decrypt the text to see if the original is restored. You may also alter the signed / encrypted messages with a text editor and then try to check the signature or decrypt afterwards to see what happens.

4.2.3.3 Certificates Retrieval Management

SECUDE 5.1 provides a function to retrieve certificates from an LDAP server. For your management system to be able to authenticate remote management systems, it must be configured in order to retrieve the public key certificates of those systems.

4.2.3.3.1 Accessing the TRUMPET LDAP Server

You will need to set two environment variables in order to enable SECUDE to access the TRUMPET LDAP server:

```
export SECUDE_LDAP_LIB=/path/to/LDAP/lib/libldap.so
export SECUDE_X500=aldebaran.nr.no:1521
```

Note: if aldebaran is not known to your DNS use:

```
SECUDE_X500="156.116.2.200:1521"; export SECUDE_X500
```

4.2.3.3.2 Querying the Server

Everybody could try to retrieve certificates from the server and thereby check the installation, by doing :

```
secude> psemaint -D
PSE pse> retrieve “ou=NR,o=TRUMPET Project”
```

and ask for the CA certificate.

4.2.4 Access Control

4.2.4.1 Access Control Principles

The TRUMPET access control architecture is based on [ITU-T X.812] which specifies an access control framework, and [ITU-T X.741] which specifies a model for controlling access to management information and operations. The access control profile AOM24322 (Access Control List (ACL) with Item Rules) [ISP 12060-9] will be used in TRUMPET.

4.2.4.2 Access Control Architecture

The basic entities and functions involved in access control are the initiator, the target, the Access Control Enforcement function (AEF) and the Access Control Decision function (ADF). The access request represents the operation and operands that form part of an attempted access. The AEF ensures that only allowable accesses, as determined by the ADF, are performed by the initiator on the target.

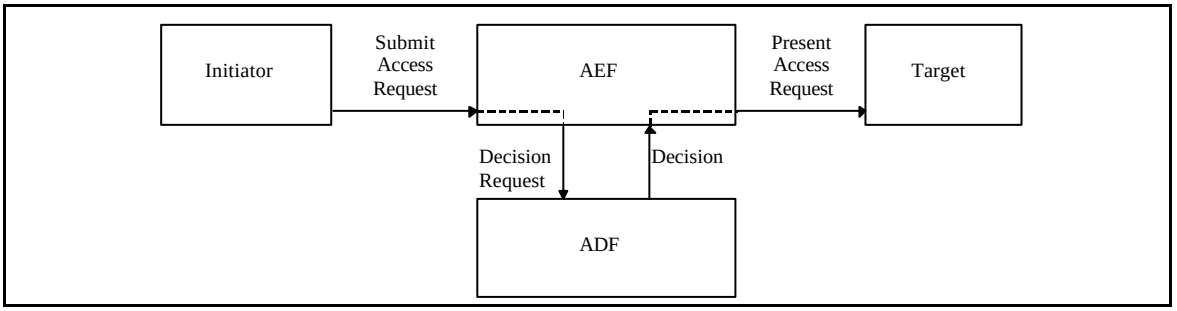

Figure 15: Generic Access Control Architecture

In TRUMPET access control is applied to management associations and management operations. Figure 16 shows the global access control architecture. The black boxes refer to locations where the access control mechanisms will be built into the system. Use of Privilege Attribute Certificates (PAC), where a privilege attribute server (PAS) in the initiator domain (or perhaps even an external TTP) signs a certificate for the access rights granted to the requesting entity, is for further study in TRUMPET.

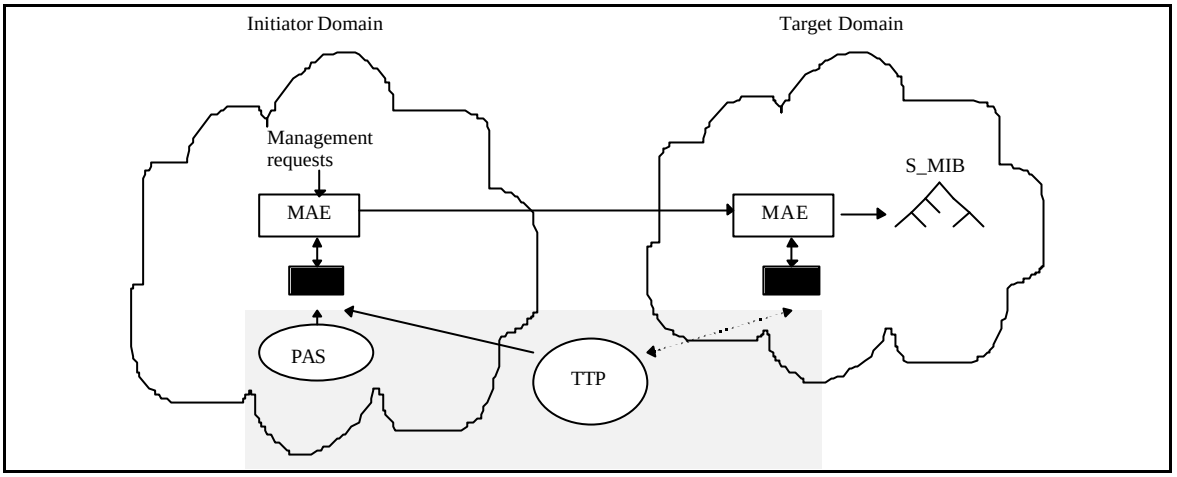

Figure 16: TRUMPET Access Control Architecture

In the initiator domain, access control is applied to outgoing management association requests. In the target domain, access control is applied to incoming management association requests and incoming management operation requests.

4.2.4.2.1 Access Control Information Model

The Access Decision Function requires information to decide whether an access request should be granted or denied. The types of access control information used are:

- the identity of the initiating management application entity (MAE) of the access request (Initiator ACI)
- the management information identities to which access has been requested (Target ACI)
- and access control rules which represent the access control policy to be applied.

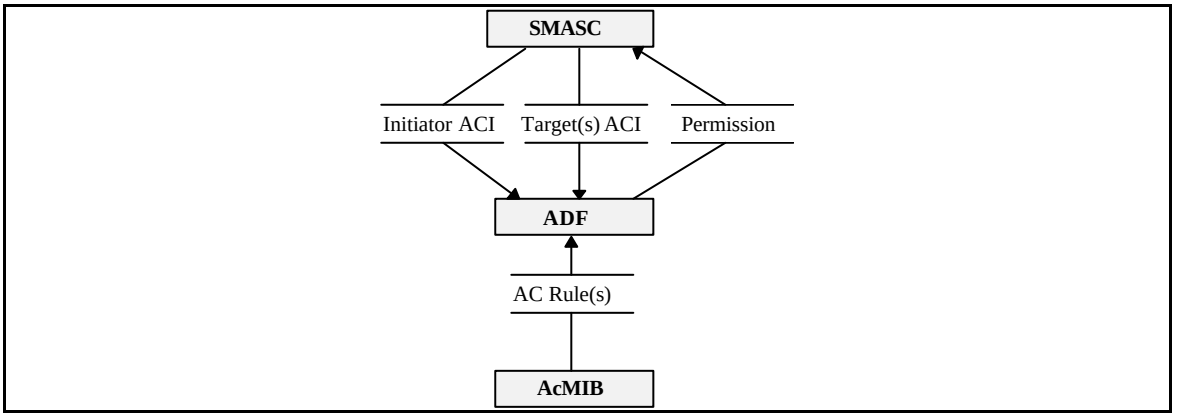

Figure 17: Access Control Information - Data Flow

Initiator ACI

The initiator ACI consists of the authenticated identity of the initiating MAE (AE Title). In TRUMPET authentication and access identity will be the same.

Target ACI

The target ACI identifies a management information entity, either a MAE or MO instance, access has been requested to.

Access Control Rules

The access control rules represent the permitted operations and the conditions upon their execution in an access control domain. There are five classifications of access control rules which are to be applied by the Access Decision Function (ADF) in the following order [ITU-T X.741]:

Global deny rules; access control rules that deny access to all targets in that domain with respect to particular initiators.

Global deny rules will be used to realise access control on management associations. If a global rule denies access for the initiator the association establishment will be denied.

Item deny rules; access control rules that deny access to specific targets.

Global grant rules; access control rules that grant access to all targets in that domain with respect to particular initiators.

Item grant rules; access control rules that grant access to specific targets.

Default rules; Default rules are used for making access decisions when no global or item rules apply.

Component	Type	Comment
Initiators	InitiatorGroup	1-*
Targets	TargetGroup	0-* (if not present => Global Rule)
Operations	Enum(Operation)	0-5
Permission	Enum(Permission)	1

Table 4: Elements of access control rule

4.2.4.2.2 Association Access Control

Access control is performed as an integral part of the establishment of a secure management association. It will be enforced in the initiator and the target domain to ensure that the initiator MAE is authorised to establish a management association to the specific target MAE.

It is assumed that the initiating entity has an identity assigned by the initiating system during the login procedure. Authentication and access identity are the same in the TRUMPET security package. Access is denied, if the initiator is not allowed to do anything on the target system.

4.2.4.2.3 Operation Access Control

The ADF bases its decision on the requested management operation and identification of the target objects together with the retained ACI, the access control rules and the contextual information. It uses the access control rules to determine whether the initiator may access a particular target in the inter-domain MIB. To do this, the <Initiator, Target> pair is mapped to the access rights (of the Initiator) in the access control rules according to the above priority. If access to the target is granted, the MAE may apply the management operation to the target.

4.2.4.3 Management of Access Control Information

4.2.4.3.1 Access Control Administration Tool

The Access Control Administration Tool provides a graphical user interface to create, change and display the access control information. It provides comfortable means to maintain the Access Control MIB which is used by the access control decision function. The GUI is structured in five tab-panels which are mapped onto their corresponding files:

- File Panel
- Group Target Panel
- Group Initiator Panel

- Rule Panel and
- Default-Rule Panel

The user will be able to create a new domain but he can not delete a domain throughout the GUI. The latter needs to be done on the operating system layer. The remainder of this section describes the various GUI commands.

File Panel Window

This panel allows specify the address of the administration server host and the name of the Access Control MIB. First, the user needs to enter the address of the server host in the text field “Server Address”. After pressing the Return button or the button “Contact” the connection will be created.

The text field “Directory Location” is used to enter the pathname of the directory where the Access Control MIB is located or going to be created. Following this, three actions are available to the user: "New" creates a new MIB, "Open" opens an existing MIB, and "Save" stores current changes to the selected location.

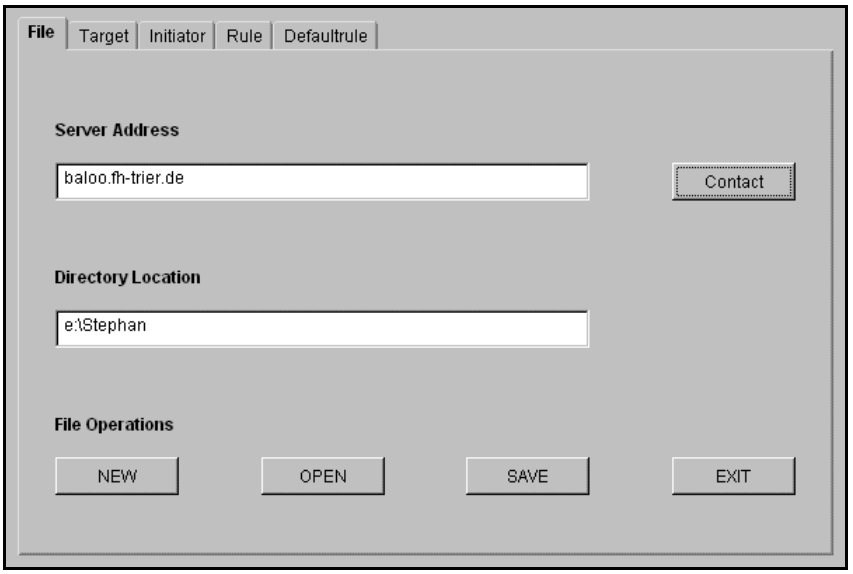

Figure 18: File Panel

Target Panel Window

The user can manage the target - file with this window. By clicking on “<NEW>” in the group target list a new group target is created. A dialog will appear where the user can edit the identity of a group target. Click on the entries in the group target list, but not the “<NEW>, for details in the target list. If a entry in the group target list is double-clicked the identity of a group target is changeable. One click on the “<NEW>” causes the “New Target Dialog” to appear, if before a group target was selected. With this dialog the user will be able to create a new target with identity, level and scope. The new generated target belongs to the chosen group target. If the user double-clicks on an entry of the target list the chosen target attributes can be changed.

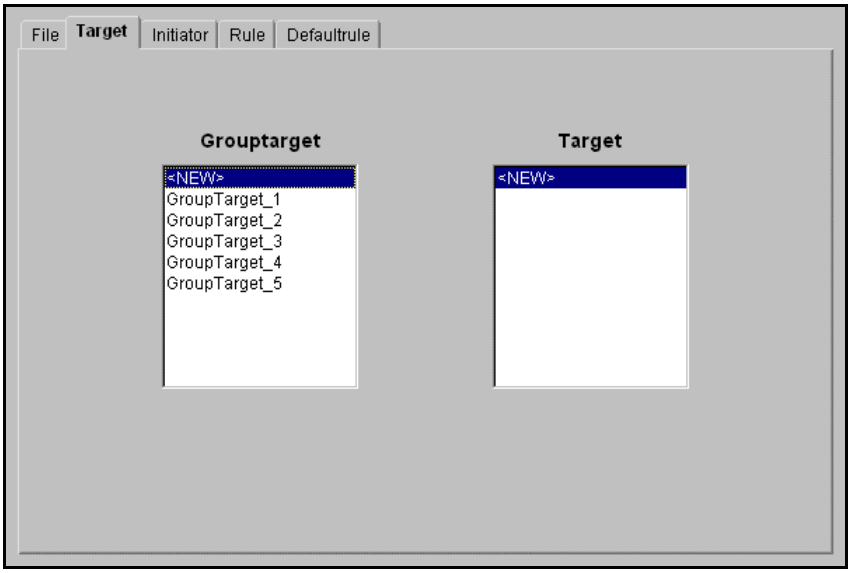

Figure 19: Target Panel

Initiator Panel Window

With this window the user can make changes in the initiator file. The functionality is the same like the first one mentioned above. The only difference is that the initiator do not have either a level or a scope.

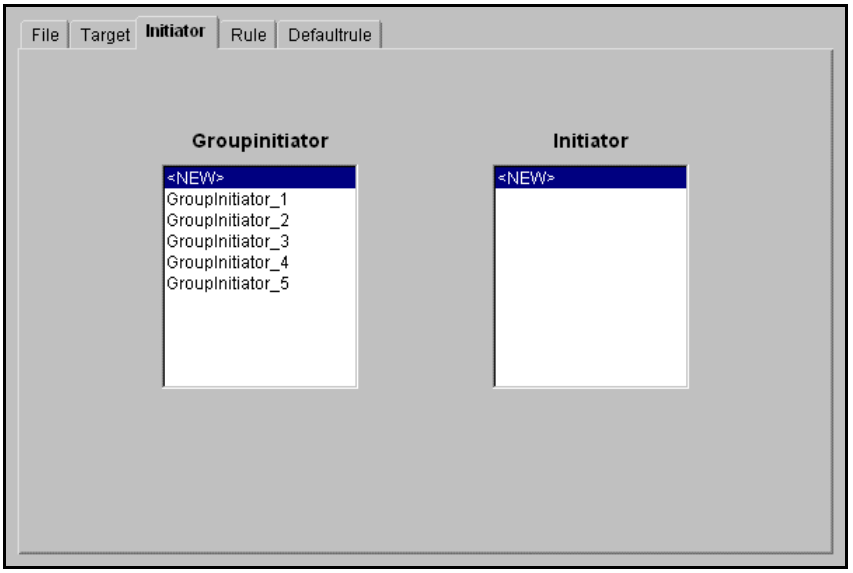

Figure 20: Initiator Panel

Rule Panel Window

The rule file was changed and was created by the rule panel window. In the first list called “Rule ID:” the user can edit the name of a rule. In the other lists it is possible to add or remove operations, group initiators and group targets belonging to the chosen rule. For adding an item the user should click on the “<ADD>” entry and for deleting he has to double-click on the item which should be removed. In the permission part the user is able to select only one of the possibilities as stated in Figure 3.

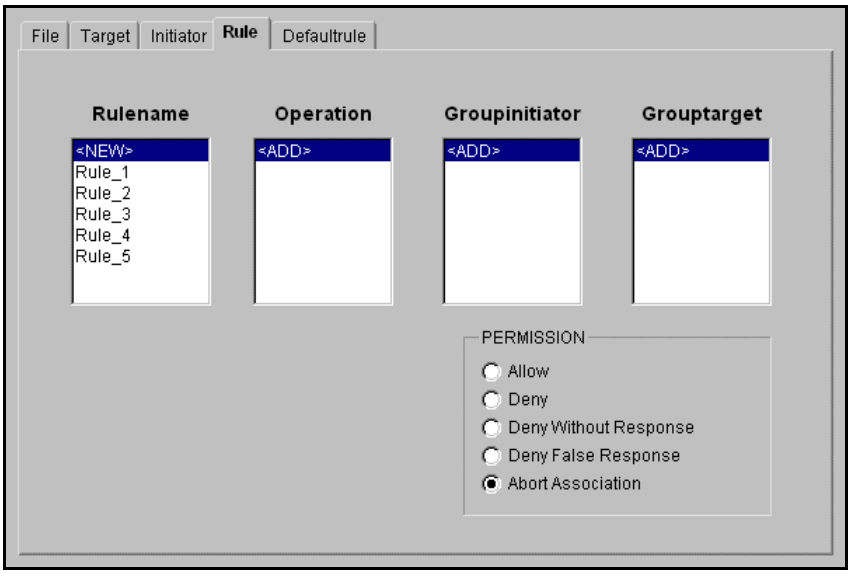

Figure 21: Rule Panel

Default Rule Window

Here the user can change the default rule file. For every action like “GET” for instance there are five possibilities available.

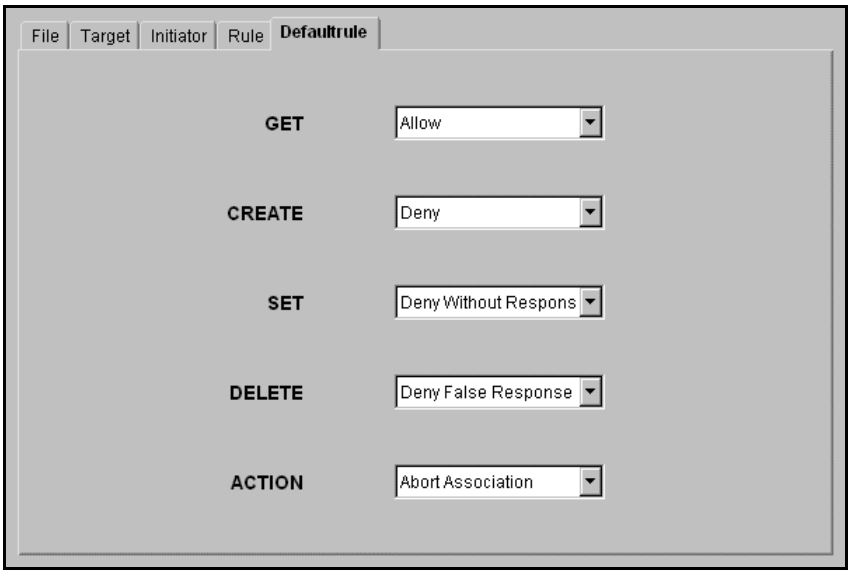

Figure 22: Default Rule Panel

4.2.4.3.2 Access Control Configuration Files

The access control domains used in a management application are configured by ASCII files contained in UNIX directories. These directories are specified in the security profile for each <initiator,target> pair. The directory must contain the following files (filenames are fixed) :

- *initiator.MIB* contains a set of initiator groups which can be referred by rules.

  An initiator group definition consists of a name followed by at least one initiator name (string DN). Note, that each element has to be placed on a separate line. A definition will be closed by a line starting with the character '*#*'.

  Example :

```
; Example Domain - Initiators (Managament Application entities)

EPFL
    cn=MAE1, ou=EPFL, o=TRUMPET Project
    cn=MAE2, ou=EPFL, o=TRUMPET Project
#
GMD
    cn=MAE1, ou=GMD, o=TRUMPET Project
    cn=MAE2, ou=GMD, o=TRUMPET Project
#
NR
    cn=MAE1, ou=NR, o=TRUMPET Project
    cn=MAE2, ou=NR, o=TRUMPET Project
#
Ascom
    cn=MAE1, ou=Ascom, o=TRUMPET Project
    cn=MAE2, ou=Ascom, o=TRUMPET Project
#
```

- *target.MIB* contains a set of target groups which can be referred by rules.

  An target group definition consists of a name followed by at least one target. Note, that each element has to be placed on a separate line.

  A target is defined by the name of a base object instance (string DN) optional followed by scope in brackets. A scope is specified according to this:

  [*] whole sub-tree

  [-n] base to nth Level

  [n] individual Level n

  A definition will be closed by a line starting with the character '*#*'.

  Example :

```
; Example Domain - Targets

; the user subtree
User876394426
    user=876394426, systemId=xuser_SL_GMD_de[*]
#
```

- *rule.MIB* describes the rules of the access control domain

  A rule definition consists of a name followed by at least one initiator group reference, a set of target group references and an access right specification. The different parts are separated by a line starting with the character '-' (minus).

  The access rights are defined starting with the permission followed by an optional list of operations. If no operation is specified the rule is valid for all operations.

  A definition will be closed by a line starting with the character '*#*'.

  Example :

```
; Example Domain - Access Control Rules

AssociationDenied_EPFL
    EPFL
    -
    -
    deny
#

AllAllowed_GMD
    GMD
    -
    -
    allow
#

GetAllowed_Ascom
    Ascom
    -
    -
    allow get
#

UserSubtreeAllowed_NR
    NR
    -
    User876439426
    -
    allow
#
```

- *ruleDefault.MIB* contains the default rule of the access control domain

  A default rule is specified by a line with one permission for each operation separated by ':'. The sequence of operations is :

  get, set, create, delete, action

  Example :

```
; Default permission for
; get:set:create:delete:action
deny:deny:deny:deny:deny
```


4.2.5 Operation of a Certification Authority

4.2.5.1 Certification Hierarchy

TRUMPET makes use of the certification hierarchy presented in Figure 23. The Root CA is operated by NR and corresponds to an Inter-Domain Management CA. In other words TRUMPET operates its own PKI, the Root CA in Figure 23 is the root of a private certification hierarchy.

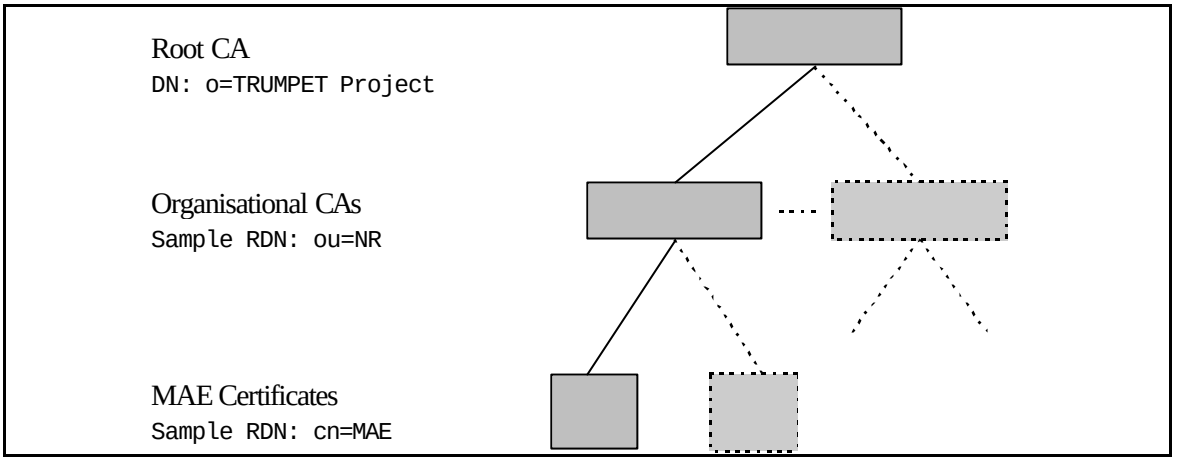

Figure 23: Certification Hierarchy

The Organisational CAs in Figure 23 represent the CAs of the participating providers. Each provider running a TMN system that participates in inter-domain management shall operate a CA. These CAs issue certificates for application entities, but it is a local matter for a particular organisational CA to decide if it will have subordinate CAs to issue the user certificates.

Organisations wanting a certificate for an MAE in TRUMPET can just send a certification request for such an MAE according to Section 4.2.3.2 and ask NR to create and operate a CA for them. However, organisations insisting on administrating a CA on their own, can do so and the instructions for how to do this is given below. Organisations having NR operate their CA do not have to care about the contents of the rest of this chapter.

4.2.5.2 Having your Organisational CA Certified by the Root CA

Before doing the operations described below you will need to set the environment variables as specified in Section 4.2.3.2.

4.2.5.2.1 Creating your Organisational CA

A CA is created by:

```
secude> cacrt -v -c new.ca -k 1024
```

You will be prompted for a PIN code to be used in future access to the CA. You will also be prompted for the DN of the CA (unless you supply this as the last parameter to *cacrt*, which you may do). A key pair will be generated for you and the CA and it's PSE will be put on a directory named `$HOME/new.ca`.

http://www.darmstadt.gmd.de/secude/Doc/htm/utilcagr.htm gives the details of this command.

4.2.5.2.2 Make a Certification Request for your Organisational CA

Having created an organisational CA, you will need to make a certification request for it. You do this by doing :

```
secude> pem CERT-REQ -c new.ca -o cert.pto
```

The certificate request will be put on the file *cert.pto*. You then should send the *cert.pto* file by mail to the certifying CA (i.e. to trumpet@nr.no).

4.2.5.2.3 Install a Certificate of Your Organisational CA

A certificate issued as the result of a certification request, is accepted by the requester by doing :

```
secude> pem -v -I -c new.ca -i cert.issued -u yes
```

The file *cert.issued* contains the issued certificate. The -I option tells that scanned issuer certificates shall be stored as trusted keys in the local PKList. The PKList contains public keys to be trusted in the validation of signatures and message encryption.

PEM will detect which certificate you are trying to install, and ask you if you want to do this.

4.2.5.3 Running your Certification Authority

4.2.5.3.1 Issue a Certificate using Your CA

A certifying CA will issue the following command in order to answer to a certification request :

```
secude> certify -v -l 990630215959Z -c certifyingCA.ca -C TRUE cert.pto
cert.issued
```

PEM will show the prototype certificate and ask if you would like to sign it. The -l option will ensure that the issued certificate is valid until end of June 1999. No certificate renewal actions should thereby be necessary for TRUMPET partners during the project lifetime. The issued certificate will be put on the file *cert.issued*. The -C TRUE option tells that the certified entity will be able to operate as a CA. If this shall not be the case, then `-C FALSE` should be used instead.

4.2.5.3.2 Issue of Certificate Revocation Lists

To issue a Certificate Revocation List you may first identify the certificate(s) to revoke. You then revoke the certificate(s) by doing :

```
secude> psemaint -c new.ca
PSE new.ca> revoke
```

if *new.ca* has issued the certificate. You will be asked for the serial number of the certificate to revoke. If you do know the serial number, the

```
PSE new.ca> causers
PSE new.ca> caserialnumbers <DN>
```

commands may be helpful. You will be asked if you would like to sign the CRL, and you should do so.

```
secude> pem CRL -c new.ca -o pemCRLs.txt
```

The above command will put the CRL in a PEM message to be distributed. For a user to install a CRL in the local PSE, a simple PEM SCAN will do :

```
secude> pem -i pemCRLs.txt
```

The file pemCRLs.txt contains the actual CRL.

The :

```
PSE new.ca> caprolong
```

will prolong the validity of an existing CRL, in case there are no new certificates to revoke.

4.2.6 Running the Certificate Directory Server

In TRUMPET there is one LDAP server, located at and maintained by NR (at least until some organisation insist on operating their own CA). Access rights are set up such that only retrieval is permitted from outside, and certificate retrieval can be done as explained in Section 4.2.3.3.

Sections 4.2.6.1 and 4.2.6.2 gives the directory structure. Section 4.2.6.3 is for the time being for NR only.

4.2.6.1 Directory Structure

The directory structure should resemble the certification hierarchy. Figure 24 shows the structure that is used in TRUMPET. The dotted objects are objects needed for each organisation being supported. An application certificate is stored in an *applicationEntity* belonging to the organisation owning the application.

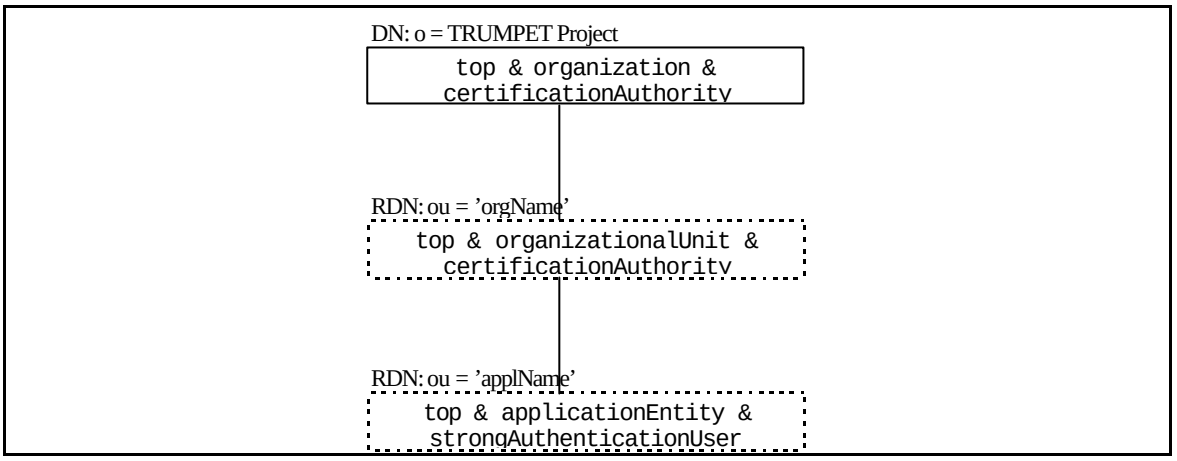

Figure 24: Certificates Directory Structure

4.2.6.2 LDIF Description of the Server

An LDIF description for the TRUMPET server will look something like this (the rootdn from *etc/slapd.conf* must be present in the directory structure):

```
dn: o=TRUMPET Project
objectClass: top
objectClass: organisation
objectClass: certificationAuthority
o: TRUMPET Project
cACertificate:
pemCRL:

dn: ou=NR,o=TRUMPET Project
objectClass: top
objectClass: organisationalUnit
objectClass: certificationAuthority
cn: NR
cACertificate:
pemCRL:

dn: cn=MAE,ou=NR,o=TRUMPET Project
objectClass: top
objectClass: applicationEntity
objectClass: strongAuthenticationUser
cn: MAE
userCertificate:
```

The last entry is for administrative purposes. New *organizationalUnit* nodes and subordinates will be added according to how NR is inserted above.

4.2.6.3 Updating the Server with new Certificates and CRLs

This is done by using the commands below:

```
secude> psemaint -c my.ca -D
PSE my.ca> caucert2dir
PSE my.ca> enter [cACertificate | revocationList]
```

`caucert2dir` is used when the CA wants to insert an issued user certificate, `enter` is used to insert its own CA certificate or CRL.

4.2.7 Security Support Object

The SSO is a set of API to provide security services. It interfaces with an commercial dynamic library (SECUDE) which has to be in the "dynamic library path".

Secude is a security product that provides a generic (GSS) security interface and also a toolkit to manage keys and certificates. These both aspects are closely linked, and the result of the GSS API hardly depends on the key management.

For a MAE, the use of the SSO object needs at least to first create a User PSE and Credentials (see Management of authentication keys chapter).

4.2.8 Secure Management Association

The Secure Management Association is that component which provides security to the MAE by the co-ordination of the trumpet security objects (Access Control, Security Profiles and SSO) and by the generation of security events to the SELF.

Communication with the SELF

The parameters of the communication can be configured using the environment variables:

- *XUSER_MANAGER_ID* which specifies the Managed Object Instance for a manager MAE (default value = 3),
- *XUSER_AGENT_ID* which specifies the Managed Object Instance for an agent MAE (default value = 4),
- *SELF_HOST* which specifies the hostname on which the SELF is running (default name = localhost),
- *SELF_PORT* which specifies the port on which the SELF is listening (default value = 50095).

Co-ordination of the security objects

In the D8 document, four Security Profiles have been defined, with their security rules. The co-ordination of the security object directly stems from this definition.

Initially designed to run test, the possibility to re-define the security rules for each profile has been added. This option is use when the environment variable RULES_PATH is found and when this variable leads to a file named "rulesProfiles.txt". Otherwise the security rules used for the four profiles are those defined in the D8 document.

Configuration of the security rules for each profile

In the file rulesProfiles.txt, the configuration of each profile is made on three criterions:

- The type of authentication (nickname = AUTH),
- The type of control on the accesses (nickname = AC),
- The type of security to apply on the exchanges (nickname = SEAL).

Note: the generation of security events is always active.

The configuration of the profiles is made in the order: NULLP/MIN/BASIC/ADV.

The rules definition has to respect the format: AUTH=*value* AC=*value* SEAL=*value*. Where the values correspond to:

- AUTH :
  - 0 = simple authentication,
  - 1 = secured unilateral authentication (only the initiator authenticates itself),
  - 2 = secured mutual authentication.
- AC :
  - 0 = no Access Control neither on associations nor on managed object operations,
  - 1 = Access Control on associations,
  - 2 = Access Control on associations and on managed object operations.
- SEAL :
  - 0 = no security applied on exchanges,
  - 1 = integrity applied: signature of the outgoing messages, verification of the incoming messages,
  - 2 = confidentiality applied: encryption of exchanges.

Note: the lines that don't respect the format "AUTH=*value* AC=*value* SEAL=*value*" are considered as comment.

For example the default Trumpet rules would be defined this way:

```
####################
# The default rules #
####################
AUTH=0 AC=0 SEAL=0
AUTH=0 AC=1 SEAL=0
AUTH=1 AC=2 SEAL=1
AUTH=2 AC=2 SEAL=2
```

4.2.9 Adapter Object

The Adapter object is a set of API to secure the XMP function, it modifies the MAE's exchanges to insert security.

This API doesn't need a specific configuration effort.

4.2.10 Audit and Alarm

4.2.10.1 Overview of Audit and Alarm Management

Security audit and alarm management is dealing with :

- auditing, i.e. collecting information, about the usage of security services or mechanisms, and more generally sensitive resources protected using the security services. The information collection is based on the events generated by the security mechanisms and forwarded towards management applications and recorded into logs. The collected information is analysed in order :
  - to verify the effectiveness of a given security policy and of its implementation
  - to find out whether resources are misused - and if they are, who performed the misuse and how it occurred.
- recovering from abnormal situations, e.g. when auditing concludes to a security breach, by asking for changes in the security policy, restricting the access to some service, etc.

Audit and recovery also includes a management facet dealing with administering the logs and the forwarding of events, selecting the events that will be considered as being relevant, etc.

In this schema, the audit analysers (either off-line or real-time) detect security problems and generate security events. These events are forwarded and caught by the security alarm processing application, handled and filtered-out by the security alarm processing application which might trigger recovery actions by providing the security recovery management with the necessary information.

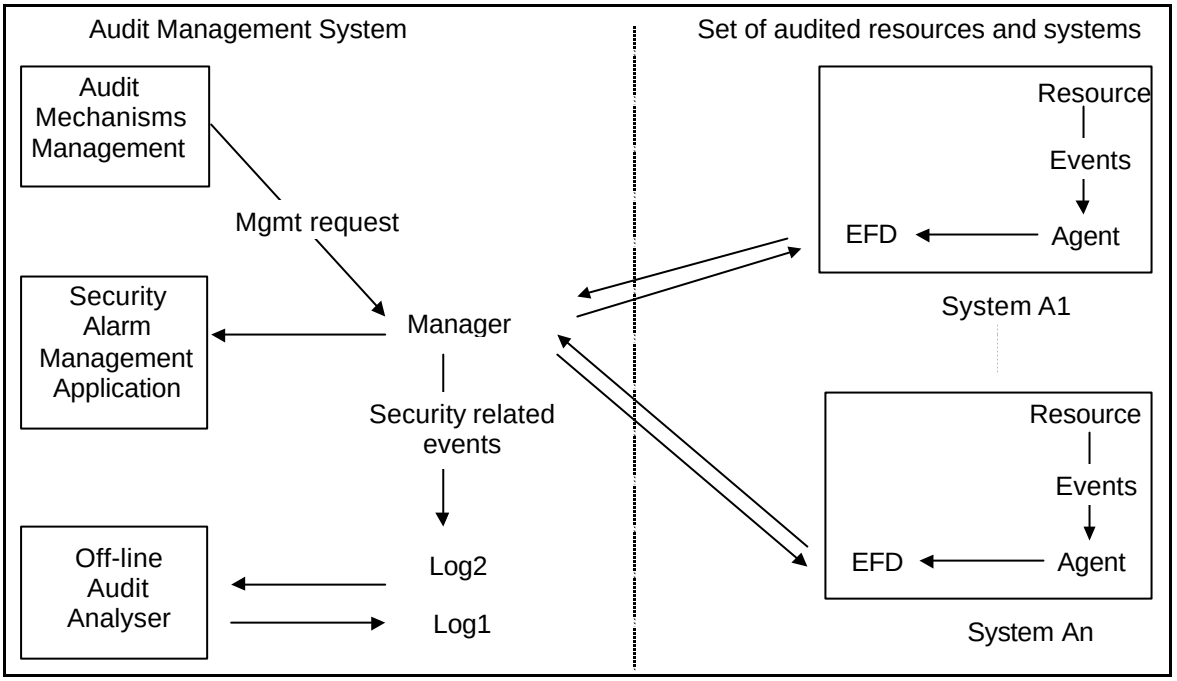

Figure 25 : Auditing on Systems Providing no Logging Capabilities

4.2.10.1.1 Security Event Forwarding and Logging

The security mechanisms behave as event generators, i.e. sending security relevant events using the M-EVENT-REPORT service. These events are forwarded towards e.g. the security alarm management application and / or the log management application – more generally those of the applications grouped under the Audit and Alarm Management "hood", i.e. :

- the audit mechanism management which also provides for backing-up audit trails which are close to overflow;
- both off-line and real-time event analysers are providing for detecting security related problems within the audited (sub)domain. When such a problem is found or suspected, these applications are generating security alarms that can be handled by the security alarm management application;
- the security alarm management application purpose is to raise a flag when security problems have been encountered. Such kinds of situations require careful analysis, probably performed by a human being who either decides to :
  - ignore the alarm, since no further action is required;
  - triggers some recovery action.

The event pre-processing function translates a local notification into potential event report. The potential event reports are distributed to all EFDs. The event forwarding discriminator is used to determine which event reports are to be forwarded to a particular destination during specified time periods.

The figure below summarises the management interactions implied by security event reporting.

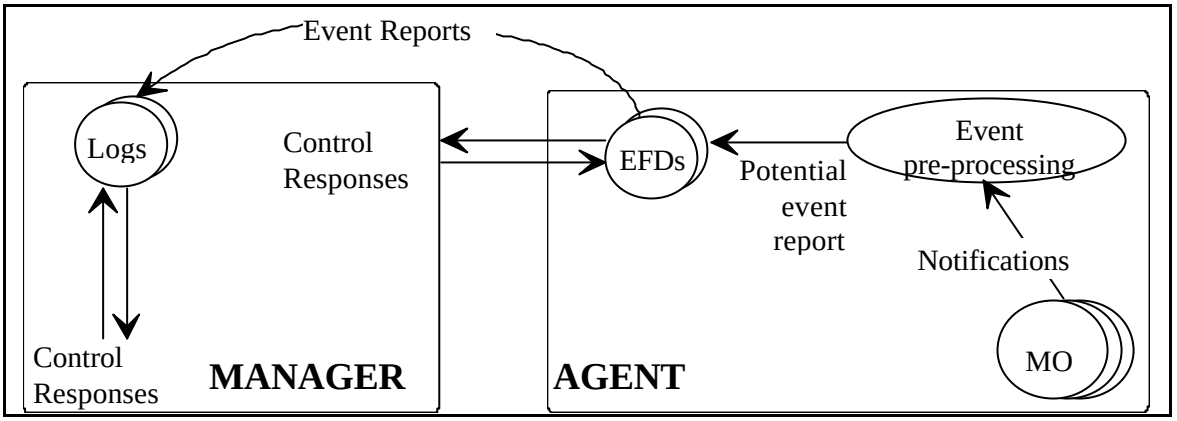

Figure 26 : Interactions for Security Event Reporting

The managed objects notify security related events. Only relevant events are selected by the event forwarding discriminator and reported to the manager. The manager can modify event selection conditions according to the security policy. The collected events, especially alarms, can be reported in real time, or logged into security audit trails for further analysis. The events to be logged are selected according to the criteria defined in the log object.

The Event Report Management Function provides services by which event reports can be distributed. Event report distribution means the selection of events to be reported to some designed system, or process, within some selected time period. These selections are done by a filtering process using the "Event Forwarding Discriminator" managed object. Event Forwarding Control is the ability to initiate, terminate, suspend, or resume event reporting through the manipulation of Event Forwarding Discriminator managed objects.

4.2.10.1.2 Management of Event Forwarding and Logging Mechanisms

For security audit management purpose, management services are required to :

- create, modify and delete any objects or attributes of managed objects which specify the selection criteria for security relevant events, i.e. event forwarding discriminator and audit trail;
- initiate and terminate the generation of security audit messages;
- initiate and terminate the generation of security audit records;
- initiate and terminate the generation of security alarms.

4.2.10.1.2.1 Audit Objects to be Managed

The objects that have to be created and managed are security audit trails and event forwarding discriminators (EFDs). Security audit trails are used for recording security relevant events generated within the local environment and EFDs purpose is the forwarding of events for analysis and / or recording purpose, e.g. when no local log facility is available.

Security audit trails are defined as logs of which records correspond to security relevant events. The logs properties provide for :

- selection of events that are to be logged by a management system in a particular log;
- an external system to modify the criteria used in logging events;
- an external system to determine whether the logging criteria were modified or whether log records have been lost;
- control over time during which logging occurs by suspending, resuming and modifying the logging scheduling;
- retrieving and deleting log records;

- an external system to create and delete logs.

Logs (audit trails) store security relevant events. The audit trail pre-processing function receives notifications from managed objects (object identified within a given policy as being security relevant) within the local system and forms potential audit trail records. These audit trail records are (conceptually) distributed to all logs contained within the local open system - these records are perceived as a discriminator input object for the purpose of discrimination by the audit trail. The characteristics the input object must satisfy to be logged are specified by the audit trail discriminator construct. When selected for logging, the information is supplemented with additional information generated as part of the logging process (record identifier, logging time).

Event forwarding discriminators are managed objects allowing the selection of event reports to be sent to particular, specified, managing systems. They are also providing for :

- control over the forwarding of events by suspending, resuming and modifying the scheduling of their forwarding;
- the ability to modify the conditions used in the reporting of events;
- the ability to specify back-up locations when no primary location is available.

4.2.10.1.2.2 Audit Mechanism Configuration

Defining an audit configuration for a system or a security service consists in stating which means have to be used and how these are configured, which are the objects to be created and what the values of these attributes are. Once defined, the application will provide the end-user with the ability to name and save the defined configuration.

Setting up an audit then consists in :

- determining those of the managed objects and the events they are generating which have to be audited;
- selecting the trail(s) in which the selected events will be recorded.

Selecting objects and events subject to auditing thus involves the setting of audit trails and/or EFDs discriminator constructs. In effect, the discriminator construct is a filtering mechanism which acts on attributes of the discriminator or log input objects. The discriminator construct is a set of assertions about the presence or values of attributes, such assertions can be grouped together using logical operators; it thus provides the audit administrator with means for indicating the system which events must be either recorded or forwarded (e.g. by subject or object identity).

In order to be able to configure other systems or security services based on the re-use of a defined configuration, most of the provided information can be re-used - the value of the discriminating construct excepted. This type of value has to be adapted to each system or security service especially when the discrimination is based on the managed object class and/or managed object instance. These values have to be changed according to the system and/or security service to be audited this can either be done automatically (if rules for writing discriminator constructs were defined) or the end-user can be requested by the application to update the discriminating constructs according to the system and / or security service.

Because of the relationship between audit and recovery through :

- their complementary goals,
- the generation of, and the reaction to, security alarms,
- the common means (EFDs) used for forwarding relevant events towards remote audit trails and security alarms administrators,

and the similarities of the tasks to be performed, it makes sense to configure security audit and detection of simple security alarms at the same time. "Simple alarm detection" is used here in contrast to real-time intrusion detection and refers to events generated by managed objects of which the types are referenced as security alarms, i.e. integrity violation, operational violation, physical violation, security service or mechanism violation and time domain violation.

4.2.10.1.2.3 Management of Security Event Forwarding

The event report management is realised with :

- the definition of flexible event report control service which allows systems to select which event reports are to be sent to particular managing systems;
- the specification of the destinations (e.g. the identities of managing systems) to which event reports are to be sent;
- the specification of a mechanism to control the forwarding of event reports, for example by suspending and resuming their forwarding;
- the ability for an external managing system to modify the conditions used in the reporting of events;
- the ability to designate a backup location to which event reports can be sent if the primary location is not available.

4.2.10.1.3 Security Alarm Management

4.2.10.1.3.1 Alarm Management Configuration

The mechanisms needed for security alarms management can be classified into three functions : alarms collection, alarms examination and alarms processing :

- Alarms collection deals with reporting and logging the alarms, and uses resources similar to those used for auditing purpose, namely EFDs and logs; therefore the related section will focus on the specific aspects of the management of those resources;
- Configuration of alarms examination includes such activities as configuring alarm display;
- Alarms handling requires to configure the actions to be taken on receipt of an alarm :
  - ignore,
  - submit to the security alarm administrator,
  - (forward to the security recovery component for immediate triggering of a recovery mechanism.)

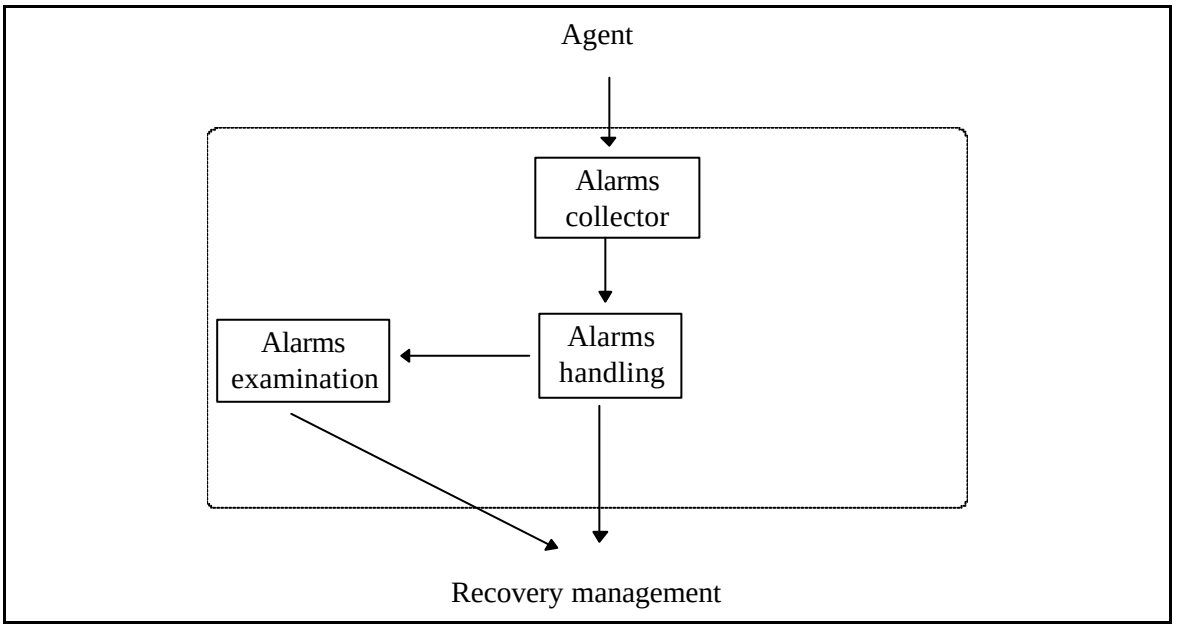

Figure 27 : Overview of Alarm Management

4.2.10.1.3.1.1 Alarm Collection Configuration

Although the nature of the security alarms requires that they must normally be reported, the capability to select those of the alarms which will be forwarded within the managed systems will be provided to the security alarm administrator. This capability would turn out to be useful in such a situation as repetitive alarm generation due to maintenance works, or in order to dispatch security alarms to different alarm collectors according to their types, causes, etc. The latter point deserves to be developed, since the EFDs

are the means to physically reflect the splitting of a system into several domains in the case a managed system contains objects or mechanisms belonging to distinct domains or sub-domains.

In order to avoid having alarms mixed with other events types, EFDs dedicated to security alarms forwarding are to be used within the agents. The responsibility for configuring those EFDs is given to the security alarms administrator. The configuration application provides the administrator with the capabilities to specify EFDs and discriminator constructs with a similar interface to that used for audit messages forwarding; the interface is specialised to fit the attributes which are specific to alarm notifications.

Because there is no strong need for splitting the alarms filtering function between the local EFD filter and the log filter located at the alarms collector application level, configuration of security alarms logs will implicitly use a pass-through filter. Thus the log mechanism is only used to register an alarm notification as a *securityAlarmReportRecord*. Therefore the specification of security alarms logs will present a simplified but similar aspect as that of security audit trail logs, where only the attributes *administrativeState*, *logFullAction*, *maxLogSize* and *capacityAlarmThreshold* will have to be specified using the same techniques as for audit trails.

4.2.10.1.3.1.2 Configuration of Security Alarms Examination

Since alarms displaying will reflect the possible subdivision of the domain into several domain portions, the configuration of this management function is used as the means to define that subdivision. Two different techniques for grouping the objects are provided to the administrator : topological grouping of systems and functional grouping of alarms. The first technique permits the administrator to display the alarms sorted by their origin, the second one, sorted by their nature.

Topological grouping is achieved by determining sorting criteria on the distinguished name of object instances.

Functional grouping is described by selecting a combination of :

- the potential alarm detector :
  - security mechanisms classes or sets of mechanisms classes which generate security alarms, for instances access control mechanisms to enable a global watching of unauthorised access attempts, or audit mechanisms to have a quick view on error processing alarms ;
  - classes of objects capable of generating security alarms ;
  - application entity, this selection based on the alarm detector identity provide for the ability to separate alarms generated by audit trail analysers;
- the event type;
- the alarm cause.

4.2.10.1.3.1.3 Configuration of Security Alarms Handling

Depending on various criteria to be specified by the alarms handling configuration function, uprising alarms are submitted to various kinds of treatment or a combination of treatment, namely :

- immediate forwarding of the alarm to the recovery processing when an immediate reaction is of importance (a common example is the detection of a threshold of unsuccessful authentication attempts on a same account which would require – if the authentication policy states it – to break the communication where those attempts occur);
- transfer to the alarms displaying function for acknowledgement and, possibly, recovery decision;
- ignore the alarm.

Combination of two treatments is allowed to the alarm administrator ; let us take the above example of a threshold of unsuccessful authentication attempts : in addition to the immediate recovery decision to break the communication, this alarm should be submitted to the alarm administrator in order to take a mean term recovery action such as locking the threaten account.

4.2.10.1.3.2 Security Alarms Processing

A first aspect of Security Alarm Management is the off-line analysis of the alarms having been generated. For this purpose the mechanisms and solutions defined for the off-line security auditing will be re-used, i.e. logging capabilities and audit trail analysis.

A second aspect is the detection and the handling of the security alarms that occurred in the considered audit-domain and to display them to the security alarm administrator who is thus able to react accordingly, i.e. by taking into account the information provided by the alarm.

4.2.10.1.3.2.1 Displaying Security Alarms

After collection and logging an alarm has passed through the alarms handling function which might have "decided" to submit it to the administrator for examination. The alarms examination function first requires the alarm to be sorted according to the criteria defined for the subdivision of the security alarms domain into domain portions. This sorting is applied by using the following attributes which permit to identify the origin or the nature of this alarm :

- the *securityAlarmDetector* which indicates the security mechanism identifier, object instance which has detected the alarm,
- the *managedObjectClass* and *managedObjectInstance* which give the class and instance of the mechanism,
- the *serviceProvider* which identifies the managed object submitted to the risk,
- the *serviceUser* which designates the object responsible for the infringement,
- the *eventType* and *securityAlarmCause*.

A global view of the alarm state is given to the administrator by displaying the state of every domain portions with a visual representation of its state : no pending alarms in the domain, alarms waiting for acknowledgement. Depending on the option chosen by the administrator, this global view either indicates that there are problems in a (group of) system(s), or that there are alarms of this or that type.

Then, the administrator can zoom-in on the domain portion he wants to investigate and clear by clicking on the corresponding icon.

The security alarms are displayed in a tabular format, alarms of the same type concerning a same object are grouped in a single entry (line) in the table — double clicking on this line brings-up a window providing the detail of all the alarms belonging to the group.

Each table entry may provide all or some of the following information :

- **Status indication** : tells whether the alarm has been acknowledged and/or ignored;
- **Time of first alarm** : date of the first generation of the alarm concerning a given object;
- **Time of last alarm** : date of the last generation of the alarm concerning the same object;
- **Count** : the number of occurrences of the alarm;
- **Event type** : one of integrity violation, operational violation, physical violation, security service or mechanism violation or time-domain violation;
- **Security alarm cause** : specifies the cause of the security alarm;
- **Security alarm severity** : defines the significance of the security alarm, i.e. one of indeterminate, critical, major, minor or warning.
- *Remark* : this item in the table could be replaced by setting the colour of the line to a suitable value; the value of this information corresponds to the highest severity of a group of events;
- **Security alarm detector** : identification of the alarm detector;
- **Service user** : identifies the service-user whose request for service led to the generation of the security alarm;

- **Service provider** : identifies the intended service-provider of the service that led to the generation of the security alarm;
- **Additional information** : the following information is provided as a *managementExtension* and might be used for filtering alarms.
  - subject : already provided by "Service user".
  - object : identifies the object on which the application of "operation" led to the generation of this alarm.
  - object security level : qualifies the degree of protection of the object.
  - subject sensitivity : qualifies the access-rights of the subject.
  - operation : specifies the operation involved in the alarm.

4.2.10.1.3.2.2 Handling Security Alarms

The security alarm(s) to be handled are first selected by the user and it (they) might :

- simply be ignored and removed from the display (the problem is known and no further action is required), clicking on an "ignore" button would provide the necessary means;
- require subsequent recovery action to be undertaken - the alarm being acknowledged once the security recovery operation has been triggered by the security alarm administrator.

Thus, associated with the alarm display window, the interaction means required for fulfilling the previously described task has to be available. As an example, the additional information might be input in an area below the displayed alarms; clicking on a "forward for recovery" button would send the relevant information (alarm(s) and the information provided by the alarm administrator) to the recovery management application.

4.2.10.2 Management of Audit and Alarm

The Graphical User Interface of the Security Audit and Alarm management applications is a collection of user friendly mask windows which are defined and realised to help the user to access easily the event forwarding discriminator construction and the alarm reporting function.

The GUI is structured in the following way :

- The top level GUI
- The GUI for alarm viewing
- The GUI for EFD management
- The GUI for log management
- The graphical editor for building event filters.

The main goal of the GUI is to allow the user not necessarily familiar with the security management concept and the OSI model to be able to access the management functions. This is the reason for which the GUI should be user-friendly not only as to how the information is displayed, but also in the manner in which the user participates in the management process.

Top-Level Window

This window allows to start the TRUMPET management session. The administrator can select the security services to select in the *Administration* menu : currently only the audit management is implemented; potentially, management of the security policy, management of the authentication and access control services and key management could be added. The *Security Status* menu displays the number of security alarms for each severity level. Two buttons allow to display more information about the alarms.

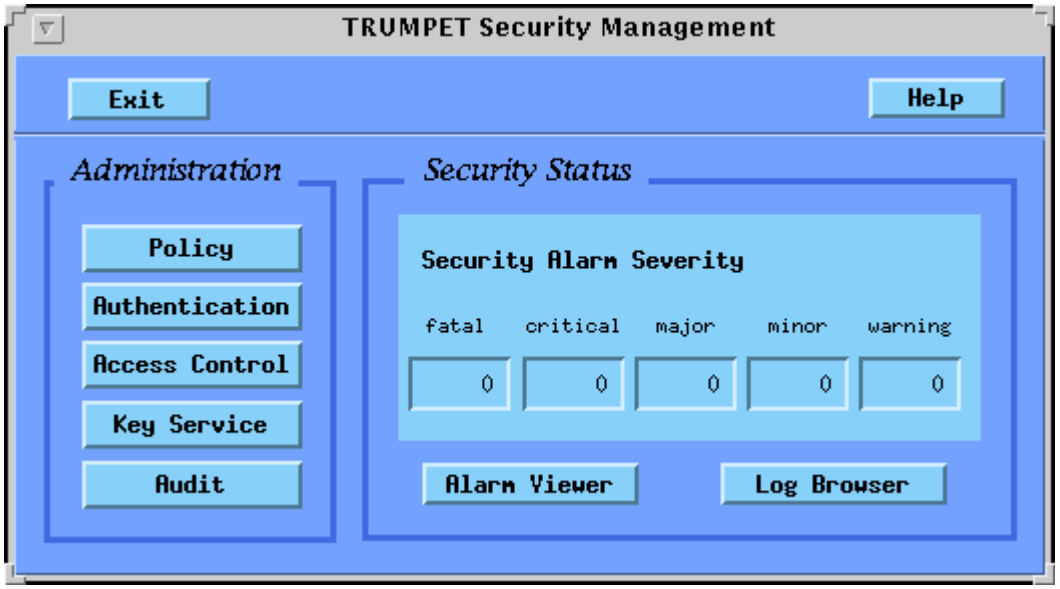

Figure 28: Top Level Management Window

Agent Configuration Window

This window allows to see each agent configuration from which security alarms will be collected. In its menu bar, the administrator can select an audit manager configuration containing some agent bitmaps which he can move on the trials bit map and perform actions on them.

Agent-EFD Control Window

This window gives some useful information on the selected agent and controls the EFD on it. The agent control panel shows the name and the current co-ordinates of the agent. The user can give the current operational state of the agent (present, running, locked, etc.). Using the EFD panel, the user can create a new EFD instance in the selected agent. A scroll list shows the EFDs currently associated with the agent; by choosing one of the EFD identifier from the list, the user may delete, edit or get help from it. When the user clicks on the Modify or Create button, the EFD construction mask window is displayed.

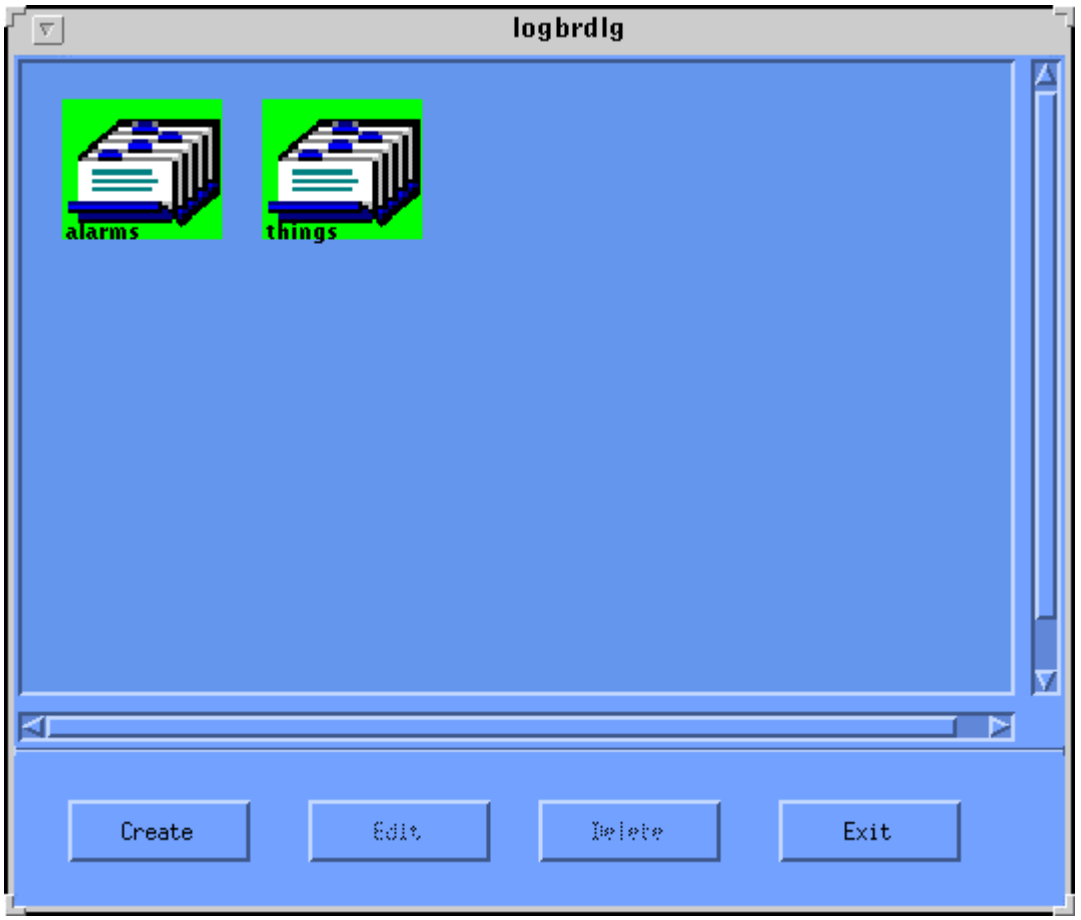

Figure 29 : Agent Control Window

EFD Construction Mask Window

This window contains all the attributes of the EFD class. The security administrator can specify the appropriate value to build a new EFD or edit an existing EFD. By clicking on the corresponding button, the administrator is displayed a specific window to fill all the EFD attributes fields : Discriminator Construct, Start Time, Stop Time, Intervals of Days, Week Mask, Destination, Backup Destination List, Active Destination, Administrative State, Operational State, Availability Status, Confirmed Mode.

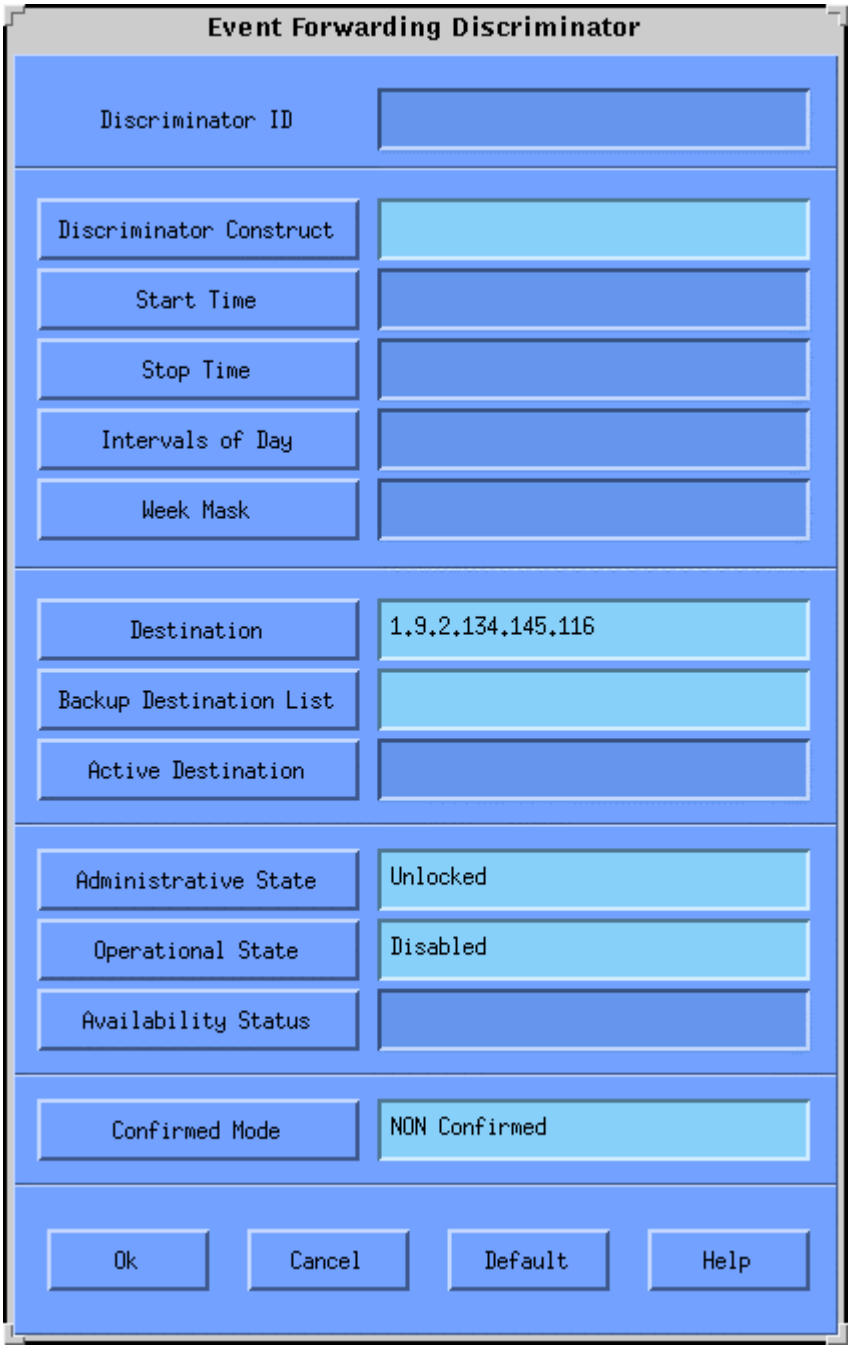

Figure 30 : EFD Construction Window

Filters Editor Window

This window allows to construct a discriminator graphically. On the drawing area, the administrator has to click on a rectangle bitmap and to control it by using the mouse menu button.

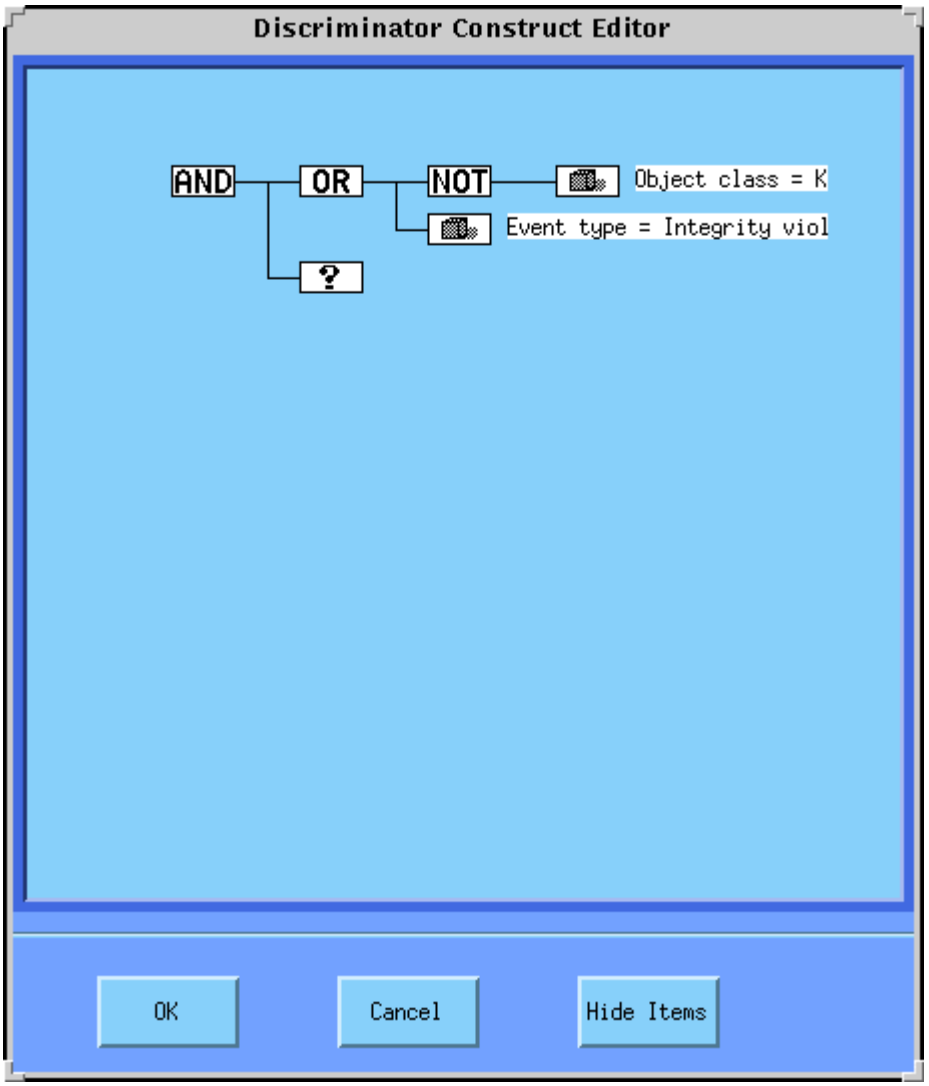

Figure 31 : Filter Edition Window

Control Window for Managing Logs

This windows is used for log management. The existing logs are displayed as icons with log names below. The security auditor can select a log with the mouse to display the contents of the log or delete the log. He can create a new log (Log Creation Window). By double-clicking on the log icon, the administrator can display the log attributes.

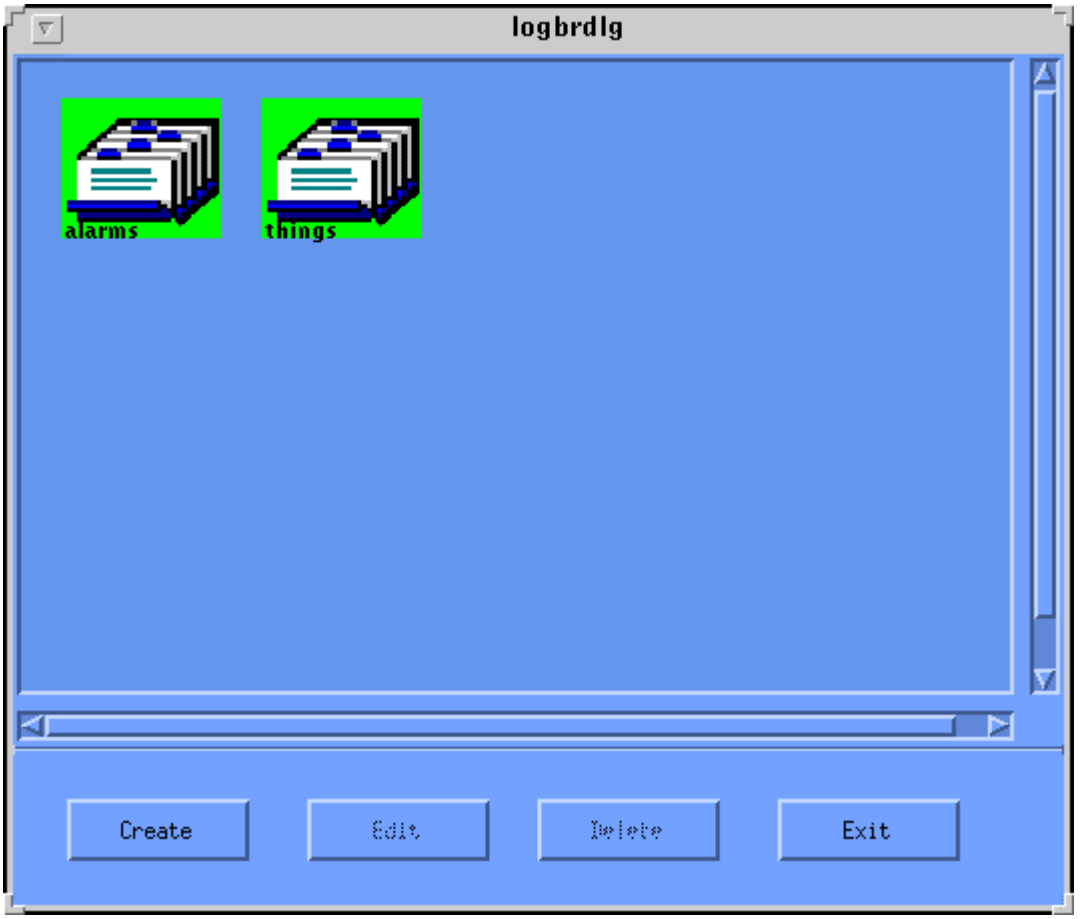

Figure 32 : Log Management Control Window

Log Creation Window

This window allows the creation of a new log. A default log can be created or the administrator can create a customised log by specifying – using the same windows as for the EFDs – the discriminator construct, the start time, the stop time, the intervals of days, the week mask. He can modify the operational and administrative state of the log.

Log Attributes Display Window

The administrator can display and modify the attributes of the selected log.

Alarm Viewer Window

This window shows the collected security alarms. Only 10 alarms can be displayed simultaneously. The alarm buffer size is limited. The older alarms will be discarded when the buffer is full.

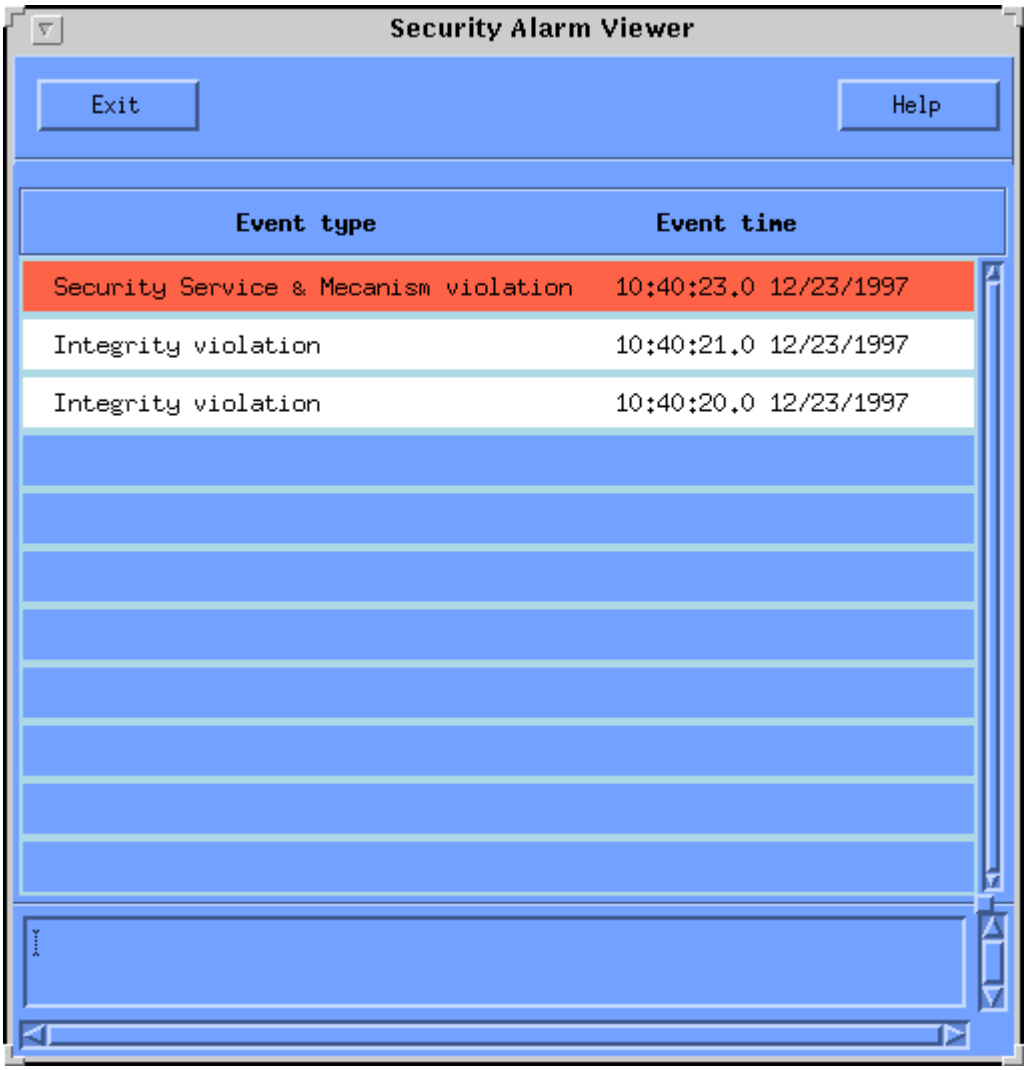

Figure 33 : Alarm Viewer Window

4.2.11 SELF

The SELF is the Audit and Alarm agent. It runs using the UNIX shell script “runSelf” from the directory /Trumpet.dev/src/securityPackage/self/audagent. It listens to a specific port and when an event is received form the SMASC or the Management system, it pre-process it and sends a CMIS M-EVENT-REPORT to the Audit and Alarm manager. There two ways to run the SELF script. Either without an argument or using an integer as argument which denotes the listening port of the SELF. In the first case there is a default listening port.

5 SERVICE MANAGEMENT DEVELOPERS MANUAL

5.1 CPN User Application

5.1.1 Engineering Object Model

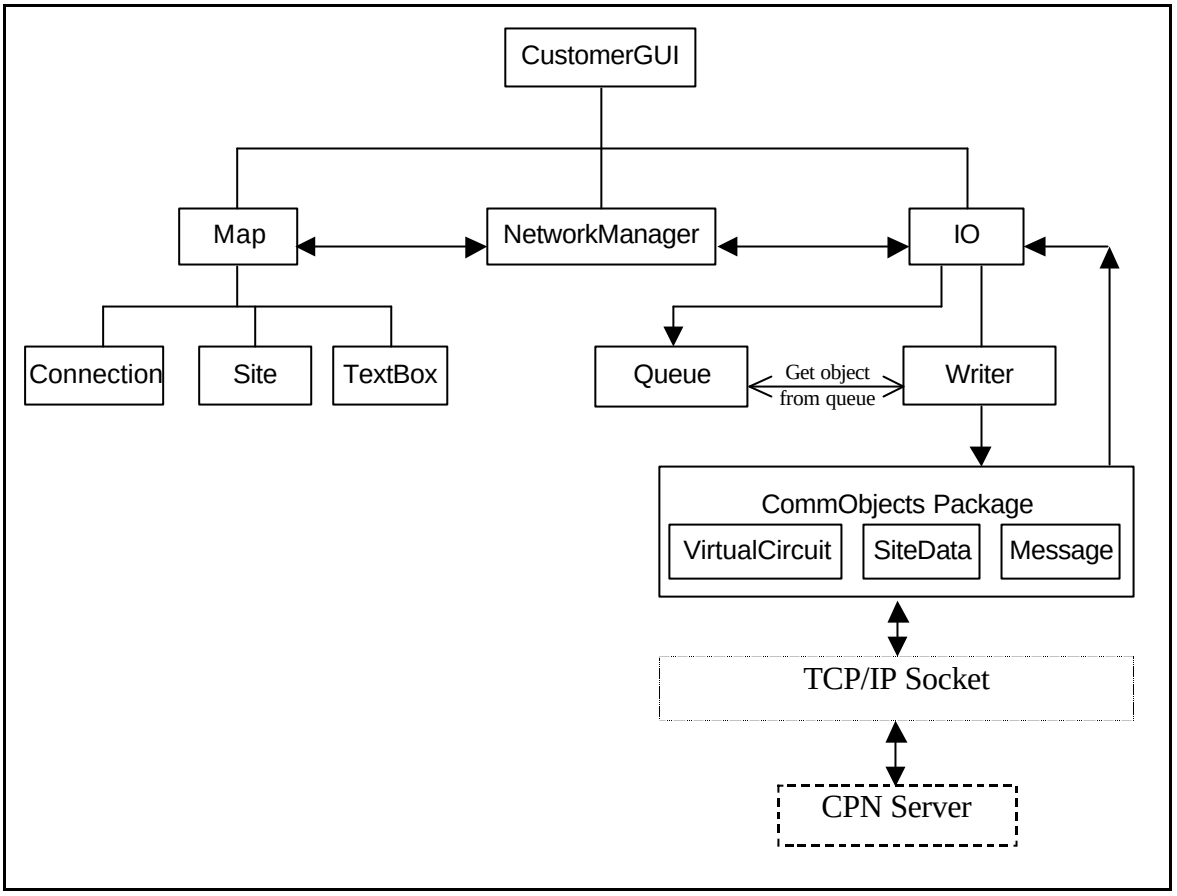

Figure 34: Engineering Object Model of the CPN User Application

The CustomerGUI, Map and NetworkManager components extend class NameConstants in order to share global constants.

- **CustomerGUI**: is an applet designed to display a map on which sites and connections can be displayed and modified. The map on which the network is displayed is a background image of the applet. Buttons and checkboxes are placed on the screen in order to allow modification to the network. Events relating to the use of the buttons, mouse actions etc. are caught and methods of the Map class are called to make network modifications.
- **Map**: the Map component manipulates the map image displayed in the CustomerGUI applet to show the location of sites and the connections between them. Map also holds lists of the current sites and the connections between them.
- **Connection**: an instance of this type exists for each connection on the map. It holds the information required for each connection and has methods to modify or to obtain information on the connection.
- **Site**: an instance of this type exists for each site on the map. It holds the information required for each site and has methods to modify or to obtain information on the site.
- **IO**: this component performs the communication between the CustomerGUI and the CPN Server. It uses a Queue class to store outgoing messages. The CallbackPanel methods are then used to process incoming objects.

- **Queue**: this class is used as a FIFO buffer for the objects to be sent to the CPN Server. The IO send methods are used to transmit each object as it is removed from the stack
- **Writer**: this class is a separate thread started by the IO class. It waits for objects to be placed on the Queue, objects are then removed from the queue and sent by the object methods to the CPN Server.
- **CallbackPanel**: this class contains the methods for handling the reception of objects from the CPN Server.
- **NetworkManager**: this class, derived from the CallbackPanel, is used to control all communication between the CustomerGUI and the CPN Server. Map functions make calls to this class when transmitting objects and NetworkManager calls Map methods when objects are received from the CPN Server and the Map needs to be updated.
- **VirtualCircuit**: this class, a member of the CommObjects package is used to hold a representation of a virtual circuit. It also has methods which can transmit and receive virtual circuit details (i.e. this object)
- **SiteData**: this class, a member of the CommObjects package is used to transfer Site information to and from the CPN Server.
- **Message**: this class, a member of the CommObjects package is used to transfer a list of text messages to and from the CPN Server.

5.1.2 Supported Component Interfaces

The CPN user application (GUI) has only one interface, which is a connection to the CPN Server component via a TCP/IP socket. The actions required of the CPN Server by the GUI are determined by a transmitted keyword and its following data (when applicable), the following table defines the keywords and the expected following data.

Keyword	Class Used For Transmission	Associated Data Types	Data Contents
UserDetails	Message	String, String	Username, Password
LDAPDetails	Message	String, String	VASP LDAP Username, Password
ReserveConnection	VirtualCircuit	VirtualCircuit	Connection details
deleteConnection	Message	String	Connection Identifier
modifyConnection	Message	String, String, String	Connection Id, Stop time, Bandwidth
getConnections	Message		
GetSites	Message		
AddSite	SiteData	SiteData	Site details
DeleteSite	Message	String, String	Site name, location
getBandwidths	Message	String	Connection type for required bandwidth (one of Voice, Video or Data)
VirtualCircuit	VirtualCircuit	VirtualCircuit	New connection details sent by the CPN Server, normally in response to a getConnnections request.

The classes referenced above support the following interfaces for transmitting their data.

METHOD

SiteData.writeSite

SYNOPSIS

```
public void writeSite (DataOutputStream dout)
```

DESCRIPTION

This method sends the variables of the class using the passed output stream.

METHOD

SiteData.readSite

SYNOPSIS

```
public void readSite (DataInputStream din)
```

DESCRIPTION

This method sets the variables of the class with data read from the passed input stream.

METHOD

VirtualCircuit.writeVC

SYNOPSIS

```
public void writeVC (DataOutputStream dout)
```

DESCRIPTION

This method sends the variables of the class using the passed output stream.

METHOD

VirtualCircuit.readVC

SYNOPSIS

```
public void readVC (DataInputStream din)
```

DESCRIPTION

This method sets the variables of the class with data read from the passed input stream.

METHOD

Message.addString

SYNOPSIS

```
public void addString (String s)
```

DESCRIPTION

This method adds the passed string to the list of strings for this message.

METHOD

Message.write

SYNOPSIS

```
public void write (DataOutputStream dout)
```

DESCRIPTION

This method sends each string of the message using the passed output stream.

5.1.3 Known Bugs

The following problem has been encountered using the Solaris JDK 1.1.x:

Bug Id: 4069784

Synopsis: TimeZone.getDefault() returns incorrect time zone.

Category: general:classes_util_i18n

Reported Against: 1.1.2; 1.1; 1.1.3; 1.1.4; 1.1beta3; 1.1fcs; 1.1.1; 1.1.5

State: Closed, duplicate of 4059431

5.2 CPN Server

5.2.1 Engineering Object Model

The CPN structure closely follows that described in deliverable 8. The component hierarchy is shown below:

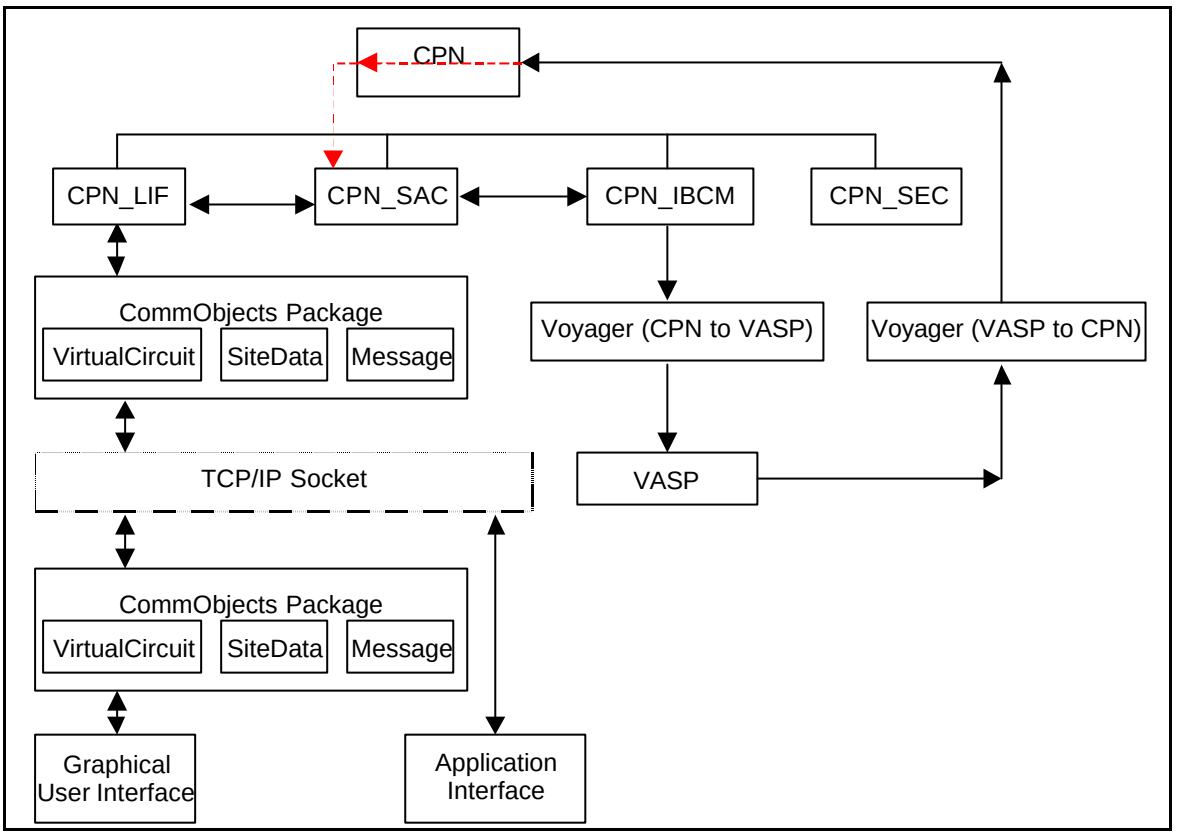

Figure 35: Engineering Object Model of the CPN User Application

All CPN components except for CPN_IBCM extend class CPN_CONSTANTS in order to share global constants between all components.

- **CPN:** invokes new instances of the other classes required, constructing them with suitable parameters to enable communication with the GUI and VASP. Callbacks from the VASP via Voyager are also handled in this component as the use of Voyager precludes any other component performing this function. A new CPN_LIF class is invoked each time a connection is made to the local socket on which the CPN resides. This component also receives callbacks from the VASP using Voyager. This requires that the CPN be made "Voyager aware" in order that it can be accessed as a remote object from the VASP.
- **CPN_LIF:** is the CPN local interface for interfacing with the local system (currently via the GUI application). This component is threaded which allows more than one user to attach to the CPN. When constructed, the LIF registers itself with the CPN_SAC component. The CPN_SAC keeps a table of each user connection with an associated id.
- **CPN_SAC:** the service access control point provides a common interface between local facilities and the VASP. All calls from the LIF and callbacks received from the CPN_IBCM and the CPN components are passed through the SAC component.
- **CPN_IBCM:** provides the communications interface from the CPN to the VASP and uses Voyager as the interface medium. Voyager allows an association to be made to a remote instance of the VASP and call methods of the VASP object such as associate (), get () etc. directly.
- **CPN_SEC:** the security interface will handle security issues such as certification and key exchange.
- **VirtualCircuit**: this class, a member of the CommObjects package is used to hold a representation of a virtual circuit. It also has methods which can transmit and receive virtual circuit details (i.e. this object)

- **SiteData**: this class, a member of the CommObjects package is used to transfer Site information to and from the CPN GUI.
- **Message**: this class, a member of the CommObjects package is used to transfer a list of text messages to and from the CPN GUI.

5.2.2 Supported Component Interfaces

The CPN interfaces on one side to the graphical user interface using a TCP/IP socket and on the other side to the VASP using Voyager using secure sockets. Much of the communication between the CPN Server and the CPN GUI is through the VirtualCircuit, SiteData and Message classes, refer to the CPN User Application sections for their interface descriptions.

5.2.2.1 Interface to the Graphical User Interface

The CPN_LIF interfaces to the GUI via a socket. More than one GUI can be connected to the CPN at any one time, a new instance of a CPN_LIF is created for each GUI requesting connection. The LIF then listens on the socket for any requests from the GUI. The LIF component has a run method, which is executed during the life of the component. The run method continuously calls the method mainParseLoop, which processes data received from the GUI connection. Requests from the GUI processed by the mainParseLoop take the form of a keyword followed by associated data (when applicable). When a keyword has been detected the associated SAC component method is called. The following keywords and associated data can be processed, the actions required and the SAC method used to complete the action.

- **User Details**: the message header from the GUI contains the string “UserDetails”. The next two tokens in the string are taken to be the distinguished name and the password for the VASP LDAP server. If tokens are received as expected then call the Make Association method in the CPN_SAC as below**:**

```
CPN_SAC.MakeAssociation (distinguished_name, password)
```

- **LDAP Details**: the message header from the GUI contains the string “LDAPDetails”. The next two tokens in the string are taken to be the distinguished name and the password for the VASP LDAP server. If tokens are received as expected then call the Make Association method in the CPN_SAC as below**:**

```
CPN_SAC.MakeAssociation (distinguished_name, password);
```

- **Reserve Connection**: the message header is “ReserveConnection”. This is followed by a Virtual Circuit definition. A new instance of the virtual circuit class is created and then its read method is used to read the values received from the GUI. Once the Virtual Circuit has been read successfully, the ReserveConnection method in the CPN_SAC is called as below.

```
CPN_SAC.ReserveConnection (VC);
```

  Where VC is the created instance of the virtual circuit class.

- **Delete Connection**: the message header is “deleteConnection”. This is followed by a string token which is the connection identifier string for the connection to be deleted. The CPN_SAC method ReleaseConnection is called as below:

```
CPN_SAC.ReleaseConnection (connection_id);
```

- **Modify Connection**: the message header is “modifyConnection”. This is followed by three string tokens, the first is the connection identifier. The second is the new stop time of the connection and the final string is the new bandwidth of the connection. Once the tokens have been read the associated virtual circuit stored by the LIF is updated and the SAC method ModifyConnection is called as below:

```
CPN_SAC.ModifyConnection (VC);
```


Where VC is the updated virtual circuit class, found from the connection id.

- **Get Connections**: the message header from the GUI is "getConnections", call the GetConnections method in the LIF as below**:**

```
CPN_SAC.MakeAssociation (distinguished_name, password)
```

- **Get Sites**: the message header is "GetSites". The SAC method GetSiteVector is called as below, to return a list of sites which is then sent to the connection GUI:

```
Vector sites = CPN_SAC.GetSiteVector ()
```

- **Add Site**: the message header is "AddSite". This is followed by a SiteData definition. A new instance of the SiteData class is created and then its read method is used to read the values received from the GUI. Once the SiteData has been read successfully, the addSite method in the CPN_SAC is called as below.

```
CPN_SAC.addSite (site);
```

Where site is the created instance of the SiteData class.

- **Delete Site**: the message header is "DeleteSite". This is followed by two string tokens which specify the name and the location of the site to be deleted. Once the name and location have been read successfully, the deleteSite method in the CPN_SAC is called as below.

```
CPN_SAC.deleteSite (name, location);
```

- **Get Bandwidths**: the message header is "getBandwidths".This is followed by a string token which is the type of connection whose bandwidths are required, this can be one of "VOICE", "VIDEO" or "DATA". Once the connection type has been read the local LIF method getBandwidths is called as below:

```
Message bw = getBandwidths (connection_type);
```

5.2.2.2 Interface to the VASP

The interface between the CPN and the VASP is made using the Voyager package. The CPN_IBCM component performs all interfacing to the VASP. Refer to the VASP Customer Service section for a description of the supported interface.

VASP to CPN Interface:

In order for the VASP to return information to the CPN Server the CPN object requires a Voyager aware version of itself, the VCPN class. An instance is constructed and started from the main() method in the CPN class. This object is then referenced remotely by the VASP by means of the Voyager package. Two CPN methods are defined that the VASP can call, they are:

```
public void eventReport (String ConnectionID, boolean status);
public void eventReport (Entry details);
```

Note: The first method should be considered obsolete and be replaced with calls to the second version of the method.

When called these methods take the information passed and call the appropriate CPN_SAC method to complete the processing and pass to the required GUI user connections.

5.3 VASP Customer Server

5.3.1 Engineering Object Model

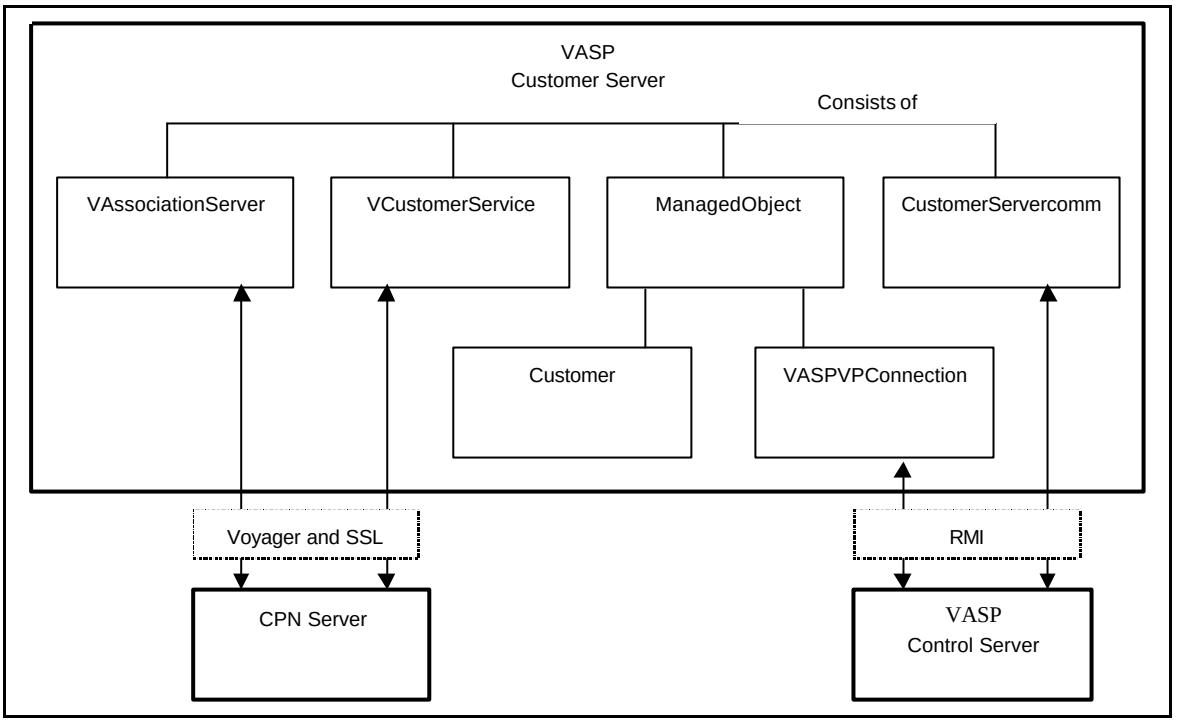

Figure 36: Engineering Object Model of the VASP Customer Server

? **VAssociationServer:** is the local interface for a CPN to start an association with the VASP. It provides a method to create new objects to provide services.

? **VCustomerService:** is the interface for the CPN to provide services to a Customer. It allows the management of the Customer and VASPVPConnnections objects.

? **ManagedObject:** is the base class of the objects to be managed by the CPN. The major function of ManagedObject class is to interface the LDAP Directory Server, by providing facilities to build/modify/release the entries.

? **Customer:** this object contains information about the customer as the name, the id and password. This object derives from the ManagedObject class and keeps its image in the LDAP Directory Server up to date.

? **VASPVPConnection:** this object contains information about a connection as termination points, bandwidth and schedule. It provides methods for its creation/modification/release, and performs the suitable requests to the VASP Control Server. This object derives from the ManagedObject class and keeps its image in the LDAP Directory Server up to date.

? **CustomerServercomm:** is the interface for the VASP Control Server to report events or request's result from the PNOs.

5.3.2 Supported Interfaces

5.3.2.1 Interface to the CustomerServer as seen by the CPN.

The interface between the CPN and the Customer Server is made through the Voyager ORB.

There are two steps for a CPN to interface with the CustomerServer. Firstly the CPN register a virtual VASP object (VAssociationServer), calls an association method to obtain a reference to another virtual VASP object (VCustomerService). Secondly the VcustomerService (dedicated to this CPN) provides methods to communicate with the VASP.

Each call to the following methods sends an Event to the CMA event manager with explicit information.

5.3.2.1.1 VAssociationServer methods.

METHOD

VAssociationServer.Associate

SYNOPSIS

```
public synchronised VCustomerService associate (
                            String DistinguishedName,
                            String password,
                            String address)
```

DESCRIPTION

This method first creates a new connection to the LDAP MIB with the passed DistinguishedName and Password. Then a new VCustomerService object is created to provide management services to the CPN. If successful, this object is returned. Otherwise the exception is thrown to the CPN.

5.3.2.1.2 VCustomerService methods.

METHOD

`VcustomerService.get`

SYNOPSIS

```
public void create(    String Stype,
           Entry Eentry)
```

DESCRIPTION

Creates a new object of Stype class with the attributes set in Eentry. This implicitly references the new object in the LDAP MIB. The implemented classes are “Customer” and “VASPVPConnection”.

In case of a “VASPVPConnection” creation, a request to the ControlServer is performed to reserve a new connection with the passed characteristics. And an event is send to the CMA event manager to inform the request of reservation.

If unsuccessful an exception is thrown to the CPN.

METHOD

`VcustomerService.get`

SYNOPSIS

```
public EntrySet get(   String BaseDN,
           int scope,
           String filter,
           String[] SAattrs)
```

DESCRIPTION

Performs a search in the LDAP MIB tree, with the passed parameters:

- the base DN which determines the subtree to apply the search to,
- the kind of scope to perform,
- the filter to select the entries.

Returns to the CPN a set of entries which match with the passed arguments. Each entry contains only the selected attributes specified in SAattrs.

If unsuccessful an exception is thrown to the CPN.

METHOD

`VcustomerService.modify`

SYNOPSIS

```
public Entry modify(String BaseDN, String conId, AttributeList attList)
                                   throws CMISException, LDAPException
```

DESCRIPTION

Performs a search in the LDAP MIB subtree determined by the BaseDN, to find the VASPVPconnection identified by conID. Then replaces the attributes of the connection according to the passed attribute list, and requests to the ControlServer the modification of the connection.

An event is send to the CMA event manager to inform the request of modification.

If successful the modified entry is returned.

If unsuccessful an exception is thrown to the CPN.

METHOD

`VcustomerService.delete`

SYNOPSIS

```
public void delete(String BaseDN, String conId) throws CMISException
```

DESCRIPTION

Performs a search in the LDAP MIB subtree determined by the BaseDN, to find the VASPVPconnection identified by conID. Then requests to the ControlServer the release this connection.

An event is send to the CMA event manager to inform the request of release.

If unsuccessful an exception is thrown to the CPN.

5.3.2.2 Interface to the CPN as seen from the CustomerServer.

The interface between the CPN and the Customer Server is made through the Voyager ORB.

To interface with the CPN the CustomerServer registers a virtual CPN object (VCPN), which provides two methods to communicate.

METHOD

`VCPN.eventReport`

SYNOPSIS

```
public void eventReport(Entry details)
```

DESCRIPTION

Sends to the CPN an Entry that contains an image of the current state of an object.

METHOD

`VCPN.eventReport`

SYNOPSIS

```
public void eventReport(String ConnectionID, boolean status)
```

DESCRIPTION

Sends to the CPN the status of a connection. This function is not used any more.

5.3.2.3 Interface to the ControlServer as seen by the CustomerServer.

The interface between the CPN and the Customer Server is made using Java Remote Invocation Method (RMI).

Three methods are offered by the Vasp Control Server:

METHOD

`ControlServer.reserveConnection`

SYNOPSIS

```
public void reserveConnection(String          VaspVPId,
                              String          srcCustId,
                              String          tgtCustId,
                              String          sourceAddr,
                              String          targetAddr,
                              VaspScheduleType schedule,
                              VaspQoSTypeOpt   qosParsOpt)
```

DESCRIPTION

Requests to reserve a connection between a peer entities with the specified parameters.

METHOD

`ControlServer.modifyConnection`

SYNOPSIS

```
public void modifyConnection(String                VaspVPId,
                             VaspScheduleTypeOpt   schedOpt,
                             VaspQoSTypeOpt        qosParsOpt)
```

DESCRIPTION

Requests to modify a reserved connection according to the passed parameters.

METHOD

`ControlServer.releaseConnection`

SYNOPSIS

```
public void releaseConnection(String VaspVPId)
```

DESCRIPTION

Requests to release a reserved connection.

5.3.2.4 Interface to the CustomerServer as seen by the ControlServer.

The interface between the CPN and the Customer Server is made using Java Remote Invocation Method (RMI).

Seven methods are offered by the CustomerServer.

Each call to the following methods sends an Event to the CMA event manager with explicit information.

METHOD

`CustomerServercomm.allocateSegment`

SYNOPSIS

```
public void AllocateSegment(String ConnectionID, String pnoID,
                            String pnoSegmentID, String accesspoint1,
                            String accesspoint2)
```

DESCRIPTION

Memorises the allocation of a segment of a connection.

METHOD

`CustomerServercomm.allocateConnection`

SYNOPSIS

```
public void allocateConnection(String ConnectionID, boolean status)
```

DESCRIPTION

Updates the status of the connection in the LDAP MIB, according to the result of the allocation.

Informs the CPN of the connection new state.

METHOD

`CustomerServercomm.activateConnection`

SYNOPSIS

```
public void activateConnection(String VaspVPId, boolean status)
```

DESCRIPTION

Updates the status of the connection in the LDAP MIB, according to the result of the activation.

Informs the CPN of the connection new state.

METHOD

`CustomerServercomm.deactivateConnection`

SYNOPSIS

```
public void deactivateConnection(String VaspVPId, boolean status)
```

DESCRIPTION

Updates the status of the connection in the LDAP MIB, according to the result of the deactivation.

Informs the CPN of the connection new state.

METHOD

`CustomerServercomm.modifyAccepted`

SYNOPSIS

```
public void modifyAccepted(String   VaspVPId, boolean status)
```

DESCRIPTION

Updates the status of the connection in the LDAP MIB, according to the result of the modification.

Informs the CPN of the connection new state.

METHOD

`CustomerServercomm.releaseNotify`

SYNOPSIS

```
public void releaseNotify(String   VaspVPId)
```

DESCRIPTION

Updates the status of the connection in the LDAP MIB.

Informs the CPN of the connection new state.

Remove the connection Entry from the LDAP MIB.

METHOD

`CustomerServercomm.notify`

SYNOPSIS

```
public void notify(String VaspVPId, VaspReasonType    reason)
```

DESCRIPTION

Receives the notification.

5.3.3 Version, Release history

Version 3: Septembre 98.

This release support SSL, but not Access Control.

5.4 VASP Control Server

5.4.1 Engineering Object Model

The controlServer's class hierarchy is rather flat. The only hierarchy that exists, depicted below, has to do with Managed Objects (MO).

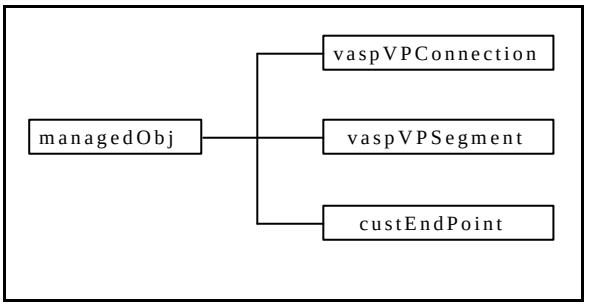

Figure 37: Engineering Object Model of the VASP Control Server

The controlServer keeps a local copy of the connection MIB for its own use. The objects that comprise the MIB are instances of the three leaf classes above. The class *managedObj* is an abstract class containing all the operations pertaining to the containment tree representing the MIB. For example, by inheriting from *managedObj*, all MIB objects are able to insert themselves into or remove themselves from the containment tree.

The class *VaspVpnManager* is the core of the controlServer and is instantiated only once. This only instance is responsible for communicating with the customerServer on the one side, and the PnoConnectionManager on the other. It receives customer requests from customerServer and further negotiates with and forwards the requests to the relevant PNOs, and vice versa, receives notifications from the PNOs and if necessary forwards them to the customerServer. As a result of the different incoming requests and notifications the MIB (containment tree) is updated.

The class *ControlServer* acts like a proxy of the *vaspVpnManager* with respect to the customerServer and likewise has only one instance. In other words, it is the *ControlServer* that really receives the requests from the customerServer. After converting some of the parameters from VASP-domain types to PNO-domain types, It forwards the requests to the *vaspVpnManager*.

The rest of the classes are only support the functionality of the controlServer such as, reading the route table or keeping a list of PNOs with active connections to the controlServer.

5.4.2 Supported Component Interfaces

The VASP controlServer module interfaces on one side with the VASP customerServer module and on the other side with the PnoConnectionManager object. This latter is an OrbixWEB proxy object representing the Xuser-Agent.

5.4.2.1 ControlServer as seen by the CustomerServer

This defines the methods offered by the controlServer to the customerServer and is used to relay customer requests to the controlServer.

```
interface VPNService
{
public void reserveConnection( String VaspVPId,
                               String sourceCustId,
                               String targetCustId,
                               String sourceAddr,
                               String targetAddr,
                               VaspScheduleType schedule,
                               VaspQoSTypeOpt qosParsOpt )
throws RemoteException, vaspException;

public void modifyConnection( String VaspVPId,
                              VaspScheduleTypeOpt  schedOpt,
                             VaspQoSTypeOpt   qosParsOpt)
throws RemoteException, vaspException;

public void releaseConnection( String VaspVPId )
throws RemoteException, vaspException;
}
```

5.4.2.2 ControlServer as seen by the PnoConnectionManager

This defines the notifications that the ControlServer can receive from the PnoConnectionManager.

```
interface VPConnectionServiceEventHandlerOperations
{
public void
activateConnectionNotify( XuserTypes.NameType pnoId,
                          XuserTypes.NameType vpConnectionId,
                          int status )
throws  IE.Iona.Orbixe.CORBA.SystemException;
```

```
public void
deactivateConnectionNotify( XuserTypes.NameType pnoId,
                            XuserTypes.NameType vpConnectionId,
                            int status )
throws  IE.Iona.Orbixe.CORBA.SystemException;

public void
releaseConnectionNotify( XuserTypes.NameType pnoId,
                         XuserTypes.NameType vpConnectionId,
                         XuserTypes.ReleaseReasonType reason )
throws  IE.Iona.Orbixe.CORBA.SystemException;

public void
connectionNotify( XuserTypes.NameType pnoId,
                  XuserTypes.ReasonType reason,
                  String eventInformation )
throws  IE.Iona.Orbixe.CORBA.SystemException;
}
```

5.4.3 Version, Release history

Version 3: August 1998.

5.5 VASP CORBA/TMN Gateway

5.5.1 Engineering Object Model

The VASP CORBA/TMN Gateway has been introduced throughout the implementation design to provide the glue between the JAVA-based VASP management system and the TMN management solution provided for the PNO domain. Its primary purpose is to map between the TMN Xuser interface to an JAVA-based API which can be integrated with the VASP Control Server.

The gateway provides a set of adapter objects which exhibit a subset of the TMN Xuser interface. The interfaces of the adapter objects are defined using the interface definition language (IDL) which is part the CORBA specification [OMG CORBA]. At the programming level the IDL interfaces are mapped to suitable programming constructs (i.e. an JAVA API) according to language bindings. The implementation design of the gateway as shown in Figure 38 represents a refinement of the computational design for the PNO Service Layer Management which has been described in D8, Section 8.1.2 [TRUMPET-D8].

The object *CORBA/TMN Gateway* Server presents the initial object of the gateway which registers the gateway application with the communications infrastructure. Moreover, it creates two factory objects which are provided to the *VASP Control Server* to create and delete instances of the service objects. The gateway provides two kinds of service objects called *VPConnectionService* and *VPSubscriptionService* which represent the core interface to the PNO Service Management. These objects provide the required functionality to manage VP connections and to maintain customer access points. Additionally, the *VPConnectionServiceEventHandler* provides means to forward event reports of the PNO Service Management to the VASP Control Server. This object is part of the VASP Control Server and provides functions which may be invoked by the gateway to indicate event reports as described in D8, section 8.1.2.

The core implementation of the service objects *VPConnectionService* and *VPSubscriptionService* is provided by the *XuserMgrRequestHandler* which maps the operations of the service objects down to CMIP requests using the XMP/XOM API of HP OpenView DM. The request handler also serves as a co-ordinator

for the CORBA and HP-OV communication channels. It realises an event loop which is waiting for indications of CORBA IIOP or CMIP protocol requests.

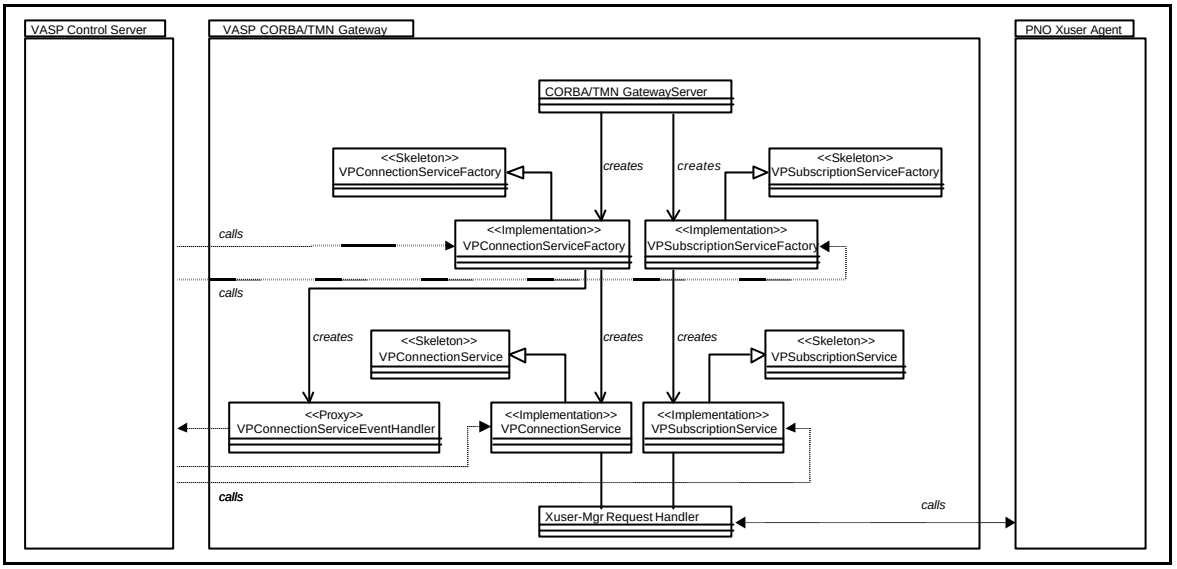

Figure 38: Engineering Object Model of the VASP CORBA/TMN Gateway

5.5.2 Supported component interfaces

Figure 39 presents an overview on the interfaces supported by the VASP CORBA/TMN gateway. All the supported interfaces are provided to the VASP Control Server which utilises the VP connectivity services of the PNO Service Layer Management. For the notification on event reports received from the PNO Service Layer Management a *VPConnServiceEventHandler* interface is required which is provided by the VASP Control Server. Moreover, the VASP CORBA/TMN gateway makes use of the Xuser interface which is provided by the PNO Xuser Agent.

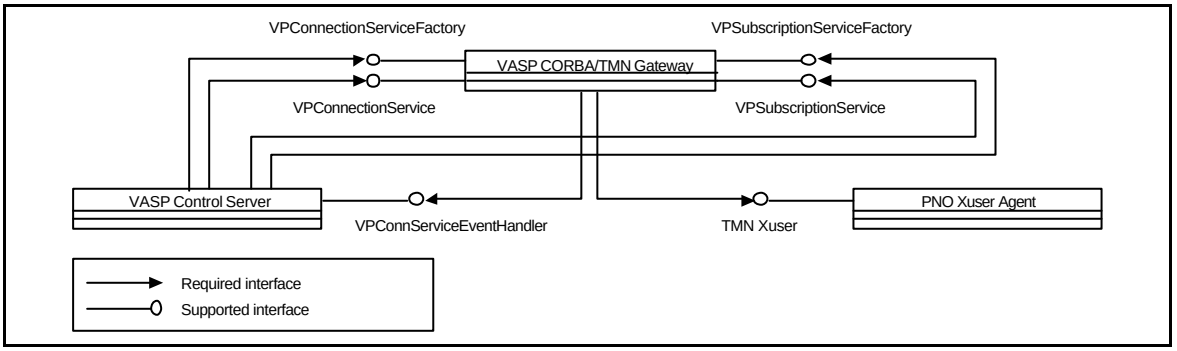

Figure 39: Required and supported interfaces of the VASP CORBA/TMN Gateway

The remainder of this section only describes the details of the supported interfaces

5.5.2.1 VP Connection Service Factory

FUNCTION

```
PnoConnectionMgr::VPConnectionServiceFactory::create()
```

SYNOPSIS

```
VPConnectionService create(
      in XuserTypes::NameType pnoId,
      in VpnManager::VPConnectionServiceEventHandler eventHandler);
```

DESCRIPTION

Creates a new service object of type *VPConnectionService* for the interaction with the Service Layer Management System of the PNO identified by *pnoId*. The new service object is associated with an event handler which is referenced by the *eventHandler* parameter.

ARGUMENTS

- *pnoId*: Identifies the PNO Service Layer Management System. The identifier may be either a presentation string containing the global distinguished name of the PNO MAE or a number which can be mapped to the distinguished name according to an mapping table of the CORBA/TMN gateway.
- *eventHandler*: Contains an object reference to an event handler object *VPConnectionServiceEvent-Handler* which is associated with the new service object. The event handler is triggered by the service object if an event report from the PNO Service Layer Management System is indicated.

RESULTS

Returns an object reference to the new service object on success. If the service object cannot be created a nil object reference is returned.

BUGS

An exception type should be defined to indicate possible failures (i.e., PNO not found, communication failure, etc.)

FUNCTION

```
PnoConnectionMgr::VPConnectionServiceFactory::delete()
```

SYNOPSIS

```
void delete(
      in VPConnectionService vpConnectionServiceRef);
```

DESCRIPTION

Deletes the service object of type *VPConnectionService* identified by the object reference given in *vpConnectionServiceRef*. After successful deletion of the service object the event handler object which had been be associated with service object will not be triggered anymore to indicate event reports.

ARGUMENTS

- *vpConnectionServiceRef*: Contains an object reference to an service object of type *vpConnection-ServiceRef*.

RESULTS

∅.

BUGS

An exception type should be defined to indicate possible failures of this operation (i.e., object not found).

5.5.2.2 VP Connection Service

FUNCTION

```
PnoConnectionMgr::VPConnectionService::reserveConnection()
```

SYNOPSIS

```
XuserTypes::ReserveConnectionResultType reserveConnection(
      in XuserTypes::ReserveConnectionInfoType connectionInformation)
            raises (ConnectionRequestFailure);
```

DESCRIPTION

Reserves a new VP connection at the associated PNO according to the details given by the parameter *connectionInformation*.

ARGUMENTS

The parameter *connectionInfo*, contains the following components:

- *userId*: Identifies the user of the connectivity service offered by the PNO. According to the TRUMPET scenario this parameter identifies the VASP.
- *sourceE164AddressOpt*: May contain the E164 source address of the customer access point.
- *destinationE164Address*: Contains the E164 destination address of the customer access point.
- *connectionProtectionLevelOpt*: May contain the protection level for the VP connection. Possible values (if not omitted) are *protected*, *unprotected-lowpriority*, *unprotected-highpriority*.
- *routingCriteriaOpt*: May contain customised settings for the routing algorithm implemented as part of the PNO's connectivity service. Currently no options have been defined for the routing algorithm used by the TRUMPET PNO Service Layer Management System.
- *directionality*: Identifies the directionality of the requested VP connection. The value may be either *unidirectional* or *bidirectional*.
- *schedule*: Defines the schedule for the requested VP connection.
- *qosParametersOpt*: May contain a set of QOS parameters for the requested VP connection.

RESULTS

Returns an object type *ReserveConnectionResultType* on successful operation. This object contains the connection id for the schedule connection and may additional contain the select source customer access

point. If the operation fails an exception of type *ConnectionRequestFailure* is raised which contains an object of type *ReasonType*. In the current implementation of the CORBA/TMN gateway this object will contain an error number.

BUGS

Instead of a general *ConnectionRequestFailure* exception which provides an error number, there should be rather several exception types defined for different types of errors which may occur.

SEE ALSO
`PnoConnectionMgr::VPConnectionServiceFactory`

FUNCTION
`PnoConnectionMgr::VPConnectionService::modifyConnection()`

SYNOPSIS
```
void modifyConnection(
     in XuserTypes::ModifyConnectionInfoType connectionInformation)
          raises (ConnectionRequestFailure);
```

DESCRIPTION

Modifies the characteristics of an pending or scheduled VP connection at the associated PNO as identified by the parameter *connectionInformation*. Modifications of both the schedule and the set of QoS parameters are possible.

ARGUMENTS

The parameter *connectionInfo*, contains the following components:

- *userId*: Identifies the user of the connectivity service offered by the PNO. According to the TRUMPET scenario this parameter identifies the VASP.
- *connectionId*: Identifies the VP Connection to be modified.
- *scheduleOpt*: may contain a new schedule for the given VP connection.
- *qosParametersOpt*: May contain a set of QOS parameters for the given VP connection.

RESULTS

Nothing is returned on successful operation. If the operation fails an exception of type *ConnectionRequest-Failure* is raised which contains an object of type *ReasonType*. In the current implementation of the CORBA/TMN gateway this object will contain an error number.

BUGS

Instead of a general *ConnectionRequestFailure* exception which provides an error number, there should be rather several exception types defined for different types of errors which may occur.

SEE ALSO
`PnoConnectionMgr::VPConnectionServiceFactory`

FUNCTION
`PnoConnectionMgr::VPConnectionService::releaseConnection()`

SYNOPSIS
```
void releaseConnection(
      in XuserTypes::ReleaseConnectionInfoType connectionInformation)
            raises (ConnectionRequestFailure);
```

DESCRIPTION

Releases an pending or scheduled VP connection at the associated PNO as identified by the parameter *connectionInformation*.

ARGUMENTS

The parameter *connectionInfo*, contains the following components:

- *userId*: Identifies the user of the connectivity service offered by the PNO. According to the TRUMPET scenario this parameter identifies the VASP.
- *connectionId*: Identifies the VP Connection to be released..

RESULTS

Nothing is returned on successful operation. If the operation fails an exception of type *ConnectionRequest-Failure* is raised which contains an object of type *ReasonType*. In the current implementation of the CORBA/TMN gateway this object will contain an error number.

BUGS

Instead of a general *ConnectionRequestFailure* exception which provides an error number, there should be rather several exception types defined for different types of errors which may occur.

SEE ALSO
`PnoConnectionMgr::VPConnectionServiceFactory`

5.5.2.3 VP Subscription Service Factory

FUNCTION
`PnoConnectionMgr::VPSubscriptionServiceFactory::create()`

SYNOPSIS
```
VPSubscriptionService create(in XuserTypes::NameType pnoId);
```

DESCRIPTION

Creates a new service object of type *VPSubscriptionService* for the interaction with the Service Layer Management System of the PNO identified by *pnoId*.

ARGUMENTS

- *pnoId*: Identifies the PNO Service Layer Management System. The identifier may be either a presentation string containing the global distinguished name of the PNO MAE or a number which can be mapped to the distinguished name according to an mapping table of the CORBA/TMN gateway.

RESULTS

Returns an object reference to the new service object of type *VPSubscriptionService* on success. If the service object cannot be created a nil object reference is returned.

BUGS

An exception type should be defined to indicate possible failures (i.e., PNO not found, communication failure, etc.)

FUNCTION
`PnoConnectionMgr::VPSubscriptionServiceFactory::delete()`

SYNOPSIS
```
void delete(in VPSubscriptionService vpSubscriptionServiceRef);
```

DESCRIPTION

Deletes the service object of type *VPSubscriptionService* identified by the object reference given in *vpSubscriptionServiceRef*. After successful deletion of the service object the event handler object which used to be associated with service object will not be triggered anymore to indicate event reports.

ARGUMENTS

- *vpSubscriptionServiceRef*: Contains an object reference to an service object of type *vpSubscription-ServiceRef*.

RESULTS

∅.

BUGS

An exception type should be defined to indicate possible failures of this operation (i.e., object not found).

5.5.2.4 VP Subscription Service

FUNCTION

```
PnoConnectionMgr::VPSubscriptionService::createAccessPoint
```

SYNOPSIS

```
void createAccessPoint(
      in XuserTypes::IdentifierType userId,
      in XuserTypes::NameType accessPointId,
      in XuserTypes::E164AddressType E164Address)
            raises (InvalidAccessPoint);
```

DESCRIPTION

Registers a customer access point with PNO Service Layer Management System for the user identified by the parameter *userId*. Note, that in the TRUMPET scenario the user role is always taken by the VASP management system which may register access points on behalf of its customers. The new access point is identified by *accessPointId* which serves as the value for the naming attribute of the created object instance within the PNO Service Layer Management System. The third parameter defines an globally unique number which is associated with the new access point.

ARGUMENTS

- *userId*: Identifies the user of the connectivity service offered by the PNO. According to the TRUMPET scenario this parameter identifies the VASP.
- *accessPointId*: Serves as the value for the naming value of the object instance to be created within the PNO Service Layer Management System. For the given user the identifier provided has to be unique.
- *E164Address*: Defines an globally unique number which is associated with the new access point.

RESULTS

Nothing is returned on successful operation. If the operation fails an exception of type *InvalidAccessPoint* is raised which indicates an invalid access point identifier or an invalid E164 address.

BUGS

Additional exception types should be defined to indicate the error conditions.

SEE ALSO

```
PnoConnectionMgr::VPSubscriptionServiceFactory
```


FUNCTION
```
PnoConnectionMgr::VPSubscriptionService::deleteAccessPoint
```

SYNOPSIS
```
void deleteAccessPoint(
      in XuserTypes::IdentifierType userId,
      in XuserTypes::NameType accessPointId)
            raises (NotFound);
```

DESCRIPTION

Removes a customer access point from the PNO Service Layer Management System for the user identified by the parameter *userId*. Note, that in the TRUMPET scenario the user role is always taken by the VASP management system which may register access points on behalf of its customers. The access point to be deleted is identified by *accessPointId*. Note, that access points can only be deleted by its creator.

ARGUMENTS

- *userId*: Identifies the user of the connectivity service offered by the PNO. According to the TRUMPET scenario this parameter identifies the VASP.
- *accessPointId*: Serves as the value for the naming value of the object instance to be deleted within the PNO Service Layer Management System. For the given user the identifier provided has to be unique.

RESULTS

Nothing is returned on successful operation. If the operation fails an exception of type *InvalidAccessPoint* is raised which indicates an invalid access point identifier.

BUGS

Additional exception types should be defined to indicate the error conditions.

SEE ALSO
```
PnoConnectionMgr::VPSubscriptionServiceFactory
```

5.5.2.5 Event Reporting Service

The interface to the event reporting service supported by the CORBA/TMN gateway is not accessible to external applications, but it is used internally by the gateway to report management events to the CMA messaging system (CMA-TMS).

Report_event.h

The Xuser manager calls the following functions just before issuing a management request to the Xuser agent. They are used in the action_req.cc file, which contains the code to build CMIP requests from the Xuser C constructs, and take as input argument the request information.

```
void report_reserve_req (const SG_ReserveGBCConnectionInformation&
info);
void report_modify_req  (const SG_ModifyGBCConnectionInformation& info);
void report_release_req (const SG_ReleaseGBCConnectionInformation&
info);
```

Similarly, the Xuser manager calls the following functions just after having received the result to a management request from the Xuser agent. They are used in the action_cnf.cc file, which contains the code to decode CMIP results into Xuser C constructs, and take as input argument the request result.

```
void report_reserve_cnf (const SG_ReserveGBCConnectionResult& result);
void report_modify_cnf  (const SG_ModifyGBCConnectionResult& result);
void report_release_cnf (const SG_ReleaseGBCConnectionResult& result);
```

XuserVaspMgmtServiceReporter.h

The functions of the report_event.h interface use the instance of the singleton class XuserVaspMgmtServiceReporter to emit the management event to report to the CMA messaging system over a TCP socket. This class emission is implemented in the public method

```
int emitEvent(
const RWCString& eventType,
const RWCString& serviceReportCause,
const RWCString& actionName,
const RWCString& actionInfo);
```

The XuserVaspMgmtServiceReporter object uses the following private data members to complete the event information:

```
struct sockaddr_in     m_meAddress;
RWCString         m_serviceId;
RWCString         m_moc;
```

These data members are initialized at launch time from the configuration file cma_tms.cfg: m_moc has a constant value that identifies the Xuser manager ("XuserVaspManagementService", corresponding to a code value of "6").

5.5.3 Version, Release history

Version 2, June 1998.

5.6 PNO Xuser-Agent

5.6.1 Engineering Object Model

The PNO Xuser-Agent realises the PNO Service Layer Management System which provides the VP Connection Management Service. This service is utilised by the VASP Management System to manage a segment of an end-to-end virtual path between to customer access points of the public network domain.

The implementation design of the Xuser-Agent is based on the computational model of the PNO Service Layer Management which has been described in D8, Section 8.1.2 [TRUMPET-D8]. The TMN Xuser-interface provided the Xuser-Agent has been adopted from the MISA project according to the agreements between the TRUMPET and MISA projects. However, TRUMPET uses only a subset of the Xuser-interface, namely those functions for the connection management (MISA Path Provisioning Ensemble [MISA-D3-A1]) as well as some basic functions provided with the subscription management (MISA Subscription Ensemble [MISA-D3-A2]).

The engineering object model of the PNO Xuser Agent is depicted in Figure 40. The object *PNO Xuser Agent Co-ordinator* constitutes the initial object of the Xuser-Agent which registers the application with the communications infrastructure. Subsequently, it creates the basic agent object, namely *the MIT Manager* and *Agent Request Handler*. The *MIT Manager* is responsible for the internal data management of the Management Information Base and it provides functions to invoke operations on the contained Managed Objects. The *Agent Request Handler* realises an event loop which awaits indications for CMIP operation requests. Besides that, the *Agent Request Handler* creates the *Dispatcher* object which maps CMIP requests to operations called on the Managed Object implemenation objects maintained by the *MIT Manager*.

Operations invoked on a Managed Object will change the state of the Management Information Bases and may also result in operation invocations on a managed resource. In particular this is the case for the operations provided by *GBCServiceProvider* which are mapped internally to corresponding operation invocations on the underlying PNO Network Layer Management System. The handling of the operations of the Network Layer Management System is provided by PNO Network Management.

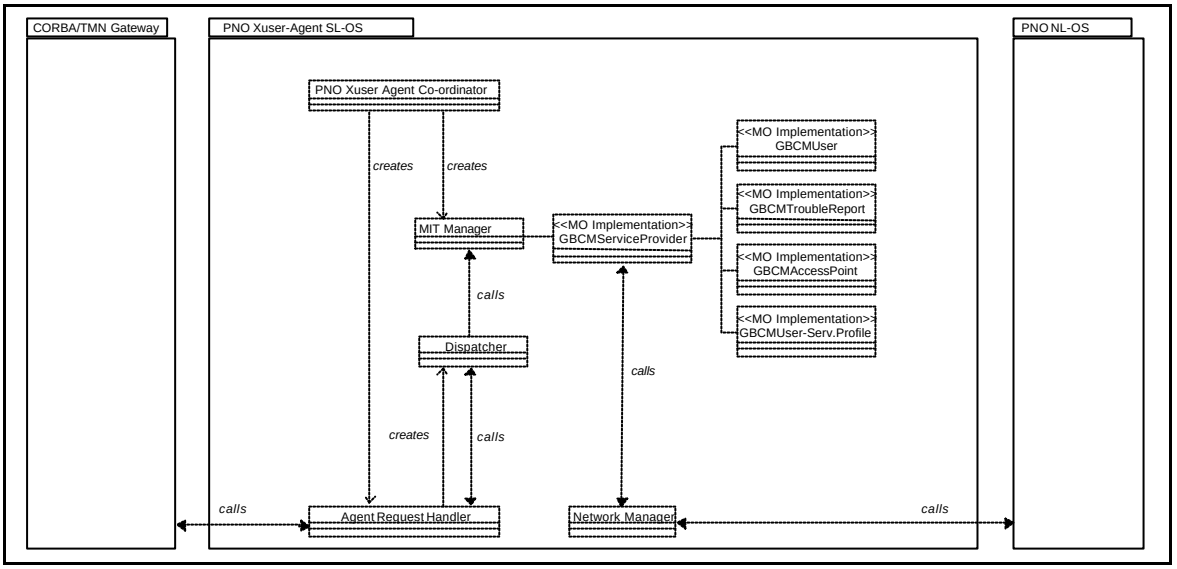

Figure 40: Engineering Object Model of the PNO Xuser-Agent

5.6.2 Supported component interfaces

The external interface provided by the Xuser-Agent corresponds to the Xuser-specification which has been developed by the MISA project [MISA-D3A1]. It is therefore not described here in detail. The latest release of the MISA Xuser specification which has been used as a basis for the implementation of the PNO Xuser-Agent can be found in Section XXX. In the remainder of this section only the basic operations provided for the connection management are described.

5.6.2.1 GBCServiceProvider

ACTION TYPE
`reserveGBCConnection`

BASE OBJECT CLASS
`GBCMServiceProvider`

BASE OBJECT INSTANCE
`GBCMServiceProvider instance`

DESCRIPTION

This action is performed by the GBCM User which requests a GBC connection reservation from the GBCM Service Provider. The result of this action is the acceptance or reject of the connection reservation request (regarding the start time, the stop time and eventually the periodicity requested). If the connection reservation is rejected, the reason is returned (not available resources, not possible in the interval time,...). If the connection reservation is accepted, a gBCConnection object instance is created.

ACTION INFORMATION

The information type *reserveGBCConnectionInformation*, contains the following components:

- *gBCMUserId*: Identifies the user of the connectivity service offered by the PNO.
- *sourceE164Address*: May contain the E164 source address of the customer access point.
- *destinationE164Address*: Contains the E164 destination address of the customer access point.

- *connectionProtectionLevel*: May contain the protection level for the VP connection. Possible values (if not omitted) are *protected*, *unprotected-lowpriority*, *unprotected-highpriority*.
- *routingCriteria*: May contain customised settings for the routing algorithm implemented as part of the PNO's connectivity service. Currently no options have been defined for the routing algorithm used by the TRUMPET PNO Service Layer Management System.
- *gBCType*: Identifies the type of connectivity service to be used. The current Xuser-specification defines the ATM and SDH Path Provisioning Service (APPS, SPPS).
- *gBCDirectionality*: Identifies the directionality of the requested VP connection. The value may be either *unidirectional* or *bi-directional*.
- *gBCSchedule*: Defines the schedule for the requested VP connection.
- *gBCPPSparameters*: May contain a set of QOS parameters for the requested VP connection.

ACTION RESULT

On successful operation the action results of type *modifyGBCConnectionResult* contains the connection id for the schedule connection and may additional contain the select source customer access point. If the operation fails the result contains an error number.

ACTION TYPE
`modifyGBCConnection`

BASE OBJECT CLASS
`GBCMServiceProvider`

BASE OBJECT INSTANCE
`GBCMServiceProvider instance`

DESCRIPTION

This action is performed by the GBCM User requesting the modification of the GBC connection. In case of SPPS (SDH), it is possible that modification is not supported. In this case the action request will be rejected."

ACTION INFORMATION

The information type *modifyGBCConnectionInformation*, contains the following components:

- *gBCMUserId*: Identifies the user of the connectivity service offered by the PNO.
- *gBCConnectionId*: Identifies the VP Connection to be modified.
- *gBCSchedule*: May contain a new schedule for the given VP connection.
- *gBCPPSparameters*: May contain a set of QOS parameters for the given VP connection.

ACTION RESULT

On successful operation the action results of type *modifyGBCConnectionResult* contains nothing. If the operation fails the result contains an error number.

ACTION TYPE
`releaseGBCConnection`

BASE OBJECT CLASS
`GBCMServiceProvider`

BASE OBJECT INSTANCE
`GBCMServiceProvider instance`

DESCRIPTION

This action is performed by the GBCM User requesting the clearing down of the GBC Connection. This will delete the *gBCConnection* object instance.

ACTION INFORMATION

The information type *releaseGBCConnectionInformation*, contains the following components:

- *gBCMUserId*: Identifies the user of the connectivity service offered by the PNO.
- *gBCConnectionId*: Identifies the VP Connection to be released.

ACTION RESULT

On successful operation the action results of type *releaseGBCConnectionResult* contains nothing. If the operation fails the result contains an error number.

5.6.3 Version, Release history, Known bugs

5.6.3.1 Version / release history

Version 2.2b: July 16th 1998 - Final version compiled for the Scottish Trial which some problems with the M4 gateway interface. This version complies to the current MISA Xuser specification version 2.5

5.6.3.2 Known bugs

Sometimes, the Xuser scheduler tries to trigger activate/deactivate messages for connections which no longer exist. This results in Xuser-Agent console error message indicating that the respective connection id have not been found. However, this bug doe not degrade the operation of the Xuser-Agent.

5.7 PNO CMA-Based NMS

5.7.1 Engineering Object Model

The figure below represents the architecture of the PNO Xuser agent with the integration of CMA-based components:

- The MOSE (Managed Object Server Entity) maintains an object model, which acts as a “bridge” between the Xuser MIB and the Fore MIB.
- The AE (Application Entity) implements the functions of the snmp-ifc.h interface in terms of CMA functionality: create/delete MOSE objects, set attributes.
- The ME (Mediation Entity) handles the translation of MOSE operations into SNMP requests to the Fore switch.

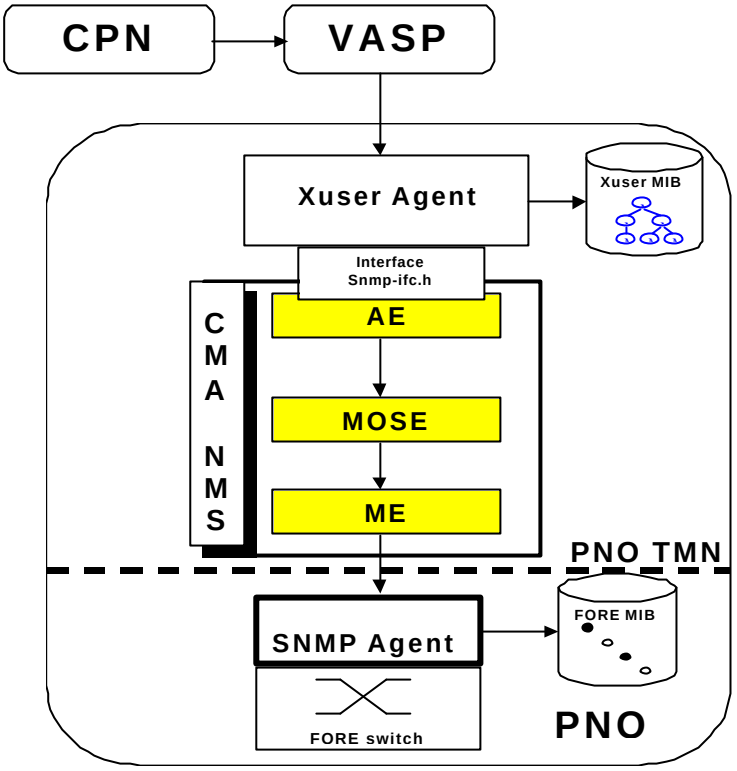

Figure 41: PNO Domain Architecture with CMA Components

Figure 41 shows four levels of inter-process communications. When the Xuser Agent calls one operation (reserve, activate, deactivate or release connection) the AE create an AE_Xuser_Adapter instance and passes the parameters to a Connection Manager instance which provides management of connections. The Connection Manager instance firstly sent call Subscribe/unSubscribe to the CMA MOSE, then performs set operations on the MOSE object model. The CMA MOSE Events Handler captures the calls from AE and pass commands to the MOSE View instance. Now MIT Manager of the View instance create/delete managed object according to Subscribe/unSubscribe calls. When Subscribe operation on new managed object is successfully completed and MO included into MIT this one can accept get/set operation from the View instance. According to scope parameter of the get/set operation changing of value of the attribute performs at MOSE or at ME level. If set/get operation is destined to the ME level than this operation is transmitted by Mediation instance to the Agent Manager of ME. The Agent Manager performs set/get operation directly to the SNMP Agent and write/read the value in physical level, that is on managed Network Element (NE).

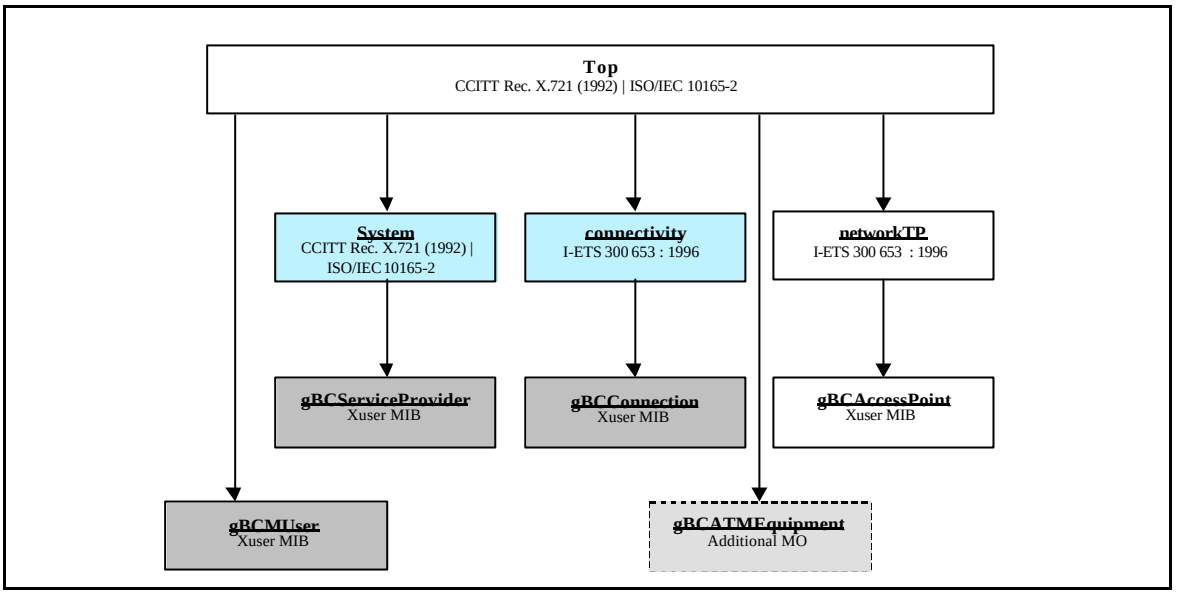

Figure 42: Inheritance Tree of the MOSE Information Model

The information model implemented by the MOSE is a subset of the Xuser MIB, except for the new **gBCATMEquipment** object class, which was added for practical purposes of CMA NMS, represents GBC ATM equipment of a PNO. A PNO can use more than one ATM switch to establish a gBCConnection. The object class gBCATMEquipment permits to extend Xuser specification to cover ATM equipment management requirements.

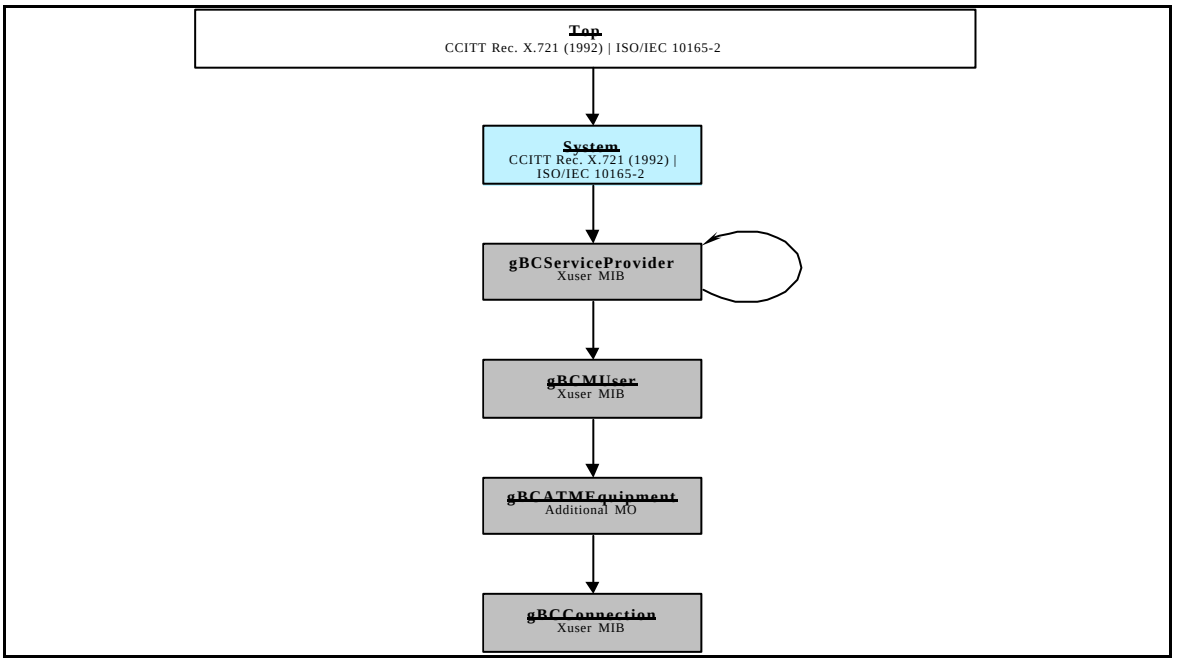

Figure 43: Containment Tree of the MOSE Information Model

The definition of the object model for CMA NMS is founded on analysis of both MIB: Xuser and FORE and on engineering of relationships, comportment and states of the chosen object classes.

The object class System owns an entity of the Global Service Provider, which represents an abstract Service Provider. The Global Service Provider can own other Service Providers and each Service Provider owns a set of end-users. An end-user owns a set of equipment that he uses to establish an end-to-end connection. And finally, the gBCATMEquipment owns a set of GBC connection, which represents a locally established connection between two ATM switches of the same PNO or an interconnection between two PNO domains.

5.7.2 Supported component interfaces

The interface supported by the CMA-NMS defines the management functionality of the FORE switch, used by the Xuser agent. The following functions are implemented (taken from the interface file snmp-ifc.h).

All functions return an int, which is the status of table entry. The return values are:

2 - for creation (reservation) successful
1 - for validation (activation) successful
4 - for invalidation (deactivation) successful
-1 - for operation failed

```
int reservePath( CommunityName communityName,
             IPAddress agentIPAdd,
             Port inputPort,
             Port outputPort,
             enum Directionality directionality,
             struct TrafficDescriptor trafficDescript,
             enum QosClass qosClass,
             enum Reason reason,
             Vpi* inputVPI,
             Vpi* outputVPI
             );

int activatePath( CommunityName communityName,
              IPAddress agentIPAdd,
              Port inputPort,
              Port outputPort,
              enum Directionality directionality,
              enum Reason reason,
              Vpi inputVPI,
              Vpi outputVPI
              );

int deActivatePath( CommunityName communityName,
                IPAddress agentIPAdd,
                Port inputPort,
                Port outputPort,
                enum Directionality directionality,
                enum Reason reason,
                Vpi inputVPI,
                Vpi outputVPI
                );
```



```
int releasePath( CommunityName communityName,
                 IPAddress agentIPAdd,
                 Port inputPort,
                 Port outputPort,
                 enum Directionality directionality,
                 enum Reason reason,
                 Vpi inputVPI,
                 Vpi outputVPI
                 );
```

5.7.3 Version, Release history

CMA-NMS 1.0 (unique version for the Sophia demo), September 1998.

5.8 PNO Messaging System Adapter

5.8.1 Engineering Object Model

The PNO messaging system is part of the Trumpet messaging system which is shown in the following figure. The Trumpet messaging system is an architecture designed and implemented in order to capture and display security and management events happened inside the system during operation. Events can be security events coming from the Trumpet security package as well as management events.. Events can be generated in all three domains i.e. CPN/VASP/PNO.

- CPN domain. Events may be generated by the CPN Server.
- VASP domain. Events are generated by the Customer Server (management events) and by the Xuser Manager (both kind of events, security events through the SMASC).
- PNO domain. The Xuser Agent can generate events of both types, security events through the SMASC.

At the VASP level, management events generated in the CPN domain are sent to the Mediation Entity (ME) of the CMA system. In addition CMA receives management events form the Xuser Manager. The instance of the SELF that runs at the VASP level receives only security events from the SMASC. The instance of the SELF that runs at the PNO level receives both security and management events coming from the SMASC and Xuser Agent respectively. Shaded boxes in the following figure represent programs built from the CMA Framework.

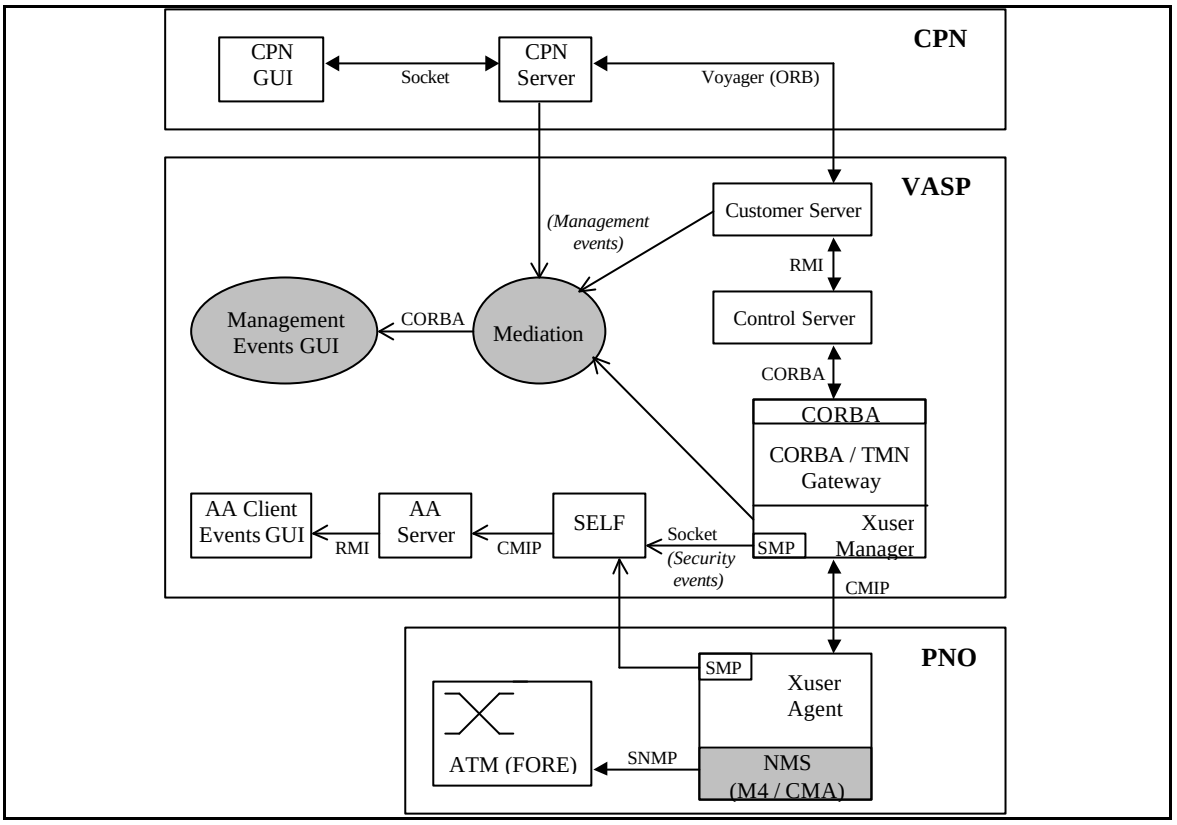

Figure 44: Trumpet Messaging System

Identification of Services

In order to better clarify a security or management event security and management service classes have been implemented. So we have five security service classes and four management service classes.

Security service classes

- Authentication service,
- Access Control service
- Key Management service
- Integrity/Confidentiality service

Management service classes

- CPN Management service
- Customer Server service
- Xuser VASP service
- Xuser PNO service

5.8.2 Supported component interfaces

There are two interfaces at the PNO level. The first one in between the Audit and Alarm Agent (AA Agent) and the Xuser/SMASC and the second one between the AA Agent and the Audit and Alarm Manager (AA Manager). The first one is realised using UNIX TCP/IP sockets and the second interface is realised using the CMIP protocol.

The Xuser agent communication defines the relevant trace points for TRUMPET. That means protocol exchanges during association establishment and release and the management operation primitives The relevant trace points are:

- start and stop of the agent
- Association requests and replies
- Association release requests and replies
- Incoming management operation indications (create/modify/delete)
- Outgoing management operation results & errors

Through SMASC the security events are sent to the Audit and Alarm agent. Both, Xuser agent and SMASC use the audit and alarm library, a library designed in order to accommodate management and security events before these are sent to the audit agent.

The basic function in this library is the send_info function which has the following parameters:

```
void send_info(
      enum  Service_Type service,
      int service_id,
      char  *EType,
      char  *Suser,
      char  *Sprovider,
      char  *Info_msg
);
```

The service_id parameter is used to identify the part of the system, a security or management event has been produced from. For the PNO level we have two identifiers denoting the two instances of the Xuser agent at this level.

Xuser Agent -> 4

Xuser Agent (second instance) -> 5

The EType parameter identifies the security events. The set of security events is left for discussion in section 6.8.2. The Suser parameter is used to identify the DN of the entity that caused the event. The Sprovider denotes the target entity. The Info_msg field is used to convey information relevant to a security or management event. In case of a security event information about the security profile or credentials and generally any kind of information that help identify better the event. In case of a management event the Info_msg field accommodates the trace message relevant to the management operation.

The interface between the AA Agent and the AA Manager is realised using the CMIP protocol and particularly the CMIS M-EVENT-REPORT. The parameters of this report is shown in the following table. More details of the contents of the report and how it applies to our implementation will be given in section 6.8.2.

Parameter name	Req/Ind	Rsp/Cnf
Invoke identifier	M	M(=)
Mode	M	-
Managed object class	M	U
Managed object instance	M	U
Event type	M	M(=)
Event time	-	U
Event information	U	-
Current time	-	U
Event reply	-	C
Errors	-	C

Table 5: M-EVENT-REPORT parameters

Mode: specifies the mode requested for the operation, either confirmed or non-confirmed.

Managed object class: identifies the class of the object in which the event occurred.

Managed object instance: identifies the object in which the event occurred.

Event type: specifies the type of the event to be reported.

Event time: time of the event generation

Event information: the information provided by the service reporting the event, i.e.

- Service report cause: the following causes are defined:
- request for service (event generated because of a request for the provision of a service);
- denial of service: a request for service has been denied;
- response from service: a request for service has been satisfied;
- service failure: an abnormal condition that caused the service to fail has been detected during the provision of a service;
- service recovery: a service has recovered from an abnormal condition;
- other reason: the actual cause is specified in the other parameters of the report.
- Notification identifier: UNUSED
- Correlated Notifications: UNUSED
- Additional text: planned to be used by the audit trail analysers and either used by the security alarm application or the recovery management application;
- Additional information: planned to be used by the audit trail analysers and either used by the security alarm application or the recovery management application.

Current time:[appears in the response to the request] time at which the response was generated

Event reply: [appears in the response to the request] the reply to the event report.

Errors: [appears in the response to the request] error notification for the operation.

5.8.3 Version, Release history

The CMIP interface is built inside the HP-OV platform. The Audit and Alarm (AA) library latest release is version 3 , July 1998.

5.9 VASP and PNO Management Event Reporting

5.9.1 Engineering Object Model

The CMA TRUMPET Messaging System (CMA-TMS) for management events supports two related kinds of operations: creation of EFDs in the ME, and graphical display of events in the GUI AE.

5.9.1.1 Creation of EFDs

Figure 45 shows the object model of the CMA-TMS concerned with the creation of EFDs.

To create an EFD in the ME, a graphical panel is available for easy definition of the filters to be set. The corresponding command is sent via CORBA through the MOSE and reaches an EFD handler in the ME. This handler checks the validity of the filter and creates the new EFD object in the ME.

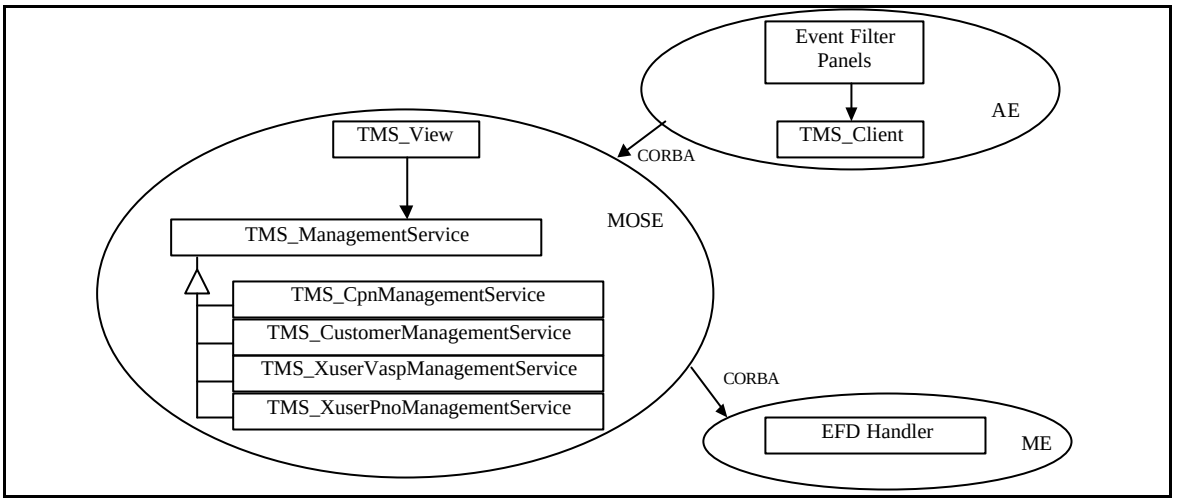

Figure 45 : Creation of EFDs in the CMA Based Messaging System

5.9.1.2 Graphical Display of Events

The processing of TRUMPET events by the CMA based system is pictured on Figure 46. At the ME level, an incoming event is processed in the following way:

- A character string is received by socket from some process in the Trumpet system. This handled by a special “Socket Agent Manager”, which also parses the string to recognize its different fields
- From the information in the string, an X.721 alarm object is created in the internal CMA representation. This is done by an “Event Handler” object.
- The event handler object compares the alarm object to all the EFDs present in the ME and, for every match, it forwards the event through the MOSE to the interested applications.

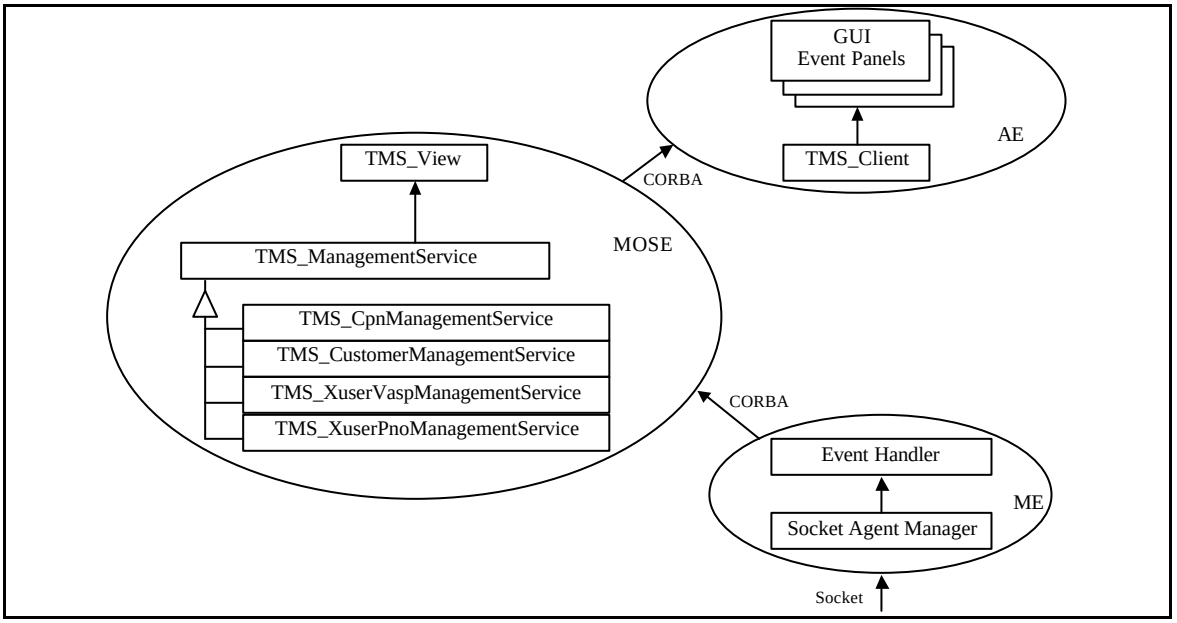

Figure 46 : CMA Processing of TRUMPET Management Events

5.9.2 Supported component interfaces

The role of the Messaging System is to display graphically event reports emitted by various servers in the TRUMPET management system. The interface offered by the CMA-TMS to the TRUMPET servers for event reporting is at the socket level, like the interface offered by the security AA (Audit and Alarm) application for the processing of events concerned with security aspects (see Section 6.7).

This section presents the specifications of the events that the CMA-TMS can handle: TRUMPET programs that emit management events should follow them to format the information sent over the socket to the CMA-TMS so it can be processed correctly.

Security and Management events in all the TRUMPET system share a common model to represent management entities and a common structure for the event information (see Section 6.7). In this model, all events contain the following information:

- Managed Object Class (MOC)
- Managed Object Instance (MOI)
- Specific event information

The MOC and MOI identify an object that may emit an event. Objects that generate events are seen as instances of some services, either security services or management services, as defined in the following list of MOCs:

0 : AuthenticationService
1 : KeyManagementService
2 : IntegrityConfidentialityService
3 : AccessControlService
4 : CpnManagementService
5 : CustomerManagementService
6 : XuserVaspManagementService
7 : XuserPnoManagementService

The MOC of an object that generated an event is recognized by using an integer code in the event data. The MOI of the object is determined by using another integer code, the "serviceId". The serviceId is a number that uniquely identifies all the management entities in the current configuration of the TRUMPET system. The attribution of the serviceId is agreed as a convention by TRUMPET developers:

1 : CPN server
2 : Customer server
3 : Xuser Manager
4 : Xuser Agent
5 : Xuser Agent

New entries may be added if the system grows.

For the management events handled by the CMA system, the information model is inspired from the X.740 ITU standard, which defines two notification types: serviceReport and usageReport. Currently, we use only the serviceReport type. The OIDs are defined as follows in X.740:

securityAuditTrail-Notification = 2.9.2.8.10

serviceReport notification OID = 2.9.2.8.10.1
serviceReportCause OID = 2.9.2.8.0.1

serviceRequest ServiceReportCause ::= {serviceReportCause 1}

serviceResponse ServiceReportCause ::= {serviceReportCause 3}

The information contained in a management event is:

1. **service Id**. This is the integer code converted as a string.

2. **MOC**. This is the integer code of the management service converted to character string.
3. **event type**. This is the OID of the notification. In this case, it will be constant and equal to 2.9.2.8.10.1 (serviceReport notification).
4. **serviceReportCause**. This is an attribute of the serviceReport notification. This attribute is an OID, one of:

   serviceRequest = 2.9.2.8.0.1.1

   serviceResponse = 2.9.2.8.0.1.3
5. **Action name**. A character string that identifies the action to report. The exact name may differ from one entity to the other (e.g. what the Xuser manager calls "reserveGBCConnection" may be known as "Reserve Connection" for the customer server).
6. **Action Info**. A character string conveying any relevant information (e.g. some of the action arguments or part of the action result).

For each event, all of the above 6 parts of information must be present in the event data as character strings, concatenated by using the character '$' as separator. The resulting string is then sent to the ME over a TCP socket. The ME hostname and port number will be specified in a configuration file. These parameters can be read directly from the file by the client processes (CPN server, customer server, Xuser manager), or provided as command line arguments, or set as environment variables.

The concerned servers (CPN server, customer server, Xuser manager) should generate an event each time they are about to call or they have just returned from a management operation: reserve connection, modify connection, or release connection. To summarize:

"<serviceId>$<MOC>$2.9.2.8.10.1$<service report cause>$<action name>$<action info>"

As an example, the Xuser manager may generate an event containing the following data, just before sending a CMIP action request for connection reservation to the PNOs:

"3$6$2.9.2.8.10.1$2.9.2.8.0.1.1$reserveGBCConnection$userId = 876394426 / destination = 123456789 / gBCDirectionality = 0"

The above string is decoded by the CMA ME into the following information:

3 :	serviceId of the Xuser manager
6 :	code of the XuserVaspManagementService (MOC)
2.9.2.8.10.1 :	event type (constant, OID for serviceReport notification)
2.9.2.8.0.1.1 :	service report cause (here it is serviceRequest)
reserveGBCConnection :	name of the action called by the Xuser manager.
userId = 876394426 ...etc... :	any additional information as text.

5.9.3 Version, Release history

CMA TRUMPET Messaging System 1.0 (version used for the Scottish and Sophia trials), May 1998.

6 SECURITY DEVELOPERS MANUAL

6.1 Adapter Object

6.1.1 Engineering Object Model

The X/Open Management Protocol (XMP) API provides a common access mecanism to both CMIP and SNMP services. XMP employs the X/Open OSI-Abstract-Data Manipulation (XOM) API to manipulate variables and parameters.

XMP consists of a library of C functions that reflect the services of CMIS and SNMP. It embodies the object-oriented OSI model of management. Manager applications use XMP to access managed objects, and agents applications use XMP to respond. XMP can be used synchronously or asynchronously.

The functions in XOM are used to create, examine, modify and destroy the arguments to XMP functions. It provides a generalized data handling mechanism, and manipulates data types that arise from Abstract Syntax Notation 1 (ASN.1) definitions.

As reflected in Figure 47, all network management applications can use XMP, with XOM, for standards-based process-to-process communications.

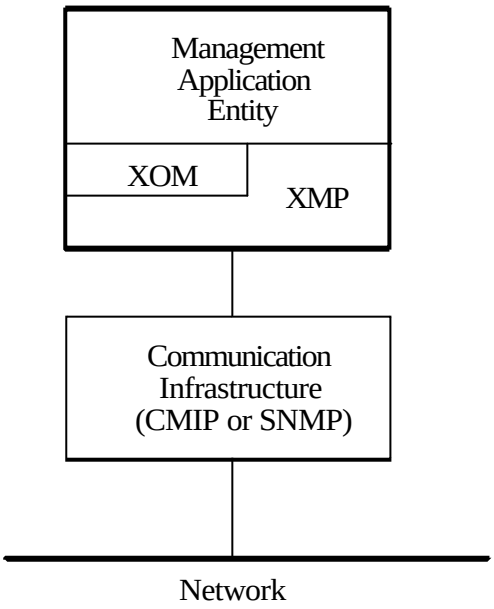

Figure 47: XOM, XMP and MAE

The XMP Adapter is responsible for securing the XMP function calls, which in turn map to CMIP requests and responses.

The approach chosen in Trumpet is to modify the MAE's source code to include security-related calls to the adapter where appropriate. The MAE is then made "security-aware".

As the adapter must protect all PDU exchanged between manager and agents applications, implementers have to insert security-related code before XMP "send request" function calls to protect the outgoing PDU ; and after XMP "get response" function calls to deprotect the incoming PDU.

The adapter interface is designed to simplify this process, as all the functions defined closely match that of XMP, having the same name and accepting similar arguments.

6.1.1.1 Secured Association Establishment

When using XMP, you can either rely on TMN Platform Infrastructure to control associations, providing automatic establishment and release, or you can retain application control over these functions. The Association Control Service Element (ACSE) extensions to the XMP API give you this capability.

Explicit association control is very useful in certain situations :

- an application may need to set certain ACSE parameters in order to interoperate with peer applications based on other TMN Platforms,
- with explicit association control, an application can establish multiple associations with a peer, each association having its own context. This can be useful if several kinds of transactions are occurring between pair of applications,
- a management association established using the XMP ACSE extensions is never terminated unless one side explicitly terminates it, or unless connectivity between the two sides is lost. Therefore, a management application can set up an association and use it as a test of connectivity with the remote peer : connectivity can be tested at any time by sending a message on that association.

Management associations established with Automatic Connection Management (ACM) are **shared associations**, that can be used by any Management Application Entity residing on the same host. By contrast, a management association established by two applications, both using the ASCE extensions, is a **private association** between those applications, and can only be used by them.

When using the Trumpet security package, secured associations estalibshed between applications must be private : each secured association rely on a distinct security context that must be negotiated during the association establishment, and terminated when releasing the association. Therefore, the ACSE extensions must be used by client Management Application Entities when they want to establish secured associations.

The XMP ACSE extensions add five functions to the XMP API. They are listed and descibed in Table 4. For more details, see the manpage for each function.

ASCE Function	Desciption
`mp_assoc_req()`	Called after mp_bind() to request the establishment of a connected session.
`mp_assoc_rsp()`	Used to reply to a previously invoked association request.
`mp_release_req()`	Used to request the release of a connected session.
`mp_release_rsp()`	Used to reply to a previously invoked release request.
`mp_abort_req()`	Used to abort a management association. This service is defined as non-confirmed.

Table 6 : ACSE Functions

The XMP Adapter provides a set of functions that are responsible for negociating the security context. Calls to these functions must be inserted in the MAE's source code before any ACSE function is called. They act by inserting and/or transforming proper parameters in the XOM structures that are later used by the ACSE functions.

Table 5 lists and describes the ACSE-related functions that are provided by the XMP Adapter.

XMP Adapter Function	Desciption
`sp_assoc_req()`	Called before mp_assoc_req() to supply the necessary security parameters for the establishment of a connected session.
`sp_assoc_rsp()`	Called before mp_assoc_rsp() to supply the necessary security parameters for the reply to a previously invoked association request.
`sp_release_req()`	Called before mp_release_req() to supply the

	necessary security parameters to request the release of a connected session.
`sp_release_rsp()`	Called before mp_release_rsp() to supply the necessary security parameters to release a connected session.
`sp_abort_req()`	Called before mp_release_rsp() to supply the necessary security parameters to abort a connected session.

Table 7 : ACSE-related Adapter functions

Figure 48 outlines the steps needed in order to establish and release a secured association. For the sake of simplicity, calls to XOM functions are not presented, but the main parameters needed to XMP functions are indicated where appropriate.

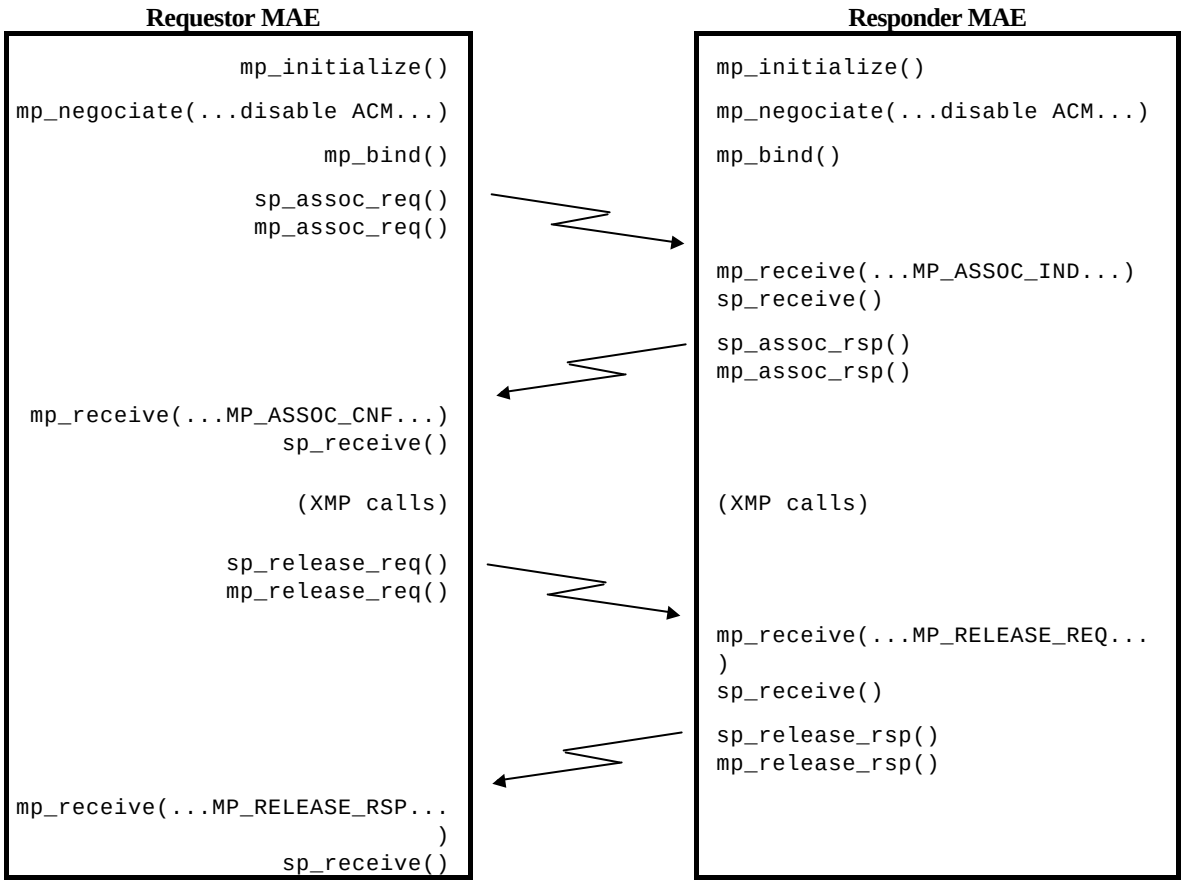

Figure 48: Secured association establishment & release

The sequence of operations is exactly the same as for a non-secured association ; calls to the Trumpet security package are just are inserted before all XMP functions that send data and after all XMP functions that receive data (the `mp_receive()` and `sp_receive()` functions are described later).

6.1.1.2 Secured XMP Requests

As previously mentionned, XMP supports the seven CMIS services through the CMIS OM package. The CMIS services are mapped to XMP function names, as shown in Table 6.

CMIS service	XMP functions	Description (of request only)

ACTION	`mp_action_req()` `mp_action_rsp()`	Requests that the responder perform one of the actions defined for an object.
CANCEL-GET	`mp_cancel_get_req()` `mp_cancel_get_rsp()`	Requests that the responder terminate servicing an earlier "get" request that has not yet completed.
CREATE	`mp_create_req()` `mp_create_rsp()`	Requests that the responder create an instance (object) of the specified object class.
DELETE	`mp_delete_req()` `mp_delete_rsp()`	Requests that the responder destroy a particular instance (object) of an object class.
EVENT-REPORT	`mp_event_report_req()` `mp_event_report_rsp()`	Issues one of the notifications (events) defined for a managed object.
GET	`mp_get_req()` `mp_get_rsp()`	Requests that the responder supply the value(s) of one or more object attributes.
SET	`mp_set_req()` `mp_set_rsp()`	Requests that the responder modify the value(s) of one or more object attributes.

Table 8 : XMP functions supporting CMIS services

As mentioned earlier, security-related calls to the adapter have to be made before XMP "send request" function calls and after XMP "get response" function call.

These calls are described in Table 7 below, they basically share the same input parameters as the corresponding XMP function calls. Their role is either to protect or deprotect an XOM object, provided as an input argument, and providing the result in an output OM object.

Protection of the OM input arguments, depending on the security context and policy requirements, can consist of confidentiality and/or integrity.

The returned protected XOM objects must not be modified or tampered in any way after they are produced ; they can only be send to the peer entity through the corresponding XMP call.

CMIS service	XMP functions	Corresponding Adapter functions
ACTION	`mp_action_req()` `mp_action_rsp()`	`sp_action_req()` `sp_action_rsp()`
CANCEL-GET	`mp_cancel_get_req()` `mp_cancel_get_rsp()`	`sp_cancel_get_req()` `sp_cancel_get_rsp()`
CREATE	`mp_create_req()` `mp_create_rsp()`	`sp_create_req()` `sp_create_rsp()`
DELETE	`mp_delete_req()` `mp_delete_rsp()`	`sp_delete_req()` `sp_delete_rsp()`
EVENT-REPORT	`mp_event_report_req()` `mp_event_report_rsp()`	`sp_event_report_req()` `sp_event_report_rsp()`
GET	`mp_get_req()` `mp_get_rsp()`	`sp_get_req()` `sp_get_rsp()`
SET	`mp_set_req()` `mp_set_rsp()`	`sp_set_req()` `sp_set_rsp()`

Table 9 : CMIS-related Adapter functions

6.1.1.3 Secured Asynchronous Operations

When a synchronous function call is performed, the function does not return unless the effect of the call is complete. In opposition, asynchronous function calls do start some process and return. They are used by applications that need to do multiple independent function calls, an example being a network management application that interrogates multiple distinct network equipment.

The XMP API allows you to make any call (except `mp_cancel_get_req()`) synchronously, and to use any requester function asynchronously.

When you make synchronous requester calls, the parameters returned by the responder application are available through the `result_return` OM object, provided the request was successfully processed.

When you make an asynchronous function call, the XMP interface first determines if the call is valid. If so, the transaction with the responder is initiated. Your application is then allowed to continue processing while the request is serviced. If a response is expected, you must later call `mp_receive()` to determine the outcome of the request.

When such asynchronous function calls are secured with the Trumpet security package, the requester application must call the `sp_receive()` function in order to exploit the information provided by the `mp_receive()` function.

On the responder side, the application has no knowledge whether the call is performed synchronously or asynchronously, therefore the sequence of operations is the same as described in the previous chapter.

6.1.2 Supported component interfaces

NAME

sp_abort_req - Protects the parameters to an ACSE association abort request.

SYNOPSIS

```
#include <xom.h>
#include <tsp.h>

SP_status sp_assoc_req(
     OM_private_object session,
     OM_private_object context,
     OM_object argument
     OM_object *protected);
```

DESCRIPTION

This function is used to protect the parameters to an ACSE association abort request just before the corresponding XMP function call is made.

Parameters

session	The OM object against which the operation will be performed. It must be a private OM object previously retumed as part of an **Assoc-Argument** or **Assoc-Result** object. This object must belong to an ACM-disabled workspace.
context	Represents the management context to be used for this operation. This must be a private OM object or the constant **Default-Context** { MP_DEFAULT_CONTEXT }.
argument	The unprotected information supplied as the argument of an Abort operation. It is an instance of a subclass of the OM class **Abort-Argument**.
protected	Returned upon successful completion of the function call. The protected information that is to be supplied to `mp_abort_req()`.

RESULTS

This function returns :

MP_SUCCESS

MP_NO_WORKSPACE

MP_INVALID_SESSION

MP_INSUFFICIENT_RESSOURCES

MP_ACCESS_CONTROL_FAILURE

NAME

sp_action_req - Protects the parameters to a CMIS Action request.

SYNOPSIS

```
#include <xom.h>
#include <tsp.h>

SP_status sp_action_req(
     OM_private_object session,
     OM_private_object context,
     OM_object argument,
     OM_object *protected);
```

DESCRIPTION

This function is used to protect the parameters to a CMIS Action request just before the corresponding XMP function call is made.

Parameters

session	An OM object that identifies the management session in which the operation/notification will be requested. This must be a private OM object previously returned from `mp_bind()`. This object must belong to an ACM-disabled workspace.
context	The context in which the operation should be performed. This must be a private OM object or the constant **Default-Context** { MP_DEFAULT_CONTEXT }.
argument	An unprotected OM object which provides the information about the Action request and the data for that action. It is an instance of a subclass of the OM class **Action-Argument**.
protected	Returned upon successful completion of the function call. The protected information that is to be supplied to `mp_action_req()`.

RESULTS

This function returns :

MP_SUCCESS

MP_NO_WORKSPACE

MP_INVALID_SESSION

MP_INSUFFICIENT_RESSOURCES

SP_NO_SECURITY_CONTEXT

SP_ACCESS_CONTROL_FAILURE

SP_AUDIT_ALARM_FAILURE

NAME

sp_action_rsp - Protects the parameters to a CMIS Action reply.

SYNOPSIS

```
#include <xom.h>
#include <tsp.h>

SP_status sp_action_rsp(
     OM_private_object session,
     OM_private_object context,
     OM_object response,
     OM_object *protected);
```

DESCRIPTION

This function is used to protect the parameters to a CMIS Action reply just before the corresponding XMP function call is made.

Parameters

session	An OM object that identifies the management session in which the operation/notification was requested. This must be a private OM object previously returned from `mp_bind()`. This object must belong to an ACM-disabled workspace.
context	The context in which the operation should be performed. This must be a private OM object or the constant **Default-Context** { MP_DEFAULT_CONTEXT }.
response	An unprotected OM object supplied as a response information about the Action request. It is an instance of one of the following OM classes : **Action-Result**, **Linked-Reply-Argument**, **Absent-Object** or **Service-Error**.
protected	Returned upon successful completion of the function call. The protected information that is to be supplied to `mp_action_rsp()`.

RESULTS

This function returns :

MP_SUCCESS

MP_NO_WORKSPACE

MP_INVALID_SESSION

MP_INSUFFICIENT_RESSOURCES

SP_NO_SECURITY_CONTEXT

SP_ACCESS_CONTROL_FAILURE

SP_AUDIT_ALARM_FAILURE

NAME

sp_assoc_req - Protects the parameters to an ACSE association establishment request.

SYNOPSIS

```
#include <xom.h>
#include <tsp.h>

SP_status sp_assoc_req(
     OM_private_object session,
     OM_private_object context,
     OM_object argument
     OM_object *protected);
```

DESCRIPTION

This function is used to protect the parameters to an ACSE association establishment request just before the corresponding XMP function call is made.

Parameters

session	The OM object against which the operation will be performed. It must be a private OM object previously returned from `mp_bind()`. This object must belong to an ACM-disabled workspace.
context	Represents the management context to be used for this operation. This must be a private OM object or the constant **Default-Context** { MP_DEFAULT_CONTEXT }.
argument	The unprotected information that represents the argument of an Associate operation. It is an instance of a subclass of the OM class **Assoc-Argument**.
protected	Returned upon successful completion of the function call. The protected information that is to be supplied to `mp_assoc_req()`.

RESULTS

This function returns :

MP_SUCCESS

MP_NO_WORKSPACE

MP_INVALID_SESSION

MP_INSUFFICIENT_RESSOURCES

MP_ACCESS_CONTROL_FAILURE

NAME

sp_assoc_rsp - Protects the parameters to an ACSE association establishment reply.

SYNOPSIS

```
#include <xom.h>
#include <tsp.h>

SP_status sp_assoc_rsp(
     OM_private_object session,
     OM_private_object context,
     OM_object response
     OM_object *protected);
```

DESCRIPTION

This function is used to protect the parameters to an ACSE association establishment reply just before the corresponding XMP function call is made.

Parameters

session	The OM object against which the operation will be performed. It must be a private OM object previously returned from `mp_bind()`. This object must belong to an ACM-disabled workspace.
context	Represents the management context to be used for this operation. This must be a private OM object or the constant **Default-Context** { MP_DEFAULT_CONTEXT }.
response	The unprotected information supplied as a response of an Associate operation. It is an instance of a subclass of the OM class **Assoc-Result**.
protected	Returned upon successful completion of the function call. The protected information that is to be supplied to `mp_assoc_rsp()`.

RESULTS

This function returns :

MP_SUCCESS

MP_NO_WORKSPACE

MP_INVALID_SESSION

MP_INSUFFICIENT_RESSOURCES

SP_ACCESS_CONTROL_FAILURE

NAME

sp_cancel_get_req - Protects the parameters to a CMIS Cancel-Get request.

SYNOPSIS

```
#include <xom.h>
#include <tsp.h>

SP_status sp_cancel_get_req(
     OM_private_object session,
     OM_private_object context,
     OM_object argument,
     OM_object *protected);
```

DESCRIPTION

This function is used to protect the parameters to a CMIS Cancel-Get request just before the corresponding XMP function call is made.

Parameters

session	An OM object that identifies the management session in which the operation will be requested. This must be a private OM object previously returned from `mp_bind()`. This object must belong to an ACM-disabled workspace.
context	The context in which the operation should be performed. This must be a private OM object or the constant **Default-Context** { MP_DEFAULT_CONTEXT }.
argument	An unprotected OM object which provides the information about which Get operation is to be cancelled. It is an instance of a subclass of the OM class **Cancel-Get-Argument**.
protected	Returned upon successful completion of the function call. The protected information that is to be supplied to `mp_cancel_get_req()`.

RESULTS

This function returns :

MP_SUCCESS

MP_NO_WORKSPACE

MP_INVALID_SESSION

MP_INSUFFICIENT_RESSOURCES

SP_NO_SECURITY_CONTEXT

SP_ACCESS_CONTROL_FAILURE

SP_AUDIT_ALARM_FAILURE

NAME

sp_cancel_get_rsp - Protects the parameters to a CMIS Cancel-Get reply.

SYNOPSIS

```
#include <xom.h>
#include <tsp.h>

SP_status sp_cancel_get_rsp(
     OM_private_object session,
     OM_private_object context,
     OM_object response,
     OM_object *protected);
```

DESCRIPTION

This function is used to protect the parameters to a CMIS Cancel-Get reply just before the corresponding XMP function call is made.

Parameters

session	An OM object that identifies the management session in which the operation will be requested. This must be a private OM object previously returned from `mp_bind()`. This object must belong to an ACM-disabled workspace.
context	The context in which the operation should be performed. This must be a private OM object or the constant **Default-Context** { MP_DEFAULT_CONTEXT }.
response	An unprotected OM object supplied as a response information about the result of the Cancel-Get operation. It is an instance of one of the following OM classes : **Absent-Object** or **Service-Error**.
protected	Returned upon successful completion of the function call. The protected information that is to be supplied to `mp_cancel_get_rsp()`.

RESULTS

This function returns :

MP_SUCCESS

MP_NO_WORKSPACE

MP_INVALID_SESSION

MP_INSUFFICIENT_RESSOURCES

SP_NO_SECURITY_CONTEXT

SP_ACCESS_CONTROL_FAILURE

SP_AUDIT_ALARM_FAILURE

NAME

sp_create_req - Protects the parameters to a CMIS Create request.

SYNOPSIS

```
#include <xom.h>
#include <tsp.h>

SP_status sp_create_req(
     OM_private_object session,
     OM_private_object context,
     OM_object argument,
     OM_object *protected);
```

DESCRIPTION

This function is used to protect the parameters to a CMIS Create request just before the corresponding XMP function call is made.

Parameters

session	An OM object that identifies the management session in which the operation will be requested. This must be a private OM object previously returned from `mp_bind()`. This object must belong to an ACM-disabled workspace.
context	The context in which the operation should be performed. This must be a private OM object or the constant **Default-Context** { MP_DEFAULT_CONTEXT }.
argument	An unprotected OM object which provides the information about the managed object to create and any data values attributes of the managed object. It is an instance of a subclass of the OM class **Create-Argument**.
protected	Returned upon successful completion of the function call. The protected information that is to be supplied to `mp_action_req()`.

RESULTS

This function returns :

MP_SUCCESS

MP_NO_WORKSPACE

MP_INVALID_SESSION

MP_INSUFFICIENT_RESSOURCES

SP_NO_SECURITY_CONTEXT

SP_ACCESS_CONTROL_FAILURE

SP_AUDIT_ALARM_FAILURE

NAME

sp_create_rsp - Protects the parameters to a CMIS Create reply.

SYNOPSIS

```
#include <xom.h>
#include <tsp.h>

SP_status sp_create_rsp(
     OM_private_object session,
     OM_private_object context,
     OM_object response,
     OM_object *protected);
```

DESCRIPTION

This function is used to protect the parameters to a CMIS Create reply just before the corresponding XMP function call is made.

Parameters

session	An OM object that identifies the management session in which the operation will be requested. This must be a private OM object previously returned from `mp_bind()`. This object must belong to an ACM-disabled workspace.
context	The context in which the operation should be performed. This must be a private OM object or the constant **Default-Context** { MP_DEFAULT_CONTEXT }.
response	An unprotected OM object supplied as a response information about the Create request. It is an instance of one of the following OM classes : **Create-Result**, **Absent-Object** or **Service-Error**.
protected	Returned upon successful completion of the function call. The protected information that is to be supplied to `mp_create_rsp()`.

RESULTS

This function returns :

MP_SUCCESS

MP_NO_WORKSPACE

MP_INVALID_SESSION

MP_INSUFFICIENT_RESSOURCES

SP_NO_SECURITY_CONTEXT

SP_ACCESS_CONTROL_FAILURE

SP_AUDIT_ALARM_FAILURE

NAME

sp_delete_req - Protects the parameters to a CMIS Delete request.

SYNOPSIS

```
#include <xom.h>
#include <tsp.h>

SP_status sp_delete_req(
     OM_private_object session,
     OM_private_object context,
     OM_object argument,
     OM_object *protected);
```

DESCRIPTION

This function is used to protect the parameters to a CMIS Delete request just before the corresponding XMP function call is made.

Parameters

session	An OM object that identifies the management session in which the operation will be requested. This must be a private OM object previously returned from `mp_bind()`. This object must belong to an ACM-disabled workspace.
context	The context in which the operation should be performed. This must be a private OM object or the constant **Default-Context** { MP_DEFAULT_CONTEXT }.
argument	An unprotected OM object which provides the information about the managed object to delete. It is an instance of a subclass of the OM class **Delete-Argument**.
protected	Returned upon successful completion of the function call. The protected information that is to be supplied to `mp_delete_req()`.

RESULTS

This function returns :

MP_SUCCESS

MP_NO_WORKSPACE

MP_INVALID_SESSION

MP_INSUFFICIENT_RESSOURCES

SP_NO_SECURITY_CONTEXT

SP_ACCESS_CONTROL_FAILURE

SP_AUDIT_ALARM_FAILURE

NAME

sp_delete_rsp - Protects the parameters to a CMIS Delete reply.

SYNOPSIS

```
#include <xom.h>
#include <tsp.h>

SP_status sp_delete_rsp(
     OM_private_object session,
     OM_private_object context,
     OM_object response,
     OM_object *protected);
```

DESCRIPTION

This function is used to protect the parameters to a CMIS Delete reply just before the corresponding XMP function call is made.

Parameters

session	An OM object that identifies the management session in which the operation will be requested. This must be a private OM object previously returned from `mp_bind()`. This object must belong to an ACM-disabled workspace.
context	The context in which the operation should be performed. This must be a private OM object or the constant **Default-Context** { MP_DEFAULT_CONTEXT }.
response	An unprotected OM object supplied as a response information about the Delete operation. It is an instance of one of the following OM classes : **Delete-Result**, **Linked-Reply-Argument**, **Absent-Object** or **Service-Error**.
protected	Returned upon successful completion of the function call. The protected information that is to be supplied to `mp_delete_rsp()`.

RESULTS

This function returns :

MP_SUCCESS

MP_NO_WORKSPACE

MP_INVALID_SESSION

MP_INSUFFICIENT_RESSOURCES

SP_NO_SECURITY_CONTEXT

SP_ACCESS_CONTROL_FAILURE

SP_AUDIT_ALARM_FAILURE

NAME

sp_event_report_req - Protects the parameters to a CMIS Event-Report request.

SYNOPSIS

```
#include <xom.h>
#include <tsp.h>

SP_status sp_event_report_req(
     OM_private_object session,
     OM_private_object context,
     OM_object argument,
     OM_object *protected);
```

DESCRIPTION

This function is used to protect the parameters to a CMIS Event-Report request just before the corresponding XMP function call is made.

Parameters

session	An OM object that identifies the management session in which the operation will be requested. This must be a private OM object previously returned from `mp_bind()`. This object must belong to an ACM-disabled workspace.
context	The context in which the operation should be performed. This must be a private OM object or the constant **Default-Context** { MP_DEFAULT_CONTEXT }.
argument	An unprotected OM object which provides the information about the event to be generated. It is an instance of a subclass of the OM class **Event-Report-Argument**.
protected	Returned upon successful completion of the function call. The protected information that is to be supplied to `mp_event_report_req()`.

RESULTS

This function returns :

MP_SUCCESS

MP_NO_WORKSPACE

MP_INVALID_SESSION

MP_INSUFFICIENT_RESSOURCES

SP_NO_SECURITY_CONTEXT

SP_ACCESS_CONTROL_FAILURE

SP_AUDIT_ALARM_FAILURE

NAME

sp_event_report_rsp - Protects the parameters to a CMIS Event-Report reply.

SYNOPSIS

```
#include <xom.h>
#include <tsp.h>

SP_status sp_event_report_rsp(
     OM_private_object session,
     OM_private_object context,
     OM_object response,
     OM_object *protected);
```

DESCRIPTION

This function is used to protect the parameters to a CMIS Event-Report reply just before the corresponding XMP function call is made.

Parameters

session	An OM object that identifies the management session in which the operation will be requested. This must be a private OM object previously returned from `mp_bind()`. This object must belong to an ACM-disabled workspace.
context	The context in which the operation should be performed. This must be a private OM object or the constant **Default-Context** { MP_DEFAULT_CONTEXT }.
response	An unprotected OM object supplied as a response information about the Event-Report. It is an instance of one of the following OM classes : **Event-Report-Result**, **Absent-Object** or **Service-Error**.
protected	Returned upon successful completion of the function call. The protected information that is to be supplied to `mp_event_report_rsp()`.

RESULTS

This function returns :

MP_SUCCESS

MP_NO_WORKSPACE

MP_INVALID_SESSION

MP_INSUFFICIENT_RESSOURCES

SP_NO_SECURITY_CONTEXT

SP_ACCESS_CONTROL_FAILURE

SP_AUDIT_ALARM_FAILURE

NAME

sp_get_req - Protects the parameters to a Get request.

SYNOPSIS

```
#include <xom.h>
#include <tsp.h>

SP_status sp_get_req(
     OM_private_object session,
     OM_private_object context,
     OM_object argument,
     OM_object *protected);
```

DESCRIPTION

This function is used to protect the parameters to a Get request just before the corresponding XMP function call is made.

Parameters

session	An OM object that identifies the management session in which the operation will be requested. This must be a private OM object previously returned from `mp_bind()`. This object must belong to an ACM-disabled workspace.
context	The context in which the operation should be performed. This must be a private OM object or the constant **Default-Context** { MP_DEFAULT_CONTEXT }.
argument	An unprotected OM object which provides the information about which attributes are to be retrieved. It is an instance of a subclass of the OM class **Get-Argument**.
protected	Returned upon successful completion of the function call. The protected information that is to be supplied to `mp_get_req()`.

RESULTS

This function returns :

MP_SUCCESS

MP_NO_WORKSPACE

MP_INVALID_SESSION

MP_INSUFFICIENT_RESSOURCES

SP_NO_SECURITY_CONTEXT

SP_ACCESS_CONTROL_FAILURE

SP_AUDIT_ALARM_FAILURE

NAME

sp_get_rsp - Protects the parameters to a Get reply.

SYNOPSIS

```
#include <xom.h>
#include <tsp.h>

SP_status sp_action_rsp(
     OM_private_object session,
     OM_private_object context,
     OM_object response,
     OM_object *protected);
```

DESCRIPTION

This function is used to protect the parameters to a Get reply just before the corresponding XMP function call is made.

Parameters

session	An OM object that identifies the management session in which the operation will be requested. This must be a private OM object previously returned from `mp_bind()`. This object must belong to an ACM-disabled workspace.
context	The context in which the operation should be performed. This must be a private OM object or the constant **Default-Context** { MP_DEFAULT_CONTEXT }.
response	An unprotected OM object supplied as a response information about the Get operation. It is an instance of one of the following OM classes : **Get-Result**, **Linked-Reply-Argument**, **Absent-Object** or **Service-Error**.
protected	Returned upon successful completion of the function call. The protected information that is to be supplied to `mp_get_rsp()`.

RESULTS

This function returns :

MP_SUCCESS

MP_NO_WORKSPACE

MP_INVALID_SESSION

MP_INSUFFICIENT_RESSOURCES

SP_NO_SECURITY_CONTEXT

SP_ACCESS_CONTROL_FAILURE

SP_AUDIT_ALARM_FAILURE

NAME

sp_receive - Unprotects the result or notification to an asynchronous operation.

SYNOPSIS

```
#include <xom.h>
#include <tsp.h>

SP_status sp_receive(
     OM_private_object session,
     OM_private_object context,
     OM_object protected,
     OM_object *result);
```

DESCRIPTION

This function is used to unprotect the partial or complete result of an invoked management operation, or its reported management notification.

Parameters

session	An OM object that identifies the management session in which the operation or notification was performed. This must be a private OM object previously returned from `mp_bind()`. This object must belong to an ACM-disabled workspace.
context	The context in which the operation was performed. This must be a private OM object ; the constant **Default-Context** { MP_DEFAULT_CONTEXT } is explicitly not permitted.
protected	An protected OM object obtained through a preceeding `mp_receive()`function call. The abstract class of this object is dependent on the value of the *primitive* and *completion_flag* parameters. It might be an instance of one of the following OM classes : **Action-Argument, Action-Result, Cancel-Get-Argument, Absent-Object, Create-Argument, Create-Result, Delete-Argument, Delete-Result, Event-Report-Argument, Event-Report-Result, Get-Argument, Get-Result, Set-Argument, Set-Result, Assoc-Argument, Assoc-Result, Release-Argument, Release-Result** or **Abort-Argument**.
protected	Returned upon successful completion of the function call. This object is of the same OM class as the *protected* argument.

RESULTS

This function returns :

MP_SUCCESS

MP_NO_WORKSPACE

MP_INVALID_SESSION

MP_INSUFFICIENT_RESSOURCES

SP_NO_SECURITY_CONTEXT

SP_AUDIT_ALARM_FAILURE

NAME

sp_release_req - Protects the parameters to an ACSE association release request.

SYNOPSIS

```
#include <xom.h>
#include <tsp.h>

SP_status sp_release_req(
     OM_private_object session,
     OM_private_object context,
     OM_object argument
     OM_object *protected);
```

DESCRIPTION

This function is used to protect the parameters to an ACSE association release request just before the corresponding XMP function call is made.

Parameters

session	The OM object against which the operation will be performed. It must be a private OM object previously returned as part of an **Assoc-Argument** or **Assoc-Result** object. This object must belong to an ACM-disabled workspace.
context	Represents the management context to be used for this operation. This must be a private OM object or the constant **Default-Context** { MP_DEFAULT_CONTEXT }.
argument	The unprotected information that represents the argument of a Release operation. It is an instance of a subclass of the OM class **Release-Argument**.
protected	Returned upon successful completion of the function call. The protected information that is to be supplied to `mp_release_req()`.

RESULTS

This function returns :

MP_SUCCESS

MP_NO_WORKSPACE

MP_INVALID_SESSION

MP_INSUFFICIENT_RESSOURCES

SP_NO_SECURITY_CONTEXT

SP_ACCESS_CONTROL_FAILURE

SP_AUDIT_ALARM_FAILURE

NAME

sp_release_rsp - Protects the parameters to an ACSE association release reply.

SYNOPSIS

```
#include <xom.h>
#include <tsp.h>

SP_status sp_assoc_rsp(
     OM_private_object session,
     OM_private_object context,
     OM_object response
     OM_object *protected);
```

DESCRIPTION

This function is used to protect the parameters to an ACSE association release reply just before the corresponding XMP function call is made.

Parameters

session	The OM object against which the operation will be performed. It must be a private OM object previously returned as part of an **Assoc_Argument** or **Assoc-Result** object. This object must belong to an ACM-disabled workspace.
context	Represents the management context to be used for this operation. This must be a private OM object or the constant **Default-Context** { MP_DEFAULT_CONTEXT }.
response	The unprotected information supplied as a response to a Release operation. It is an instance of a subclass of the OM class **Release-Result**.
protected	Returned upon successful completion of the function call. The protected information that is to be supplied to `mp_release_rsp()`.

RESULTS

This function returns :

MP_SUCCESS

MP_NO_WORKSPACE

MP_INVALID_SESSION

MP_INSUFFICIENT_RESSOURCES

SP_NO_SECURITY_CONTEXT

SP_ACCESS_CONTROL_FAILURE

SP_AUDIT_ALARM_FAILURE

NAME

sp_set_req - Protects the parameters to a Set request.

SYNOPSIS

```
#include <xom.h>
#include <tsp.h>

SP_status sp_get_req(
     OM_private_object session,
     OM_private_object context,
     OM_object argument,
     OM_object *protected);
```

DESCRIPTION

This function is used to protect the parameters to a Set request just before the corresponding XMP function call is made.

Parameters

session	An OM object that identifies the management session in which the operation will be requested. This must be a private OM object previously returned from `mp_bind()`. This object must belong to an ACM-disabled workspace.
context	The context in which the operation should be performed. This must be a private OM object or the constant **Default-Context** { MP_DEFAULT_CONTEXT }.
argument	An unprotected OM object which provides the information about which attributes are to be modified. It is an instance of a subclass of the OM class **Set-Argument**.
protected	Returned upon successful completion of the function call. The protected information that is to be supplied to `mp_set_req()`.

RESULTS

This function returns :

MP_SUCCESS

MP_NO_WORKSPACE

MP_INVALID_SESSION

MP_INSUFFICIENT_RESSOURCES

SP_NO_SECURITY_CONTEXT

SP_ACCESS_CONTROL_FAILURE

SP_AUDIT_ALARM_FAILURE

NAME

sp_set_rsp - Protects the parameters to a Set reply.

SYNOPSIS

```
#include <xom.h>
#include <tsp.h>

SP_status sp_action_rsp(
     OM_private_object session,
     OM_private_object context,
     OM_object response,
     OM_object *protected);
```

DESCRIPTION

This function is used to protect the parameters to a Set reply just before the corresponding XMP function call is made.

Parameters

session	An OM object that identifies the management session in which the operation will be requested. This must be a private OM object previously returned from `mp_bind()`. This object must belong to an ACM-disabled workspace.
context	The context in which the operation should be performed. This must be a private OM object or the constant **Default-Context** { MP_DEFAULT_CONTEXT }.
response	An unprotected OM object supplied as a response information about the Set operation. It is an instance of one of the following OM classes : **Set-Result**, **Linked-Reply-Argument**, **Absent-Object** or **Service-Error**.
protected	Returned upon successful completion of the function call. The protected information that is to be supplied to the `mp_set_rsp()`.

RESULTS

This function returns :

MP_SUCCESS

MP_NO_WORKSPACE

MP_INVALID_SESSION

MP_INSUFFICIENT_RESSOURCES

SP_NO_SECURITY_CONTEXT

SP_ACCESS_CONTROL_FAILURE

SP_AUDIT_ALARM_FAILURE

6.1.3 Version, Release history, Known bugs

0.1 First level of capability: authentication.

6.2 Secure Management Association

6.2.1 Engineering Object Model

The Secure Management Association Support Component (SMASC) is that component of the management system which provides the management applications with the means to secure the management association with other management applications located in another management system. The main purpose of the *SMASC* is to isolate the security-related components from the application code. This approach has the following advantages:

- the security can easily be added to / removed from an existing application, without affecting its internal structure,
- the security-related code can be designed, programmed and verified independently by security-aware personnel,
- the addition of auditing capacities for security-related events is made easier and safer,
- the resulting code can easily be customised to accommodate new security policies.

On behalf of a management entity, the **SMASC** authenticates and control the access to peer management applications; it also initialises the security context for further security services to be used on the association, in particular it establishes a session secret key if the requested Quality of Protection requires integrity and / or confidentiality of communicated data.

The following figure shows the main components of the security architecture, the internal structure of the *SMASC* and the contract interfaces of the *SMASC* to other components. The security services of the *SMASC* can be accessed through *Adapter* Components. The purpose of the *Adapter* Components is to transfom technology specific syntax (e.g. XOM objects [XOM]) to generic data structures (e.g. BER encoding). With this approach, platform specific code can be restricted to the Adapter Component and the SMASC can be reused without major modifications for other management platforms.

The SMASC is interfaced with:

? the Security Event Logging and Forwarding (SELF) component to keep track of all relevant security events occurring on the management associations.

The *SMASC* is decomposed into object classes as follows:

? a Secure Management Association Support (*SMAS*) object which co-ordinates the behaviour of the whole component,

? a *SSO* object, which provides generic security services such as peer authentication, integrity, encryption and digital signatures, The *SSO* is implemented using existing commercial products (SECUDE) which provides:

  ? a standard [RFC 1508] GSS interface to supply generic security services. This interface is coordinated to the *CerticateHandling* object.

  ? a *CertificateHandling* object, which performs key handling such as caching, fetching certificates from directories and checking certificates revocation lists. The *CertificateHandling* object can interact with extemal services for certificate and CRL distribution, for example offered in conjunction with a CA TTP. The LDAP protocol (RFC 1777, 1995) is usable for fetching of certificates and CRLs.

? a *SecProfile* object to provide the profile and the Access Control files to use to interact with a remote MAE,

? a AccessControl object to control the access to the management association and the validity of operation on managed objects.

The SMASC is also interfaced with:

? the Security Event Logging and Forwarding (SELF) component to keep track of all relevant security events occurring on the management associations.

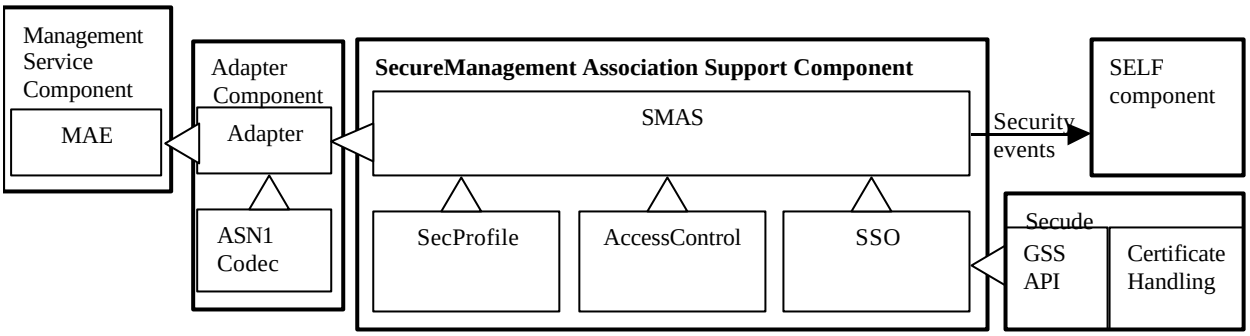

Figure 49: Graphical Representation of the Secure Management Association Component

Note that in order to establish a secure management association, a management entity must have disabled any type of automatic connection mechanism and take explicit control over the association.

6.2.2 Supported component interfaces

6.2.2.1 Function: smasc_init

NAME

smasc_init - Initialises the SMASC.

SYNOPSIS

```
#include <smasc.h>

smasc_error smasc_init(
     char *name,
     smasc_role role);
```

DESCRIPTION

This function is used by the Adapter to set up the SMASC intrinsic variables, and to acquire the credentials for future secured association processes.

The intrinsic variables setup consists of:

- the socket address setup to communicate with the SELF,
- the managed object instance setup to be used in the events sent to the SELF,
- the security rules setup of the four profiles.

Parameters

name	The MAE's LDAP Distinguished Name.
role	The role of entity for the future connection : SMP_SERVER, SMP_CLIENT

Environment variables

SELF_HOST	The hostname for the SELF socket address (default value ="localhost").
SELF_PORT	The port for the SELF socket address (default value =50095).
XUSER_MANAGER_ID	The managed object instance (MOI) for a "SMP Client" (default value =3).
XUSER_AGENT_ID	The managed object instance (MOI) for a "SMP Server" (default value =4).

RULES_PATH The path to find the security rules configuration file (mandatory named "rulesProfiles.txt"). The default security rules are set according to D8 specifications.

RESULTS

This function returns to the Adapter :

SMP_ERROR	Null string given as name.
	The role value isn't valid.
SMP_AUTH_F	The acquisition of the credentials failed.
SMP_OK	The initiation succeeded.

EVENTS

This function sends to the SELF the following events:

Service Type	*Event Type*	*Initiator DN*	*Target DN*	*Additional Info*
Authentication	AuthFailure	Initiator DN	Null	Message=<No credentials acquired>
Authentication	Notification	Initiator DN	Null	Message=<Server authenticated>
Authentication	Notification	Initiator DN	Null	Message=<Client authenticated>

6.2.2.2 Function: smasc_assoc_req

NAME

smasc_assoc_req - Request for a token to establish a secured association.

SYNOPSIS

```
#include <smasc.h>

smasc_error smasc_assoc_req(
     char *target,
     smasc_token *token);
```

DESCRIPTION

This function is used by the Adapter (in a **client task**) to initiate and continue the secured association process by providing tokens to be sent to the target MAE.

The number of exchanges for a secured association isn't a priori known, the association process needs to be continued until success.

This function first determines the security profile, then (if requested) verifies the permission of the association, and provides a token for association process.

Parameters

target	The target MAE's LDAP Distinguished Name.
token	Returned upon successful completion of the function call. The authentication token to be sent to the target MAE.

RESULTS

This function returns to the adapter:

SMP_ERROR	The entity has not or wrongly been initiated before.
SMP_AUTH_F	The security profile hasn't been found.
	The association isn't allowed.
	The authentication failed.
	The association has already been established.
SMP_OK	The secured association succeeded. A returned token with a no-null length has to be sent.
SMP_CONTINUE	The secured association needs to be continued.

EVENTS

This function sends to the SELF the following events:

Service Type	***Event Type***	***Initiator DN***	***Target DN***	***Additional Info***
Authentication	Authentication Failure	Initiator DN	Target DN	Message=<Try to initiate an association already established>
Authentication	Authentication Failure	Initiator DN	Target DN	Message=<Security profile not found>
Access Control	Unauthorised AccessAttempt	Initiator DN	Target DN	Profile=<*XXX*> Message=<Association permission not allowed>
Authentication	Notification	Initiator DN	Target DN	Profile=<*XXX*> Message=<Authentication succeeded>
Authentication	Authentication Failure	Initiator DN	Target DN	Profile=<*XXX*> Message=<Authentication failed>
Key Management	Key Expired	Initiator DN	Target DN	Profile=<*XXX*> Message=<Key expired>

6.2.2.3 Function: smasc_assoc_rec

NAME

smasc_assoc_rec - Reception of a token for a secured association.

SYNOPSIS

```
#include <smasc.h>

smasc_error smasc_assoc_rec(
     char *target,
     smasc_token token);
```

DESCRIPTION

This function is used by the Adapter (in a **client task**) to continue a secured association process by receiving tokens from the target MAE.

The number of exchanges for a secured association isn't a priori known, the process needs to be continued until success.

Parameters

target	The target MAE's LDAP Distinguished Name.
token	The authentication token received from the target MAE.

RESULTS

This function returns:

SMP_ERROR	The entity has not or wrongly been initiated before.
	The "smasc_assoc_req" function hasn't been called before.
SMP_AUTH_F	The authentication failed.
SMP_OK	The secured association succeeded.
SMP_CONTINUE	The secured association needs to be continued.

EVENTS

This function sends to the SELF the following events:

Service Type	*Event Type*	*Initiator DN*	*Target DN*	*Additional Info*
Authentication	Authentication Failure	Initiator DN	Target DN	Profile=<*XXX*> Message=<Null authentication token received>
Authentication	Notification	Initiator DN	Target DN	Profile=<*XXX*> Message=<Authentication succeeded>
Authentication	Authentication Failure	Initiator DN	Target DN	Profile=<*XXX*> Message=<Authentication failed>
Key Management	Key Expired	Initiator DN	Target DN	Profile=<*XXX*> Message=<Key expired>

6.2.2.4 Function: smasc_assoc_ind

NAME

smasc_assoc_ind - Indication of received token for secured association.

SYNOPSIS

```
#include <smasc.h>

smasc_error smasc_assoc_ind(
     char *target,
     smasc_token token);
```

DESCRIPTION

This function is used by the Adapter (in a **server task**) to accept and follow the secured association process by receiving tokens from the target MAE.

The number of exchanges for a secured association isn't a priori known, the process needs to be continued until successful.

This function first determines the security profile, then (if requested) verifies the permission of the association, and accepts tokens for the association process.

Parameters

target	The target MAE's LDAP Distinguished Name.
token	The authentication token received from the target MAE.

RESULTS

This function returns:

SMP_ERROR	The entity has not or wrongly been initiated before.
SMP_AUTH_F	The authentication failed.
SMP_OK	The secured association succeeded.
SMP_CONTINUE	The secured association needs to be continued.

EVENTS

This function sends to the SELF the following events:

Service Type	***Event Type***	***Initiator DN***	***Target DN***	***Additional Info***
Authentication	Authentication Failure	Initiator DN	Target DN	Message=<Try to initiate an association already established>
Authentication	Authentication Failure	Initiator DN	Target DN	Message=<Security profile not found>
Access Control	Unauthorised Access Attempt	Initiator DN	Target DN	Profile=<*XXX*> Message=<Association permission not allowed>
Authentication	Notification	Initiator DN	Target DN	Profile=<*XXX*> Message=<Authentication succeeded>
Authentication	Authentication Failure	Initiator DN	Target DN	Profile=<*XXX*> Message=<Authentication failed>
Authentication	Authentication Failure	Initiator DN	Target DN	Profile=<*XXX*> Message=<Null authentication token received>
Authentication	Authentication Failure	Initiator DN	Target DN	Profile=<*XXX*> Message=<Authentication succeeded with a usurped name>
Key Management	Key Expired	Initiator DN	Target DN	Profile=<*XXX*> Message=<Key expired>

6.2.2.5 Function: smasc_assoc_rsp

NAME

smasc_assoc_rsp - Response token for a secured association.

SYNOPSIS

```
#include <smasc.h>

smasc_error smasc_assoc_rsp(
     char *target,
     smasc_token *token);
```

DESCRIPTION

This function is used by the Adapter (in a **server task**) to continue the secured association process by providing response tokens to be sent to the target MAE.

The number of exchanges for a secured association isn't a priori known, the process needs to be continued until successful.

This function only continues the process of association.

Parameters

target	The target MAE's LDAP Distinguished Name.
token	Returned upon successful completion of the function call. The next authentication token to be sent to the target MAE.

RESULTS

This function returns:

SMP_ERROR	The entity has not or wrongly been initialised before.
	The "smasc_assoc_ind" function hasn't been called before.
SMP_OK	The secured association succeeded. A returned token with a no-null length has to be sent.
SMP_CONTINUE	The secured association needs to be continued.

EVENTS

This function doesn't send events to the SELF.

6.2.2.6 Function: smasc_ release_token

NAME

smasc_release_token - Releases a token allocated by a smasc_assoc function.

SYNOPSIS

```
#include <smasc.h>

smasc_error smasc_release_token(
     smasc_token token);
```

DESCRIPTION

This function is used by the Adapter to release tokens allocated by the "smasc_assoc" function.

Parameters

token	Token to be released.

RESULTS

This function returns:

SMP_ERROR	The token points on an invalid address.
SMP_OK	The release operation succeeded.

EVENTS

This function may not send events to the SELF.

6.2.2.7 Function: smasc_ assoc_close

NAME

smasc_assoc_close - Closes a secured association.

SYNOPSIS

```
#include <smasc.h>

smasc_error smasc_assoc_close(
     char *target);
```

DESCRIPTION

This function is used by the Adapter to close a secured association, with the release of :

- the security profile variables,
- the access control variables,
- the SSO variables.

Parameters

target	The target MAE's LDAP Distinguished Name.

RESULTS

This function returns:

SMP_ERROR	No secured association with this target initiated or completed.
SMP_OK	The operation succeeded.

EVENTS

This function doesn't send events to the SELF.

6.2.2.8 Function: smasc_ close

NAME

smasc_close - Closes the use of SMASC.

SYNOPSIS

```
#include <smasc.h>

smasc_error smasc_close();
```

DESCRIPTION

This function is used by the Adapter to close the use of SMASC.

This will close all the secured connections, and release SMASC intrinsic variables.

Parameters

No parameters

RESULTS

This function returns:

SMP_OK	The operation succeeded.

6.2.2.9 Function: smasc_op_perm

NAME

smasc_op_perm - .Get permission for an operation on an object

SYNOPSIS

```
#include <smasc.h>

smasc_op_error smasc_op_perm(
     char *target,
     smasc_op operation,
     char *objet);
```

DESCRIPTION

This function is used by the Adapter to get the permission to perform an operation on an object.

This function according to the association profile rules verifies (if requested) the permission of the operation.

The result “deny and abort operation” automatically closes the secured association.

Parameters

target	The target MAE’s LDAP Distinguished Name.
operation	Type of CMIP operation to perform: SMP_GET SMP_SET SMP_ACTION SMP_CREATE SMP_DELETE
object	The CMIP object’s Distinguished Name.

RESULTS

This function returns:

SMP_OP_ERROR	The entity has not or wrongly been initialised before.

	The secured association hasn't been established before.
	The operation value is invalid
SMP_OP_OK	The operation is allowed.
SMP_OP_DENY,	The operation is denied.
SMP_OP_DENY_WITHOUT_RESP	The operation is denied and no response has to be done.
SMP_OP_DENY_FALSE_RESP	The operation is denied and a false response has to be given.
SMP_OP_ABORT	The operation is denied and the connection has been closed.

EVENTS

This function sends to the SELF the following events:

Service Type	*Event Type*	*Initiator DN*	*Target DN*	*Additional Info*
Access Control	Unauthorised AccessAttempt	Initiator DN	Target DN	Profile=<*XXX*> Message=<Permission denied for *operation* on *objectDN*>
Access Control	Unauthorised AccessAttempt	Initiator DN	Target DN	Profile=<*XXX*> Message=<Permission denied without response for *operation* on *objectDN*>
Access Control	Unauthorised AccessAttempt	Initiator DN	Target DN	Profile=<*XXX*> Message=<Permission denied and false response to give for *operation* on *objectDN*>
Access Control	Unauthorised AccessAttempt	Initiator DN	Target DN	Profile=<*XXX*> Message=<Permission denied and association aborted for *operation* on *objectDN*>

6.2.2.10 Function: smasc_seal

NAME

smasc_seal - Seals a message.

SYNOPSIS

```
#include <smasc.h>

smasc_op_error smasc_seal(
     char *target,
     smasc_token msgToken,
     smasc_token *sgnToken,
     smasc_sealing *type);
```

DESCRIPTION

This function is used by the Adapter to seal or sign a message to send.

This function according to the association profile rules applies:

- a seal operation,

- a sign operation,

- no operation.

Parameters

target	The target MAE's Distinguished Name.
msgToken	The input message.
sgnToken	Returned token that contains : - the input message encrypted and signed in case of a seal operation, - the signature of the input message in case of a sign operation.
type	Returned value that indicates the operation performed : - SMP_SEAL: input message has been encrypted and signed - the sgnToken has to be sent, - SMP_SIGN: input message has been signed - both sgnToken and msgToken have to be sent, - SMP_CLEAR : no operation performed - the msgToken has to be sent.

RESULTS

This function returns:

SMP_ERROR	The secured association hasn't been established before.
SMP_OK	The operation succeeded.
SMP_FAILURE	The encryption or signature operation failed.

EVENTS

This function doesn't send events to the SELF.

6.2.2.11 Function: smasc_unseal

NAME

smasc_unseal - .Unseal a message.

SYNOPSIS

```
#include <smasc.h>

smasc_op_error smasc_unseal(
     char *target,
     smasc_token *msgToken,
     smasc_token sgnToken);
```

DESCRIPTION

This function is used by the Adapter to unseal or verify a received message.

This function according to the association profile rules applies :

- a unseal operation,

- a verify operation,

- no operation.

Parameters

target	The target MAE's LDAP Distinguished Name.
msgToken	Output clear message token in case of unseal operation. Input clear message token in case of verification.
sgnToken	Input token that contains : - the input message encrypted in case of unsealing operation , - the signature of the clear message in case of verification operation, - null token in case of null operation.

RESULTS

This function returns:

SMP_ERROR	The secured association hasn't been established before.
SMP_OK	The operation succeeded.
SMP_FAILURE	The decryption or verification operation failed.

EVENTS

This function sends to the SELF the following events:

Service Type	***Event Type***	***Initiator DN***	***Target DN***	***Additional Info***
Integrity Confidentiality	Information Modification Detected	Initiator DN	targetDN	Profile=<*XXX*> Message=<Information modification detected >
Integrity Confidentiality	Duplicated Information Detected	Initiator DN	targetDN	Profile=<*XXX*> Message=<Duplicated information detected >
Integrity Confidentiality	Breach Of Confidentiality Detected	Initiator DN	targetDN	Profile=<*XXX*> Message=<Breach of confidentiality detected >
Key Management	Key Expired	Initiator DN	targetDN	Profile=<*XXX*> Message=<Key expired>

6.2.3 Version, Release history

V3.0 Complete version with the configuration of the security profiles rules.

6.3 Security Profile Management

6.3.1 Engineering Object Model

The **SecPolicyInfo** interface as described in D9 Section 4.8 and supported by the **SecPolicy** object described in D9 Section 4.7.2 is realized by a C++ class definition with public interface:

```
class SecurityPolicy {
public:
  SecurityPolicy();
  ~SecurityPolicy();
  SecurityProfile* secProfileQuery(DistName&       initiatorTitle,
                                   ManagingRole    initiatorRole,
                                   DistName&       responderTitle,
                                   QoP);           //is not being used
};
```

Security profiles are read from file (for format of this file cf. Section 4.3) in the *SecurityPolicy* constructor.

6.3.2 Supported component interfaces

The *SecurityPolicy* class does not make use of any other TRUMPET component. It supports an interface for the SMASC.

Security rules for interacting with a remote peer are provided by calling the *secProfileQuery* method. The return type of *secProfileQuery* is *SecurityProfile* and the public part of this class definition is:

```
class SecurityProfile {
public:
  RWCString      aCDirectory();
  SecProfileType secProfile();
};
```

The *aCDirectory* method returns a value that is the name of a directory containing access rules to be used by the access control component (passed on to this module by the SMASC). Four different security profiles are recognized and may be returned by the *secProfile* method (although only two are actually supported by TRUMPET). The profile values are given in the definition of *SecProfileType*:

```
enum SecProfileType {
  NULLP,   //supported by TRUMPET
  MIN,     //not supported
  BASIC,   //supported
  ADV      //not supported
};
```

The parameters to *secProfileQuery* are of three different types; *ManagingRole*, *QoP* and *DistName*. *ManagingRole* can be either agent or manager and is defined as an enumeration type:

```
enum ManagingRole {
```



```
  AGENT_ROLE,
  MANAGER_ROLE
};
```

As *QoP* is not in use in TRUMPET, it is defined as an enumeration type with one possible value only:

```
enum QoP {
  NULLQ = 0
};
```

The class *DistName* is used by several components in the security package. The aspects of *DistName* relevant to the interface to an object of class *SecurityPolicy* can be described like this:

```
class DistName {
public:
  DistName();
  DistName(const char*);

  const char* dN() const;

  //equality operator overload
  int operator== (const DistName& dn) const;
  int operator!= (const DistName& dn) const;
};
```

The *secProfileQuery* method throws three exceptions that all have one public method each:

```
class UnknownInitiator          { public: char* msg(); };
class UnknownResponder          { public: char* msg(); };
class BadInitiatorResponderPair { public: char* msg(); };
```

It is up to the SMASC to dispose of the message returned with a *delete* operation on the returned value.

The types *SecurityProfile*, *SecProfileType*, *ManagingRole*, *QoP* and all three exceptions are defined along with *SecurityPolicy* in the *secpolicy.hh* header file. *DistName* is used more extensively in the security package and is defined elsewhere (but included by *secpolicy.hh*).

Example

The intended usage of the *SecurityPolicy* class from the SMASC can be illustrated like this:

```
#include <trumpet/secpolicy.hh>

SecurityPolicy   sPol;

DistName         initiatorTitle("cn=bm, OU=NR,  O=TRUMPET Project");
DistName         responderTitle("cn=bm, OU=UCL, O=TRUMPET Project");
ManagingRole     initiatorRole = MANAGER;
```



```
SecurityProfile* sP;

try {
  sP = sPol.secProfileQuery(initiatorTitle,
                            initiatorRole,
                            responderTitle,
                            NULLQ);
}
catch (UnknownInitiator ui) {
  msg = ui.msg(); cout << msg; delete msg;
}
catch (UnknownResponder ur) {
  msg = ur.msg(); cout << msg; delete msg;
}
catch (BadInitiatorResponderPair irp) {
  msg = irp.msg(); cout << msg; delete msg;
}

//some code to consider the attributes of *sP should
//be inserted here, before sP is being disposed of

delete sP;
```

6.3.3 Version, Release history

- September 1997: Initial version.
- November 1997: New version released after first integration meeting.
- January 1998: Code checked with *purify* and new version released.

6.4 Access Control

6.4.1 Engineering Object Model

This is the object model for the access control component implementation:

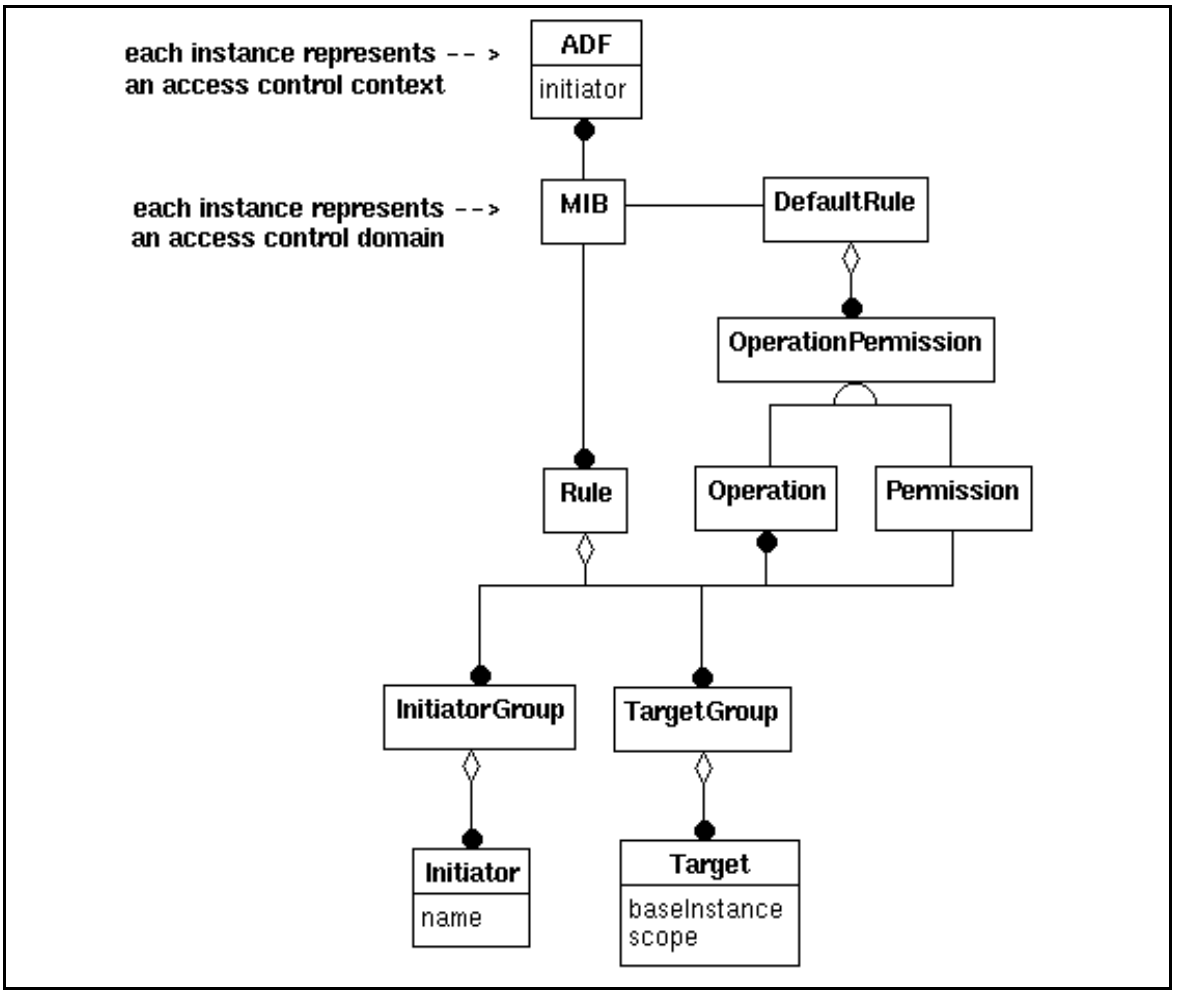

Figure 50: Engineering Object Model of the Access Contro Servicel

6.4.2 Supported component interfaces

The access control component interfaces with the SMASC. It does not depend on other TRUMPET system components.

The class ***ADF*** provides the external interface of the access control component:

```
class ADF {
public:
     ADF (const DistName& initiator, const RWCString& domain);
     Permission associationPermission (const DistName& target);
     Permission operationPermission (
          Operation operation,
          RWTValSlist<ac_target>& targets,
          RWTValSlist<ac_target>& denied);
};
```

METHOD

ADF: generates a new access decision context

SYNOPSIS

```
#include <trumpet/securityPackage/acControl/adf.hh>
ADF (const DistName& initiator, const RWCString& domain);
```

DESCRIPTION

This function is used to instantiate a new access decision context. It contains the name of the associated initiator and the set of access control rules to be used.

ARGUMENTS

- *initiator*: the application entity title of the requesting MAE (distinguished name form).
- *domain*: identifies the access control domain.

RESULTS

This function may return: **Allow, DenyWithResponse, DenyWithoutResponse, AbortAssociation**

METHOD

associationPermission: determine access control permission for association between given MAEs

SYNOPSIS

```
#include <trumpet/securityPackage/acControl/adf.hh>

AccessDecision associationPermission (const DistName& target)
```

DESCRIPTION

This function determines the access permission of the initiator MAE to establish an association to the target MAE.

ARGUMENTS

- *target*: the application entity title of the peer MAE to connect to. This must be a distinguished name.

RESULTS

This function may return: **Allow, DenyWithResponse, DenyWithoutResponse, AbortAssociation**

METHOD

operationPermission: determine access permissions to perform operation on given management information objects.

SYNOPSIS

```
#include <trumpet/securityPackage/acControl/adf.hh>

AccessDecision operationPermission (
          OM_sint primitive,
          List<ac_target>& targets,
```



```
        List<ac_target>& denied);
```

DESCRIPTION

This function is used to determine the access permissions of the initiator to perform the given operation on the set of managed objects.

ARGUMENTS

- *primitive* Identifies which type of operation has been requested. Possible values are: MP_GET_IND, MP_SET_IND, MP_ACTION_IND, MP_CREATE_IND, MP_DELETE_IND
- *targets* Identifies the set of managed objects selected for the operation. The type of list elements is *ac_target*. which is defined as:

```
struct ac_target {
     OM_object object;
     AccessDecision permission;
     void* link
     };
```

  - *object* identifies a MO instance. It is an instance of OM class Base-Managed-Object-Id.
  - *permission* returns the permission to perform the operation determined by the access decision function.
  - *link* can be used by the caller to establish a link between the structure and the MO instance.

RESULTS

- *targets*: subset of objects access is granted. *permission* is set to Allow.
- *denied*: set of object access is denied. *permission* is set to one of the following values: **DenyWithResponse, DenyWithoutResponse, AbortAssociation**.

This function may return: **Allow, DenyWithResponse, DenyWithoutResponse, AbortAssociation**

6.4.3 Version, Release history

v1.0	Initial Release	Nov 97

6.5 Security Administration Tool

6.5.1 Engineering Object Model

The Security Administration Tool is divided into a server and a client. The server is doing the operations such as reading the configuration file, read/write or CMIP requests. The client is an applet (a GUI accessible in a browser) and is collection of windows to help the user access easily to the audit and alarm management (event filtering, displaying and logging) as well as to the access control and security profile management.

6.5.1.1 The Security Administration Server

The Security Administration Server performs all the operations that cannot be performed by the client which is an applet and cannot make read/write operations and use native code.

The AA manager server uses a C library which interface with XOM/XMP API for the creation and deletion of EFD, the reception of events, the read/write in a log file with XOM format. The AA manager server is written in Java and uses natives to access the C library.

The Server communicates with its client through RMI (Remote Management Interface) of Java. The exchanged messages are:

- creation or deletion of EFD,
- confirmation of creation or deletion of EFD,
- received alarms or non-security events,
- logged alarms or non-security events.

6.5.1.2 The Security Administration Client

The Security Administration Client is a GUI that is composed of several windows. The main goal of the GUI is to allow the user not necessarily familiar with security management concept to be able to access the management functions. This is the reason why the GUI should be user-friendly not only as to how the information is displayed, but also in the manner in which the user participates in the management process. The GUI provides the following elements:

- Top Level Window;

  This window allows to start the management session. The administrator can select the security service in the administration menu: Security Profile Management, Access Control or Audit. The Security Status menu displays the number of security alarms for each severity level. Two buttons allow to display more information about the alarms and management events.

- EFD Management;

  The window allows the management of the EFD. A scrolled list allows the selection of a host. The create and delete buttons allow the creation on the selected host of an EFD and the deletion on the selected host of the selected EFD. A list displays all the EFD that have been created for a selected host.

- Alarm Viewer;

  This window shows the collected events. For the alarms the severity is displayed and for the non security events, the severity is empty.

- Log Management.

  This window enables the display of the logged events with the Edit button and the deletion of the log with the button Delete.

The client also contains a Java thread which is responsible for receiving events from the server: either alarms, management events or EFD creation or deletion confirmation.

6.5.2 Supported component interfaces

6.5.2.1 Interface between the Java AA Manager Server and the SELF

The communications interface between the Java AA Manager Server in SELF is the same as one, which is used between the Motif AA Manager and the SELF. It is described in Section 6.7.2.

6.5.2.2 Interface between the Java AA Manager Client and Server

```
public interface RemoteManager extends java.rmi.Remote {

      public Vector readHosts () throws java.rmi.RemoteException;
      public boolean createEfd (String hostname)
            throws java.rmi.RemoteException;
      public CMISEvent receiveEvent () throws java.rmi.RemoteException;
      public void deleteEfd (String hostname, int efd)
            throws java.rmi.RemoteException;
      public CMISEvent readLog (int fd) throws java.rmi.RemoteException;
      public int beginReadLog () throws java.rmi.RemoteException ;
      public void endReadLog (int fd) throws java.rmi.RemoteException;
      public boolean deleteLog () throws java.rmi.RemoteException;
      public void print () throws java.rmi.RemoteException;
}
```

6.5.2.2.1 Method: readHosts

SYNOPSIS

```
      public Vector readHosts () throws java.rmi.RemoteException;
```

DESRIPTION

This method returns a vector which is a list of the possible hosts where EFDs can be created.

6.5.2.2.2 Method: createEfd

SYNOPSIS

```
      public boolean createEfd (String hostname)
            throws java.rmi.RemoteException;
```

DESRIPTION

This method creates an EFD on the given machine. The returned boolean indicates whether the creation has succeeded or not.

6.5.2.2.3 Method: receiveEvent

SYNOPSIS

```
      public CMISEvent receiveEvent () throws java.rmi.RemoteException;
```

DESRIPTION

This method waits until an event is received and returned this event.

6.5.2.2.4 Method: deleteEfd

SYNOPSIS

```
      public void deleteEfd (String hostname, int efd)
            throws java.rmi.RemoteException;
```

DESRIPTION

This method deletes an EFD identified by its number and machine.

6.5.2.2.5 Method: readLog

SYNOPSIS

```
      public CMISEvent readLog (int fd) throws java.rmi.RemoteException;
```

DESRIPTION

This method reads and returns one event in the log file identified by its file descriptor.

6.5.2.2.6 Method: beginReadLog

SYNOPSIS

```
public int beginReadLog () throws java.rmi.RemoteException ;
```

DESRIPTION

This method opens the log file and returns this file descriptor.

6.5.2.2.7 Method: endReadLog

SYNOPSIS

```
public void endReadLog (int fd) throws java.rmi.RemoteException;
```

DESRIPTION

This method closes the log file identified by its file descriptor.

6.5.2.2.8 Method: deleteLog

SYNOPSIS

```
public boolean deleteLog () throws java.rmi.RemoteException;
```

DESRIPTION

This method deletes the log file identified by its file descriptor.

6.5.2.2.9 Method: print

SYNOPSIS

```
public void print () throws java.rmi.RemoteException;
```

DESRIPTION

This method prints the current status of the server.

6.5.3 Version, Release history, Known bugs

An exception from Symantec Coffee is raised when the Alarm Viewer is edited.

6.6 Security Support Object

6.6.1 Engineering Object Model

The SSO is a set of API to provide security services. It interfaces with an commercial (Secude v5.1c) GSS library.

SECUDE (formerly SecuDE - Security Development Environment) is a security toolkit which incorporates well known and established symmetric and public-key cryptography. It offers a library of security APIs, and also tools to manage keys and certificates. These both aspects are closely linked, and the result of the GSS API hardly depends on the key management.

This object provides convenient security APIs, especially adapted to the Trumpet environment.

6.6.2 Supported component interfaces

The trumpet interface offers the following functions to the SMASC:

6.6.2.1 Functions: server_acquire_creds & client_ acquire_creds

FUNCTION

server_acquire_creds

client_ acquire_creds

SYNOPSIS

```
Int      server_acquire_creds(char      *service_name,      gss_cred_id_t
*server_creds)
Int      client_acquire_creds(char      *service_name,      gss_cred_id_t
*server_creds)
```

DESCRIPTION

Imports service name and acquires credentials for it. The service name is imported with gss_import_name, and service credentials are acquired with gss_acquire_cred. If either operation fails, an error message is displayed and an error is returned.

ARGUMENTS

- service_name The ASCII service name. (Input)
- server_creds The GSS-API service credentials. (Output)

RESULTS

SSO_FAILURE	Either service name importation or credentials acquisition failed.
SSO_SUCCESS	Both operations succeeded.

6.6.2.2 Function: client_establish_context

FUNCTION

client_establish_context

SYNOPSIS

```
int client_establish_context(
                  int             req_flag,
                  gss_cred_id_t   client_creds,
                  char          * service_name,
                  gss_ctx_id_t  * gss_context,
                  gss_buffer_t    recv_tok,
                  gss_buffer_t    send_tok)
```

DESCRIPTION

Establishes a GSS-API context with a specified target service and returns the context handle. The target service name is imported as a GSS-API name and a GSS-API context is established with the corresponding service. The default GSS-API mechanism is used, and specified authentication options are requested. If successful, the context handle is returned. If unsuccessful an error code is returned. Else the process has to be continued.

ARGUMENTS

- Req_flag The specified options for the authentication. (input)

  In Trumpet we use:

  - SSO_BASIC_FLAG for unilateral authentication with replay and out of sequence detection,

- SSO_ADV_FLAG to add mutual authentication.

- client_creds Client credentials, from gss_acquire_cred (input)
- service_name The ASCII target service name. (input)
- gss_context The GSS-API context to establish. (output)
- recv_tok The received token. (Input)

  Has to be set to = GSS_C_NO_BUFFER for the first call.
- send_tok The token to send. (Output)

  A non-null token has to be sent even if SSO_SUCCESS is returned.

RESULTS

SSO_SUCCESS	The authentication succeeded and the context is established.
SSO_CONTINUE	The authentication process has to be continued.
SSO_FAILURE	The authentication process failed for an unspecified reason.
SSO_EXPIRED	The authentication failed: the target key validity date has expired.

6.6.2.3 Function: server_establish_context

FUNCTION

server_establish_context

SYNOPSIS

```
int server_establish_context(     gss_cred_id_t server_creds,
                gss_ctx_id_t *context,
                gss_buffer_t client_name,
                gss_buffer_desc recv_tok,
                gss_buffer_desc send_tok)
```

DESCRIPTION

Establishes a GSS-API context as a specified service with an incoming client, and returns the context handle and associated client name. Any valid client request is accepted. If a context is established, its handle is returned in context and the client name is returned in client_name and SUCCESS is returned. If unsuccessful an error code is returned. Else the process has to be continued.

ARGUMENTS

- server_creds Server credentials, from gss_acquire_cred. (Input)
- context The GSS-API context to establish. (Output)
- client_name The client's ASCII name. (Output)
- recv_tok The received token. (Input)
- send_tok The token to send. (Output)

  A non-null token has to be sent even if SSO_SUCCESS is returned.

RESULTS

SSO_SUCCESS	The authentication succeeded and the context is established.

SSO_CONTINUE	The authentication process has to be continued.
SSO_FAILURE	The authentication process failed for an unspecified reason.
SSO_EXPIRED	The authentication failed: the target key validity date has expired.

6.6.2.4 Function: trumpet_release_buffer

FUNCTION

trumpet_release_buffer

SYNOPSIS

```
void trumpet_release_buffer(gss_buffer_desc buffer)
```

DESCRIPTION

Release a buffer allocated by a GSS-API function. This concerns the ouput gss_buffer_t buffers, for examples: tokens to send, or the service names.

ARGUMENTS

- buffer Buffer to release (Input)

RESULTS

None

6.6.2.5 Function: trumpet_delete_sec_ctx

FUNCTION

trumpet_delete_sec_ctx

SYNOPSIS

```
int trumpet_delete_sec_ctx(gss_ctx_id_t * ctx)
```

DESCRIPTION

Deletes an established GSS-API context of security. This operation needs to be performed by the client and the server.

ARGUMENTS

- ctx Context to delete (Input)

RESULTS

SSO_SUCCESS	The operation succeeded.
SSO_FAILURE	The operation failed.

6.6.2.6 Function: trumpet_release_cred

FUNCTION

trumpet_release_cred

SYNOPSIS

```
int trumpet_release_cred(gss_cred_id_t * cred)
```

DESCRIPTION

Releases the acquired credentials. This operation needs to be performed by the client and the server.

ARGUMENTS

- cred Credentials to release (Input)

RESULTS

SSO_SUCCESS	The operation succeeded.
SSO_FAILURE	The operation failed.

6.6.2.7 Function: trumpet_seal

FUNCTION

trumpet_seal

SYNOPSIS

```
int trumpet_seal(gss_ctx_id_t context,
            gss_buffer_t in_buf,
            gss_buffer_t out_buf)
```

DESCRIPTION

Encrypts and signs the input buffer, according to the established context of security. If successful the result of encryption is returned in the output buffer. Otherwise an error is returned.

ARGUMENTS

- context The established GSS-API context (Input)
- in_buf The buffer to encrypt. (Input)
- out_buf The result of encryption. (Output)

RESULTS

SSO_SUCCESS	The operation succeeded.
SSO_FAILURE	The operation failed.

6.6.2.8 Function: trumpet_sign

FUNCTION

trumpet_sign

SYNOPSIS

```
int trumpet_sign(gss_ctx_id_t context,
            gss_buffer_t msg_buf,
            gss_buffer_t sgn_buf)
```

DESCRIPTION

Signs the input buffer, according to the established context of security. If successful the signature is returned in the output buffer. Otherwise an error is returned.

ARGUMENTS

- context The established GSS-API context (Input)
- msg_buf The buffer to sign. (Input)
- sgn_buf The result of signature. (Output)

RESULTS

SSO_SUCCESS	The operation succeeded.
SSO_FAILURE	The operation failed.

6.6.2.9 Function: trumpet_unseal

FUNCTION

trumpet_unseal

SYNOPSIS

```
int trumpet_unseal(gss_ctx_id_t context,
            gss_buffer_t in_buf,
            gss_buffer_t out_buf)
```

DESCRIPTION

Decrypts and verifies the signature of the input buffer, according to the established context of security. If successful the result is returned in the output buffer. Otherwise an error is returned.

ARGUMENTS

- context The established GSS-API context (Input)
- in_buf The buffer to decrypt. (Input)
- out_buf The result of decryption. (Output)

RESULTS

SSO_SUCCESS	The operation succeeded.
SSO_IMD	An information modification has been detected.
SSO_DUPLICATE	A duplication of token has been detected.
SSO_BOF	A breach of confidentiality has been detected.

SSO_EXPIRED	The operation failed: the target key validity date has expired.
SSO_FAILURE	The operation failed for an unspecified reason.

6.6.2.10 Function: verify

FUNCTION

trumpet_ verify

SYNOPSIS

```
int trumpet_verify(gss_ctx_id_t context,
              gss_buffer_t msg_buf,
              gss_buffer_t sgn_buf)
```

DESCRIPTION

Verifies the signature of the message buffer, according to the established context of security. The result of the verification is returned.

ARGUMENTS

- context — The established GSS-API context (Input)
- msg_buf — The buffer to sign. (Input)
- sgn_buf — The result of signature. (Input)

RESULTS

SSO_SUCCESS	The operation succeeded.
SSO_IMD	An information modification has been detected.
SSO_DUPLICATE	A duplication of token has been detected.
SSO_BOF	A breach of confidentiality has been detected.
SSO_EXPIRED	The operation failed: the target key validity date has expired.
SSO_FAILURE	The operation failed for an unspecified reason.

6.6.3 Version, Release history, Known bugs

6.6.3.1 Version / release history

v2.0	Complete Version	May 98

6.6.3.2 Known bugs

With Secude v5.1c a secured association with an outdated certificate generates a generic authentication failure error instead of a key expired error.

6.7 Audit and Alarm

6.7.1 Engineering Object Model

6.7.1.1 Introduction

The implementation of the audit management is described in this section with emphasis on the Graphical User Interface, the Event Forwarding Discriminator management and the security related event collecting and displaying. In the Graphical User Interface, a collection of user friendly mask windows are defined and realised to help the user to access easily to the event forwarding discriminator construction and the alarm reporting function. The security related event management is implemented using the XMP API.

The implementation work is composed of:

- The basic audit management function library
- The top level GUI
- The GUI for alarm viewing
- The GUI for EFD management
- The GUI for log management
- The graphical editor for DiscriminatorConstruct

6.7.1.2 Implementation Architecture

The audit management is performed by the security alarm management application and the audit trail management application.

These management applications collect security-related events from a set of known agents according to the conditions defined in the Event Forwarding Discriminators. The alarm manager allows these events to be displayed in human readable format, as the audit trail manager saves them into a log for further analysis.

The audit management is implemented in one process.

The GUI provides:

- an agent viewer to show the location and the state of the agents
- an agent controller to manage the EFD in that agent (EFD creation, deletion, etc.)
- an alarm viewer to display collected security alarms
- an event log browser

The Security Alarm Manager implements the following management functions:

- event forwarding discriminator management
- security alarm collection (the alarm detection is performed by the agent side)
- alarm reporting

The Audit Trail Manager logs not only security alarms (`SecurityServiceAndMechanism-Violation`: a subset of security-related events) but also `ServiceReport`. These events are very valuable resources for further security audit analysis.

6.7.1.3 GUI

This section presents the design of the GUI for managing audit functions, including Event Forwarding Discriminators and security-related event collecting and displaying. The corresponding sponsor component, accepting messages from the audit management application and mapping them to real resources mechanisms is not discussed here.

In the GUI a collection of user-friendly mask windows are defined and realised to help the user access easily the event forwarding discriminator construction and the alarm reporting function. The security-related event management is implemented using the XMP API.

The main goal of the GUI is to allow the user not necessarily familiar with the security management concept and the OSI model to be able to access the management functions. This is the reason for which the GUI should be user-friendly not only as to how the information is displayed, but also in the manner in which the user participates in the management process.

Top-Level Window

This window allows to start the TRUMPET management session. The administrator can select the security services to select in the *Administration* menu: currently only the audit management is implemented; potentially, management of the security policy, management of the authentication and access control services and key management could be added. The *Security Status* menu displays the number of security alarms for each severity level. Two buttons allow to display more information about the alarms.

Agent Configuration Window

This window allows to see each agent configuration from which security alarms will be collected. In its menu bar, the administrator can select an audit manager configuration containing some agent bitmaps which he can move on the trials bit map and perform actions on them., see agent specifications ...

Agent-EFD Control Window

This window gives some useful information on the selected agent and controls the EFD on it. The agent control panel shows the name and the current co-ordinates of the agent. The user can give the current operational state of the agent (present, running, locked, etc.). Using the EFD panel, the user can create a new EFD instance in the selected agent. A scroll list shows the EFDs currently associated with the agent; by choosing one of the EFD identifier from the list, the user may delete, edit or get help from it. When the user clicks on the Modify or Create button, the EFD construction mask window is displayed.

EFD Construction Mask Window

This window contains all the attributes of the EFD class. The security administrator can specify the appropriate value to build a new EFD or edit an existing EFD. By clicking on the corresponding button, the administrator is displayed a specific window to fill all the EFD attributes fields: Discriminator Construct, Start Time, Stop Time, Intervals of Days, Week Mask, Destination, Backup Destination List, Active Destination, Administrative State, Operational State, Availability Status, Confirmed Mode.

Filters Editor Window

This window allows to construct a discriminator graphically. On the drawing area, the administrator has to click on a rectangle bitmap and to control it by using the mouse menu button.

Control Window for Managing Logs

This windows is used for log management. The existing logs are displayed as icons with log names below. The security auditor can select a log with the mouse to display the contents of the log or delete the log. He can create a new log (Log Creation Window). By double-clicking on the log icon, the administrator can display the log attributes.

Log Creation Window

This window allows the creation of a new log. A default log can be created or the administrator can create a customised log by specifying – using the same windows as for the EFDs – the discriminator construct, the start time, the stop time, the intervals of days, the week mask. He can modify the operational and administrative state of the log.

Log Attributes Display Window

The administrator can display and modify the attributes of the selected log.

Alarm Viewer Window

This window shows the collected security alarms. Only 10 alarms can be displayed simultaneously. The alarm buffer size is limited. The older alarms will be discarded when the buffer is full.

6.7.1.4 Basic audit management function library

This library provides some basic functions to:

- initialise the XMP library
- create OpenView kernel EFD
- create, delete, enable and disable EFD in the specified agent
- wait an event (with time out)

This library is written upon the XMP interface.

6.7.1.5 Interprocess Communication Interface Library

The Interprocess Communication Interface Library defines a specific protocol and a set of service functions to allow one or more management application processes to communicate with the GUI process. The protocol is based on message passing mechanisms.

Principle

Management applications can not be implemented using event driven scheme. They have to perform a loop to wait for the CMIP messages and GUI commands. The X11 main loop does the same thing. It is possible to add into X11 main loop the CMIP messages .

The GUI implemented in X11 and Motif incorporates the messages from management processes (`XtAppAddInput`). The XMP based applications will include the GUI command handler into their main loop.

The GUI is responsible to load and shutdown a XMP process. The GUI can also control the XMP process execution, for example pause and resume the alarm collection.

6.7.2 Supported component interfaces

- The manager audit and alarms communicates with the agent ovead of the HP OpenView 4.21 platform through XMP. The GDMO model used for XOM objects is the GDMO X721 (ems.mib) of the platform. The manager manages object of class HPEventForwardingDiscriminator.
- The manager exchanges with SELF agent with XMP: Event Report which field Event Info is of class SecurityAlarmInfo (in ems.mib) or ServiceReport (GDMO X740).

Contents of the Event Report

Name of the attribute (GDMO)	**Contents**
Managed Object Class	Service which raise the event (integer converted in oid)
Managed Object Instance (in the attribute value of the first ava)	Instance of the service (integer)
Event Time	actual time (generalised time)
Event Type	oid of the notification
Event Info	Security Alarm Info or Security Audit Info

Contents of the Security Alarm Info

Name of the attribute (GDMO)	**Contents**
Security Alarm Cause	oid of the security alarm cause

Security Alarm Severity	integer
Security Alarm Detector	empty
Service User (attribute Detail)	oid of the entity which caused the raising of the alarm
Service Provider (attribute Detail)	oid of the entity which has been «attacked »

Contents of the Security Audit Info (Service Report)

Name of the attribute (GDMO)	**Contents**
Service Report Cause	oid of the service report cause
Additional Text	oid of the entity which caused the raising of the report
Additional Information	oid of the entity which has been «attacked »

6.7.3 Version, Release history, Known bugs

6.7.3.1 Version / release history

Actually, the agent of the HP OpenView platform does not implement the behaviour of all the packages of X721 GDMO. The Duration package is not implemented in the ovead agent, the GUI implements the Duration package (week mask, intervals of day, ...), but since it is not implemented in the agent, setting these attributes in the manager will have no effect on the behaviour.

6.8 SELF

6.8.1 Engineering Object Model

There is a number of XOM objects used to implement the audit agent. These are as follows:

- feature_list (MP_feature): negotiate the features of the platform environment where the agent runs.
- attributeId , ava , ds_rdn , ds_dn (OM_descriptor): they are used to identify an instance of a security service.
- bmoi: (OM_descriptor): is used to identify a managed object class which is a class of a service. An integer is used converted to an OID.
- bmoc: (OM_descriptor): is used to identify an instance of a managed object class i.e. an instance of a service using an integer.
- eType: (OM_descriptor): it contains the OID of the notification.
- sUser: (OM_descriptor): DN of the initiator MAE.
- sProvider: (OM_descriptor): DN of the target MAE.
- addInfo: (OM_descriptor): OID of the target MAE in case of emmision of a service report notification.
- addText: (OM_descriptor): OID of the initiator MAE in case of emmision of a service report notification.
- sACause: (OM_descriptor): OID of the security alarm cause.
- sASeverity: (OM_descriptor): Severity of the event (MAJOR / MINOR / WARNING / CRITICAL / INTERMEDIATE)
- srCause: (OM_descriptor): OID of the service report cause.

- saInfo: (OM_descriptor): Security Alarm Info in case of a service report notification.
- setEventReport: (OM_descriptor) contains a CMIP event report.
- eInfo - Security Alarm Info.

6.8.2 Supported component interfaces

6.8.2.1 Introduction

In the following figure the SELF interfaces are shown. The SELF is the agent part of the security audit and alarm architecture. It accepts notifications from the SMASC about security events that happened in the environment. These notifications are communicated as string through a TCP/IP socket interface. The agent transforms these notifications into CMIP event reports and sends them to the security audit and alarm manager in order to be displayed on the operator's screen. The interface between the agent and the manager is a CMIP interface.

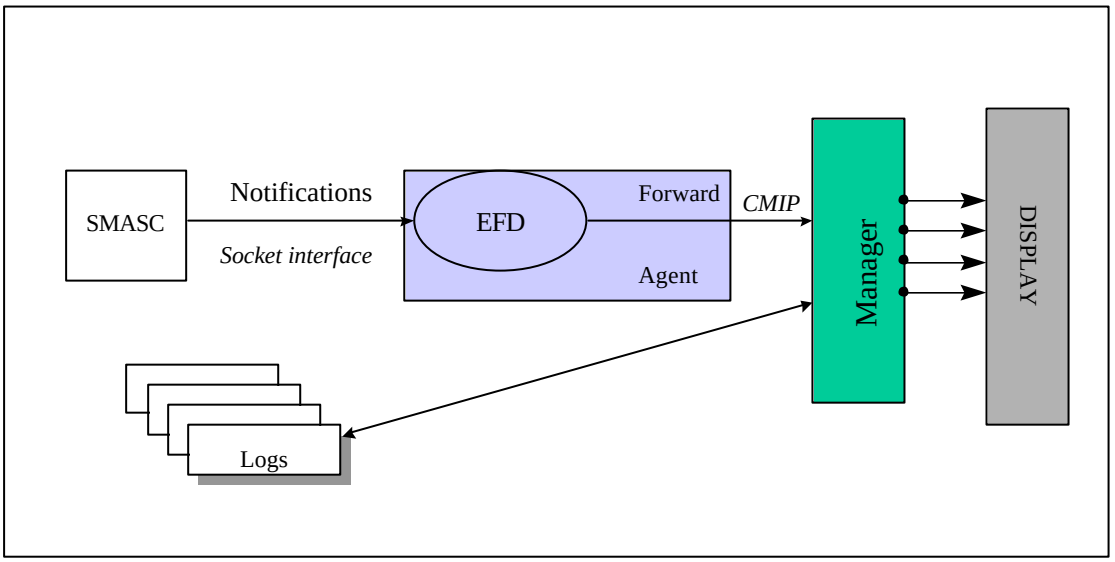

Figure 51: SELF Interfaces

6.8.2.2 Interface between the SELF and the SMASC

The SMASC and the SELF communicate through TCP/IP sockets. There is a number of security events that the SMASC sends to the agent and these are listed below.

```
/* Security events as defined in D9 + "NULL string event" in order to
use the serviceReport Notification */
#define AUTH_F "authenticationFailure"
#define KEY_EXP "keyExpired"
#define IMD "informationModificationDetected"
#define DI "duplicateInformation"
#define BOC "breachOfConfidentiality"
#define UAA "unauthorizedAccessAttempt"
#define OOS "outOfService"
#define HAA "outOfHoursActivity"
#define UR "unspecifiedReason"
#define NO_EVENT "NULL"
```

In what follows the structure of the notification information received by the agent process is described:

FUNCTION open_AA_connection

SYNOPSIS

```
void open_AA_connection (char *hostname, int Port)
```

DESCRIPTION

This function is used to open a connection with the agent process.

ARGUMENTS

- *hostname* defines the hot where the SELF runs, the IP address of the host can also be given.
- *Port* is the listening port of the SELF agent.

FUNCTION send_info

SYNOPSIS

```
void send_info( enum Service_Type service, int service_id, char *EType,
char *Suser, char *Sprovider, char *Info_msg)
```

DESCRIPTION

This is the function used to send the notification.

ARGUMENTS

- *service* defines a service chosen from the available set of services defined in section 5.9 {Auth,KeyMgmnt,Int_Conf,AC,CPN_Ser,Cust_Server_Ser,Xuser_VASP_Ser,Xuser_PNO_Ser};
- *service_id* is used to identify the part of the system, a security or management event has been produced from.
- *EType* defines the event from the list of available events described above.
- *Suser* represents the DN of the initiator MAE.
- *Sprovider* represents the DN of the target MAE.
- *Info_msg* is used to convey information relevant to a security or management event

FUNCTION send_AA_notif

SYNOPSIS

```
int send_AA_notif (char *str)
```

DESCRIPTION

This function sends actually the notification by filling in the notification string with other information, which is the library version and the instance of the service.

FUNCTION close_AA_connection

SYNOPSIS

```
void close_AA_connection ()
```


DESCRIPTION

This function closes the connection to the agent process and should always be called after sending the notification.

6.8.2.3 Interface between the SELF and the Audit Manager

This interface is accomplished though a number of XOM objects (all of type OM_descriptor) since the communication between the agent and the manager is done using the CMIP protocol:

- setEventReport: this is the XOM object representing the CMIP event report sent to the manager. It follows the X736, X740 standards and the type of information set includes the Managed Object Class, the Managed Object Instance the Event Type the Event Time and the Event Info. The Event Info is realised by the following two XOM objects.
- eInfo: this object is used for a security related event.
- saInfo: this object is used when a service report notification is sent.

In here we describe the format of the CMIS event reports sent to the audit and alarm manager by the SELF.

Contents of the Event Report

Name of the attribute (GDMO)	**Contents**
Managed Object Class	Service which raise the event (integer converted in oid)
Managed Object Instance (in the attribute value of the first ava)	service identifier
Event Time	actual time (generalized time)
Event Type	oid of the notification
Event Info	Security Alarm Info or Security Audit Info

Contents of the Security Alarm Info

Name of the attribute (GDMO)	**Contents**
Security Alarm Cause	oid of the security alarm cause
Security Alarm Severity	integer
Security Alarm Detector	empty
Service User (attribute Detail)	DN of the entity which caused the raising of the alarm + trace information
Service Provider (attribute Detail)	DN of the entity which has been «attacked »

Contents of the Security Audit Info (Service Report Notification)

Name of the attribute (GDMO)	**Contents**

Service Report Cause	oid of the service report cause
Additional Text	DNof the entity which caused the raising of the report + trace information
Additional Information	DN of the target entity

Always a management event will generate a ServiceReport Notification.

6.8.3 Version, Release history

The latest version is version 2.0 , July 1998.

7 REFERENCES

[Gos95] The Java Language Environment, A White Paper, James Gosling, Henry McGilton, Sun Microsystems, October 1995.

[Kern83] The ANSI C Programming Language, Brian Kernighan and Dennis Ritchie, Prentice Hall, 1983.

[MISA-D3-A1] Initial MISA High Level Design, Annex A1: Xuser Interface Definition, ACTS AC080 MISA Deliverable 3, Annex A1, September 1996

[MISA-D3-A2] Initial MISA High Level Design, Annex A2: Path Provisioning Ensemble, ACTS AC080 MISA Deliverable 3, Annex A2, September 1996

[Orbix96a] Orbix 2 Programming Guide, IONA Technologies, October 1996.

[Orbix96b] The Orbix Architecture, IONA Technologies, November 1996.

[Orfali97] Client/Server Programming with JAVA and CORBA, Robert Orfali and Dan Harkey, John Wiley & Sons, Inc., ISBN 0-471-16351-1, USA, 1997.

[OMG] http://www.omg.org

[OMG-970301] IDL/Java Language Mapping, Joint Revised Submission, OMG TC Document orbos/97-03-01, 19/3/1997

[RFC 1508] RFC 1508, “Generic Security Service Application Program Interface”, September 1993

[Str86] The C++ Programming Language, Bjarne Stroustrup, Addison Wesley, 1986

[TRUMPET-D6] ACTS AC112 Trumpet Deliverable 6, NIL-Security Prototype Report, February 1997.

[TRUMPET-D8] ACTS AC112 TRUMPET Deliverable 8, "Detailed Component and Scenario Design ", June. 1997

[TRUMPET-D9] ACTS AC112 TRUMPET Deliverable 9, " System Component Specification ", October. 1997

[TRUMPET-D11B] ACTS AC112 TRUMPET Deliverable 11B, "Implementation (Prototype)", January. 1998

[UML] Unified Modelling Language, RATIONAL Software Corporation, http://www.rational.com/

[XOM] X/Open Company Limited & X.400 API Association, XOM, OSI-Abstract-Data Manipulation API, CAE Specification, 1991

8 ACRONYMS

API	Application Programmer's Interface
ATM	Asynchronous Transfer Mode
BSP	Binary Space Partition
CA	Certification Authority
CIM	Common Information Model
CIS	Communications Interface System
CMIP	Common Management Information Protocol
CMIS	Common Management Information Service
CMISE	CMIS Element
CORB	Common Object Request Broker
A	Architecture
CPN	Customer Premises Network
CRL	Certificate Revocation List
DII	Dynamic Invocation Interface
DN	Distinguished Name
DSI	Dynamic Skeleton Interface
EFD	Event Forwarding Discriminator
EML	Element Management Layer
EMS	Event Management System
GBC	Global Broadband Connection
GBCM	Global Broadband Connectivity Management
GDMO	Guidelines for the Definition of Managed Objects
GUI	Graphical User Interface
HMM	Hyper-Media Management
HPOV	Hewlert-Packard Open View
HTML	Hyper-Text Mark-up Language
HTTP	Hyper-Text Transfer Protocol
IBC	Integrated Broadband Connection
IDL	Interface Definition Language
IIOP	Internet Inter-ORB Protocol
IP	Internet Protocol
JDBC	Java DataBase Connectivity
JDK	Java Development Kit
JLDAP	Java LDAP
JMAPI	Java Management API
LAN	Local Area Network
LDAP	Lite Weight Directory Access Protocol
LIF	Local Interface
MIB	Management Information Base
MO	Managed Object
MOS	Managed Object Server
NEL	Network Element Layer
NEV	Network Element View
NL	Network Layer
NML	Network Management Layer
NMS	Network Management System
NV	Network View
ODP	Open Distributed Processing
OMG	Object Management Group
ONP	Open Network Provision
ORB	Object Request Broker
OS	Operation System
OSF	Operation System Function
OSI	Open Systems Interconnection
PC	Personal Computer
PNO	Public Network Operator
PTT	Postal, Telegraph and Telephone
QAF	Q Adapter Function
QATM	ATM QAF
QoS	Quality of Service
RDN	Relative Distinguished Name
RMI	Remote Method Invocation
SAC	Service Access Control
SDH	Synchronous Digital Hierarchy
SII	Static Invocation Interface
SL	Service Layer
SME	Small and Medium Enterprise (Market)
SNC	Subnetwork Connection (as in releaseSNC, etc.)
SNMP	Simple Network Management Protocol
SOHO	Small Office Home Office (Market)
SSI	Static Skeleton Interface
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TMN	Telecommunications Management Network
TTP	Trusted Third Party
UML	Unified Modelling Language
URL	Universal Resource Locator
VASP	Value Added Service Provider
VP	Virtual Path
VPI	Virtual Path Identifier
WAN	Wide Area Network
WPx	Work Package Number x
XMP	X/Open Management Protocols API
XOM	X/Open OSI-Abstract-Data Manipulation API

