Project no. IST-033576

XtreemOS

Integrated Project
BUILDING AND PROMOTING A LINUX-BASED OPERATING SYSTEM TO SUPPORT VIRTUAL ORGANIZATIONS FOR NEXT GENERATION GRIDS

First Specification of Security Services

D3.5.3

Due date of deliverable: 31/05/2007
Actual submission date: 29/05/2007

Start date of project: June 1^{st} 2006

Type: Deliverable
WP number: 3.5
Task number: 3.5.2

Responsible institution: Rutherford Appleton Laboratory,
Science & Technology Facilities Council,
Harwell Science and Innovation Campus,
Didcot, Oxon OX11 0QX, United Kingdom
Editor & and editor's address: Erica Y. Yang and Amit D. Lakhani

Version 2.4 / Last edited by Erica Yang / 29/05/07

Project co-funded by the European Commission within the Sixth Framework Programme		
Dissemination Level		
PU	Public	
PP	Restricted to other programme participants (including the Commission Services)	√
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Revision history:

Version	Date	Authors	Institution	Section affected, comments
0.0	27/03/07	Amit Lakhani	CCLRC	first draft
0.1	01/05/07	Philip Robinson	SAP	second draft
0.2	03/05/07	Erica Yang	STFC	(with Gregor Pipan)added 1st version of AEM use cases and APIs - JobCreation, ResourceMatching and JobExecution
0.3	04/05/07	Amit Lakhani	STFC	added 1st version of resource management use cases and APIs
0.4	04/05/07	Gregor Pipan	XLab	added 1st version of AEM use cases and APIs - Resource Negotiation
0.5	09/05/07	Erica Yang	STFC	added 1st version of user management use cases and APIs
0.6	09/05/07	Philip Robinson	SAP	included general use cases and those for XtreemFS
0.7	09/05/07	Erica Yang	STFC	added 1st version of security services, in particular, mutual authentication
0.8	10/05/07	Ian Johnson	STFC	added use cases for VO Management
0.9	14/05/07	Amit Lakhani	STFC	Added Background-Federated Resource Management
1.0	14/05/07	Erica Yang	STFC	split and updated mutual authentication into two sections in chapter 4: introduction and mutual authentication
1.1	14/05/07	Philip Robinson	SAP	Added background section on data management and architecture section on Authorization
1.2	15/05/07	Amit Lakhani	STFC	Added Secure Communications section in Specifications of Security Services
1.3	15/05/07	Philip Robinson	SAP	Added new section on Isolation and updated the Authorization section
1.4	16/05/07	Amit Lakhani	STFC	Edited Specification of Security Services, Updated and edited Use cases chapter, added bibliography file
1.5	17/05/07	Erica Yang	STFC	removed APIs and implementation notes, added open issues chapter
1.6	17/05/07	Erica Yang	STFC	restructured introduction chapter, updated open issues
1.7	17/05/07	Amit Lakhani	STFC	Edited XFS Use Case section, Added package hyperref for easier navigation
1.8	17/05/07	Erica Yang	STFC	Modified Open issue - authentication, executive summary
1.9	22/05/07	Erica Yang	STFC	removed diagram from background AEM section and added address
2.0	24/05/07	Amit Lakhani	STFC	Updated Resource Management Use case diagrams, included internal reviewer comments
2.1	29/05/07	Erica Yang	STFC	added citations, modified Spec-authN, Spec-Intro
2.2	29/05/07	Erica Yang	STFC	added appendix, incorporated reviewers' comments by amended BG-Isolation, updated spec-intro, spec-authn, openissues-authn, usecase-VO Mng, use case - User Mng
2.3	29/05/07	Erica Yang	STFC	slightly revised introduction chapter and added open issues - resource sharing
2.4	29/05/07	Erica Yang	STFC	corrected a few typos

Contents

List of Figures

1 Executive Summary

This document is the first specification of security services for XtreemOS. It first presents the outcomes of an extensive investigation into the overall security requirements of the XtreemOS system architecture, including both F-layer system and G-layer Grid services. The presentation follows a bottom-up approach starting from describing the F-layer security requirements and solutions of existing Linux, and those of Kerrighed, the Single System Image (SSI) system, to those of the XtreemOS G-layer Grid infrastructual services. The result of this study has been successful as it consolidates our understandings on the XtreemOS security requirements while giving us a platform to derive *solid and tangible* use cases from the essential aspects of the XtreemOS system.

The core of this deliverable is the specification of a set of security services for XtreemOS. They are designed to support the XtreemOS VO-centric approach towards Grid computing and include: VO membership, identity, attribute, credential distribution, and policy services. Together, they offer a foundation to support a range of key XtreemOS security functionalities, including mutual authentication, authorization, secure communication, and isolation in a VO-centric Grid environment. As XtreemOS is a Grid operating system, the major challenge for the security design is to strive a fine balance between scalability and usability (e.g. the realization of transparent Grid access to end users).

This deliverable is backed by a wide-range selection of XtreemOS use cases, including management of VOs, users, resources, data, and application execution. The aim is to illustrate how to secure an XtreemOS-based Grid system using the set of foundation security services specified.

The essence of this deliverable captures a snapshot of the ongoing architectural design and implementation work undertaken by work package 3.5. The document itself is a milestone that records our understanding of the security challenges and exploration of solutions. To the work package as a whole, this is still an ongoing process. We would expect the incorporation of refinements and adjustment of our design to reflect our on-going interactions with other work packages. Therefore, this document ends with a brief discussion of the open issues that may produce an impact on the future development of this work package.

2 Introduction

This section defines some fundamental concepts and models used in this deliverable and describes how we organize this document.

A working definition of Virtual Organization (VO), taken from D3.5.2 - Security Requirements for a grid-based OS, is described as follows:

"A VO can be seen as a temporary or permanent coalition of geographically dispersed entities (individuals, groups, organizational units or entire organizations) that pool resources, capabilities and information to achieve common objectives. There usually will be legal or contractual arrangements between the entities. The resources can be physical equipment such as computing or other facilities, or other capabilities such as knowledge, information or data."

2.1 Security Concept and Model

The XtreemOS security concept and model follows four notions of entity (subject or object) protection and policy enforcement, as depicted in figure and explained below, based on the Virtual Organization (VO) concept established in the Grid Computing community.

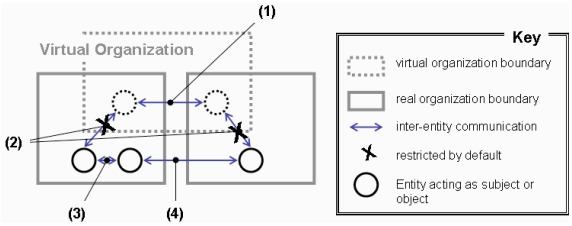

Figure 1: A general security model for XtreemOS, based on boundaries and privileges surrounding VO membership

1. Permissions of interactions between entities within a VO boundary are decided based on authorized VO membership and roles issued

2. Interactions between entities internal to a VO and those outside the virtual boundary (i.e. non-members) are restricted by default and require special privileges to interact; otherwise the non-member must go through the membership-joining process

3. The existence of a VO should not interfere with the mechanisms protecting interactions between entities in the same domain that are not acting as members of a particular VO

4. Policies governing established, non-VO cross-domain interactions should still be supported and their security enforced according to authorizations agreed on between the interacting domains

Note that this is technically a difficult model to enforce and is for the most part theoretical, in that external parties cannot override the policies of a local administrator. Secondly, the virtual existence of a VO suggests that the interpretation of the boundary may differ from domain to domain. For this reason the core VO-related security enforcement mechanisms are to be implemented within the operating system, assuming that installations of XtreemOS are obtained from reliable sources[1]. Core VO-related security enforcement mechanisms include those concerned with enabling confidentiality of stored data, confidentiality of data communicated over networks, integrity of stored data, integrity of communicated data, identification and authentication of users, authorized access to resources and services, guaranteed access to resources and services by authorized parties, accountability of data access and service execution, isolation of data and services, according to the model of boundaries and privileges surrounding VO membership.

2.2 Structure

The remaining of this document is organized as follows.

Chapter 3 presents outcomes of an extensive background research to this deliverable. It elaborates on the key functional description on and security requirements of both F-layer system and G-layer Grid services of XtreemOS. The F-layer XtreemOS system services cover conventional Linux operating systems and Kerridge, the Single System Image (SSI) system. The readers who are familiar with the functional aspects of XtreemOS can jump straight into the elaboration on the security requirements.

Chapter 4 is a specification of the security services for XtreemOS. This chapter consists of two parts. The first presents a high level description of the overall approach we are undertaking and elaborates the list of security services we define. The second details how these services are used to support four key security functionalities of XtreemOS, they are: mutual authentication, authorization, secure communication, and isolation.

[1] Typically, this should be assured by XtreemOS system administrators.

Chapter 5 presents a range of use cases to explain how the set of security services defined in Chapter 3 can be used to support management of VOs, users, resources, data and application execution in XtreemOS.

Chapter 6 is a brief summary of a list of challenging open issues on the design of the security services. The inclusion of this list is to demonstrate our awareness of potential obstacles in the deployment, development and integration stages of our project.

Chapter 7 is an appendix, which offers a preliminary overview of the format and content of the major security credential used in XtreemOS.

3 Background

This chapter provides an overview of functional and non-functional requirements of XtreemOS, which have influenced the security architecture presented in this document. These surround the conceptual models, applications and technologies that contribute to the overall features of XtreemOS.

3.1 Linux Security Overview

There are many facets of Linux security and we only focus on existing Linux security mechanisms that could be leveraged or extended by VO security services in WP3.5.

3.1.1 Fundamental Linux Security

Fundamental Linux security is achieved with user and group management, as well as file permissions. Based on this, the Linux Kernel implements a Discretionary Access Control (DAC) model. DAC means that users and programs have discretion over the objects (e.g. files, directories, sockets) in their control. Owners of objects could determine, modify, or grant the access rights of objects at their will. DAC is a simple but powerful solution that is important for keeping the integrity of each user's data in a multi-user environment.

UID, root and file permissions A Linux user, identified by a unique number (UID), can belong to one or more groups identified by group IDs (GIDs). Each process is associated with UID and GID information (called *process credentials*) when it is created. When a process accesses a file, the kernel checks the process credentials to determine which set of permission bits (e.g. `rwx` for *o/g/w*) of the file will be applied. Once a malicious process gains success to impersonate the root (the super user with UID of 0), the kernel will bypass permission checks of this process, thus, it has all privileges to perform system administration actions. That is why traditional DAC model of Linux has been heavily criticized for its vulnerability.

Capabilities The Linux Kernel also supports POSIX.1e capabilities that impose granular access control on processes. A capability is a flag that manifests whether the process can perform a specific operation. Comparing to traditional *uid-root-permission bits* model in which a process can either do everything or do nothing, the capabilities model provides more constrained access control as a program could be only granted a limited number of operations. Unfortunately the current capabilities support in Linux is limited and not applicable to file systems.

Access Control List (ACL) POSIX ACLs extends the traditional POSIX file system object permission model (rwx permissions for user, group and others) and allows to specify different read, write and execution permissions not only for one, but for a list of users and groups. POSIC ACLs are defined in POSIX 1003.1e draft 17 . Sponsorship for this family of standards was withdrawn, which means that they are unfinished, even though they seem to be in quite mature a state. Patches that implement draft 17's ACLs have been available for various versions of Linux for several years and they are now part of the 2.6 Linux kernel.

3.1.2 Pluggable Authentication Module(PAM)

PAM (Pluggable Authentication Modules) is a suite of shared libraries conforming to a set of abstraction APIs, i.e. PAM APIs, which covers security-related tasks including authentication, authorization, logging, accounting, and so on. PAM allows the implementation of security schemes to be independent of Linux system services and applications. All PAM-aware system services (i.e. those rely on PAM APIs for authentication and authorization) could easily switch from using one security mechanism to another, without the need of changing their source codes. This is gernerally accomplished by configuring the use of PAM libraries in external files, like config files in `/etc/pam.d`.

PAM deals with four separate types of management tasks: *authentication management, account management, password management,* and *session management*. For each type of task, multiple PAM modules could be configured to form an invoking chain (or stack), which allows for a modular and flexible development of PAM plugins. The Linux-PAM library consults the contents of the PAM configuration file and loads the modules that are appropriate for the requesting application. Textual information, required from/or offered to the user, can be exchanged through the use of the application-supplied conversation function.

As a de facto standard interface of authentication, PAM has been supported by most Linux/Unix systems. Based on a generic framework, new PAM modules keep on emerging to fit with different sort of applications needs.

3.1.3 Linux Security Module (LSM) framework

LSM provides a collection of hooks in kernel which make it possible for developers to custom special security check policies for various objects access. The LSM framework is policy agnostic in itself and only provides interfaces between kernel objects access and various secure policy check codes which are implemented in different kernel modules.

Access control is nothing but a way to check whether a subject (for example a process) can execute an operation (read/write/execute) on an object (for example

a file). To support this, the LSM provides a hook for every place in kernel which is required to make such a check. In the LSM, each object and subject have a label which is defined and interpreted by concrete policy modules. A policy module utilizes the subject/object label pair to determine whether the required operation can be done or not.

The LSM framework is not a replacement but an enhancement to traditional DAC and ACL of Linux. The classical Linux DAC checking is performed before running LSM hook code whenever the kernel is about to access internal objects (tasks,inodes,etc.). Different secure policies can be implemented on top of LSM, such as the SELinux implementation.

3.1.4 Security Enhanced Linux (SELinux)

There are a variety of projects that provide enhanced Mandatory Access Control (MAC) by patching Linux kernels. The MAC model follows the principle of least privilege. In a MAC-based environment, application capabilities and privileges are set by pre-defined policies rather than owners of resources. The attacker is limited to the actions allowed by the system's security policy.

Among those MAC enhancements of Linux, SELinux may be the only one that was accepted into the mainline Linux 2.6 kernel series, as well as some distributions like Fedora Core and Gentoo. SELinux-enabled kernel enforces MAC policies that confine user programs to the minimum amount of privilege they require to do their jobs. Even being a root user, access to resources could also be denied according to predefined rules. SELinux operates independently of the traditional Linux access control mechanisms.

In SELinux every process or file has a context which is comprised of three parts: an identity, a role, and a domain (also called *type*). The identity is the name of the Unix account or system build-in default user names. The role determines which domains are permitted and is used to restrict the transitions to other domains. A domain is a sandbox-like combination of subjects (e.g. *processes*) and objects (e.g. *files*) that may interact with each other. For example, someone who logs in as a regular user, with a regular user role, can never enter into administration domain. This kind of access control is called Type Enforcement (TE). The context of a file is stored within the file system as extended attributes, and the process contexts are stored by the kernel.

The SELinux policy describes the access permissions for all subjects and objects, i.e., the entire system of users, programs, and processes and the files and devices they act upon. Policies could be customized and configured for achieving different levels of access control.

3.2 Overview of Kerrighed Security

In this section we overview the Kerrighed security issues. As the type of installation of the Kerrighed distribution also implies associated security risks, we first present its three types:

- using `NFSROOT` and booting the disk-less nodes using the PXE mechanism.
- using `NFSROOT` and booting nodes with installed Kerrighed.
- manually installing Kerrighed on nodes and manually starting the Kerrighed session.

Security risks associated with each of the installation types are specifically mentioned whenever they present a special kind of risk. In the following, we start with assumptions about the environment and proceed with scenarios originating from these assumptions.

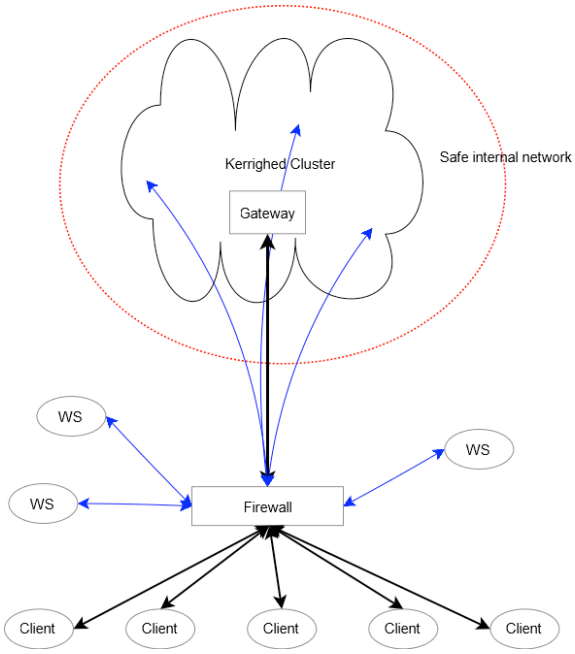

Figure 2: Kerrighed deployment scenario.

Figure 2 shows the recommended deployment scenario for Kerrighed clusters. It shows that the Kerrighed cluster should be deployed on a private network (VPN, dedicated switch, VLAN, etc.), using one cluster node as the gateway — holding the usernames for the cluster users. The whole cluster should be hidden behind the firewall. Users should connect to the gateway through the firewall, using SSH[2]

[2]We are aware of the current SSH exploits – these cannot be avoided.

protocol, while any outbound (i.e., requests for web services that are not in the private network) requests originating from the cluster should also go through the firewall. In the following paragraphs we give a more in-depth explanation of the Kerrighed deployment scenario and the associated security risks.

Use of private network: The Kerrighed distribution is currently not designed to operate on the publicly accessible network – it currently does not implement secure communications between internal nodes.

Cluster is hidden behind a firewall: The Kerrighed cluster should be located behind a firewall that allows incoming connections only from well-known IPs. The connections originating in the Kerrighed cluster should also be filtered and allowed only if they are well-known, thus reducing the chance of compromising the system by malicious services. Of course, if the user's machine is already compromised and he connects to the cluster, the chances are that the cluster will be vulnerable, but this human factor cannot be solved only by IP filtering.

Users should SSH themselves to the Gateway: The gateway is part of the cluster and has the information about all users that connect to the cluster. If the firewall allows the connection (the user is trying to connect from a well-known IP), the gateway will accept the connection only if presented with a valid certificate and then spawn the corresponding user session.

Cluster is static in terms of adding new nodes: The Kerrighed distribution currently does not allow hot-plugging of new nodes. Currently `NFSROOT` is most used, but support for hot-plugging is under development. If the nodes are disk-less, i.e. ramdisk is used, then the absence of dynamicity in nodes addition/retraction (and also use of `NFSROOT`) results in absence of nodes with different set of installed software. In case where nodes have disks and `NFSROOT` is used only for Kerrighed sessions (started manually by `modprobe kerrighed` and `krgadm cluster start`), the software that is installed locally on nodes should be verified or the administrator should not grant the installation permissions to the users.

Kerrighed should be used on secure distribution of Linux: The fact that Kerrighed consists of set of patches for the standard kernel, module and a set of user tools serves the conclusion that Kerrighed is as secure as the base underlying distribution. Currently we are not aware of any security issues with Kerrighed patches and module.

The summed up assumptions and facts about the Kerrighed deployment environment thus are:

- *Internal communications security*
  - Kerrighed cluster is working on a private network (VPN/dedicated switch/VLAN).

  - There is no hot-plugging of new nodes - use of common `NFSROOT` on disk-less nodes, but support for hot-plugging is under development. The case where nodes have disks and thus locally installed software should be closely monitored by the system administrator.
  - Kerrighed is as secure as the underlying Linux.
- *Inside/outside security*
  - Kerrighed cluster is behind a firewall.
  - Firewall allows connections to/from well known IPs.

Installing Kerrighed and associated security risks The general use-case scenario for installing Kerrighed is given in on the Kerrighed web site[3]. It describes installation on one machine which then acts as a boot server for other nodes in the cluster. Currently the Kerrighed distribution does not support hot-plugging, hence the number and IP addresses of the nodes are hardcoded into the NFSROOT initalization as shown below. It is evident that changes to the Kerrighed cluster initialization require root privileges.

```
group {
 filename "/pxegrub";
 option grub-menu = concat("(nd)/grub/", host-decl-name);
 option root-path "/NFSROOT/kerrighed";
 host ssi1 { fixed-address 192.168.0.101;
hardware ethernet xx:xx:xx:xx:xx:xx; }
 host ssi2 { fixed-address 192.168.0.102;
hardware ethernet xx:xx:xx:xx:xx:xx; }
 host ssi3 { fixed-address 192.168.0.103;
hardware ethernet xx:xx:xx:xx:xx:xx; }
 host ssi4 { fixed-address 192.168.0.104;
hardware ethernet xx:xx:xx:xx:xx:xx; }
}
```

Internal communications security If we assume that we're running on a safe internal network, we can reduce the threat of attacks on the internal Kerrighed communications. Still, if the user connects from compromised machine he may also compromise the cluster. Internal DOS attack (cluster nodes are sending packets to one node) is possible using properly forged raw packets. Using this approach

[3] `http://www.kerrighed.org/wiki/index.php/Kerrighed_on_NFSROOT`

the Kerrighed cluster can crash, but in order to do that, the user has to have credentials that allow him to send raw packets, which in turn means, that the user has root rights, i.e., he can do anything he pleases.

We proceed with scenario where we add a compromised machine to the cluster. As the current distribution does not support stable hot-plugging, we can conclude that there are no attack possibilities here[4]. Later on, when hot-plugging is possible, the node being added will have to be checked against malicious software. As setting up and adding a new machine to the cluster is usually administrator's work, the integrity of the new machine is up to him. If we now focus to the current common and recommended use of the Kerrighed cluster by using common file system tree (i.e. the `NFSROOT`), we can conlude that this is the only possible attack point. If users introduce malicious software (either on the `NFSROOT` or on nodes' local disks, if they are present), they can crash the cluster, but in order to do that, they have to have proper rights – thus it is critical that the system administrator has granted proper rights (restrictions) on the users.

Finally, we observe that Kerrighed is a patch to the standard Linux kernel along with `kerrighed` module and a set of user tools. Currently we're not aware of any exploits introduced by these changes, so we assume that Kerrighed is as secure as the underlying Linux distribution – we recommend Debian, which is considered to be one of the most stable and secure Linux distributions. Additionally, Debian also supports use of PAM and LSM.

Inside/outside security Integral part of the overall Kerrighed cluster security is the firewall which controls the barrier between inside and outside of the cluster. First and foremost, the users have to use well-known IPs in order to connect via SSH to the gateway. If their jobs need services or data that is available outside the cluster, then they need to provide the IPs of their services to the firewall and also provide properly secure channels for communication. Still, the security of the cluster is dependent on the human factor.

Security issues In this section we present the identified security risks and give our assessment of their current status.

- *If Kerrighed cluster is running on a exposed, public network, the messages between nodes can be easily intercepted, as the communication between nodes is currently unencrypted.*

- *When Kerrighed will support hot-plugging new nodes, it will be possible*

[4]Please note that this conclusion will have to be revised with progress of Kerrighed development.

that new nodes contain compromised software. Also, when nodes are not disk-less, the software installed locally may be malicious.

Using clusters in business environments includes having good network and system administrators. Network administrators are responsible for proper setup of the private network and firewall, while system administrators must check (or better set up) the new node. Proper set up of the network and firewall is of utmost importance, as it prevents sending the data to third party services outside the cluster network. The verification of the software installed on the new node software is important in order to prevent malicious internal actions (e.g., cluster crashing, etc.). We conclude that both threats are moderate or low.

Overall, the security risks associated with use of the Kerrighed cluster with current Kerrighed distribution are scaled down to good system and network administration and the underlying Linux distribution and kernel.

3.3 Virtual Organization - Sharing under Isolation

One of the major benefits of introducing the concept of Virtual Organization into the IT world is the potential of transforming (existing or new) IT systems to fulfill the requirements of project-oriented groups, which, in today's world, typically span across multiple physical organizations and administrative domains. However the establishment of a VO has to be in accordance with local IT and data protection policies. As a result only dedicated users or user groups are allowed to share dedicated resources and data while company dependent security guidelines and data protection policies apply for all other employees. Thus it is essential for a VO framework to handle subjects and roles they can act in, data/resource objects and the granted rights to.

3.3.1 Virtual Organization and Sharing of Data and Resources

Sharing within a Virtual Organization can take place on two different levels. On one hand this is sharing of resources, e.g. computing power which has no consequences to users data. On the other hand, data, objects and interfaces could be shared which renders common shared data. Independent of the kind of sharing, this has to be defined by a set of policies which only empower dedicated users.

Resource sharing - A lot of work has been done in order to make effective use the resources of IT systems. Often, the experience of resource utilization is much improved by giving users and applications the illusion that they have an exclusive access to all the available resources. A resource manager, a piece of software, is in place to handle the load of users and enable efficient operations. Typically, it

is very important to note that neither users nor their applications should deal with the actual resource management at the system level. For example, the management of concurrent access to data or resources should be managed by the resource manager.

Technically, there are multiple ways to achieve a better resource utilization. Storage can be, for example, reused by the use of copy-on-write technologies which ensure that only the differences between the data belonging to different users will be stored. Such technologies can also be used to execute multiple instances of a program in a very memory efficient way. They can also improve the startup time of applications or even that of virtual machines when being used in that context.

On the other hand, data should still be possible to be shared between users and their applications even users work in different projects. It is very important to note that resource managers are no longer present to (1) mediates access to data, objects or interfaces; or (2) ensure data/object consistency.

Data sharing - Exchanging information among entities in a VO is essential for achieving a common goal. Typically, only a few entities in this construct have the full access privileges, while some others have much restricted access rights. This is very much an application-specific issue and operating systems cannot mediate access requests with a predefined set of strategies based on which a resource manager has implemented.

Enabling such accesses depends on the subjects, objects and access rights involved in the applications executed on the IT system. Since users in a Virtual Organization are represented by their VO identifier, authorizations represented by access rights and credentials can be bound to their identifier. Hence, it is essential to identify the subjects, objects/resources in the context of a VO and a operating system.

Subjects in an VO and operating system context

- Owner of physical infrastructure (hardware, network and storage).
- Administrators of physical infrastructure.
- Users:
  - Local users
  - Global users, VO user, VO initiator
- Roles: Depending if a rule based access control concept is used, rules can be applied to:

  - VO administrators and users
  - Local administrator roles

Data objects and resources in an operating system context

- Filesystems, volumes, files
- IPC
  - Pipes
  - Shared memory
  - Semaphores
- Service access points and network sockets
- Methods of objects/function calls

The above is a list of data objects that can be used for data sharing among Virtual Organizations and the access to them is needed to be controlled. However, the access rights which have to be controlled depend on the type and access methods of the data objects and sources. Both VO users and their rights within a VO have to be managed and be made available via a suitable infrastructure. Although these issues are discussed in the remaining parts of this deliverable, they will be tackled in greater detail in the next specification of security services.

3.3.2 Virtual Organisation and Isolation

One of the fundamental goals of XtreemOS is to provide an abstraction that makes the complexities of distributed hardware and secure resource sharing between different sites transparent. The Virtual Organization concept provides such an abstraction, but in addition it also prevents unauthorized access and thus logically isolate the Virtual Organization. This can also be seen as the counterpart to the sharing aspect within a Virtual Organization by denying access to all other resources and data. This becomes increasingly important for Grid systems that handle critical data and have high integrity requirements for data and processes. Isolation therefore presents relatively hard security requirements for XtreemOS.

Functional Description The resources of a VO may be physically dispersed and under the governance of different administrators. A VO defines a logical boundary around a set of specified resources, which indicates some default access constraints. That is, a resource r1 accessible in VO1 is not accessible in VO2.

Each physical resource in a VO is therefore assigned a VO-specific reference identifier, which may differ from their global identifier when outside of the boundary. Any member and resource in a VO has the following elements:

- *Entity attributes* that describe their identities, capabilities and functionalities
- *Objects* that hold computational data and state information, which change as they interact
- *Interfaces* that allow for them to interact
- *Services* (or processes) that are executed as a result of interaction; services perform operations on local objects or invoke local or remote interfaces

A VO is also considered to be an entity with unique attributes, such that these attributes are inherited by its members and resources. This VO-wide attribute set is referred to as the *VO Context*. The above elements can however be used in none or more VO contexts; Elements used in 0 contexts are referred to as *root elements*, such that they are the attributes, objects, interfaces and services in the real world. Within a VO they may be created as pointers, aliases or special instances of these root elements. The goals of isolation are also defined with respect to this categorization of elements:

- *Attribute Isolation* ensures that selected attributes used to identify a resource in a particular VO context are the only set of attributes that can reference that resource, such that the resource cannot be referenced outside of the VO context (e.g. machinex <– VO1)

- *Object Isolation* ensures that application, computational and state data assigned to VO context vo cannot be viewed or altered by processes in vo'. In addition, any memory allocated for these objects cannot be used for objects outside of the VO context.

- *Interface Isolation* ensures that invocations of resources in one VO context cannot be used externally or to leak information outside contexts.

- *Service/Process Isolation* ensures that the execution of services or processes outside of the VO context do not interfere with the execution of those internal, even when failure occurs[5].

A VO that requires all of these isolation types to be satisfied is known as *hard-isolation*. Isolation in a distributed system is nevertheless a hard problem. Isolation in an inter-domain system, such as in a Grid environment, is even more

[5]Could also argue that mutual isolation is perhaps necessary, to ensure that internal processes and services do not interfere with external services and processes.

difficult for reasons beyond technical. XtreemOS should be capable of achieving this level of isolation, as a fundamental objective, yet capable of flexibly adjusting the level of isolation based on the requirements of the application.

Overlap with Security Requirements The security services implemented in XtreemOS should provide functionalities to support different levels of isolation as well as the enforcement of the logical VO boundary and context. However, these requirements are not exclusive from each other, such that the same mechanisms can be used together with other mechanisms for different types of isolation.

1. **Security requirements for VO context creation and boundary enforcement:** A VO context requires a unique name attribute that is a universally unique identifier (UUID). There are already three UUID generation functions in standard Linux available in the uuid/uuid.h header library, using different algorithms for determining randomness and uniqueness. These may be part of e.g. a *create-vo-context* method.

   ```
    void uuid-generate(uuid-t out);
   void uuid-generate-random(uuid-t out);
   void uuid-generate-time(uuid-t out);
   ```

   Secondly, the authenticity of the context must be verifiable as created and issued by an authorized node in the XtreemOS. We would therefore require flexible authenticity stamp and check functions that take the inputs of target (e.g. 'vo-context'), method (e.g. 'md5') and reference (e.g. 'key').

2. **Security requirements for attribute isolation:** In order to achieve attribute isolation, any attribute must be appended with that of the vo-context. Therefore, this has to be enforced when attributes are assigned to or registered by resources when joining a VO. These are actually two different protocols and functions:

   (a) *issue-attributes*: based on the identity, VO context and requirements of a subject (i.e. a resource user or process), a trusted *attribute-issuing* component should provide the relevant attributes required by the subject.

   (b) *check-attributes*: there needs to be a fully trusted component available on each node, acting as a reference monitor, which can inspect claimed attributes and validate that they belong to a given VO.

3. **Security requirements for object isolation:** the security mechanisms should ensure that actions performed on objects and their memory conform to a set

of isolation policies. A sandbox is therefore required that performs the following checks on operations: (i) object-create, (ii) object-write, (iii) object-read, (iv) object-copy and (v) object-delete.

4. **Security requirements for interface isolation:** the mechanisms for interface isolation are related to the read and write checks for object isolation at a finer granularity. The same protection mechanism can be used or a second level of defense can be set up that checks for finer-grained attributes of calling processes and the parameters of the call related to the VO context within which the resource runs[6].

5. **Security requirements for service/process isolation:** this adds requirements for failure handling to the above set of isolation protection mechanisms. Firstly, services and processes of one VO context should execute within an independent physical address space. Secondly, provisions should be made for *secure*[7] checkpointing and rollback during their execution.

   In terms of achieving isolation across domains, it must be initially assumed that the administrators of each domain, members and resource of a VO act trustworthily with respect to the isolation policies. Issues with maintaining trust and reputations may be beyond the technical scope of XtreemOS. Note that virtualization technologies such as VMWare and OpenVC provide different means of logically yet robustly isolating execution environments. These should be explored as implementations for proving transparency of the functions requested above.

3.4 Application Checkpointing

Using application checkpointing, one can write to stable storage an image of the state of an application and restart the application at a later time or on a different set of nodes.

Saving the state of an application to stable storage might involve different strategies as requested by the user. The most simple strategy simply writes the state to a disk accessible locally, whereas more complex constraints might send the state of the application to another node of the grid, either for storage on a remote disk or in remote memory. Moreover, disks accessible locally might be disks attached to the local node as well as remote filesystems mounted locally.

[6] recall that a physical resource may be running in multiple VO contexts

[7] note that checkpointing and rollback require possibly sensitive data from memory state to be transmitted over a network to a storage location. This must be securely done and stored i.e. encrypted

3.4.1 Functional Description

In XtreemOS, application checkpointing involves:

- *The Kernel Checkpointer*, that deals with taking the snapshot and restart for an individual process
- *The System Checkpointer*, that deals with providing checkpoint management at the application unit level, and
- *The Grid Checkpointer*, that deals with providing checkpoint management at the application (or job) level.

Since Kernel and system checkpointing is performed in XtreemOS-F layer, we will consider the description of Grid Checkpointer in this section.

In XtreemOS, the need to checkpoint a job can arise in two instances 1) Due to a scheduling decision from Application Execution Management, or 2) Due to a Fault Tolerance policy defined by the user and given in the job description.

The service responsible for checkpointing a job is the `jController` service. The `jController` includes two internal services for managing job migration (`jMigration`) and checkpointing (`jCheckpointing`). The `jCheckpointing` service applies the checkpointing strategy to all the running processes. It registers the processes with the checkpointing services on the local nodes, it provides resources to store the checkpoints, it then launches jobs in the checkpoint context and it co-ordinates the checkpoints of a job running on different nodes. It is also worth noting that checkpointing a job is optional during migration of a job and is determined by policy set by the user.

In case of a fault, a `jRecovery` service will apply the fault tolerance policy as given by the user and restart the job based on a checkpoint. Both the `jRecovery` and `jCheckpointing` interact with the System Checkpointer on Grid nodes.

3.4.2 Overlap with Security Requirements

The security requirements defined by application checkpointing can be categorised into two broad types:

1. **Requirements for secure storage of checkpoints:** When an application is checkpointed, the checkpoints can either be stored in local storage or in complex multiple Object Sharing Services (OSS) as defined in XtreemOS file system (XtreemFS). Nevertheless, the secure storage of these checkpoints is of prime importance, since recovery from failures will be instantiated from these checkpoints. In addition to secure storage, checkpoints

should be accessible easily and quickly. Checkpointing should at least store the threads, Inter-Process Communications, network communications and opened files by the job being checkpointed. Also, replicas of the checkpoint files are required to preserve availability of checkpoints. However, this can be specified by the user. Such a provision is considered for scenarios where replicas are not cost-effective or if the user does not want to have checkpoint replicas.

In essence, confidentiality, integrity and availability of checkpoint files need to be maintained.

2. **Requirements for secure context switch at restart:** When an application is restarted after failure, it inherits the security context (for e.g. policies) as defined during restart and looses the security context it had upon checkpoint. Therefore, it will abide to the security policies at the time of restart and not necessary to those that prevailed at the time it was running. For such a requirement, VO policies should be embedded within the checkpoints stored either locally or in OSS.

3.5 Federated Resource Management

3.5.1 Functional Description

From the "Description of Work" it becomes clear that the XtreemOS operating system is intended to be executed on all computers, making resources from such computers as part of virtual organizations. There will be three flavors of XtreemOS - for single PCs, for clusters and for mobile platforms. On top of XtreemOS, runs a Grid application on one or several Grid nodes. Such an application can be divided into application units, where each of these application units are executed on a single node. Federated resource management deals with LinuxSSI-XOS, the SSI (Single System Image) cluster version of XtreemOS's foundation layer. In a cluster all nodes work closely so they can be viewed as a single multiprocessor computer. A Linux SSI operating system does exactly that. It provides an illusion that a cluster is a highly robust, virtual multiprocessor computer running Linux. Thus, Linux SSI-XOS is a standard Linux kernel modified in two ways – firstly, to include the modifications for VO support, and secondly, to integrate distributed resource management services to provide a single system image.

Five main directions are identified for design and implementation of LinuxSSI:

- *Building scalable SSI mechanisms*: The goals here are common to other HPC cluster technology development. It is foreseen then during the tenure of the project, clusters will reach petaflops range. So, the XtreemOS-SSI

needs to support clusters with hundreds and even thousands of nodes. These nodes should be able to use 64-bit multicore processors. With such large number of nodes the mean time between failures will decrease dramatically and XtreemOS-SSI should be able to handle such failures. To enable fulfillment of such goals, WP2.2 is working on six core functionalities – dynamic reconfigurability (dynamic node addition and removal, survival under single node failure and clean shutdown of SSI cluster), SMP nodes support (add support for shared memory parallel nodes and multicore CPUs), 64-bit processor support, removing hard-wired parameters (increase number of containers in system), support for high-speed interconnects (add support for OpenIB infiniband stack) and functional stability (benchmarking methods for detecting issues in this area).

- *Checkpoint/Restart Mechanisms*: Refer to Section 3.4 for a complete description of this functionality.

- *Reconfiguration*: In LinuxSSI, (i) new nodes are added, (ii) nodes are removed from the cluster and (ii) nodes fail. LinuxSSI has to adapt to these reconfigurations to ensure that they will not compromise application execution on a cluster. While handling of node addition and removal are characterized as basic features, handling node failures is thought of as advanced feature for LinuxSSI. *Hotplug* is the component in LinuxSSI that deals with reconfigurations. It informs other components of the reconfiguratons and it coordinates the actions taken by the other components to adapt to the reconfiguration. WP2.2 have built the *Hotplug* component and are testing it against basic and advanced features.

- *High Performance Disk I/O*: The most common approach in cluster technology is to use dedicated nodes to provide distributed/parallel file systems. Thus, the cluster is divided into compute nodes and I/O nodes. In XtreemOS, however, our view is that all nodes could potentially provide both CPU and storage resources.

- *Customizable Scheduler*: Each resource owner strives for high utilization and optimal resource usage. A predictable system keeps the user satisfaction high. User satisfaction can be achieved by resistance to failures. For developing an adaptable scheduler XtreemOS scheduling services consist of three levels. At XtreemOS-G level the `jScheduler` service finds resources through a discovery process and assigns them tasks. It is also responsible for job optimization based on various optimization criteria. Thereafter, a Service Level Agreement (SLA) is negotiated with `rAllocation`

service. The cluster level optimization goals are optimal resource utilization. This can be achieved by having reduced overhead on the resources, which results in better efficiency.

The architecture is presented in Figure 3. The `rAllocation` and `jScheduler` are Grid services whereas `LBScheduler` is a site level service. rAllocation service negotiates use of resources, and puts jobs in LBScheduler queue. The Load Balancing Scheduler- LBScheduler, performs load balancing in order to level the load on a cluster. LBScheduler is executed on each site.

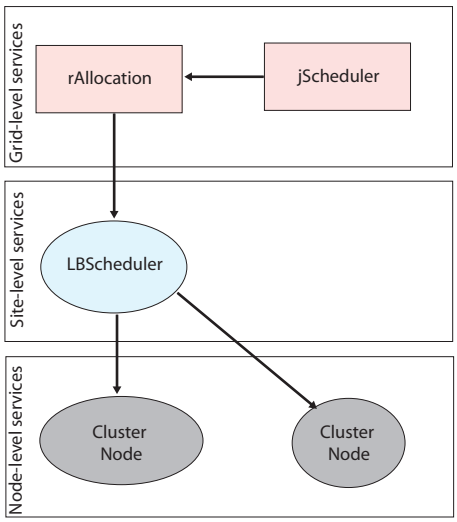

Figure 3: Scheduling Architecture

- *Virtual Organization Support*: For integration of LinuxSSI with the XtreemOS Grid services there is a need to add support for VO management services and infrastructures as defined in WP2.1. The first step is the addition of Kernel Key Retention Service (KRS) in Kerrighed. KRS is needed for transporting and attaching Grid certificates and proxies to processes. On single Grid nodes, there is mapping between the Global User ID and VO ID to local UID and GID. The local UID and GID is also used by XtreemFS for defining ownership and permissions on files. LinuxSSI will let a SSI cluster to appear in the Grid as a big SMP machine. Grid-related services which usually run on each node will run on one instance for LinuxSSI cluster. This is important for WP2.1 which is designing the mapping between local and global IDs, as the entire LinuxSSI will have a single local UID/GID.

3.5.2 Overlap with Security Requirements

It is to be noted that most of the functionalities provided by federated resource management mechanisms are at kernel level or more so for managing nodes within a cluster. In lieu of this, many of the security requirements for federated resource management concur with security requirements for highly available and scalable nodes, since both of them deal with optimal node management and maximum resource utilization. Security issues in this section deals with robustness against failures, quality of service and secure communication. Many of these are already covered in Section 3.2. However, following security requirements can be derived from the above functional description:

- *Specification of node Service Quality:* It must be possible to specify the service qualities (e.g. maximum network delay, availability of resources) for certain applications. Nodes that do not fulfill such requirements should not be selected for application execution.

- *Node properties in federations:* It must be possible to specify some required properties of federation nodes. For example, node architecture, installed web services, node libraries etc. can be specified by the user intending to submit a job. In addition, it must be possible to mount certain constraints on federation nodes. The XtreemOS API must provide mechanisms to specify such constraints.

- *Checkpointing and restart:* It must be possible to detect failures automatically, checkpoint and restart nodes. The policies governing such restarts should be applied once the node is restarted.

- *Robustness:*It must be possible to change the number of nodes that the application uses during runtime. If the number of available federation nodes changes, XtreemOS must notify the running applications. The application then decides whether it can adapt to the change. If the application can adapt to the change, it is its responsibility to rearrange any variables and computations going on. If an application cannot adapt on-the-fly to fewer nodes being available, perhaps it can be checkpointed and restarted on fewer nodes. The notification mechanism can be decided on later. It must also be possible that the running application requests a change of the number of federation nodes. A running application can request additional nodes to start processes. These additional resources have to be provided by XtreemOS (if nodes are available). Furthermore, a running application may release certain resources after terminating calculations on these nodes. These nodes are then available for the execution of other applications. XtreemOS must

be able to dynamically consider these released nodes in resource management and to provide them to other applications.

3.6 Availability and Scalability of Grid Services

This document is intended to summarize security requirements from the perspective of WP3.2. It shall enhance the information that was already stated in Deliverable D3.5.2. [3, section 7].

3.6.1 Functional Description

Before understanding the functionality of highly available services, it is important to understand the deployment of applications in XtreemOS. This section first describes the general concepts of applications/services running on the XtreemOS grid and then goes into details regarding the availability aspects.

At the core of the grid, WP3.2 provides a lightweight epidemic algorithm that runs on all XtreemOS nodes. This algorithm serves two purposes. Firstly, it establishes a robust overlay covering all nodes that are part of the XtreemOS grid. Secondly, it groups the nodes according to their parameters enabling easy searching of nodes with specific properties.

For deploying an application, the deployer/administrator has to specify the number of nodes she requires for the application. Furthermore, it is possible to provide additional static parameters that describe the attributes nodes must have in order to run the application, in addition, it is also possible to specify different attributes for different nodes. These criteria are passed to the Grid (as an input to the routing functionality of the epidemic algorithm) which, in turn, finds a set of nodes that fulfill the requirements. The number of nodes is higher than the one required, if possible. Application Execution Management is responsible for selecting the best nodes out of the whole set (considering dynamic parameters e.g. load, throughput etc.). The nodes finally selected are grouped by the system and provided with an overlay allowing the communication of any node with any other node.

An entry point to that group of nodes is returned to the administrator as a result of the search request. Now the deployer is able to deploy application code to the nodes of her application group. Furthermore, it is possible for her to build up other, application specific overlays such as a Chord ring or an MPI on top of the first, application-wide overlay. This process is supported by the WP3.2 toolbox.

Deliverable D3.2.1 [1] describes a set of independent abstractions that can be used by application developers to enable high-availability for their applications. Implementations of these abstractions are part of the WP3.2 toolbox. Application developers can plug-in tools into their applications.

During application bootstrapping the system groups all nodes, an application/job runs on, into a logical entity. Depending on the requirements the applications has, it may use different tools out of the WP3.2 toolbox. Consequently, there exist at least two kinds of overlays. A grid wide overlay and an application wide overlay. Additionally, every application may instantiate other overlays on top of the previous ones.

An application with a server-like interaction pattern may benefit from the "distributed server" abstraction. That is all nodes the application runs on are accessible with a single IP address. To enable fault-tolerance this approach does not use a proxy pattern like many other systems, but relies on the mobile extension to the IPv6 instead. Thus, the reliability of the service depends on the reliability of the network infrastructure. However, this is not a drawback, as a working network is mandatory for the service to be reachable.

Applications with critical parts may want to use the "virtual node" abstraction that allows increasing the fault-tolerance of these parts by using replication techniques that are known from object replication systems. In general it is assumed that the critical parts follow a client-server interaction scheme, that is, *clients* send requests that are processed by the *servers*, and are answered with a reply message. Internally, the requests are forwarded to all replicas, processed by all of them, and one or several replies are returned to the client depending on the error model. It is assumed, that clients accessing a virtual node are part of the same application as the virtual node i.e. it is not planned to support external access to virtual nodes. If such is required, we plan to use a setup of both distributed servers and virtual nodes.

For applications that require events to be passed from one of their nodes to another one, WP3.2 offers a publish-subscribe system. In such a system, parts of an application can register to be informed for certain events that are published by either other parts of the application, the runtime infrastructure (such as the scheduler or job monitors), the grid infrastructure (joining of new nodes, failure of existing nodes) or application external entities (other grid applications).

3.6.2 Overlap with Security Requirements

Following the above functional description multiple security issues are raised. Most of them are already covered in D3.5.2 [3] and will only be summarized here in short. Just as in the deliverable we will distinguish three security layers: grid level, application level, and host level.

At the grid level, the epidemic algorithm chooses hosts for an application according to properties hosts have and which application deployers specify during the deployment process. It is required that hosts cannot fake the properties they offer as this would enable DoS attacks.

At the application layer, it is required that external nodes can communicate with an application by predefined communication structures. Furthermore, when application nodes open a connection to an external party, this communication must appear to come from the official application interface and thus hide the node's identity. This is necessary not only to provide anonymity, but also to tolerate node failures and the movement of parts of the application to other nodes for load balancing purposes.

Similarly, arbitrary communication with nodes participating in a virtual node shall not be possible, as the request has to be forwarded to all nodes, i.e., logical 1:1 connections must be mapped to a physical 1:N connection. Accordingly, connections from replicas to other nodes must be coordinated to avoid multiple physical requests for one logical request. All these demands may be handled in the application itself or in the replication framework. However, as they are mandatory for application integrity, it seems reasonable to put the solution in OS level. Furthermore, the mechanisms to handle application communication and virtual node communication might be quite similar.

At the host level WP3.2 requires that communication integrity and confidentiality be granted.

Additional security requirements Epidemic algorithms have proved to be very resistant to non-benign failures, as they can tolerate contemporary failure of a high portion of nodes. However, they are also known to be very vulnerable to non-benign failures, as all nodes are considered equal. Since the basic algorithm of the XtreemOS grid system will be some sort of epidemic algorithm and a grid system will definitely be subject to attacks, securing the algorithm without introducing too much overhead will have to be a major concern. A first step in this direction is to find a way that allows the algorithm to distinguish between nodes that are allowed to participate in the grid, since they belong to a valid virtual organization. In a second step the challenges of an open grid system, which allows arbitrary nodes to join, is to be faced.

For virtual nodes the requirements for secure communication can be concretized. Not only communication between clients and virtual node members must be secured, but also communication between the members themselves. Especially, it must be ensured that former members that were excluded from the group for whatever reason can not follow group messages after their exclusion.

3.7 Application Execution Management

In XtreemOS, application execution is managed by Application Execution Management (AEM) services [2]. Depending on the amount of interactivity required

during application execution, applications can be classified into two categories: simple and complex applications. Simple applications do not require further interactions with users once they start execution, whilst complex applications need to support interactivity between users and runtime applications. Complex applications require session management on top of application execution management of simple applications. AEM provides a general architecture to accommodate the functional requirements of both types of applications.

3.7.1 Functional Description

AEM services can be logically grouped into three types: Job Management Services (JMSs), Resource Management Services (RMSs), Execution Management Services (EMSs) and AEM global services.

JMSs cover all the job related tasks, including job scheduling, job controlling, job monitoring, event handling, and execution management. JMSs are mostly operated on an individual job basis, that is, these services do not have a global view of the system. Once an application starts executing, JMSs become intermediates between users and runtime applications by allowing the users to monitor the runtime application information and control application execution. When a job requires multiple resources, JMSs also cover dynamic application execution management in a distributed manner.

RMSs cover all the resources related tasks, including resource monitoring, negotiation (i.e. checking with local policies and reservation), and allocation. RMSs contact node management services [8] to obtain a list of candidate nodes and perform resource selection and matching based on the static resource information (e.g. number of CPUs). Once resources are selected, RMSs further reserve the resources and perform resource allocation so that applications can execute with a guaranteed level of resource provision. In the presence of addition and release of resources, RMSs also cover the re-negotiation and re-allocation of resources to ensure the agreed level of guarantee is held.

On nodes, a submitted job will be managed by EMSs, which are responsible for executing jobs on resources.

There are two AEM global services, namely `JobDirectory` and `jResourceMatching`. By global, we mean these services are operated within the context of a VO. `JobDirectory` maintains the reference to running jobs whilst `jResourceMatching` provides resource selection and matching. Once an application starts execution, the control of the job is entirely handed over to another AEM service, `jController`. `JobDirectory` is the only way that users can get access to `jController`.

[8] Node management services are provided by WP3.2.

3.7.2 Overlap with Security Requirements

AEM services are operated in the context of a VO with the support of VO management services (aka. VOM). VOM is a logical representation of a collection of VO infrastructure services, which include, but not limited to, membership, identity, attribute, and policy management services. We will start by the general security requirements and then move on to the specific security requirements.

General security requirements for AEM services: There are two fundamental security requirements for all AEM services: 1) only registered users are allowed to access AEM services. A registered user means that the user is registered with a VO membership service; and 2) fine-grained and scalable access control should be in place to ensure AEM services are available to authorized users and they can perform efficiently even in the presence of a large number (e.g. thousands) of VO users. There are two levels of controls: one is the control to AEM services themselves (e.g. `jController` and `JobDirectory`), and the other is the control to the information provided by AEM services (e.g. accounting or monitoring information).

Security requirements for job submission: The security services should ensure that: 1) only authenticated VO users are allowed to submit jobs so that the use of VO resources is accountable; 2) the selection of VO policies should be *context-aware*. That is, VO policies should be associated with the context of job submission. Contextual information may include users attributes (e.g. role, location, origin organization); and 3) cross-VO job submission should be possible. This last requirement is probably the most complicated because it requires federation of security services (e.g. identity and policy) from multiple VOs across multiple administrative domains.

Security requirements for resource matching: One of the most important aspects of AEM resource matching is its incorporation of VO policies for resource selection. The security mechanisms for resource matching should ensure that resources selected for application execution conform to VO policies throughout the entire lifespan of the execution. The major challenge here is to ensure such conformance in the presence of dynamic resources or application execution across multiple resources.

Security requirements for job accessing: The security services for accessing remote jobs are mainly for applications that require interactions during application execution. They should ensure that: 1) users are attached to the correct sessions

of application execution; and 2) users are given the same level of privileges in all the sessions; and 3) once a job is complete, users' credentials should be safely revoked and their privileges be removed across the entire VO [9].

Security requirements for interactive sessions: The security services for accessing remote jobs are mainly for applications that require interactions during application execution. They should ensure that: 1) users are attached to the correct sessions of application execution; and 2) users are given the same level of privileges in all the sessions.

Security requirements for job execution: A job executing on resource providers should be granted appropriate access rights to access the resources it needs. These resources can locate locally on one or across more than one resource providers, or remotely (e.g. under the governance of GFS). The access rights should be guaranteed consistently even in the presence of dynamic resource changes or engaging of multiple concurrent resources through the entire lifespan of job execution. Once a job is complete, users' credentials should be safely revoked and their privileges be removed across the entire VO [10].

Security requirements for accounting: In the context of AEM, the security services for job resource accounting should ensure that resource usage is recorded with accuracy so that resource consumption is accountable. By accountability, we mean that the accounting information should associate with individual *VO* users [11]. The recorded information should also be made available to authorized VO entities so that they can run follow-on services (e.g. reputation or billing) after a job is finished.

3.8 Data Management - XtreemFS

In XtreemOS, data management is primarily a feature provided by XtreemFS, a distributed file system designed for Grid environments. As a result of this extension of traditional filesystem concepts, there are new security requirements that

[9] Upon job completion, users should still be able to access to the results with appropriate rights and should even possibly grant these rights to other users. However, this is a grey area which may involve the interactions among AEM, XtreemFS and security.

[10] Upon job completion, users should still be able to access to the results with appropriate rights and should even possibly grant these rights to other users. However, this is a grey area which may involve the interactions among AEM, GFS and security. Hence, we will not discuss the security requirements for job result accessing here.

[11] In the context of XtreemOS, accounting should be based on VO users rather than purely users with Distinguished Name (DN)

arise. This section provides an overview of XtreemFS and then a description of the security requirements that have been considered in the security architecture.

3.8.1 Functional Description

A reliable file system and effective data management are integral parts of any operating system. XtreemFS supports data management in XtreemOS by facilitating the following characteristics:

- **Object-based**: separates pure content from meta data. Object Storage Devices (OSDs) store objects and provide read/write interfaces to them, while the Metadata and Replica Catalogue (MRC) maintain the meta-data surrounding objects including replica locations. Both of these data repositories are targets for attack, such that a reference monitor mechanism that protects both object and meta-data is required. Compromise of meta-data as well as the links between meta-data and objects destroys the integrity of the system.
- **Fault tolerant**: there is an attempt to reduce the effect of host failures such that these can be automatically handled and not propagated. The security challenge here is maintaining the fault tolerance mechanisms and ensuring that it is not used as a means of attacking the XFS.
- **Scalability-objective**: i.e. increased performance demands can be matched with proportionally adding more machines - should run on standard hardware with the assumption that the data loads and requests will exceed the capacity of a single machine.
- **Organization, administrator and user autonomy**: organisations are independent, i.e. can join and leave at will, and they can work without connectivity (referred to as federation). Federations allows institutions and users to use their local system with their local data when disconnected from the Grid, while having a global data view when connected.
- **Loose coupling and asynchronous interaction**: it is loosely coupled and asynchronous, so that failures and temporary performance issues don't spread over the whole system.

There are other important assumptions of XtreemFS made with respect to its integration in a UNIX/Linux-like operating environment. These include compliance with POSIX libraries and library calls, extended meta-data, portable implementation on commodity PC hardware (i.e. no assumptions of specialized, large scale memory or processors), flexible data hiding and storage, group mechanisms and legacy support for UNIX-based applications.

3.8.2 Overlap with Security Requirements

As distributed filesystem consist of up to several thousand nodes, management becomes a greater issue. Distributed systems can become very complex, yet compliance with a specification is important. Due to the distributed nature, if something fails it is hard to figure out the responsibility and who caused it. Security mechanisms therefore seek to reduce possibility of failure by preventing against different types of malicious attackers.

Assumptions about attackers It is assumed that a cluster of OSDs of the XtreemOS filesystem is present and stores data. All participating OSDs behave as specified in respect to the OSD service and software it depends on. The users as well as the administrators do not attempt to interfere the operation. The participating OSD nodes share the same codebase or implement the same functionality in a different way. In contrast, the OSD of an attacker does not adhere to the specification and is thereby not trustworthy. It can behave in a different/unspecified way than the other OSDs in the cluster. It is assumed that an attacker can benefit from integrating a non-conformant OSD into the cluster.

Internal Attackers This section focuses on an attacker joining the set of OSDs and when successfully joined, to perform an internal attack on the operation of the cluster of OSDs, the data integrity and confidentiality. An internal attacker is much more powerful than an external one since it can perform all attacks of an external attacker locally too. In addition all the network attacks can be performed in more sophisticated way since an internal attacker may know encryption keys and can decipher the payload of the network traffic in case it is encrypted.

Uncompliant Object Storage Device This section focuses on an attacker with the aim to interfere the execution of an OSD. The attacker can also be interested to get confidential data stored on his OSD or render data integrity. The aim of the attacker is to get his malicious OSDs joined into a XtreemOS storage cluster.

Aims of an Attacker The attacker can have several aims and can perform different kinds of attacks to the OSD cluster he managed to join it. The aims are:

- *Perform a denial of service attack*: The attacker may have the goal to interfere the quality of service of the cluster or federation. Once he managed to join the cluster he is able to control the communication. He can act in a lots of different ways, from interrupting the communication and thereby letting other nodes wait to send malicious packets as external attackers can. If the

communication is encrypted the attacker is able to decipher the communication and read the information in plain. To endure his actions, the attacker may apply different strategies to remain undetected.

- *Render the integrity of data useless*: The attacker may have the goal to have access to sensitive data and modify it in a way he prefers. Therefore the attacker disobeys the policies for access control by having access to data without the mediating reference monitor. He may also be able to render the set of policies to achieve this. Especially logging and accounting data is a major goal for attacks to the integrity of the data. This also includes data parts influencing this data such as time information.

- *Accessing confidential data*: The attacker may have the goal to have access to confidential data such as business information, personal profiles, license information, etc. available in the XtreemOS OSD cluster.

Point in time of the attack The attacker can be seen as an external attacker when he tries to join the OSD cluster. Therefore some might argue that this is an external attack. However when the attacker achieves his goals he operates from an internal standpoint.

Security Functions of the MRC The MRC is a major component in the security of XtreemFS. The core interface between an MRC and the OSDs is the file access capability. When a client wants to access a file, it requests access from the metadata server, which hands out a capability to the client. The client presents that capability to a storage server as both an authentication and authorisation means and gets granted the proper access. File access capabilities are secured by encrypting them with a key that is shared between the MRC and an OSDs. They contain: the object that the client is allowed to access, the operations that the client is allowed to perform, some means for revocation (expiration time or object id versions) and some means of client authentication (its IP address for example).

4 Specification of Security Services

4.1 Introduction

One of the fundamental security requirements of XtreemOS is to provide a Grid operating system that has a security infrastructure to support single sign-on (SSO). In an XtreemOS Grid environment, a user should be granted appropriate access rights and privileges to access Grid-wide resources in a consistent and efficient manner. In this context, the consistency should be ensured throughout the lifetime of user sessions and the efficiency should be achieved regardless of the scale of the system and the geographical locations of users and resources.

This is, however, a non-trivial research as well as engineering challenge because the *scale* (e.g. up to thousands of concurrent users and resource nodes) that XtreemOS sets out to target. It is also inherently challenging because XtreemOS also allows the concurrent presence of multiple VOs, each of which may span cross multiple organizational boundaries. And commonly, each organization has their own *established* security infrastructure and practice.

However, the authentication system adopted by Grid middleware is typically independent from those in the existing security infrastructure used by operating systems. In the present design, the user needs to *explicitly* acquire and manage two *independent* sets of security credentials:

- Local credential for authenticating to the operating system [12] where the user initiates a Grid request; and
- Grid credential for authenticating to the Grid.

Typically, the former one is based on password and the latter one is based on PKI. Effectively, this design means that users need to cope with two completely different types of authentication systems. This can be inconvenient for users.

On the one hand, when a user is part of an organizational computer network, a local credential is typically a Kerberos username/password. Or, in a more general case, a local credential is a username/password pair issued by his own machine. On the other hand, Grid credential is a public/private key pair issued by a Certification Authority (CA). Managing two independent sets of credentials can be daunting for majority of non-technical users who often have limited experience of PKI.

Hence, this leads to the demand of a *scalable* SSO solution that should maximize the possibility of integrating with existing organizational security infrastructures and minimize the demand on users coping with two different types of authentication systems.

[12] This OS can be either networked or stand-alone.

Since XtreemOS is a Grid OS, we have an distinct advantage of integrating such a solution well into system-level services. The motivation is that this approach can lead to a better integrated security solution that demand minimum interactions with users, and be able to provide transparent access to resources.

The remaining of this section is divided into two parts. The first part describes what sets our solution apart from the existing ones offered by Grid middleware. The second gives a general overview of the security services offered by this first edition of specification of security services of XtreemOS.

4.1.1 What is new?

In XtreemOS, we take an integrated approach towards credential management by better integrating the Grid level authentication with system level authentication. To users, the end result will be an integrated Grid operating system that provides Single Sign On (SSO) access to Grid resources. The goal of this approach is to realize such a vision that

Once a user logs on to a Linux terminal, this user should be able to easily access Grid resources by invoking XtreemOS commands with minimum degree of awareness of the Grid.

The aim of our design is to shift the complexity of secure Grid resource access from users to XtreemOS. However, the typical security requirements (e.g. mutual authentication) for accessing Grid resources should still be satisfied.

In line with the fundamental approach taken by XtreemOS, our security solution is VO-centric. The essence of our approach is to cleanly separate user management from resource management in a VO through a set of carefully designed infrastructural security services within the scope of VO Management (VOM). The challenge is to achieve our goal without compromising scalability and efficiency. Because of the separation, the changes (i.e. addition or removal) of VO users will not impose significant impact on resource management in a VO, and, vice versa. In the end, XtreemOS should allow independent management of users and resources from multiple administrative domains.

Each VOM has an associated role, called a **VO manager**, who is in charge of running all the VOM services and is issued a public key certificate by a recognized CA. The certificate permits the VO manager to issue VO credentials to users for them to access Grid resources.

4.1.2 An Overview of Security Services

In XtreemOS, a VO user is uniquely identified by his global ID and has a selection of VO attributes. The user often affiliates with an real organization (we call it home organization), who runs a set of VOM services. A user can concurrently

possess multiple global identities (global IDs) within a VO. In the current design of VOM, global IDs are managed by VOM on a VO basis.

The approach we have adopted is top-down, starting from user and system spaces down to the kernel space of the Linux OS. In the user space, there are five fundamental security services managed by a VO manager under the VOM umbrella on a *per-VO basis* [13]:

- **Credential Distribution Authority (CDA)** issues users with VO security credential for accessing Grid-wide services and resources. In XtreemOS, we use X.509 v.3 certificate [6] as our credential format and refer such credential as XOS-Cert. CDA holds the public/private key pair of the VO manager so that they can issue signed XOS-Certs. Before obtaining an XOS-Cert, a user needs to prove its VO membership.

- **Identity service** generates and manages globally unique VO IDs and user IDs. These IDs have to be globally unique across the entire Grid regardless of the number of VOs, users, and resources. In XtreemOS, we assume each VO has a globally unique ID, called VO ID [14] The global uniqueness of user IDs is managed by VOM. Their authenticity is guaranteed by the signature of the VO manager in a XOS-Cert. The underlying assumption is that resource nodes trust the VOM manager and have pre-installed the root CA certificate of the VO manager. Based on a XOS-Cert, nodes can verify users' authenticity.

  Comparing with the scale of user population, the number of VOs is normally negligible. It is much easier and simpler to ensure the global uniqueness of VO IDs than that of user IDs. Hence, one way is to generate globally unique user IDs is to concatenate a VO ID with a VO-unique user ID. Herein, by global ID, unless specifically spelt out, we mean global user ID. In order not to be confused with the DN field in a typical X.509 certificate, the user

[13] It is very important to note that these are logical separation of security services. When these services are implemented, they may be implemented as a single component or a set of separated components. When they are implemented separately, one has to consider the mutual authentication between each pair of these components. For the initial implementation, these services should be implemented as one single component to reduce the operation complexity. Due to the high availability and scalability requirements of such a VO component, we plan to investigate the possibilities of using the highly available and scalable services currently being developed in WP3.2 in the near future.

[14] There are many ways to ensure the global uniqueness of an identity, be it numerical or alphanumeric. For some, please see [4]. Some of candidate approaches rely on the availability of a global (across all VOs) directory service to check the uniqueness of IDs. Others don't. For the moment, we tend to favor on those approaches who can provide uniqueness assurance without online interactions, although this may mean some compromise on the quality of the IDs.

ID in a XOS-Cert embedded as an attribute of the certificate [15].

- **Attribute service** provides users with VO attributes. In XtreemOS, these attributes are used in a variety of ways, including:
  - to allow nodes to perform access control to their resources; and
  - to allow nodes to map global IDs to system UIDs/GIDs. In XtreemOS, there are proposals from WP2.1, to use attributes in XOS-Cert, such as PrimaryGroup and SecondaryGroup, to map global IDs to system UIDs/GIDs to enforce some level of isolation between Grid users.

- **VO Membership Service (X-VOMS)** provides VO membership checking service that would allows systems to validate the VO membership of a user who initiates a Grid request from a Linux terminal. X-VOMS connects to a database which stores VO information, such as identity and attributes, about a user.

- **VO Policy Service (VoPS)** provides policy related services, such as policy information and decision points to VOM so that VO level access control can not only be enforced at nodes but also by VOM. One benefit of integrating policy decisions in VOM is to accommodate the flexibility of incorporating VO policies in job scheduling and resource negotiation processes.

In the system space, we provide interfaces to these security services to system level daemon services. This is to allow seamless integration of our services into OS level services that will be run as daemon. A typical example of such OS level services is Application Execution Management (AEM) services, being developed in WP3.3. In the kernel space, PAM modules are used to transparently interpret XOS-Certs for authentication and authorization.

4.2 Mutual Authentication

To be self-contained, this section starts from describing the principle of authentication, mutual authentication, and Single Sign On (SSO). The elaboration on this topics sets the foundation for the description of mutual authentication between a user and a resource node in XtreemOS. We will then explain the trust assumptions of the three authentication methods: Kerberos, PKI, and password, that XtreemOS

[15] Putting user ID in the DN field of a XOS-Cert is possible. This can be achieved by requiring users to put their Grid user ID in the certificate they send to the VO manager. If the ID is verified, the VO manager signs it and sends back an endorsed XOS-Cert. Otherwise, the XOS-Cert request is refused.

security architecture can be based upon. In the following, we refer these authentication methods as Kerberos, PKI and password, respectively. This section ends by describing how mutual authentication works using each type of authentication method.

4.2.1 Authentication Problems

Authentication is concerned with proving the identity of an entity in a network. In public key cryptography, the knowledge of a private key equals proof of identity[16]. In symmetric key cryptography, if A shares a secret with B, when A talks to another entity in a network, if the other entity can be convinced A that it has the knowledge of the secret, A is convinced that the other party is B[17].

Mutual authentication is the process of mutually authenticating two entities in a network. Each entity needs to be certain about the identity of the other. In XtreemOS, because a user's global ID is described in a XOS-Cert signed by VOM, mutual authentication becomes a problem of verifying the authenticity of the ID that a user claims to have and of verifying the ID of the other end (typically, a resource node).

Traditionally, Single Sign On (SSO) means that a user only needs to authenticate once and gain access to resource nodes owned by different organizations.

In XtreemOS, SSO has the following meaning. SSO should provide users with a seamless access to remote resources with minimized amount of awareness of the Grid environment. From a security point of view, that means that secure access to resources should demand a minimized amount of operations from users. To them, accessing remote resources should be the same as accessing networked resources in their own organization. SSO should enable direct access to Grid resources once a user has authenticated themselves to organizational network through an organization's **existing security infrastructure**, if they exist.

4.2.2 The Model

This section describes a generic authentication model for XtreemOS. This model consists of two parts: a home organization, from which users originally register their VO membership, and a foreign organization, where mutual authentication (between users and resources) takes place. This model is designed to be flexible to accommodate a wide range of authentication methods to allow users to authenticate themselves to a VO.

[16]This is assuming the holder does *not* share this private key with anybody else. The public key, as its name suggests, is a piece of public knowledge, i.e. anybody can know about it.

[17]This is assuming neither A nor B shares the secret with anybody else.

Figure 4 shows the model and a proposed protocol to facilitate mutual authentication between a client, who runs a service on behalf of a user, and a node, who runs a Grid service, in XtreemOS. The authentication within a home organization is handled by Home Authentication Authority (HAA). In the diagram, X-VOMS and CDA are presented as one single component (refer to Section 4.1.2 for more information).

The components on the left hand side of the vertical dash line are with the home organization. The node on the right hand side is with the foreign organization. The client represents a service running on a XtreemOS Linux terminal. This service understands XtreemOS commands (e.g. `XSub`) and is configured to interact with other XtreemOS components, such as X-VOMS/CDA.

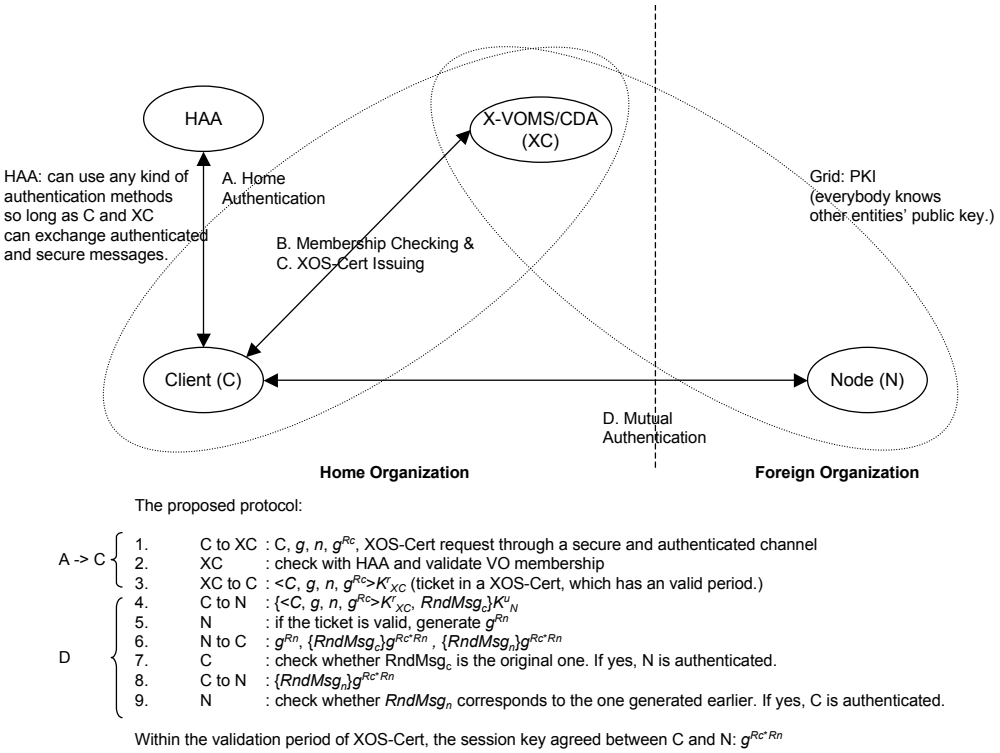

Figure 4: Mutual Authentication in XtreemOS: Model and a Proposed Protocol

The mutual authentication process consists of four sequential steps:

- **Step A (Home authentication)**: authenticate with the home organization, this is handled by HAA;
- **Step B (VO membership checking)**: obtain confirmation from X-VOMS that he is a registered VO user;

- **Step C (Requesting XOS-Cert)**: obtain a XOS-Cert from CDA, if the VO membership is confirmed; and
- **Step D (Mutual Authentication)**: perform mutual authentication between the user and an entity in the foreign organization.

It is important to note that not all these steps have to be performed every time a Grid request is issued. Steps A to C are performed the first time Grid requests are issued. An XOS-Cert can be reused within its validation period and be stored in the a temporary directory belonging to a user. So long as it is still valid, steps A to C can be skipped.

In this model, HAA is designed to serve two purposes:

- To separate authentication in a home organization from that in a foreign organization so that VO user management can independently evolve from the actual Grid environment within which resources are hosted; and
- To accommodate the need of integrating with existing authentication infrastructure (be it Kerberos, PKI, or any other kinds of authentication methods) that an organization may already operate.

Note that the definition of a home organization is the organization that a user originally registers his VO membership with. A special case of this setting is that a user does not have any home organization to authenticate with. The flexible nature of this model gives XtreemOS the opportunity to cope with such a situation as well. In this case, X-VOMS provides a password based authentication system to allow users to register with.

Typically, the authentication methods employed by the home and foreign organizations are different. As shown in the figure, the Grid side, including the X-VOMS/CDA(XC), are under the governance of PKI. That is, the XC and every node have issued a public key certificate. This is because majority of existing Grid infrastructure is based on PKI.

The authentication system at the home side is not restricted. That is, we make no assumption on which type of authentication method that the home organization adopts. The challenge here is to bridge the trust between the home organization and the foreign organization.

4.2.3 Trust Assumptions

This section describes the trust assumptions on which mutual authentication in XtreemOS is based upon.

It is important to note that different authentication methods built upon different trust assumptions. In this specification, XtreemOS aims to provide the support of three authentication methods that can be used in a home organization, they are:

1. Kerberos, a popular network based authentication system already employed by many organizations;
2. PKI, a common authentication system used by Grid middleware; and
3. A password based authentication system provided by X-VOMS.

In methods 1 and 3 above, users are identified by their registered username. In method 2, users are identified by their DN.

In our model, the VO manager, governing both X-VOMS and CDA, acts as a trusted arbitrator to bridge the trust between two worlds: the home and the foreign organizations. *To facilitate such bridging, resources in the Grid environment need to trust the XOS-Certs issued by the VO manager*. In practice, the VO manager's root CA certificate and its public key certificate need to be installed on resource nodes so that nodes can verify the authenticity of XOS-Certs. Similarly, users should be able to verify the authenticity of nodes. However, this is the mutual authentication after users obtain their XOS-Cert. The trust assumptions on the authentication in the home organization is different.

In a typical setting of XtreemOS, the user, HAA, X-VOMS, and CDA are associated with a home organization. Let us say, the home organization has its own security infrastructure, for example, Kerberos. In this case, X-VOMS needs to be configured as a Kerberized service that understands Kerberos tickets so that it is capable of verifying the authenticity of users without needing to know the password of users. The presentation of a valid Kerberos ticket is a proof of identity. Here, the example Kerberos server, i.e. krealm.ac.uk, is the HAA. Because Kerberos is a network service, it has to be online to serve as a HAA.

In the second case, where the home organization uses public key certificate to verify users, X-VOMS needs to install the root CA certificate of the user and other CA certificates in the certificate chain. To X-VOMS, the proof of knowledge of the user's private key is equivalent to the proof of identity. Here, there is no physical and separate server that users need to authenticate with. However, indirectly, the CA who issues certificates to users and X-VOMS is viewed as a HAA.

In the third case, a user is authenticated by X-VOMS because he shares a secret (i.e. password) with the X-VOMS. To X-VOMS, the proof of knowledge of the password is equal to the proof of identity. Similar to the PKI case, there is no physical and separate server that users need to authenticate with. here, X-VOMS acts as a HAA. Hence, in the third case, HAA is simply a logical entity that does not physically exist.

4.2.4 The Protocol

This section describes a protocol for facilitating mutual authentication in XtreemOS. This protocol is based on the classic Diffie-Hellman key agreement protocol[5].

At the end of our protocol, a shared secret key is agreed between communicating parties who (a) previously are unknown to each other; and (b) are under two different administrative domains, who may or may not use the same kind of authentication methods.

The following is a list of notations being used in the protocol.

- C: the client service running on behalf of a user
- N: a resource node in the Grid[18]
- XC: the component implementing the services X-VOMS and CDA
- g, n: the Diffie-Hellman (DH) parameters[19]
- R_c: a random number generated by C
- R_n: a random number generated by N
- K_Y^r: the private key of Y
- K_Y^u: the public key of Y
- <M>K_Y^u: a message M signed by Y's public key
- $\{M\}K_Y^r$: a message M encrypted by Y's private key
- $RndMsg_y$: a random message generated by Y

The steps described below are identical to those shown in Figure 4. Here, we provide detailed elaboration of the mutual authentication protocol.

1. $C \rightarrow XC$: C, g, n, g^{R_c}, XOS-Cert request through a secure and authenticated channel
2. XC: check with HAA and validate VO membership
3. $XC \rightarrow C$: <C, g, n, g^{R_c}> K_{XC}^r (ticket in a XOS-Cert, which has an valid period.)
4. $C \rightarrow N$: { <C, g, n, g^{R_c}>K_{XC}^r, $RndMsg_c$ }K_N^u
5. N: if the ticket is valid, generate g^{R_n}.

[18]Examples of a node are a normal resource node and a MRC or OSD node in XtreemFS. In fact, any service that has a public key certificate can be such a node.

[19]g is the DH exponent and n is the DH field.

6. $N \rightarrow C$: g^{R_n}, $\{RndMsg_c\}g^{R_c * R_n}$, $\{RndMsg_n\}g^{R_c * R_n}$

7. C: check whether $RndMsg_c$ is the original one. If yes, N is authenticated.

8. $C \rightarrow N$: $\{RndMsg_n\}g^{R_c * R_n}$

9. N: check whether $RndMsg_n$ corresponds to the one generated earlier. If yes, C is authenticated.

Steps 1 to 3 are done within the home organization. As stated earlier, HAA can validate the authenticity of users. Hence, this protocol relies on the authentication methods offered in the home organization to establish the secure and authenticated channel between C and XC. In the case of HAA using Kerberos, XC is configured as a Kerberos service and, thus, the channel is established based on the shared secret (given by Kerberos) between XC and the user. In the case of HAA using PKI, the traffic towards XC will be encrypted using XC's public key; whilst the traffic towards C will be encrypted using C's public key. Finally, in the case of HAA using X-VOMS password based authentication, this key is the shared password hash between the user and XC.

Note that R_c is kept *privately* by the user to perform steps 7 & 8. According to the discrete logarithm problem based on which the Diffie-Hellman protocol is derived, it is computationally hard to obtain R_c from g^{R_c}. This applies to any other discrete logarithm computation involved in this protocol. Similarly, R_n is kept *privately* by N to perform steps 6 & 8.

The ticket <C, g, n, g^{R_c}>K^r_{XC} is embedded in a time-bounded XOS-Cert. It can be reused again and again by the user until it expires.

Step 4 onwards shows how the ticket can be used by C to authenticate with any other entities, represented by a Node (N), in the Grid environment.

After the last step, a session key $g^{R_c * R_n}$ is agreed between C and N and is valid within the validation period of XOS-Cert.

The user registration process is done through out-of-band means, hence it is also not included in these online interactions.

This protocol assumes the user knows the location of X-VOMS/CDA[20]. This is to accommodate the genericity of the mutual authentication model described earlier. For example, when Kerberos is used, a Kerberos ticket is attached to the step 1 to demonstrate the endorsement of a Kerberos server. When X-VOMS password based authentication is used, a username/password is sent along with step 1 and the home authentication is actually performed by X-VOMS itself.

[20]This can be either statically configured by local workstation administrator. Alternatively, it may be possible to use a directory service, such as that currently being developed in WP3.2, to enhance the availability of the X-VOMS/CDA.

4.3 XtreemOS Authorization

This section describes how the components of the XtreemOS security architecture are used to support authorization of entities in a Grid environment.

4.3.1 Authorization Problem

The authorization problem is stated as a question: can a subject s with claimed identities $su = (su_1, ..., su_n)$, attributes $sa = (sa_1, .., sa_n)$, and credentials $sc = (sc_1, ..., sc_n)$ perform an action a, where the action has a name an and a set of parameters $ap = (ap_1, ..., ap_n)$, on a target object o, where o is a resource locator that can be resolved to a concrete end point, typically with an IP address, port number and url $o = (ip : port : url)$, given a set of constraints $q = (q_1, ..., q_n)$. Any incoming request by a subject therefore has the format $(s, o, a) = ((su, sa, sc), (an, ap), (ip : port : url))$. An object may be a file, directory, application service, process, database table/field or memory location. In distributed systems these requests are typically messages that are asynchronously exchanged and either request access to data, invoke functions, notify or respond to previous requests. Permitting a subject to perform an action that harms the target object, its owner, other users or the system is therefore the challenge of authorization, often known as *safety*. In order to determine if the performance of an action is safe, constraints and policies need to be described. A policy can either be mandatory, discretionary or inferred. Mandatory policies are enforced by the operating system regardless of the application or users. For example, an object currently being written by one subject should not be written by another. These are therefore based on a predetermined model of safety with default constraints that need to be checked and verified. Such a model for VOs has already been described in the introduction, but will be further described with respect to authorization. Discretionary policies allow the users and owners of objects to define authorization policies based on their own rules and decisions concerning what work needs to be done. The operating system may still enforce these but allows the users and owners to describe what is legal according to their discretion. Finally. In any event, any policy decision is done by retrieving a set of policies of the form $(s, o, a, q)_1, ..., (s, o, a, q)_n$ that correspond to the request (s, o, a). It is possible that there are general, default policies of form $(*, o, a, q)$, $(*, *, a, q)$ or even $(*, *, *, q)$, where $(*, *, *, *)$ should resolve to 'DENY' by default - this is however a decision of the administrator. The tasks of authorization are therefore as follows:

1. *Identity, Credential and Attribute Selection*: if the subject needs to perform an action, then it needs to use the relevant identity (or alias known to the

target), select the appropriate attributes that will be checked and acquire the correct credentials to gain.

2. *Policy specification*: determine, define and store rules that govern authorization decisions

3. *Message Interception*: provide a mechanism that listens and intercepts messages targeted at particular objects (or responses to subjects) in order to make authorization decisions

4. *Policy Retrieval*: based on the contents of the intercepted message, gather the set of policies that need to be applied in the authorization decision

5. *Information Gathering*: in addition to the policies, there is a need to gather additional context information needed to make the decision. This includes parsing $((su, sa, sc), (an, ap), (ip : port : url))$, but also may include determining the state of the object, the relevant VO etc.

6. *Policy-based Authorization Decision*: execute an authorization decision logic based on the parsed message contents, relevant policies and additional information; the decision must be a 'PERMIT' or 'DENY'. Anything else is considered as a conflict, which should by default be treated as a 'DENY'.

7. *Deny or Permit*: once a decision to 'DENY' or 'PERMIT' has been made, it must be possible to block and send a negative response to the subject, forward the message to the target object respectively.

8. *Assertions*: in addition or alternative to forwarding the permitted message, the subject may be issued with a so-called 'ASSERTION' or 'CAPABILITY', which acts as a proof that the subject is authorized to perform the action. This is again based on the requirements and distribution of the system: in some cases the authorization decision may be made at a different node or in a different domain to the object.

9. *Post Authorization Actions*: after each authorization decision, there may be a need to update a history, notify other interested parties or cache the decision according to the performance and other application requirements.

In order to achieve some of the application-dependent requirements for authorization, the authorization mechanisms need to be configurable and support integration with different types of mechanisms. A model and protocols for using the XtreemOS security services have been developed to support the above authorization functionalities.

4.3.2 Protocols and Mechanisms

The protocols and mechanisms for authorization in XtreemOS are described in the following paragraphs. Before describing these, there are two further components that are assumed to be present in any system that implements an authorization mechanism, taken from RFC 2753 [**?**] on Policy-Based Admission Control:

- *Policy Enforcement Point (PEP):* intercepts incoming and outgoing messages to and from a security domain, allowing only valid one to proceed, in order to protect the objects of the domain
- *Policy Decision Point (PDP):* makes policy-based decisions about the admission of messages intercepted by the PEP

XtreemOS provides the specification and mechanisms for configuring PEPs and PDPs so that they can decide and enforce authorization decisions. These are however not part of the XtreemOS security architecture, but considered to be part of the operating system, node or domain. The implementation and location of a PEP or PDP differs based on the distribution of the system and location of parties involved in the intercepted message.

Each of the numbered interaction points are described in a paragraph below:

(1) Identity, Credential and Attribute Selection A subject's identity can vary across different VOs. For example, in VO-1 a subject s may be identified as *"bob"*, while in VO-2 the same subject s is identified as *"bobs-machine"*. Secondly, in a very dynamic environment, the qualified identity of a subject may change due to migration or a necessity for multiple connections. A subject may maintain its own repository of alternative identities or may use an identity management service such as X-IS. Attributes are used by both subjects and objects to make claims about their properties, such as *role='scientific-analyst'* or *capacity='100 users'*. Credentials qualification statements that are relevant for gaining access to a resource, such as *is-member-VO1*, *is-manager-VO2*, *not-blacklisted* etc. An outgoing PEP should be configured in order to intercept subject requests and append them with the relevant identities, credentials and attributes, according to the message format and communications protocol used. Similarly, the PEP should be configured to know how to validate credentials and attributes of incoming messages; however, in most cases, the guidelines for validation are encoded in the message, such that the PEP knows which validation keys and algorithms to apply. Nevertheless, errors are created if these are invalid or unavailable to the PEP. For example, if the PEP does not know the authority that has issued an attribute claimed by the subject.

(2) Policy specification There are three types of authorization policies supported in XtreemOS, each having the format (s, o, a, q), as discussed above. These three types of authorization policies vary with respect to how they are (i) motivated, (ii) encoded, (iii) retrieved and (iv) decided during the making of an authorization decision:

- **Mandatory**
  - *motivation:* default logic for protecting the correctness and consistency of the system; they cannot be changed by an administrator unless the source code is altered and recompiled
  - *encoding:* policies are programmed into the application or operating system logic (e.g. in file system changing access mode to a file)
  - *retrieval:* no retrieval method; are executed inline with the application or operating system
  - *decision:* no decision point component required
- **Discretionary**
  - *motivation:* logic determined from application and systems analysis, where different roles are required for application users and components.
  - *encoding:* specified in a specialized authorization policy language and can be changed by an administrator without recompiling the source code
  - *retrieval:* a search is required on the fields of subject, object and action in order to find a policy-set per request; may be encoded in a matrix for some implementations
  - *processing:* a specific component is required to make decisions; in some cases it is the reference monitoring mechanisms of the operating system. Otherwise, highly trusted components
- **Inferred**
  - *motivation:* logic is derived when different application and system components are composed. Rules of composition may exists: example $IF worth(A) > worth(B) THEN do - policy(A)$
  - *encoding:* may be specified as inference rules in code (in the mandatory style) or as interpreted extensions (in the discretionary style)

- *retrieval:* performed when key event occur such as composition of two different components
- *processing:* the decision point is dependent on the result of the inference

By distinguishing between these three types of policies, developers of the security architecture or its extensions can make decisions regarding the necessity and implementation of policy decision points.

(3) Message Interception Messages are intercepted depending on the layer at which the PEP has been implemented. Message interception requires a basic mechanism of communications mediation, caching and redirection. In the XtreemOS security architecture, messages may be intercepted at the following layers, starting from top to bottom:

- **Grid Layer**
  - *mediation:* a PEP at the grid layer has access to communications that have been broadcasted across a Grid network or directed towards any node in a VO
  - *caching:* the messages may be cached at any node in the Grid network
  - *redirection:* messages can then be appended and forwarded to a concrete end point at the application layer
- **Application/User Layer**
  - *mediation:* a PEP at the application layer listens for messages forwarded from the Grid Layer as well as from applications
  - *caching:* it only caches messages that have originated or directed at its domain
  - *redirection:* messages are then stripped/appended and forwarded to the intended recipient objects and subjects
- **OS Kernel Layer**
  - *mediation:* a PEP at the kernel layer mediates all messages on a single physical node (or within a container)
  - *caching:* only messages on the node or container are cached
  - *redirection:* forwards to a concrete address and port

(4) Policy Retrieval Policy retrieval is done based on the properties of attributes, credentials, actions and objects stated in requests.

(5) Information Gathering Additional information to be gathered is done by inspecting the requirements for making a policy decision. That is, if a policy states *(s,o,a,'time>09:00')*, then the PDP needs to gather information regarding the current time. Therefore, the semantics of constraints need to be well specified and established among a set of cooperating PDPs and context information providers.

(6) Policy-based Authorization Decision PDPs act autonomously based on the knowledge provided to them. A simple policy-based decision is *IF (all constraints are true) THEN 'PERMIT' ELSE 'DENY'*. However, logic can become more complex as subjects will present different types of attributes and credentials within different contexts that are all valid. PDPs therefore require the use of Pluggable Decision Modules (PDMs) that are created for making decisions given a request signature. In the case of purely mandatory decisions, there are no PDMs expected.

(7) Deny or Permit A denial results in a request being ignored or an explicit 'NACK' (Negative ACKnowledgement) being returned to the requesting subject. A Permit generally forwards the request to the intended object, such that it can be handled. However, it is possible for attackers to take advantage of 'NACK' messages by initiating a denial of service attack that keeps the PEP and PDP busy. There should therefore be some means of numbering attempts (see paragraph 4.3.2) or recognizing patterns of malicious behavior.

(8) Assertions Assertions are indicators that a subject has been successfully authorized, having presented a set of attributes and credentials. An assertion is signed by the authorizing party (i.e. within the domain of the PEP and PDP that make and enforce the authorization decision). The subject may produce the assertion as an additional claim to other parties that trust the authorizing party. This is then part of a single-sign-on (SSO) mechanism that uses the proof of authorization as an attribute for authentication and credential for further authorization.

(9) Post Authorization Actions These are specified as *obligations* following an authorization decision, where there is either a mandatory, discretionary or inferred directive to perform a subsequent action to the authorization enforcement. These actions may include but are not limited to:

- *Issue Assertion*: the subject may receive a signed assertion from the authorizing party showing that it was successfully authorized
- *Count Attempts and Expiration*: there may be a counter associated with the number of times that a subject can perform an action
- *Notify and Log*: if all requests or certain types of requests are to be logged
- *Compensation*: in case there was an exception or
- *Promotion or Demotion*: once a subject is authorized it may gain additional or less privileges in the domain
- *Adaptation*: an authorization decision may be followed by a reconfiguration of the system such as customization of the interface

4.4 Secure Communications

In a previous deliverable, D3.5.1, we identified various security requirements for XtreemOS. In priority, we identified the need for secure communication and security of data in XtreemOS. While security of stored data is examined in XtreemFS, in this section we describe security of communication between various components of the XtreemOS system, especially the confidentiality and integrity of communication data. The data, being communicated, can either be User generated (for e.g. authentication credential), process generated or can be operating system specific data (for e.g. synchronization information).

4.4.1 Problem space

The underlying problem of securing communication between XtreemOS components stems from the presence of open networks in accessing resources, communicating with nodes and other operating system communications in XtreemOS. It is rightly assumed that data is transmitted over insecure channels such as Internet, so that there is a need to protect data and responses in-transit.

Such a problem of securing communication can be further disintegrated into hiding data so as to hide its presence, or actual semantics of the data (*confidentiality*) and preventing modification to the data by unauthorized users (*integrity*). In addition to these broad security requirements can be further divided into following.

- **Confidentiality of communicated data**
  - *Connection Confidentiality:* Such a service provides confidentiality of data transmitted using a continuous connection (for e.g. in a session).

- *Connectionless Confidentiality:* Such a service provides confidentiality of a single unit of data transmitted (for e.g. in a packet).
- *Traffic flow confidentiality:* A service offering confidentiality of data which might be derived from observation of traffic flows. For example, in a challenge response protocol an eavesdropper can examine the reply to a challenge by observing traffic flow to a particular node.

- **Integrity of communicated data**

  - *Connection integrity with and without recovery:* In such a service integrity of user data is secured over a continuous connection and data can be recovered from the integrity providing mechanism (with recovery) or can not be recovered from the mechanism (without recovery)
  - *Connectionless integrity:* A service providing integrity of a single unit of communicated data.
  - *Selective field integrity:* In such a service integrity of only certain fields within the entire communicated data is provided.

In addition to the above services, the following points must be noted to ensure communication between various components are secure.

- It should be possible to send messages of arbitrary lengths in the protocol setup to ensure confidentiality and integrity of communicated data.
- The mechanisms used to secure communication between participating entities should incur lower network latency and should use as less computational power as possible.
- The management of services providing confidentiality and integrity should not be cumbersome.

While the above are rather generic security requirements, we specify here the problems in the case of XtreemOS. As is examined in Section 4.2.4, at the very top there are five distinct XtreemOS security services, namely, the X-VOMS, VoPS, CDA, Identity Service and attribute service. The other entity of importance for secure communications is the VO user. While it is specified that CDA and X-VOMS are included in VO Management, the VO user and VO Manager are roles assigned to members of a VO. Figure 4 ensures mutual authentication between a VO user and a resource. However, it is a high-level description of the mutual authentication process where certain intermediate steps are deliberately missed out to be specified in this section. Such communications become prime target for security in this section.

4.4.2 Assumptions

Before specifying the actual confidentiality and integrity mechanisms, we need to examine various assumptions between components of the XtreemOS system. However, such a listing is meant to be non-exhaustive and can be refined later to ensure satisfactory provision of several security services and to resolve conflicts, if any, between them.

Firstly, it is fair to assume that an entity, for e.g. VO user, a resource etc., wishing to communicate securely with another entity in XtreemOS has access to genuine public credentials (for e.g. public key) of that entity. It is imperative that such information can either be held in a database or can be provided by a Certificate Authority (CA) included in a certificate. In the case of XtreemOS, we consider CDA as the CA for such parameters. Therefore, in the following discussion the use of a PKI certificate would indicate a certificate issued by the CDA for authentication. The CDA, as mentioned in Section. 4.2.4, is responsible for registering a User. In essence, the presence of a trusted third party to guarantee authenticity of public parameters is assumed. Secondly, communicating entities have knowledge about a wide variety of commonly used algorithms, for e.g. DES, 3DES, AES, RSA etc. This is to ensure that at a given time entities may decide to secure communication using a mutually agreed algorithm from such a set.

4.4.3 Mechanisms for secure communications

In the case of XtreemOS, we decided to use Secure Socket Layer (SSL) protocol to mutually authenticate communicating entities. While being a *de facto* standard, SSL also provides for establishing keys between communicating entities for confidentiality and integrity of application data. Thus, to ensure confidentiality and integrity of communicated data we use keys derived from the SSL protocol. While description of the overall SSL protocol is strictly out of scope of this deliverable, we elaborate on several steps of the protocol in the following description for securing communication between XtreemOS entities.

Securing User-initiated Communications

If we consider a normal VO user, we can examine communications between:

1. User and X-VOMS for obtaining the XOS-Cert,
2. User and XJobMng, and
3. User and VO Manager

It is to be noted from Figure **??** that the XJobMng, communicates with the individual nodes (or the daemon running on them) on the user's behalf. So the communication between the user and the node is not included in this discussion.

The communication between User and X-VOMS is bilateral and the security requirements therein are for mutual authentication, confidentiality (as local authentication parameters or a PKI certificate will be passed) and integrity. Specifically, connection confidentiality, traffic flow confidentiality, connection integrity and mutual authentication are required for this communication. Assuming that both the user and the X-VOMS server have their own PKI certificates in place, the first step as described for any SSL transaction is the `User_hello` to the X-VOMS server. The server is passed on the user's PKI certificate for authentication, a set of cipher suite (algorithms) are prescribed for the consequent operations and the X-VOMS server certificate is requested. The X-VOMS server sends the `server_hello`, `change_cipher_suite` request, the X-VOMS PKI certificate and the key exchange material. The user and X-VOMS server on successful certificate verification mutually generate a `master_secret` from the exchanged key material and keys for confidentiality and integrity are generated from this `master_secret`.On establishing the `master_secret` the keys are derived based on standard SSL specifications. Once these mutually agreed keys are generated, these can be used either to share a session key and encrypt the data till the validity of the session keys or these keys can be directly used as long-term keys to encrypt data between the User and the X-VOMS server. For integrity, we plan to use standard HMAC or MD5 libraries. The keys for such integrity mechanisms are the same keys as derived from the SSL protocol between the User and X-VOMS server above. For preserving traffic-flow confidentiality we use padding techniques based on standard SSL libraries.

For the communication between the User and the XJobMng, only unilateral authentication of the user is required, since the XOS-Cert is a public credential. The XOS-Cert suffices the requirement of such unilateral authentication requirement. This is because the XJobMng trusts the X-VOMS server and can validate the certificate signed by X-VOMS. The XOS-Cert will contain the Global identity of the User and the signature of the X-VOMS server (or the VO Manager). XJobMng can verify this information and submit job on the user's behalf. Since XOS-Cert have a validity period on them replay attacks are unlikely by masquerading users.

The communication between the User and the VO Manager is similar to the one between the User and the X-VOMS server. It will be bilateral and mutual authentication, along with confidentiality and integrity are essential. A similar SSL protocol will be initiated between the User and the VO Manager in such a setting.

Securing VO Manager initiated Communications

The VO Manager may need to communicate with several other entities including resources, users or Resource Owners etc. in various transactions. In fact, the VO Manager's root key is given to every user and resource whilst joining the VO (i.e. while adding a user and resource to the VO). The following VO Manager initiated communication need security i.e. the communication between:

1. the VO Manager and a User, and
2. the VO Manager and a node (or something like LocalNodeMng service on the node), and
3. the VO Manager and X-VOMS

The communication between VO Manager is described in the previous section.

The communication between the VO Manager and a node usually features when adding, removing or updating a resource (in this case only nodes). The use cases in the next chapter show detailed operations in these scenarios. The security requirements in this case are mutual authentication, connection and connectionless confidentiality and integrity. We design to use the same SSL protocol for securing communication between the VO Manager and a node. The VO Manager sends a `client_hello` request and initiates the communication. It then provides the VO Manager's certificate (signed by the CDA), the cipher suite request, and the certificate request to the node. The node, in return, sends the `server_hello`, verifies the VO Manager's certificate, prescribes a cipher suite to use, sends its own certificate and sends the key exchange material. In accordance with SSL specification both the VO Manager and the node share a `master_secret` and keys for securing subsequent communications are derived from this master secret.

The VO Manager could be in the same administrative domain as the X-VOMS server in many deployment cases. However, we need to consider cases where they both are in different domains. In such cases, we need to secure transactions, which are of prime importance. We plan to use the same SSL protocol for securing such communications.

4.5 XtreemOS Isolation

This section describes how the components of the XtreemOS security architecture are used to support different forms of isolation in a Grid environment. Isolation is more concerned with management of a single, shared resource in a Grid environment, as opposed to considering the overall network of resources in the Grid. In addition, the architecture is designed to handle the four aspects of isolation identified in the background: attribute, object, interface and service/process isolation.

4.5.1 Isolation Problem

The isolation problem in a resource sharing environment is based on resource usage. Mechanisms used as solution to the isolation problem ensure that resource-reservations made for one client remain unaffected by those made for another. That is, should the resources reserved for one client fail, this should not cause the resources reserved for any other clients to consequently become unavailable. Secondly, the set of observations, outputs and views that one client has on the shared resource should not be causally affected by interactions that the shared resource has with other clients, although there are some exceptional cases. These two properties are known as fail-safety and non-interference respectively. These are formerly stated as follows: if a single resource O is being shared by a fully-ordered set of unique subjects $S = (s_1, ..., s_n)$, where for any pair $(s_i, s_j) \in S, s_i \neq s_j$ given that $0 < i, j \leq n$;, and there are a set of *resource allocations* for each subject, such that $O = (o_1, ...o_n)$ and $isReserved(o_i, s_i), 0 < i \leq n$ then:

1. $\forall s_i \in S, fail(s_i) \nRightarrow (fail(s_j) \vee unavailable(O, s_j))$; if any subject fails during its execution, this does not cause any other subject to fail nor does it cause the shared resource to be unavailable to other live subjects.

2. $\forall o_i \in O, fail(o_i) \Rightarrow unavailable(O, s_i) \nRightarrow (fail(o_j) \vee unavailable(O, s_j))$; if any resource allocation fails, the shared resource becomes unavailable to the subject owner of the allocation but not for any other subject

3. $execute(s_i, O, x_i) | execute(s_j, O, x_j) \Rightarrow (execute(s_i, o_i, x_i) \wedge execute(s_j, o_j, x_j))$; even if multiple subjects perform concurrent executions on the shared resource, the results should appear as though they were using their allocated resources individually

4. $fail(O) \Rightarrow \forall s_i, o_i : fail(s_i) \wedge unavailable(o_i, s_i)$; if the entire shared resource fails, such as a hardware problem, then the resource allocations for each subject can no longer be available

Isolation therefore extends authorization by adding the requirements of fail-safety and non-interference. The four aspects of isolation can also be described according to these two additional requirements, as shown in table 1:

From the perspective of the VO model, isolation of a shared resource is determined by the VOs to which it is assigned. The local representation of a VO is referred to as a *XOS-Container*, which 'contains' a set of subjects and objects to be isolated for a given VO, as shown in figure 5.

As seen in figure 5, in order to support management of containers and communication with their subjects and objects, there are a set of additional components introduced into the security architecture, as described below:

Isolation	**fail-safety**	**Non-interference**
Attribute	Invalid attributes of a resource allocation o_i or its subject s_i do not invalidate the attributes of any other resource allocation o_j or subject s_j	The attributes used to describe a resource allocation o_i are not known to any non-owning subject s_j unless explicitly permitted by a policy of the container manager or hypervisor
Object	The failure of a resource allocation o_i cannot be responsible for the failure of any other resource allocation o_j	Observation of the access and execution properties of a resource allocation o_i cannot reveal knowledge about the access and execution properties of any other resource allocation o_j
Interface	The availability or unavailability of an interface between $s_i : o_i$ has no influence on the availability or unavailability of an interface $s_j : o_j$	The specification and knowledge of an interface between $s_i : o_i$ has no influence on the specification and knowledge of an interface $s_j : o_j$
Service/Process	The failure of a subject s_i cannot cause the failure s_j	The behavior of a subject s_i does not causally influence the behavior of a subject s_j

Table 1: Categorization of system isolation properties

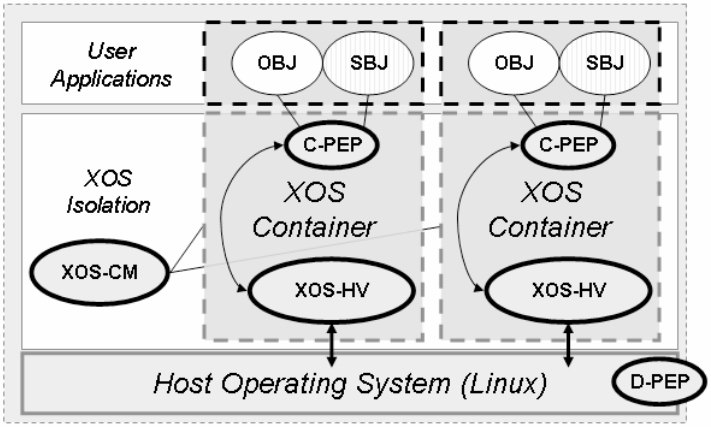

Figure 5: The components of the XOS Container concept deployed on a single operating system node

1. *XOS-CM*: the XOS Container Manager is responsible for scheduling and maintaining the lifetime of containers, as resources are shared in different VOs

2. *XOS-Hv*: the XOS Hypervisor is a component that has privileged access to the operating system on behalf of multiple containers

3. *C-PEP*: the container policy enforcement point is the supervisor for a single container and intercepts all incoming and outgoing messages to and from the container, in order to enforce access control decisions

4. *D-PEP*: the domain policy enforcement point (e.g. firewall) is the supervisor for a domain of containers and intercepts all incoming and outgoing messages to and from the domain in order to enforce domain-wide access control decisions

With these components, it is then possible to realize the four types of isolation mentioned in the background section. The protocols and mechanisms for communication and management of containers are described in the next section.

4.5.2 Isolation Protocols and Mechanisms

Seven protocols have been identified for container management and communication in order to implement isolation using the XOS security architecture. The component relationships involved in each protocol are indicated in figure 6 and described in the subsequent paragraphs.

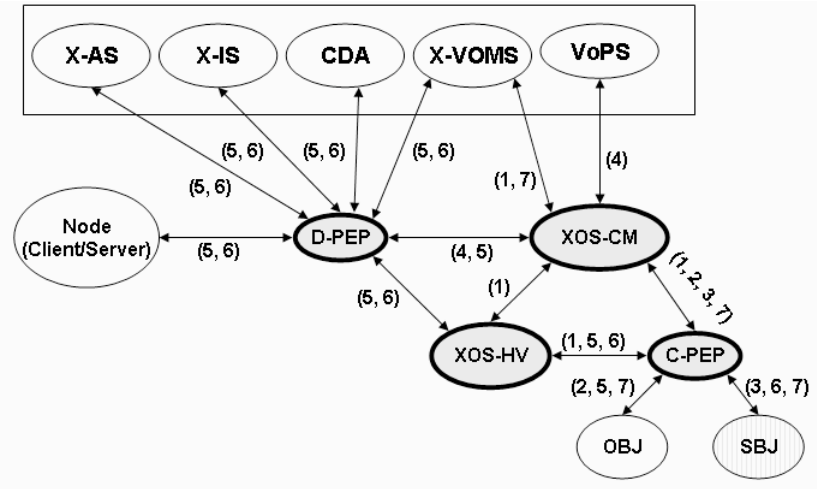

Figure 6: Component relationship diagram for using the XOS security architecture

(1) Create Container When isolation is defined as a definite requirement for collaboration in a VO, containers are created within domains (networks or machines) to partition their resource allocations and usages. Creating a container is done as follows:

1. X-VOMS → XOS-CM: include domain as member in VO and request requirements for resource allocation in VO
2. XOS-CM → new C-PEP: create new instance of container with given requirements as policies
3. XOS-CM → XOS-Hv: register address of C-PEP with a hypervisor on the domain where the container is created
4. XOS-Hv ↔ C-PEP: ensure established secure channel between hypervisor and container
5. XOS-CM ← XOS-Hv: confirm that container is set up and prepared for resource allocation in VO
6. X-VOMS ← XOS-CM: confirm availability in VO

(2) Add Object to Container Once a container is set up, then finer-grained resources can be allocated for the VO. This is a deployment and initialization action where objects such as files, databases, services or storage locations are made available to specific roles in the VO.

1. XOS-CM → C-PEP: object description
2. C-PEP → OBJ: object allocation
3. XOS-CM ← C-PEP: confirm object availability

(3) Add Subject to Container Different services or processes in a domain may be assigned the responsibility of performing given tasks at given times or due to specific events. These too have to be made available and aware of their participation in the VO.

1. XOS-CM → C-PEP: task assignment and initialisation
2. C-PEP → SBJ: assign tasks to selected subjects (process/service)
3. XOS-CM ← C-PEP: confirm task assignment complete

(4) Initialize Domain PEP :
In order for any interaction to be possible, the relevant permissions have to be set up on the domain's access control interface in addition to that of the container. This is still the first level of defence. Secondly, this is the interface made visible to the outside world and within the VO.

1. XOS-CM → C-PEP: task assignment and initialisation
2. C-PEP → SBJ: assign tasks to selected subjects (process/service)
3. XOS-CM ← C-PEP: confirm task assignment complete

(5) Handle Incoming Request to Contained Object Incoming requests to an object in a container from an external node goes through 3 levels of defence: domain, hypervisor and container. At each level, different attributes of the request are checked against different policies. The object only receives the request once each of these checkpoints have been successfully passed.

1. Node → D-PEP: request with appropriate attributes and credentials
2. D-PEP ↔ X-IS: validate node identity
3. D-PEP ↔ X-AS: validate attributes
4. D-PEP ↔ XOS-CM: validate VO existence
5. D-PEP ↔ X-VOMS: validate VO membership

6. D-PEP ↔ CDA: validate credentials
7. IF valid(request) THEN
   (a) D-PEP → XOS-HV: forward request
   (b) XOS-HV → C-PEP: check container availability and forward request
   (c) C-PEP → OBJ: check object availability and forward request to object
   (d) C-PEP ← OBJ: response
   (e) XOS-HV ← C-PEP: response
   (f) D-PEP ← XOS-HV: response
8. Node ← D-PEP: response (recall that the default response is 'DENY')

(6) Handle Outgoing Request from Contained Subject Similarly to incoming requests, outgoing requests go through three levels of defence, where the correct attributes and credentials are attached on the way out. Secondly, information that need not be exposed externally is removed from the request, cached locally and the stripped request forwarded to the next level.

1. SBJ → C-PEP: request or response
2. C-PEP → XOS-HV: place request on the XOS-HV queue for scheduling
3. XOS-HV → D-PEP: once the request can be scheduled, forward to the outgoing interface of the D-PEP
4. D-PEP ↔ XOS-CM: check existence of VO for which request is intended
5. D-PEP ↔ X-VOMS: if new membership credential is required
6. D-PEP ↔ CDA, X-AS, X-IS: get signatures for credentials, attributes and identity to perform the task
7. D-PEP → Node: forward request

(7) Remove Container Once a VO is no longer operational or in existence, the container is removed from domain.

1. X-VOMS → XOS-CM: VO is no longer active
2. X-VOMS → C-PEP: remove event
3. C-PEP ↔ OBJ, SBJ: check if all activities are completed

4. XOS-CM ← C-PEP: once activities of objects and subjects are complete, respond to the container manager

5. X-VOMS ← XOS-CM: perform any clean up actions that need to be performed e.g. accounting

5 Use Cases of Security Services

5.1 Assumptions and Use-Cases in the Architecture Derivation Methodology

The derivation of the security architecture has followed a use-case driven approach, in order to support the functional aspects of XtreemOS as effectively as possible, without introducing or assuming additional features that overload the final specification and code-base. On one hand it provides another level of security analysis for XtreemOS, while, on the other hand, it allows to validate the components that are included in the architecture. The methodology followed was to consider how to effectively enforce the general VO security model presented in 2.1 given that there are a set of functional requirements existing for XtreemOS. The realization of this model however varies per subsystem of XtreeemOS, where various aspects such as file-systems, application execution, node management and networking are focused on. This section therefore identifies and groups the set of use cases used to derive the security architecture and service specifications. Any set of activities that will lead to a use-case failing, or breaching the security rules embedded in the model, is considered as a misuse[21] of the XtreemOS functionalities. The use cases have been grouped according to specific *management scenarios* that arise in real and virtual organizations, as we seek to make the distinction between the two as transparent as possible. For example, it is envisioned that a user logging into a XtreemOS workstation should have the same operational experience when logging into a traditional networked workstation. This has resulted in 6 use case groupings (user, virtual organization, resource, application, policy and credential management), which have then emerged as the logical blocks/subsystems of the XtreemOS security architecture. These are also supported by existing literature and related frameworks for Grid Security and VO Management, such that we do not deviate tremendously from established concepts. Before outlining these use cases, we also state some assumptions concerning the technical and organizational aspects of the management infrastructure required for supporting multiple VO boundaries to be established and operate in parallel to real, physical organizations.

5.1.1 Trust Management Infrastructure

In that Grids and VOs span multiple organizational boundaries, there is always the need for trusted authorities to become involved in establishing and, in some cases, mediating interactions that cross organizational boundaries. In other cases the

[21] we did not gather a full listing of misuse cases but identified them interactively during the elicitation of security requirements

need for such authorities is subsumed by the existence of strong, trusted, ongoing links between the organizations, such that there is an agreed means of validating identity and exchanging secrets for encapsulating information[22].

Secondly, although we attempt to make limited conceptual distinction between the administration of virtual and real organization domains, there are great differences from a technical perspective. Therefore, this distinction cannot remain transparent from a technical, architectural perspective, although it remains a guiding principle for realizations of the architecture. From a technical perspective, the management infrastructure makes a clear distinction between a real and virtual organization boundary. Having made these observations, seven entities were identified in the management infrastructure, as well as their relationships, as depicted in figure 7 and described below:

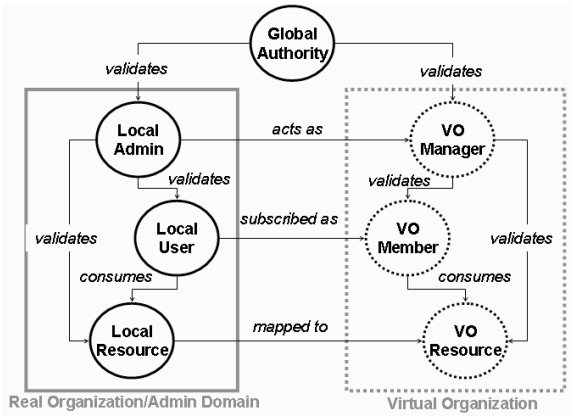

Figure 7: Trust Management infrastructure for XtreemOS Security Mechanisms and Protocols

Each entity in any instance of the management infrastructure has a particular responsibility and relationship to other entities. Relationships between entities indicate that they have particular knowledge about other entities and control over how that knowledge is used in making decisions that affect the operational integrity[23] of the infrastructure. In order for the management infrastructure to maintain its operational integrity, the entities must be trusted to perform these functions and not abuse their relationships.

- **Global Authority:** has a responsibility for issuing and validating proof that other entities possess certain attributes, such as identity, function and capability. Global authorities are trusted by various organizational domains to

[22] One example is that the organizations have an established private network connection between them, making a third-party certificate authority irrelevant for establishing the link

[23] operational integrity

perform this role, such that their compromise breaks the ability for previously unknown organizations to establish and re-establish relationships. As there is no single global authority, this typically refers to multiple authorities that trust or have agreements with each other concerning the issuing and validation of proving attributes. One common example of a global authority is a Certificate Authority (CA) in public key infrastructure (PKI), which issues and validates mainly identity certificates, using methods from public key cryptography.

- **Local Administrator:** is responsible for establishing and ensuring that security policies of a single, real domain are correctly enforced. This entails specifying boundary protection rules, regulating information flow, defining how the organization's entities are identified, determining particular constraints on internal and external interactions, and determining the global authorities to which the organization subscribes.

- **VO Manager:** plays the role of a "local administrator" within a VO. That is, the VO Manager determines the rules of interaction and resource sharing within the VO domain. However, the VO Manager does not have the overall power of enforcement as a local administrator does, in that the VO is comprised of multiple real domains, each which continue to function autonomously.

- **Local User:**is limited to access resources in a physical domain for which they have received privileges from its local administrator.

- **VO Member:** is limited to access resources in a VO for which they have received privileges from a VO manager.

- **Local Resource:** are available only within a local domain and are under the control of a local administrator

- **VO Resource:** are mapped to physical resources and are provisionally available within a VO

The remaining sub-sections now provide an overview of the use cases that build on this management infrastructure towards a comprehensive security architecture.

5.1.2 Secure Virtual Organization Management

Secure VO management is fundamental to XtreemOS, as the ability to create and manipulate the properties of VOs must be restricted, else access rights granted

on the basis of a VO's existence and membership can be exploited. A VO is essentially setting up a logical administrative boundary around multiple users and resources, such that they can be identified as a collective within a given operational context.

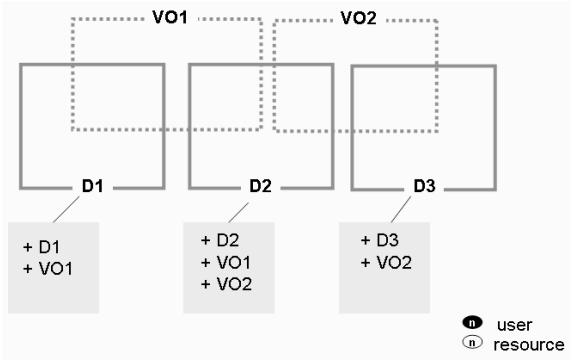

Figure 8: Illustration of virtual domains being created that spans more than one real domain

Figure 8 shows a set of physical domains $(D1, D2, D3)$, 2 VOs $VO1$ and $VO2$, with memberships $VO1 = (D1, D2)$ and $VO1 = (D2, D3)$. It is therefore seen that $D2$ maintains mappings of local group ids to $VO1$ and $VO2$. However, the domains will only share partial access to their resources with the other domains involved in the respective VOs. VOs can be created, deleted (also known as 'dissolved'), modified and queried by a system user or administrator with sufficient privileges, using lighter weight procedures as a physical, real-world domain, although achieving similar organizational features. Privileges to perform administrative tasks in a VO are associated with the role 'VO-MANAGER'. However, the possession of the 'VO-MANAGER' role does not give its holder any special, super privileges within real, physical member domains, such that the policies specified by their local administrators still hold. As membership within a VO is a prerequisite for gaining access to resources specified within the VO, it is security critical that the lifetime and constraints surrounding the operation of the VO are well managed. A VO that extends its lifetime may leave dangling capabilities and authorizations to resources that should no longer be available. VO Management and maintaining the logical boundary is in this sense a cooperative task by all members involved. That is, while consistency is not easy to enforce across domains, members act autonomously by issuing and acting on trusted events that are propagated throughout the lifetime of the VO. It is hoped that advantages for maintaining such consistency can nevertheless be gained when the resource and process hosts involved in VOs are running instances of XtreemOS.

5.1.3 Secure User Management

Having established a VO as a logical domain, it is then necessary to have a means of facilitating membership in this domain. Secure user management is concerned with ensuring that users that become members in a VO can be identified and authenticated without losing these properties in their local, home domains. One example is that users log into their networked computers and can choose from a list of VOs that they want to work in, which are available to them, in addition to the traditional network domains that their machines are allowed to join once they are authenticated to a domain server.

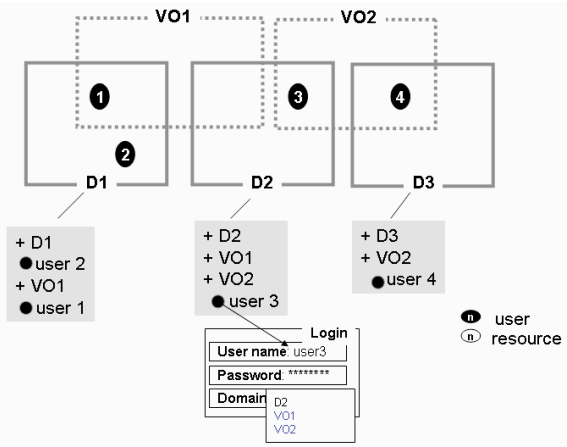

Figure 9: The inclusion of users in both the real and virtual domains i.e. VOs

Figure 9 shows the existence of 4 users with UIDs 1, 2, 3 and 4 distributed across the domains. A user is a human and a set of processes they own, which claim a need to consume or provide certain types of resources and operations in a domain or VO. User 2 is not part of VO1 and therefore has no access to objects in GID = VO1,VO2. User's 3 and 4 are in domain 2 and 3 respectively, and are both members of VO2. The figure also shows user 3 explicitly logging into domain VO2 in order to gain access to its set of shared resource objects. User management is fundamentally concerned with the addition, deletion and update of *user profiles* in a domain or VO, where a profile defines the attributes and privileges held by a human and its set of client processes. Secure user management in XtreemOS is aimed at preventing spoofs of identity, privileges and affiliation, as well as non-privileged access to resources due to leaked or falsely acquired access rights. Maintaining user identities, privileges and profiles distributed across multiple domains has the difficulty of uniformity and consistency. That is, a user may be restricted access to a resource in one domain but permitted access to e.g. a replica of that resource hosted in another domain, making user management in

XtreemOS a complex but critical security feature.

5.1.4 Secure Resource Management

Once users have been included in a VO, they will need access to resources in a similar manner to they have some access to resources in the home domain with which they can achieve particular objectives. However, the offering of resources for collaboration beyond a single domain is always associated with a set of new security risks. Secure resource management is concerned with the addition, removal and modification of nodes, services, data and operations in a VO or domain.

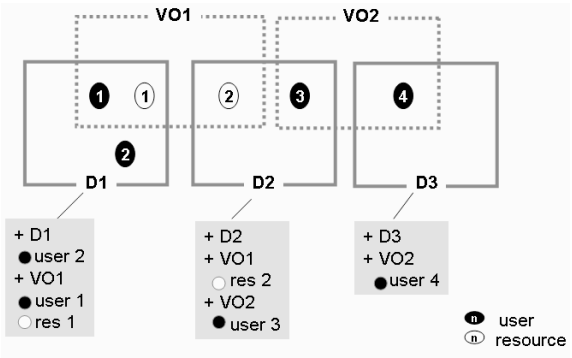

Figure 10: The inclusion of resources as instances of physical resources in VOs

Figure 10 shows now the inclusion of resource objects 1 and 2, offered by domain's 1 and 2 respectively are made available to VO1. This implies that three should be an ACL at D2 that allows $uid = guid(d1, 1) \rightarrow object2$. Nodes, services, data and operations may be contained and hosted in various ways, such that the term "resource" is an abstraction analogous to "process", where a process is said to consume a resource. Nodes are special cases of resources as they represent a full machine (which could be a virtual machine) as opposed to just an interface to storage, processing or other specialized operations. The addition of a resource to a VO or domain is a prerequisite for it being available for usage. Physical resources in a VO remain under the full control of their local administrators and physical users, but have to be shared in a controlled manner in order to maintain some agreed access rights and usage in the VO context. One means of attaining this separation of a resource into physical and virtual usage is referred to as 'virtualization', where a customized image of the physical resource is emulated for the purposes of its agreed contributions to the VO. Isolation as a security property has been discussed in an earlier section of the architecture document, where the main challenges for security are to ensure that partial access to resources can be granted to members of a VO to which the resource is assigned, while also pro-

viding a means of shielding partial failures within a VO context from propagating beyond that context. From another perspective, there is also the need for users of resources that may be virtualized to have a means of validating that these resources can really deliver the types of functionalities that they claim. This is known as *accreditation*, which is again similar to the authentication and trust attestation of users, but is done according to the functionality claims of resources.

5.1.5 Secure Application Management

Once there are users and resources included in a VO, there is some effective work that they will want to do in order to achieve the objectives for which the VO was created.

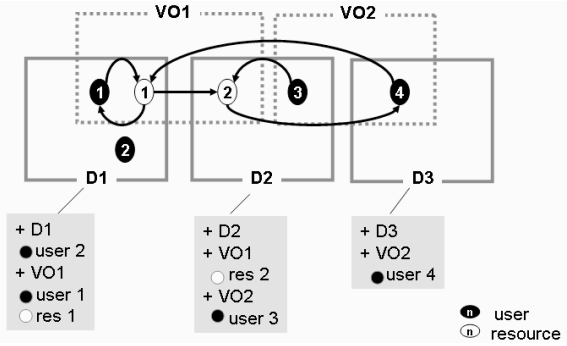

Figure 11: Application being executed in a VO as a set of interactions between distributed components

However, the way in which these tasks or jobs are performed must be regulated according to resource sharing policies that are agreed amongst the members of the particular VO. Secure application management provides the means for creating, allocating resources, starting, monitoring, checkpointing, migrating and destroying instances of applications and application components, within the constraints of the security policies specified by both resource and application owners. Recalling that access control is discretionary, it may still be possible to grant access to users for a specific task. Figure 11 shows an application being executed as different interactions between user processes and objects in the VOs. The appropriate policies need to be in place to facilitate the cross-domain interactions. The need to satisfy various sources of security policies is in itself a challenge. An instance of an application may be distributed amongst multiple resources, such that their interdependencies need to be maintained and conflicts between different policies resolved. In addition, as an application instance has both a management and service interface, it can also be treated as a resource, such that similar se-

curity mechanisms may be applied to protecting the confidentiality, integrity and availability of the instance.

5.1.6 Security Policy Management

Policies are means of specifying change in the behavior of a system at runtime. They allow a system developer to omit commitments to how the system will behave, by including a call to a policy decision point (PDP) that has a means of accepting inputs, checking policies and responding with a decision about what the system should do next. Importantly, in a VO, security policies are means of specifying which interactions should be allowed between users and resources, according to the roles and relationships.

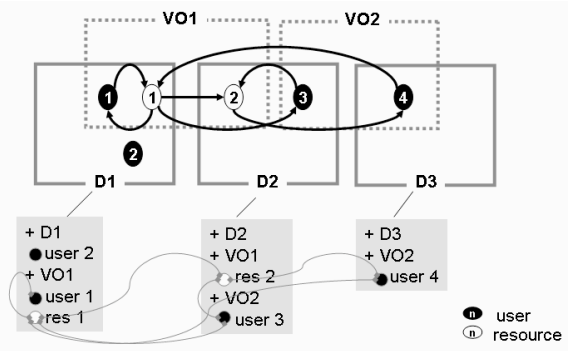

Figure 12: The security associations created between users and resources in the various VOs for the support of an application being executed

Figure 12 shows the new security associations that must be expressed in security policies to enable an application to execute across a set of processes and shared resource objects. Security policies change the behavior of a system, as they specify how to ensure that the system maintains its security requirements. Other reliable system components are therefore required in order to listen and respond appropriately to occurrences of such events, such that the policies can be enforced. In addition, the information required to detect such events is also typically distributed within the system environment.

5.1.7 Key and Credential Management

In most cases, the first step in enforcing a security policy is by checking the attributes, relationships and contexts of the users and resources involved in creating an event. As users and resources will claim these different properties, in order that interactions may proceed, they are referred to as credentials. Secondly, proving these credentials are actually associated with the claimant (user or resource),

cryptographic keys are used to attest their authenticity and that of their issuer. Key and credential management are therefore part of the foundation of the security architecture necessary in XtreemOS. Furthermore, once users and resources need to interact or communicate with each other, the authenticity and confidentiality properties of their communication must also be enforced. The ability to generate, negotiate and exchange keys in order to build up secure channels between communicating parties is also supported in the architecture.

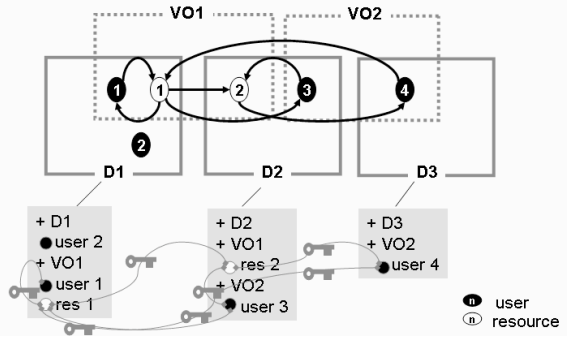

Figure 13: The inclusion of resources as instances of physical resources in VOs

Figure 13 shows a set of session keys that may be used to provide secrecy of channels between users and resources within a given context, and as specified by a set of security associations. Key and credential management includes the generation, issuing, validating and revocation of these different security elements, where the compromise of the host performing these operations breaks the overall security of the system. There are other fundamental security functions and primitives required for the security architecture to be operational, such as cryptographic algorithms and firewall support, but these have been omitted from this description as the XtreemOS is based on a standard operating system that is expected to have these fundamental mechanisms already included in its code-base. There are however cases, for the reason of performance and storage constraints, where these standard mechanisms will have to be extended or replaced.

5.2 Use Cases for VO Management

5.2.1 VO Creation

Virtual Organizations in XtreemOS can be created by any trusted user. This trust is represented by a VO certificate that the user has to obtain from the Certificate Authority before creating a VO. This signifies an agreement between the user and some physical entitiy. A user does not have to be a member of an existing VO

before creating a VO. We do not currently have the notion of VOs containing other VOs.

If VO creation is successful, the user gains the role of VO administrator in the new VO. This role is required for the user to invoke operations such as:

- adding users to a VO, updating users and removing them from a VO
- adding resources to a VO, updating resources and removing them from a VO
- destroying a VO.

Purpose: Create a VO

Actors: XtreemOS user

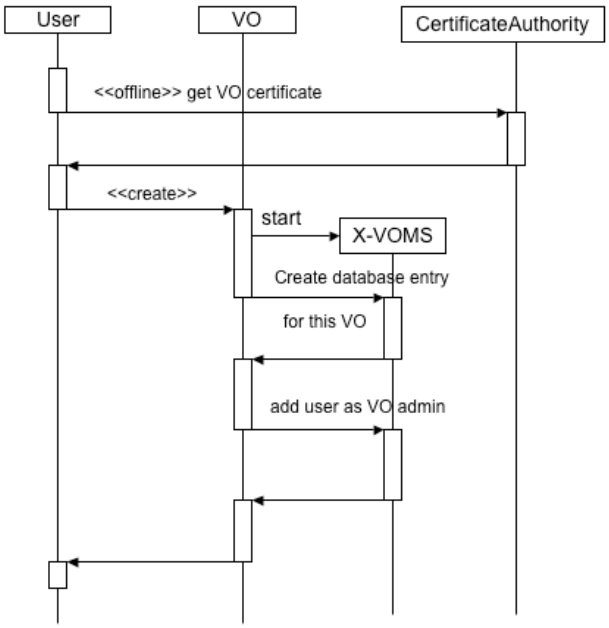

Figure 14: Sequence Diagram showing the user creating and using a VO

Pre-conditions:

- User has obtained a VO certificate from the Certificate Authority. This signifies some agreement between real entities taking place, allow the user to gain an indication of trust (the VO certificate)
- A VO with the supplied name doesn't already exist

Post-conditions:

- A VO with a given name now exists.

Invariants:

- None identified so far.

Security related functionalities:

- Authentication of user by checking the supplied credentials

Relationship to use cases/components in other work packages

- Use case provides services to: User needing to create a VO
- Use case requires services from:
  - WP3.5 - HAA to check user authentication
  - X-VOMS (VO check for pre-existence, VO creation)

Test:

- **Unauthenticated user:** If the user hasn't authenticated with a recognized HAA, the creation fails, and results in an InvalidUserException
- **VO name already exists:** This could arise because the user has already created a VO with the same name. Creation fails and results in a VOExistsException
- **Other reason prevents VO creation:** Call returns a VOCreationFailedException

5.2.2 VO Destruction

Purpose: Destroy a VO

Actors: Administrator of this VO

Pre-conditions:

- The invoking user performing the VO deletion must have authenticated with a Home Authentication Authority that XtreemOS will recognize. The credentials thus obtained are used to authenticate the user for this operation.
- The invoking user is the administrator of this VO
- There must be no jobs still running in this VO. The condition that no valid XOS-Certificates signed by this VO exist will indicate this.

Post-conditions:

- The VO no longer exists.
- The list of roles for the invoking user no longer contains an administrator role for this VO.
- All users and resources are removed from the VO.

Invariants:

- None identified so far.

Security related functionalities:

- Authentication of user by checking the supplied credentials

Relationship to use cases/components in other work packages

- Use case provides services to: VO administrator
- Use case requires services from:
  - WP3.5 - HAA to check user authentication
  - X-VOMS (VO check for existence, deletion of VO, release of VO resources, removal of VO users)

Test:

- **Unauthenticated user:** If the user hasn't authenticated with a recognize HAA, the deletion fails 'user not authenticated'
- **Unauthorized user:** If the user doesn't have VO administrator role in this VO, the deletion fails 'user not authorized'.

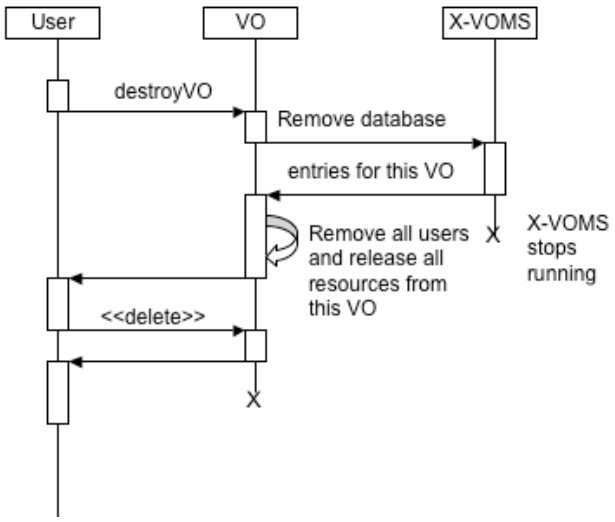

Figure 15: Sequence Diagram showing the VO administrator destroying a VO

5.3 Use Cases for User Management

5.3.1 User Registration

In XtreemOS, the user registration process turns a user into a VO user by adding entries in the X-VOMS tables. We provide two methods to allow a VO manager to introduce users into a VO.

The access control to X-VOMS is role-based. In the initial implementation, only users with the role of VO manager can add new VO users. However, the exact setting can be adapted to reflect the actual administrative policy of an organization.

Purpose: Register a VO user

Actors: A user who has the role of VO manager

Input: A user registration request

Output: Message indicating whether the registration is successful or not

Pre-conditions:

- **(1)** A VO has been created before the user sends in user registration requests.

- **(2)** A X-VOMS database has been created for this VO.
- **(3)** The very first user who registers new VO users should satisfy two pre-conditions: 1) this user has been added to the X-VOMS database; and 2) this user has an *appropriate role* [24] that would allow him to add new users.
- **(4)** There is a live X-VOMS server which accepts registration requests.

Conditions (1) to (3) are ensured by the VO lifecycle management process.

Post-conditions:

- If the return message is successful, entries about the user should be added to the appropriate tables of the X-VOMS database.

Security related functionalities:

- none.

Relationship to use cases/components in other work packages Use case provides services to: the overall XtreemOS system

Use case requires services from: WP3.5 X-VOMS

Test:

- **VO doesn't exist:** The VO that the VO manager wants to add users to does not exist.
- **Invalid user role:** The user is rejected during the registration process because he doesn't have an appropriate role.
- **Invalid user entry:** A new user entry is rejected because some information is wrong or missing.

Sequence Diagram: Figure 16 shows the interactions between a user who has the role of VO manager and the X-VOMS service during user registration process.

[24] In the initial implementation, this role will be restricted to any users with the role of VO manager.

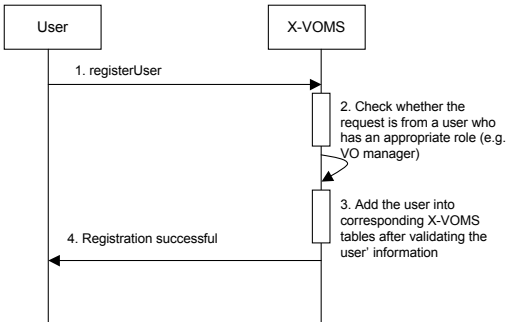

Figure 16: Sequence Diagram showing the interactions between a user who has the role of VO manager and the X-VOMS service during user registration process.

5.3.2 User Update

When a VO user's details change, his X-VOMS records need to be updated. Since X-VOMS manages back-end data with database, this process has to be transactional. That is, the X-VOMS server should ensure that the updating operation will not interfere with jobs or files that are under the old VO credential related to this user. One way to ensure that is to perform user update operations after a user' current VO credentials expire.

Purpose: Ensure correct update of user details.

Actors: User

Input: A user update request

Output: Message indicating whether the update is successful or not

Assumption: There is a live X-VOMS service which accepts update requests.

Pre-conditions:

- A user who updates his record should already has appropriate entries in the X-VOMS database.
- There are two possibilities that a user can update X-VOMS records. Either he is updating his own record; or he is updating other users' records because he has been given appropriate role(s), such as VO manager.

Post-conditions:

- none.

Security related functionalities:

- none.

Relationship to use cases/components in other work packages Use case provides services to: the overall XtreemOS system

Use case requires services from: WP3.5 X-VOMS

Test:

- **Insufficient user privileges:** The user's request is rejected because he doesn't have an appropriate role, e.g. VO manager.
- **Invalid user entry:** The user's request is rejected because essential information (e.g. VO role) is wrong or missing.

Sequence Diagram: Figure 17 shows the interactions between a user with the role of VO manager and the X-VOMS service during the updating user process.

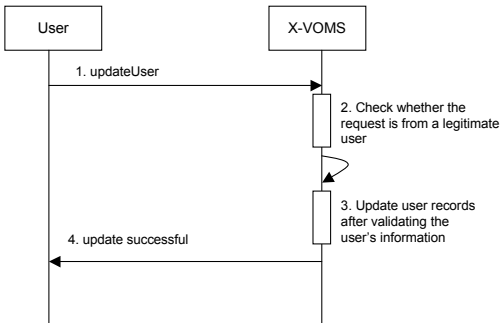

Figure 17: Sequence Diagram showing the interactions between a user with the role of VO manager and the X-VOMS service during the updating user process.

5.3.3 User Removal

When a user is no longer allowed to use VO resources, all entries related to this user should be removed from the corresponding tables of the X-VOMS database.

Purpose: Remove a user' entries from all relevant X-VOMS tables.

Actors: User

Input: the username of the user whose entries should be removed from the X-VOMS tables

Output: a message indicating whether the removal operation is successful

Pre-conditions:

- A user cannot be removed as long as there are still valid XOS-certificates associated with this user. This operation has to wait until all XOS-certficates associated with a user expire[25].

Post-conditions:

- None.

Security related functionalities:

- none.

Relationship to use cases/components in other work packages Use case provides services to: the overall XtreemOS system

Use case requires services from: WP3.5 X-VOMS

Test:

- **Invalid user:** the user issuing the request does not have appropriate rights to delete users.
- **User does not exist:** the user whose entries being deleted does not exist in the X-VOMS database.

Sequence Diagram: Figure 18 shows the interactions between a user with the role of the VO manager and the X-VOMS service during the user removal process.

[25]This is to guarantee that there is no more jobs running in the Grid environment and there are no more files remained on a resource node. Note that, at the time of writing, it is unclear how this operation should be handled regarding the user's files under XtreemFS. The major reason is that it is unclear how users are managed in XtreemFS and how does that relates to the user management in a VO. We plan to conduct further interactions with WP3.4 (i.e. XtreemFS) to clarify this point.

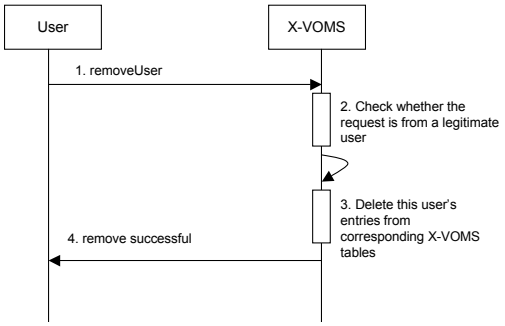

Figure 18: Sequence Diagram showing the interactions between a user with the role of the VO manager and the X-VOMS service during the user removal process.

5.4 Use Cases for Resource Management

In this set of use cases, we consider the issues related to security in management of resources in XtreemOS. We will state various assumptions and give description of the respective use cases below.

In the context of XtreemOS, resources could be defined as various entities. They could be system services, nodes, files, software licenses and in some cases even entire networks (for e.g. a cluster). Thus, we consider management of resources (adding, removing, updating), node management (selection of nodes) and the XtreemOS File System (XtreemFS).

Management of such resources is inherently a problem in Grid architectures. Such problems can be lessened by introduction of Virtual Organizations (VOs). A VO has its own VO Manager or Administrator who is responsible for management of users, resources and other functions. Thus, the problem of large-scale resource management is disintegrated and handled by each VO Manager for its own VO.

On top on all the managerial functions on resources, lies our assumption of an independent Trusted Third Party (TTP) for resources. We call such a TTP as Resource Certification Authority (ResourceCA). This entity is trusted to provide a certificate validating the claims provided by a resource. Examples of such claims could be provision of a particular service, limited memory, storage space etc. by a resource. In XtreemOS, we call this combined set of claims as Resource Contribution Expectation (RCE). The certificate from the ResourceCA would include the RCE, the ResourceCA's signature and the public key of the resource. By presenting such a certificate and on successful validation, the resource is trusted to deliver the claims it makes while joining a VO.

5.4.1 Adding a Resource to a VO

Name: Add-Resource.
Purpose: Such an operation adds a resource to a VO.
Actors: A User and a VO Manager

In this use case, a new resource is added to an existing VO. The call to add a resource is initiated by a User of XtreemOS. As is covered earlier, Users have roles in XtreemOS and a User can have a role as a VO Manager. In such a case, a VO Manager is adding a resource to the VO. For abstraction, we assume that a User adds a resource to the VO. If all the conditions are met, then a resource is successfully added to the existing VO.

Assumptions:

1. It is assumed that some form of resource specification is already in place. By this we mean that a resource should have a name, resource owner should be known, a reference to the resource is available (usually in the form of IP address of the resource), the type of the resource is known (whether it is a file, node etc.) and the resource should have obtained a certificate from the ResourceCA. We call this complete specification as RDesc. So, $RDesc = ResourceName, ResourceOwner, EPR, ResourceType, K_{rca}$ where, ResourceName = Name of the resource, ResourceOwner=UserID of the resource Owner if the Owner is a user, if not then it is just a canonical name of the resource owner, EPR = end point reference of the resource (usually the IP Address of the resource), ResourceType = type of resource, K_{rca} = Certificate from ResourceCA.
2. Since addition of resources takes place through a VO Manager, the VO Manager is assumed to be robust. The VO Manager is a bottleneck but is assumed that the VO Manager is fault-tolerant.
3. The users have the EPR of the resource being added. Since it is assumed that a user makes the call to add a resource, it is also safe to assume that the user also has the end point reference to the new resource being added to the VO.
4. The root certificate of the ResourceCA is stored in every VO (specifically in the VO Manager).

Relationship to other Work Packages

This use case provides services to WP3.2. WP3.2. includes node management and adding resource is a primary part of node management. Also, such a use case provides services to WP3.3 and WP3.4. This is because once the resource is added, some daemons need to be started for WP3.3 (resource matching) and

WP3.4 functions. The use case uses services of VO management which is an integral part of WP3.5.

Use Case Modelling

In this section, we present the use case modelling of the add resource scenario. The figure below explains various steps carried out between actors to add a new resource to a VO.

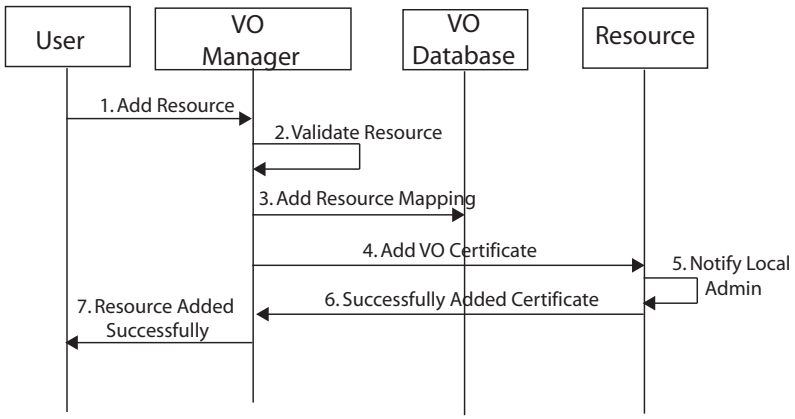

Figure 19: Sequence Diagram for Adding a Resource to an existing VO

From Fig. 19, the User first invokes a method to notify the VO Manager that he intends to add a new resource to the VO. This call would include passing the RDesc defined above to the VO Manager. The VO Manager would consequently validate the certificate given by the ResourceCA. On successful validation, the VO Manager contacts the VO Manager database and adds a mapping of the User to the RDesc provided. The VO Manager then contacts the VO and provides it with the VO Manager's certificate and asks the resource to install the VO Manager certificate. The resource notifies the local administrator to allow access to the resource from outside its local domain as it is a part of a VO. The VO Manager on receiving approval from the Resource, notifies the User that the new resource is now added to the VO.

APIs for Adding a Resource to an Existing VO

From the above use case, we can derive various method calls. Figure 20 shows the complete method calls with arguments required to fulfill the add resource use case.

So, the various method calls in this use case are:

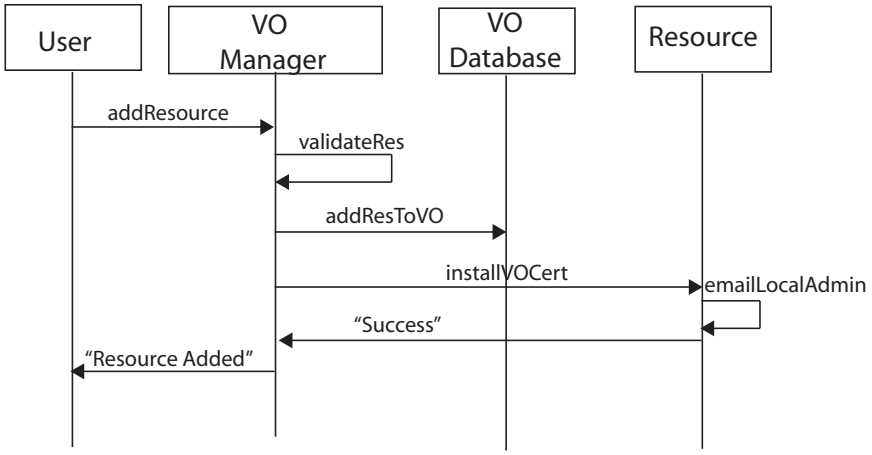

Figure 20: APIs for Adding a new Resource to an existing VO

- `addResource(VO_ID,RDesc,POLICY`[26]`)`
  Input: ID of the VO, Resource specification in the form of RDesc and any policies defined for adding resource.
  Output: A bool value stating whether the resource was added to the VO or not.
  The user invokes the above method and supplies the VO manager with the VO_ID so that the VO Manager can verify which VO the resource is to be added and the RDesc specifying the resource. The policy part is left as an optional argument because currently we have not defined what policies should be used when adding a resource. In the future, we will include possibly an array specifying the policies to be used when adding a resource.

- `validateRes(`K_{rca}`,RCE)`
  Input: The ResourceCA certificate for the resource and the RCE as given by the User for the resource.
  Output: A bool value stating whether the certificate is valid or not.
  This function is internally called within the VO Manager. The VO Manager invokes his function to verify the certificate provided by the resource. As mentioned earlier, the certificate contains the RCE, the signature of the ResourceCA and the public key of the resource. The arguments to the above function are the certificate and the RCE supplied by the resource. The VO Manager uses the root ResourceCA stored within it to verify the signature and outputs whether the certificate supplied by the User (for the resource) is valid.

[26]This is optional at this stage

- `addResToVO(VO_ID,RDesc)`
  Input: The ID of the VO and the resource specification RDesc.
  Output: Void.
  The VO Manager makes such an internal method call to add the mapping between the VO ID and the Resource Description for accounting purposes. In the later stage if the user wishes to add or remove functionality from the resource, he can issue an update command to the VO Manager.

- `installVOCert(`K_{vom}`)`
  Input: The VO Manager's certificate.
  Output: Void.
  This function is called by the VO Manager on the Resource Object to notify the Resource to install the VO Manager's certificate. Later, when a user tries to communicate with the resource for job submission the resource can then validate the user based on the certificate provided to the user by the VO Manager.

- `emailLocalAdmin(RDesc)`
  Input: Resource specification in the form of RDesc.
  Output: Void.
  The resource object calls this method to email the Local Administrator to allow access to VO members on the resource. Such a provision is to facilitate VO members from different domains to access the resources.

Additional notes

With regards to security the following notes need to be considered when implementing the above APIs:

- The method call `addResource` from the User to the VO Manager needs to be authenticated, since the VO Manager otherwise is vulnerable to Denial of Service (DoS) attacks.

- Policies for adding resources are not currently defined. However, once defined they need to be stored in the VO Manager's database. There should also be a mapping between RDesc and policies.

Tests

To verify whether the above APIs are correctly implemented, below are a list of tests:

- Reject resource: If the resource certificate is invalid, reject addition of resource to the VO. Similarly in case of conflict between specifications given by ResourceCA's certificate against RCE given by the resource, reject resource addition to the VO.

5.4.2 Removing a Resource from a VO

Name: Remove-Resource.
Purpose: Such an operation removes a resource from a VO.
Actors: A Resource Owner, the VO Manager and ResourceCA

A request for removing a resource can come from either the Resource Owner or the VO Manager may decide to remove a resource from an existing VO. The reason for removing a resource by the VO Manager may be motivated by the resource acting maliciously or not delivering as per its RCE. In the later stages of the project we will consider using resource reputation to keep track of the behavior of the resource in a VO. Another important issue is also the fact that if jobs are running on the resource then removing the resource does not make sense. In this case, an exception should be thrown to alert the Resource Owner or the VO Manager.

On successful execution of this operation, the resource is removed from the VO.

Assumptions: In this use case, assumptions identical to the add resource use case applies.

Relationship to other Work Packages

This use case provides services to WP3.2. The removal of node is also of consideration for WP3.3. The primary reason for this is when resources are being removed, if there are jobs running on the resources, the jController component need to be updated. If later we use resource reputation than the Job Directory which falls under WP3.2 needs to be updated after successful completion of this operation.

Use Case Modelling

The modelling of removing a resource is similar to the adding resource use case. The figure below explains various steps carried out between actors to remove a resource from a VO.

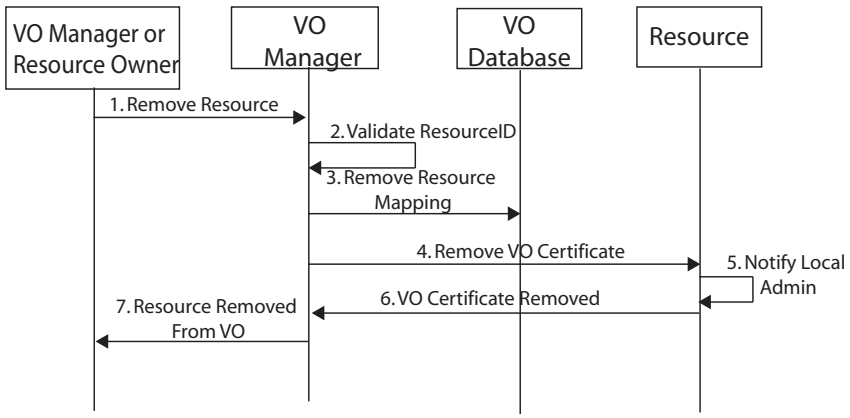

Figure 21: Sequence Diagram for Removing a Resource from an existing VO

As described above, the VO Manager (VO Admin) or the Resource Owner initiates the request to remove a resource from the VO. Such a method is invoked and the ResourceID is provided to the VO Manager for removing the resource. It is to be noted that RDesc can also be used instead of ResourceID. However, for removal by simply providing the ResourceID, rather than the whole RDesc is satisfactory. Later if there are policies defined for removing resources than the need to supply the full RDesc may arise, but it would mean minor modification to the overall API for this use case. The VO Manager then notifies its database to remove the mapping between the VO and the Resource, since the resource is being removed from the VO. On success, the VO Manager informs the resource to remove the VO Manager certificate. In this process, the resource also notifies its local administrator to remove access to the resource for this particular VO. The Local administrator then updates its access controls (possibly firewall, IDS etc) for disallowing access to the resource for the VO. On successful completion of all these tasks the Resource Owner or VO Manager receives a success message, stating the removal of resource has been accomplished.

APIs for Removing a Resource from a VO

Having generalized the use case in the above section, we will derive various function calls in this section for removing a resource from a VO.

Figure 22 shows the complete method calls with arguments required to remove a resource from a VO.

The details and description of the functions are below.

- `removeRes(ResourceID,VO_ID)`
  Input: ID of the VO from which the resource is to be removed and the ID

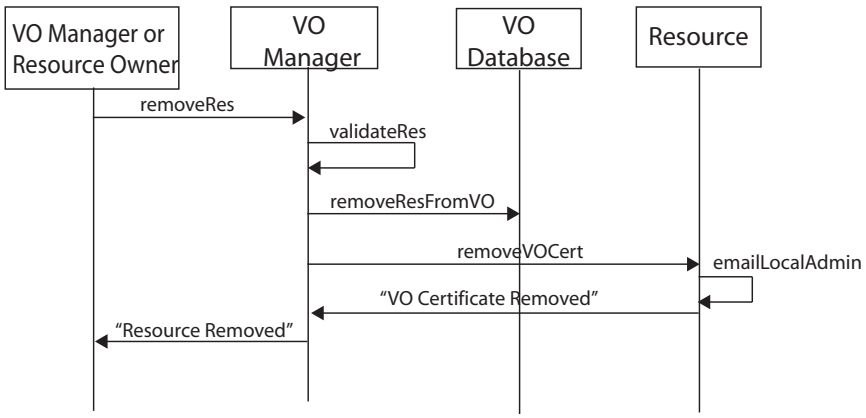

Figure 22: APIs for removing a resource from a VO

of the resource to be removed.
Output: A "Resource Removed Successfully" message.
The VO Manager or a Resource Owner calls this message by providing the ID of the resource and ID of the VO as arguments. As described earlier, at this stage ResourceID is enough rather than providing the full RDesc.

- `removeResFromVO(ResourceID,VO_ID)`
  Input: The ID of the resource to be removed and the ID of the VO from which the resource is to be removed.
  Output: A "Mapping Removed".
  By calling this function, the VO Manager notifies the VO Manager's database to remove the mapping between the VO and resource being removed. The arguments supplied in this call are the ID of the resource and the ID of the VO. The VO Manager's database has to search within various RDesc for the ResourceID provided and consequently remove the mapping between such RDesc and the VO. Once finished, the function should return a message to the VO Manager stating the mapping has been removed from the database.

- `removeVOCert(VO_ID)`
  Input: The ID of the VO.
  Output: A "Success" message.
  This method is called by the VO Manager to notify the resource to remove the VO Manager's certificate it possesses. This will disallow any VO users to use the resource in consideration in the future. The resource removes the VO Manager's certificate from its store and will return a "Success" message back to the VO Manager.

- `emailLocalAdmin(VO_ID,ResourceID)`
  Input: The ID of the VO and the ID of the resource.
  Output: Void.
  The resource object while executing the removal of VO certificate calls this method to notify the local administrator to restrict users access to itself since it is now removed from the VO. It is left to the discretion of the local administrator how the access control is established after such a call.

Once all these above operations are successful, it is guaranteed that the resource is removed from the VO.

Additional notes

With regards to security the following notes need to be considered when implementing the above APIs:

- The method call `removeRes` from the Resource Owner or the VO Manager or the Resource Owner to the VO Manager needs to be authenticated, since the VO Manager otherwise is vulnerable to Denial of Service (DoS) attacks.
- Policies for removing resources are not currently defined. However, once defined they need to be stored in the VO Manager's database. There should also be a mapping between RDesc and policies in the VO Manager's database.
- For the `removeRes` method, there should be validation of the ResourceID sent as otherwise VO Manager would be vulnerable to Denial of Service attacks.

Tests

To verify whether the above APIs are correctly implemented, below are a list of tests:

- Check the identity of the entity requesting resource removal. If the entity is not authorized to remove the resource, reject resource removal call.
- Check whether the resource received the remove certificate call and has successfully removed the certificate.

5.4.3 Updating a resource in a VO.

Name: Update-Resource.
Purpose: Such an operation updates the description of a resource in a given VO.

Actors: User (this could be the resource owner), the VO Manager and the Resource to be updated.

During XtreemOS operation in a production environment, it might happen that certain description of the resource might change. Common examples of these operations are a resource adding more memory, more storage space etc. for the VO. In such cases, we need to have an API that performs such functions. It is to be noted that updates are only successful when the resource **at least** provides the contribution it guaranteed when it joined the VO. There should be policy management in place to notify the VO Manager if there are conflicts between the update operation and the RCE specified by the resource when it was added to the VO. The VO Manager can then take suitable steps to manage such conflicts or disallow the update operation completely.

On successful execution of this operation, the description of a resource is updated for a particular VO.

Assumptions: The assumptions for this use case remain the same as for the Add-Resource and Remove-Resource use case.

Relationship to other Work Packages

This use case provides services to WP3.2. If resource reputation is considered and it depends on the resource's contribution to a VO, the reputation of the resource should also be updated on completion of this use case.

Use Case Modelling

Interactions between various entities for the Update-Resource use case are depicted below.

Figure 23 describes various steps to be followed for the update operation. The User, who can be also a resource owner, initiates a call to the VO Manager to update the resource. The new description is provided with this call. The VO Manager first validates the resource based on the certificate from the ResourceCA. It then checks any conflicts that the update operation might have with the RCE provided by the resource when the resource was first added to the VO. If there are no conflicts, the VO Manager updates its database to reflect the mapping between the updated RDesc for the resource and the VO_ID. The next step is to notify the resource to update its status. On success of all the above calls, the VO Manager returns a success message to the User stating that the new description of the resource was successfully noted.

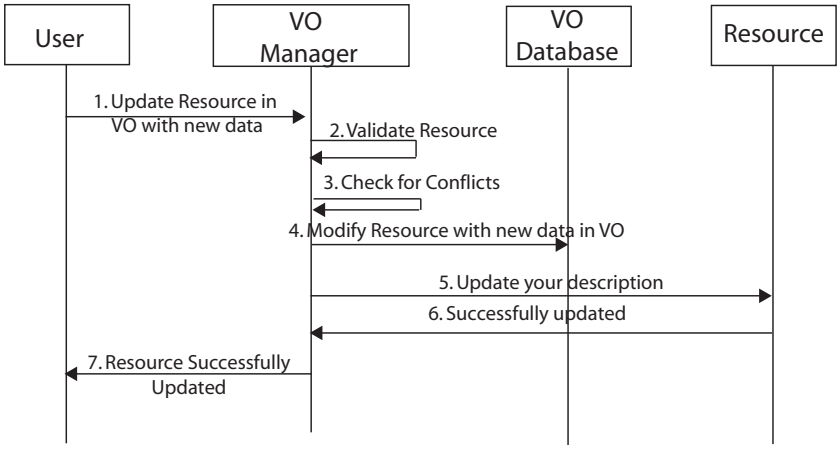

Figure 23: Sequence Diagram for Updating a Resource in a VO

APIs for Updating a Resource of a VO

Having identified the various steps in updating a resource for a particular VO, the next step is to derive concrete function calls and APIs. We provide these function calls below.

The details and description of the functions are below.

- `updateRes(VO_ID,ResourceID,newRDesc)`
  Input: ID of the VO in which the resource is to be updated, ID of the resource, new description of the resource in form of RDesc.
  Output: A "Successfully Updated" message (String).
  A User (or the resource owner) calls this message. The user specifies which VO the resource belongs to, the ID of the resource and the new specifications of the resource in the form of new RDesc. On successful completion of this operation, a string "successfully updated" is returned back to the user.

- `validateRes(ResourceID,`K_{rca}`,RCE)`
  Input: The ID of the resource, the certificate from the ResourceCA and the RCE.
  Output: An OK or successful message (String).
  This function call is initiated by the VO Manager. The VO Manager validates the ResourceID and verifies the signature on the ResourceCA's certificate. The purpose of this function is also to verify whether the ResourceID

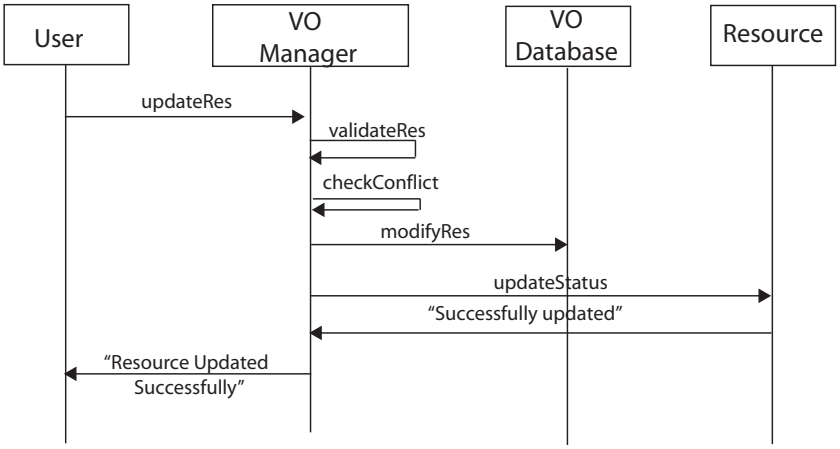

Figure 24: APIs for updating a resource in a VO

supplied is identical to the one in the ResourceCA's certificate. The VO Manager gains the ResourceCA's certificate from the newRDesc. Once the ResourceID and the certificate are verified, the VO Manager is returned an OK message.

- `checkConflict(newRDesc)`
  Input: The newRDesc supplied by the User.
  Output: An "OK" message (String)
  The purpose of this function, as discussed earlier, is to find any conflicts that might occur if the newRDesc does not satisfy the RCE supplied by the resource when it was added to the VO. The VO Manager checks its database by supplying the newRDesc given by the User. The function parses the ResourceID from the newRDesc and checks the RCE for this ResourceID from the database. A check is then made between the newRDesc and the RCE of the resource. Only when the specifications in the newRDesc are more than the RCE is the function successful. On successful checking the function returns an OK message to notify the VO Manager that no conflicts exist.

- `modifyRes(VO_ID,newRDesc)`
  Input: The ID of the VO and the newRDesc.
  Output: Void.
  The VO Manager initiates this call and updates its own database to reflect

the updates resource. The newRDesc is mapped to the VO_ID. Thus, when specification about the resource is inquired upon the newRDesc is returned instead of the old specifications.

- `updateStatus(newRDesc)`
  Input: the newRDesc.
  Output: Void.
  As the last step, the VO Manager informs the resource to update its own status. The newRDesc is supplied to the resource in this function. The resource updates itself by storing the newRDesc.

On successful completion of all the above operations the new resource specifications are establishes throughout the VO.

Additional notes

With regards to security the following notes need to be considered when implementing the above APIs:

- The method call `updateRes` from the User (Resource Owner) to the VO Manager needs to be authenticated, since the VO Manager otherwise is vulnerable to Denial of Service (DoS) attacks.
- Policies for updating resources are not currently defined. However, once defined they need to be stored in the VO Manager's database.

Tests

To verify whether the above APIs are correctly implemented, below are a list of tests:

- Check conflicts between newRDesc and RCE supplied by the resource while joining the VO. If conflicts found reject update of the resource.
- Check that the resource has updated its specifications to newRDesc from RDesc.
- Check that the return value to the user is a string.

5.4.4 Selection of Nodes.

Name: Node-Selection
Purpose: The purpose of this use case is to select nodes based on certain requirements.
Actors: A User, the Resource Matcher and the Node Manager.

An XtreemOS user would like to execute jobs on nodes. For optimizing execution of such jobs, there is a need for selecting the nodes available at that instance of time and to select the best nodes that can handle the particular job under consideration. This use case tackles operation of such scenarios where the best nodes are to be selected.

It becomes apparent to understand the underlying node overlay network for such an operation. The nodes (resources) of a VO are connected by an overlay network. The VO Manager acts as an entry point for this overlay network. By this we mean that if any component wants to interact with resources of a VO the VO Manager has the end point reference (usually the IP Address) of a set of resources. The resources are assumed to keep references for their *neighbours* in the overlay network. Thus, by contacting a resource through the VO Manager and iterating through the neighbours of the resource, it is possible to "walk" the entire overlay network. Information retrieval from the entire overlay network is enables optimally using such a scheme.

On successful completion of the Node-Selection use case a set of nodes are provided to the User, that match the requirements provided while submitting the job.

Assumptions:

- The VO Manager keeps a record of end point references of a set of nodes. These nodes act as entry point to the underlying node overlay network for a particular VO. Also, such a record is updated when a resource or resources within the record leave the VO.
- Nodes have record of end-point references of at least two of their neighbours. The overlay can be structured for e.g. using a Chord ring or can be semi-structured network for e.g. as in a DHT.
- The entity who queries for the selection of nodes already possesses the XtreemOS certificate (XOS-cert) of the user requesting the execution of the job.

Relationship to other Work Packages

This use case provides services to WP3.2 specifically. Also, services are provided to WP3.3 for obtaining resources, specifically nodes, that match the job requirements.

Use Case Modelling

The main actors in this use cases are the entities Resource Matcher and the Node Manager. The figure below shows the interactions between these entities and the

user who submits the job.

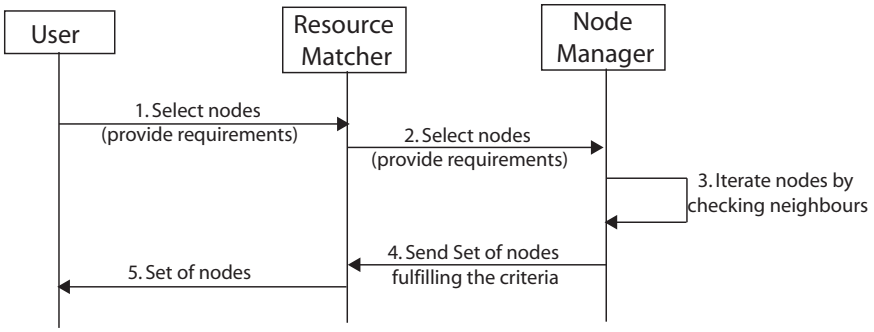

Figure 25: Sequence Diagram for selecting nodes matching job execution requirements

As can be seen in Figure 25 the user requesting a job execution first comes in contact with the Resource Matcher. It submits the job and during job execution a call to select nodes is initiated. The requirements are given in the job file submitted previously. The Resource Matcher makes a call to the Node Manager which can retrieve information about nodes by iterating through the entire node overlay network of a particular VO. It is to be assumed that the Resource Matcher already has the XOS-cert for the user, so it knows which VO the user belongs to. The Node Manager is returned a list of nodes satisfying the requirements provided which it passes to the Resource Matcher and ultimately to the User.

APIs for Selection of Nodes Satisfying a set of Requirements

From the sequence of steps above it is easier to extract exact functions that handle the selection of nodes use case. Figure 26 describes the concrete function calls required to implement the node selection scenario.

The details of the function calls are given below:

- `SelectNodes(XOS-Cert,Requirements[...])`
  Input: XtreemOS Certificate of the user and the requirements as given in the job file.
  Output: An array of nodes satisfying the requirements.
  While job execution a call is invoked for this method to select nodes satisfying the requirements. The user also sends its XtreemOS certificate for the Resource Matcher. Requirements are sent in the form of a string array specifying what is needed for each of the nodes. In return the function returns an array of Nodes satisfying the given requirements.

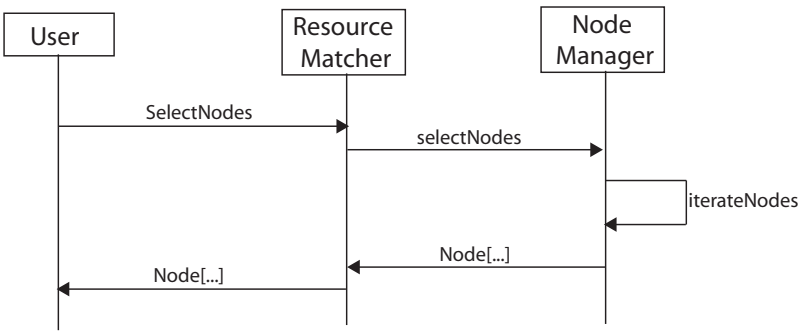

Figure 26: APIs for selecting nodes from a VO matching the given requirements

- `SelectNodes(Requirements[...])`
  Input: The requirements array as provided by the user.
  Output: An array of Nodes satisfying the requirements.
  The Resource Matcher, in turn, calls the Node Manager to select nodes that satisfy the requirements given in the job file by this message. In return a set of nodes are sent back to the resource matcher.

- `IterateNodes(Requirements[...])`
  Input: The requirements array as provided by the user.
  Output: An array of Nodes.
  The Node Manager iterates through all the nodes for the VO using the node overlay network for the VO and checks RDesc of all the nodes. If RDesc are greater than the Requirements than the node is selected and added in the set of nodes to be returned to the user.

On successful completion of the above method calls a set of nodes are returned to the user, who can then initiate the job.

5.5 XtreemOS Security support for XtreemFS

This section describes how the security services support XtreemFS (also referred to as XtFS for short) by showing the interrelationships with security components, and how the interactions between the components of XtFS are protected. As a result of XtFS being an extension of traditional file-systems, established filesystem security concepts and mechanisms have been included in its architecture and protocol specification. Nevertheless, there are technical challenges associated with realizing these mechanisms in a filesystem distributed across multiple organizational boundaries. The two main security aspects are facilitated by using the XtreemOS Security:

1. Security bootstrapping and deployment of XtreemFS using X-VOMS
    - installation of root XOS certificates for authentication
    - authentication of components
    - initialization of ACLs
    - issuing of identity certificates
2. Authentication and access control per request
    - authentication of users with clients
    - file and directory operation requests including read, write, copy and delete

These are described first as component relationships and secondly as sequence interactions.

5.5.1 Static Modeling

This section gives an overview of the interrelationships between the XtFS and XtreemOS Security components, for support of the security requirements determined for XtFS. For more information on the protocols of XtFS and how each component functions, refer to the relevant specification documents; however a brief overview is included here for comprehension of the security architecture (taken from architecture deliverable):

- **Object Storage Device (OSD):** store complete or portions of objects and implement a read/write interface to them.
- **Metadata and Replica Catalogue (MRC):** These two components store metadata (extended and POSIX) and replica locations of files. Note that from a security perspective the MRC is very critical; part of the Metadata includes the access rights to files and directories on OSDs.
- **Replica Management Service (RMS):** This component will cooperate with the rest of services to take decisions on when and where replicas are created and will also decide when replicas should be removed from the system.
- **Directory Service (DS):** a central instance for registering and querying information about file system services and volumes.
- **Client:** a client-side library that allows access to all XtreemOS features, but also allows mounting the file system as a normal UNIX file system

The most critical security component in XtFS is the MRC (Metadata and Replica Catalogue), which issues capabilities to clients requesting access to a file or directory, which is located on an OSD (Object Storage Devices) or any other their replicas. The authentication of Clients is critical before issuing them with capabilities. The authenticity, freshness and validity of capabilities needs to be assured by the MRC when issuing them to Clients, as access control to files and directories in a filesystem is the most fundamental security requirement.

Before clients can be issued with capabilities the MRC requires a means of authenticating their identities and attributes claimed. Similarly, OSDs require the means to authenticate that these clients have really been issued with the capabilities that they declare. Secondly, OSDs must also be authenticated with the MRC, otherwise rogue OSDs could provide false data to Clients or maliciously lead them to writing confidential data. Similarly, as the RMS is responsible for managing and initiating replicas, this must also be authenticated before these decisions to strip data can be made. The task of the security components is hence to maintain a set of trust relationships between the XtreemFS components in order to facilitate this.

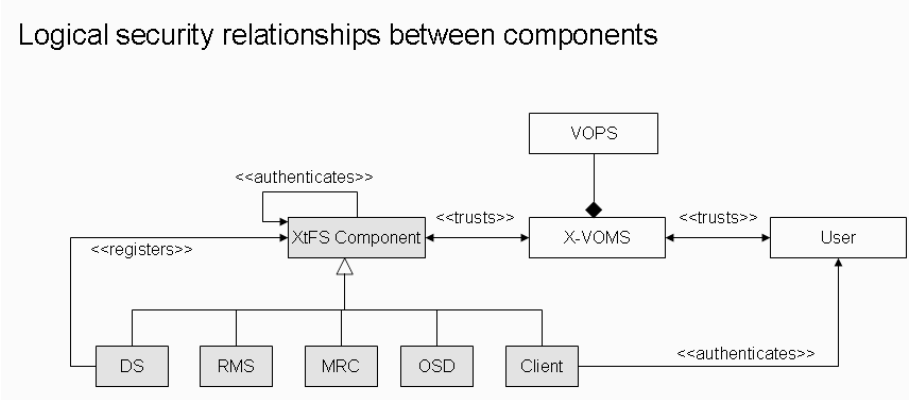

Figure 27: Component diagram showing trust relationships between XFS components

X-VOMS may again act as a trust anchor for all XtreemFS components and users. There may be multiple authorities that are trusted but that is not included here. Using X-VOMS as a trust anchor, authentication between components is possible, as well as between the clients and users. Furthermore, once components have been issued with a certificate from X-VOMS, this could be registered in the directory service in order to support discovery within XtreemFS.

However, XtreemFS requests must be handled quickly, such that the distribution of the file system components remains transparent. The MRC may cache capabilities in order to remove the need to regenerate capabilities for the same

client every time they make a request, in order to address performance problems introduced by the inclusion of additional protection and cryptographic operations in every request. Capabilities must therefore be unforgeable and have a means of proving their freshness, even if cached. It is therefore expected that a shared-secret is established between the MRC, Client and OSD involved in different requests, assuring that the capability remains unforgeable. It is noted that this introduces performance problems on requests, but these are not discussed explicitly in this document.

5.5.2 Secure Bootstrapping of XtreemFS

As XtreemFS is a distributed file system, the authenticity of components is critical for every operation. Bootstrapping is the process of ensuring that this authenticity can be determined when the components are interconnected and operations are requested. Bootstrapping is only necessary the first time a particular deployment instance is done, a problem occurs that requires redistribution and, partially, when central components such as the Directories or MRCs need to be removed.

Purpose: to bring a deployment of XtreemFS to an operational state and to ensure that each component can be authenticated during requests.

Actors: All XtFS Components, X-VOMS, Users or Applications

Input: XOS-Certs with public-keys and attributes

Output: Successful, interconnection of authenticated XtFS Components, as well as registration of attributes with the Directory Service (DS)

Implementation Note: The Directory Service could be used for storage and registration of component certificates. Performing signatures on every subsequent request could be too expensive. Initial authentication should be followed by exchange of shared secret between the components.

Pre-conditions:

- All XtFS components have been issued with XOS-Certs from a trusted X-VOMS certificate issuer
- Potential users and administrators also have a user version of XOS-Cert issued

Post-conditions:

- All component interactions can be authenticated
- It is possible for the user or application to be authenticated before receiving a capability to perform operations

Invariants:

- Components that cannot be authenticated are not allowed to be registered with the directory
- Clients with unsuccessful authentication cannot find registered components

Security related functionalities:

- **Authentication:** this is supplied by the mutual trust that the components, users and applications have in X-VOMS
- **VO membership checking:** X-VOMS could check some membership or other policies before issuing a XOS-Cert

Relationship to use cases/components in other work packages Use case provides services to: WP3.4

Test:

- **will need to test that the authentication mechanisms do not break the performance of the protocols**
- **using rogue components should not be allowed in XtFS operations**

Sequence Diagram: Figure 28 shows the interactions between the security components and XFS components during secure addition of replicas to files or directories.

It is assumed that all components have a similar trust anchor in X-VOMS, from which they have been issued with XOS-Certs. Every component also installs the root XOS-Cert of X-VOMS in order that they can validate all other XOS-Certs that have been issued.

1. All components must
2. All Clients

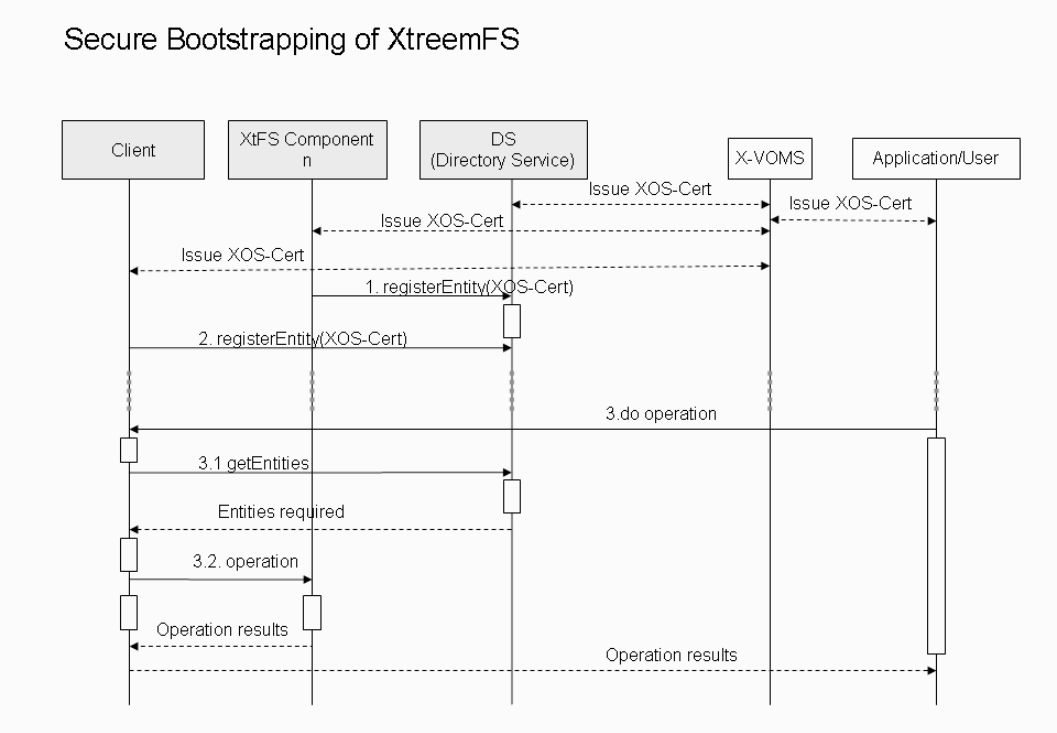

Figure 28: Sequence Diagram showing the interactions between security components and XtFS for bootstrapping

3. Application or user requests to perform a file, directory or administrative operation on XtreemFS

4. Client must be capable of authenticating the user or application

5. Once authentication is possible, the Client retrieves the information for the required entities from the DS, only if the user is authorized to retrieve this information (implies that ACLs must also be set up on the DS)

6. The Client then performs the operation according to the established protocol, and returns the results to the user

5.5.3 Initialization of ACLs based on VO policies

Although the MRC and Ds are the access controllers of the XtreemFS, there is a need for a higher-level framework for specifying and installing access control lists (ACLs). This section specifies a means of installing these ACLs, given that groups in XtreemFs can be resolved to VOs maintained by X-VOMS, using a specialized

method on the VOPS (VO Policy Server) for generating these POSIX-compliant ACLs.

Purpose: to create a group/VO and install ACLs for that group or for addition of components to groups

Actors: Client and User (with admin privileges), MRC, X-VOMS, VOPS

Input: group/VO or member parameters

Output: ACLs installed on MRC

Pre-conditions:

- All components can be authenticated using a common trust anchor
- Only administrators should be able to perform ACL and group creation operations

Post-conditions:

- ACL is created to reflect change in group and group membership

Invariants:

-

Security related functionalities:

- **Authorisation:**

Relationship to use cases/components in other work packages Use case provides services to: WP3.4 adding and removing ACLs

Test:

- **ensure that only users in a group can perform operations permitted by the group**

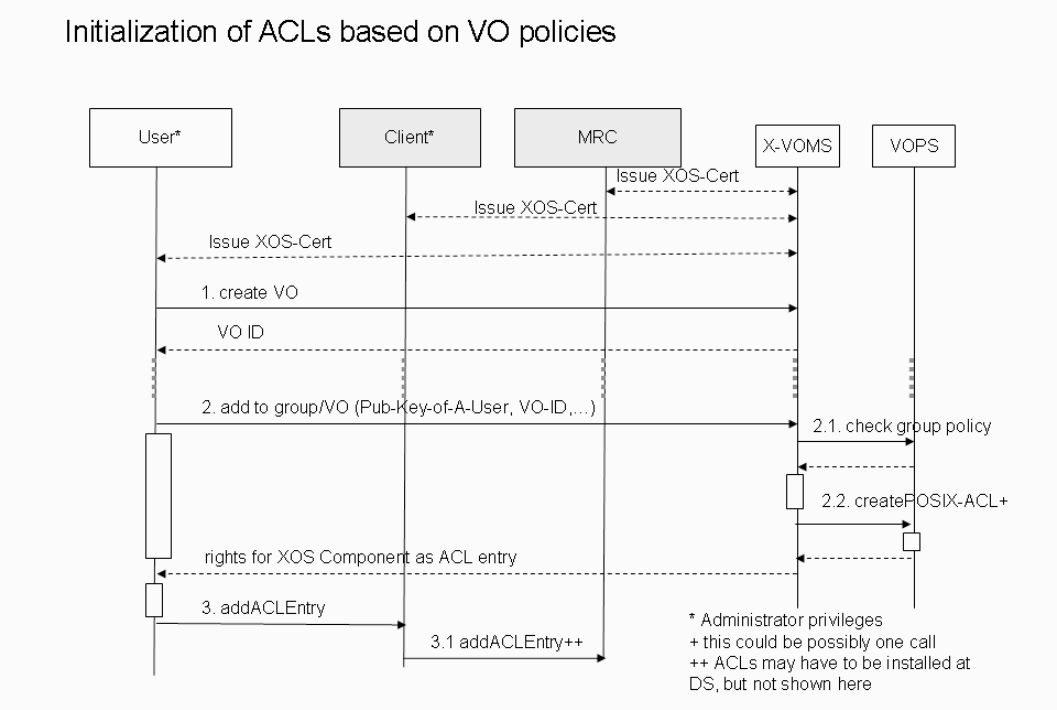

Figure 29: Sequence Diagram showing the interactions between security services and XtFS components for installing an ACL

Sequence Diagram: Figure 29 shows the interactions between the security components and XFS components during the initialisation of an ACL.

1. creation of a local group may be elevated to creation of a global VO; this returns a globally-unique VO-ID, which has a specified format

2. adding of a user to a global group entails sending the public-key of the target user to X-VOMS; internally to a Client in XtFS, the public-key would be mapped to a local uid

   (a) Before a user can be added, the group policy needs to be checked to see if their attributes are valid for the group/VO

   (b) this is a method for transforming a set of policies specified for users in a VO to POSIX-compliant ACLs

3. user as administrator requests that ACL entry be created for the MRC

   (a) Client sends ACL entry request to MRC

5.5.4 Secure File Operations

The standard protocol for performing any operation on a file is (i) Client makes request to MRC, (ii) MRC checks if request is valid and issues capabilities, as well as list of replica OSDs involved and (iii) Client uses the capabilities to perform operations on set of OSDs. These operations need to be secure between the Client and User as well as between the Client and OSDs. Integrity checks are particularly important, as segments of striped files can easily be altered without easy detection.

Purpose: to ensure authentication and integrity of file operations

Actors: Client, MRC, OSDs

Input: request from Client to do read, write, delete, copy and other file operations

Output: operation performed

Implementation Note: There is currently no specification for how exactly a shared secret is established between the Client and OSDs, nor the generation of a session key between the Client and User, but such mechanisms are required.

Pre-conditions:

- The Client, MRC and OSDs can be authenticated using the same trust anchor

Post-conditions:

-

Invariants:

-

Security related functionalities:

- **Authentication:**
- **Integrity:**

Relationship to use cases/components in other work packages Use case provides services to: WP3.4 reading, writing, copying files

Test:

- **non-authenticated operations are always denied; no capability can be issued**

Sequence Diagram: Figure 30 shows the interactions between the security components and XtFS components during standard file operations.

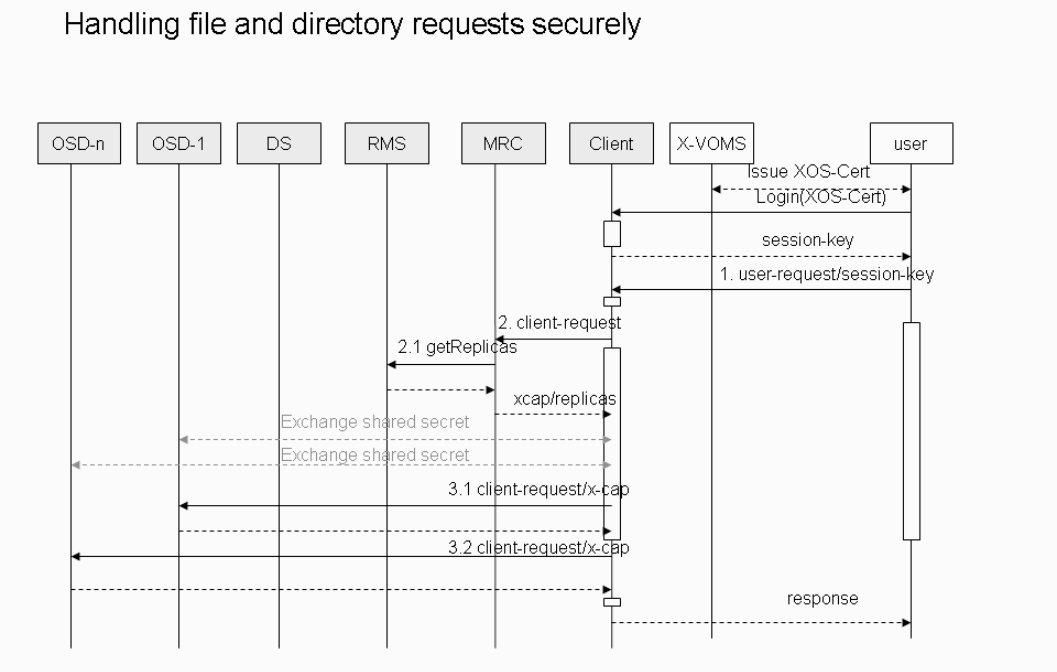

Figure 30: Sequence Diagram showing the interactions between security services and XtFS components during standard file operations

The above is the standard protocol for XtreemFS operations. The challenge of the security mechanisms are to ensure that they are not too expensive for the frequency and size of requests/responses to be made.

5.6 Use Cases for AEM

This section describes how AEM makes use of our security services by detailing the interactions between the AEM components and the security services described in 4.1.2.

5.6.1 Job Creation

The AEM job creation process is typically triggered by a user typing the command `XSub` to a XtreemOS Linux terminal. This process involves three types of security components: HAA, X-VOMS, and CDA.

Purpose: Ensure only registered VO user can create jobs on VO resources and only jobs from validated (i.e. having a valid XOS-Cert) users can have a job entry on JobDirectory.

Actors: User

Input: XSub command and parameters, including a choice of attributes, such as VO role and group.

Output: A valid JobID

Implementation Note: Authenticating against HAA should only need to be performed once. The security tokens (e.g. Kerberos tickets) issued by HAA can be reused within its lifetime.

Pre-conditions:

- The user has pre-registered with a X-VOMS recognized HAA.

Post-conditions:

- A valid job entry is created on the JobDirectory. [27]

[27] A job with a valid JobID is not necessarily a job that has been successfully launched and/or executed on a Grid resource node. There are other factors that may influence whether a job is executed successfully or not. For example, a node can deny the execution of a successfully submitted job based upon the VO attributes in a XOS-Cert.

Invariants:

- When the authentication against HAA has been done, the job creation process can reuse an existing security token, **if it is still valid**, to access X-VOMS directly and bypass the HAA.

Security related functionalities:

- **User validation:** Validating user's home authenticity using security credentials (e.g. Kerberos ticket, X.509 certificate, X-VOMS based password) with HAA;
- **VO membership checking:** Checking whether a user is registered with a VO by validating this with X-VOMS; and
- **Credential requesting:** Obtaining a XOS-Cert for job creation via contacting CDA.

Relationship to use cases/components in other work packages Use case provides services to: WP3.3

Use case requires services from: WP3.5 HAA, X-VOMS, CDA

Test:

- **non-registered user**

Sequence Diagram: Figure 31 shows the interactions between the security components and AEM components during security services during the AEM job creation process.

5.6.2 Resource Matching

Resource matching provides functionality of matching resources against requirements and user credentials, retrieved from XOS-cert. The selection process is performed in two steps. In the first step resources are matched based on static information, which functionality is provided by Node management service (WP3.2). At the second step, dynamic information is matched, and only the resources, which satisfy both matching criteria, are returned to scheduling module.

Purpose: Find resources, which match specific criteria, defined in the JSDL.

Actors: Grid level service on behalf of the user

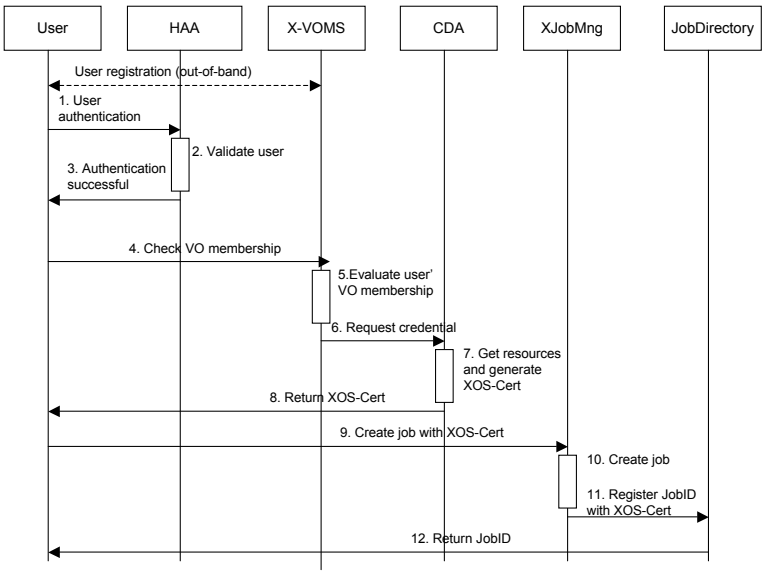

Figure 31: Sequence Diagram showing the interactions between security services and AEM components during the AEM Job Creation Process

Input: XOS-cert, VO name, JSDL

Output: List of matched resources

Pre-conditions:

- XOS-cert has been created and passed to the resource matching procedure

Post-conditions:

- None.

Invariants:

- None.

Security related functionalities:

- **Static information resource matching** is based on functionality, which does not allow complex interaction during the matching process, which results, that at this step the only user specific information checked is VO resource membership.

- **Dynamic information resource matching** on other hand contacts each candidate resource, and checks any remaining resource property, which has to be satisfied. At this step the XOS-cert is presented to the resource, so that the resource can authenticate the legitimacy of the access.

Relationship to use cases/components in other work packages Use case provides services to: WP3.3 jResMatching

Use case requires services from: WP3.2 Node management

Test:

- **Empty VO.** jResMatching service should return empty set of resources, when VO provided by XOS-cert does not contain any resource

- **Invalid ticket XOS-cert.** jResMatching service is presented with invalid ticket. Function should return empty set of resources.

Sequence Diagram: Figure 32 shows the interactions between the security components and AEM components during the AEM Resource Matching process.

5.6.3 Resource Negotiation

Resource negotiation is a protocol performed by jScheduler when submitting a job to a resource. When a suitable resource for a job is found, there is no guaranty, that the resource will be able to perform a job inside the parameters required by user (i.e. SLA level, QoS). Therefore a protocol is envisioned, where the resource and job controller can negotiate the use of resources. After the resource has accepted a job into execution, it holds i.e. in job queue, until the job start time. At this time the job is then send to the execution manager, who will perform the actual job execution (i.e. creating a process)

Purpose: Process of submitting a job to a resource.

Actors: Grid level service on behalf of the user in communication with Resource manager

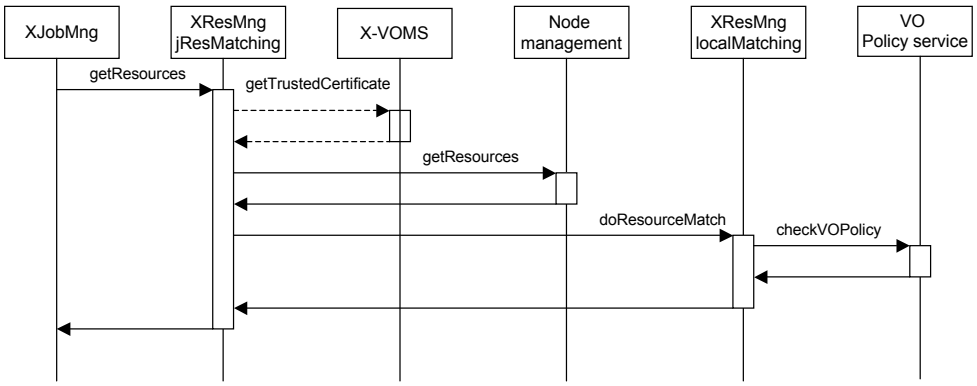

Figure 32: Sequence Diagram showing the interactions between security services and AEM components in the AEM Resource Matching Process

Input: XOS-cert, JSDL

Output: confirmation of success

Pre-conditions:

- XOS-cert has been created and passed to the resource negotiation procedure

Post-conditions:

- None.

Security related functionalities:

- **User authentication and authorization:** with the XOS-cert the negotiation process receives all the user related credentials, which are first checked with the trusted certificate the request validity, and then based on any local policy a resource can deny access to them.
- **SLA:** during the negotiation period a service level agreement - SLA can be reached, which binds both User (job owner), and the Resource owner to honour the agreement reached.

Relationship to use cases/components in other work packages: N/A.

Test:

- **SLA with finish time in past:** Allocation service must not accept such job in execution.
- **Invalid ticket - XOS-cert:** Allocation service is presented with invalid ticket. Negotiation should not get accepted.

Sequence Diagram: Figure 33 shows the interactions between the security components and AEM components during security services for AEM resource negotiation process.

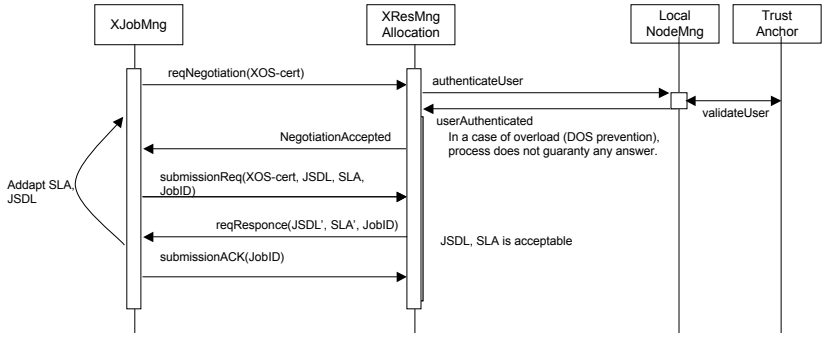

Figure 33: Sequence Diagram showing the interactions between security services and AEM components during AEM Resource Negotiation Process

5.6.4 Job Execution

The AEM job execution starts after resource has been allocated on a node. The local job execution component (i.e. XExecMng) of AEM executes $fork$ command to start a new proces, sets its UID/GID, and runs the command $exec$ to start executing this process. While the process is running on a node, the node-level VO support modules (from WP2.1) monitor and record its resource consumption. When the process is finished, a message will be sent back to XExecMng to signal it to relay the accounting record back to the VO manager. This is to ensure the accountability of resource usage on nodes.

Purpose: Ensure resource consumption on each node is accountable by relaying accounting information back to an appropriate VO manager (VOM).

Actors: XResMng

Input: A message from XResMng triggering the spawn of a new job process

Output: Accounting record of a job execution process being relayed back to VOM

Assumption: The communication channel between a node and VOM is mutually authenticated and secure, for example, via SSL. This is to ensure that accounting information is relayed back to the VOM that the job request is originated from and the node is not bogus (i.e. faking the accounting record to charge users who haven't submitted a job).

Pre-conditions:

- Node resource has been allocated to a *valid* user.

Post-conditions:

- The accounting information of a job execution process is correctly relayed to VOM

Security related functionalities:

- **VO accounting:** ensure that an accounting record is correctly associated with a user

Relationship to use cases/components in other work packages Use case provides services to: WP3.3

Use case requires services from: WP3.5 VO accounting server

Test:

- **Invalid user:** The VO accounting service rejects an accounting record because of the global ID presented by the XExecMng is invalid.

Sequence Diagram: Figure 34 shows the interactions between the security components and AEM components during security services for AEM job execution process.

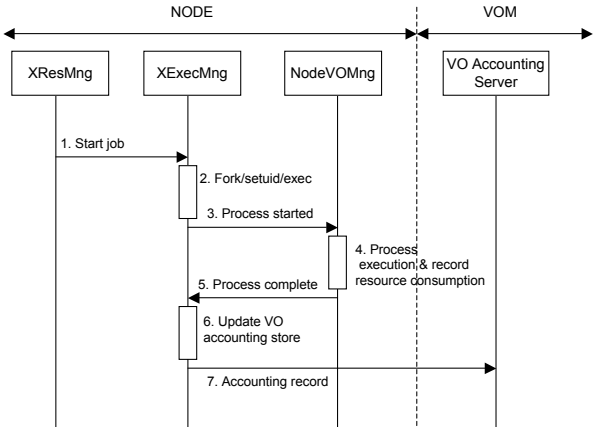

Figure 34: Sequence Diagram showing the interactions between security services and AEM components in the AEM Job Execution Process

6 Open Issues

6.1 Scalability of the Authentication System

One of the major challenges of developing an authentication system for a large-scale Grid environment is its scalability. To XtreemOS, there is no exception. The authentication system presented in 4.2.4 is designed with the aim of easy integration with the existing security infrastructure that has been well operated in large corporate organizations. The model that the system is based upon is flexible in that it can still accommodate the more general situation that there isn't any existing infrastructure to bootstrap the trust from.

The protocol presented in Section 4.2.4 basically generating a short-lived public key for a user and then using that key to establish a shared secret between the user and each online service. The classic Diffie-Hellman protocol is leveraged to achieved both goals in one single protocol. Because of this reason, no additional key agreement protocol is needed after the mutual authentication process is performed. One of the alternative is to give users the flexibility of choosing any public key cryptography algorithms (e.g. RSA). But with this alternative approach, there would be extra key establishment steps involved. Therefore, it seems that our proposed protocol should have advantage over other types of public key algorithms. However, the exact advantage (i.e. capable of coping more concurrent clients) of our protocol is yet to quantify.

On the other hand, our protocol relies on an online CDA service (operating on a per-VO basis). In contrast to conventional PKI who is typically offline, the benefit of making credential services online is evident. It offers the advantage of providing short-lived certificates, which effectively reduce the impact of compro-

mised certificates. Thus, the classic certificate revocation problem is mitigated. The downside of this approach is that the online availability of this service becomes crucial to the success of VO operations. Because of this reason, we plan to explore the possibilities of using the highly available and scalable services currently being developed in WP3.2 to improve the robustness of our services.

There are open questions remaining, they are:

- is the proposed model a complete new VO model to the existing ones (e.g. VOMS) using in Grid middleware? If so, what are the key differences? Does our solution provide a satisfactory solution to the new model?
- does this strategy really gives us the better scalability than its a pure PKI based authentication system? Note that the proposed model itself can cope with both symmetric key based and public key based authentication protocols.
- will it imply significant changes to the current practice of Grid authentication? If so, will this strategy be worthwhile for XtreemOS as a whole?

6.2 Flexibility vs Complexity of Authorisation

The major open question for any authorisation system is always: how open should the policy and attribute types be? On one hand, especially in a dynamic, heterogenous environment like Grid Computing, there is a need for flexibility and configurability, which leads to requirements for specifying generic policies and attributes. However, a flexibility and a highly generic architectural specification shift complexities to implementation and management. There is therefore a need to agree on the appropriate level of flexibility with respect to the types of attributes and constraints used to make policy-based authorisation decisions.

6.3 Technicalities of using Virtualization for Isolation

Isolation has been included as a security requirement since the days of large-scale, timesharing computer architectures, where traditional multiprogramming techniques appeared to have their limitations. The theory of virtualization for isolation has been proposed since the late 1960's, but there was never sufficient computational resources available for it to be realized on the scale that it is being realized today. In any event there are still some major decisions to be made concerning if to use this as our sole isolation strategy:

1. will the platforms supporting XtreemOS have the capacity to handle virtualization?

2. if selected, what approach to virtualization is most suitable - full vs. paravirtualization?

3. do the range of applications support making it a core service offered by XtreemOS security, or should this be left as an extension?

4. is it necessary to support all 4 aspects of isolation - attribute, object, interface and process? Note that if there is no need for object and process isolation, virtualization techniques could be an overkill as a solution for isolation.

6.4 Resource Sharing across Multiple VOs

As briefly discussed in Section 3.3, resource sharing across VOs is of vital importance to the XtreemOS project as a whole. The key challenges are to (a) ensure efficient resource management and utilization; and (b) deliver a certain level of quality assurance despite the concurrent presence of users who may or may not belong to the same VO. Effectively, this can be treated as an isolation problem. In a Grid OS, such as XtreemOS, isolations can coexist on many different levels.

The very basic level of isolation can be achieved by the use of UIDs/GIDs for different users[28]. However, WP3.5 only provides the necessary information, such as VO attributes of a user, to allow other aspects of the system to deal with the handling of basic isolation in XtreemOS. Currently, this functionality is currently being provided by WP2.1, who uses the VO attributes[29], to create UIDs/GIDs for users.

An more advanced level of isolation can possibly be achieved by using virtualization techniques. Due to the time constraint, such techniques and in-depth investigation of the implications of sharing under isolation have just been marginally mentioned in the current deliverable. We plan to explore them in the next edition of this specification.

[28] Fundamentally, Linux has three basic elements: processes, files, and I/O channels. All these elements are associated with UIDs/GIDs. That is, the creation, modification, deletion of all of these are based on UIDs/GIDs. So long as two users are under different UIDs, their processes, files, I/O channels are isolated. Well, they may be able to see the files etc. of the others, but they cannot access/modify/delete the files/processes/I/Os of each other. And, files/directories are somehow special because it has 9-bit access control associated with them.

[29] Currently, we are still in a process of finalizing a list of VO attributes that will be embedded in a XOS-Cert. We believe, this is a cross-WPs issue that may have a great impact on many aspects of the design of XtreemOS.

7 Appendix - XtreemOS Certificate (XOS-Cert)

This appendix describes the format and content of a XOS Certificate (XOS-Cert). To the best of our knowledge, this information is correct at the time of writing. However, the exact content that goes into the certificate is subject to changes. To a large extend, we perceive such changes be necessary and healthy as they would allow us room to accommodate future extensions to our protocol. Meanwhile, these changes may also be necessary to reflect our ongoing active interactions with other WPs.

Overall, in comparison, the format of XOS-Cert is expected to be much stable than its content.

7.1 XOS-Cert - Format and the Certificate as a Whole

As pointed out in Section 4.1, the format of a XOS-Cert is that of a X.509 v.3 public key infrastructure certificate [6]. XOS-Certs are short-lived. Compared with long-term public key certificates, the lifetime of XOS-Cert is much shorter. Typically, the validation period of XOS-Certs is in the matter of days, rather than years. The XOS-Cert is signed by a VO manager.

As with VOMS certificates, XOS-Cert also contains VO attributes, which are defined in a X-VOMS database. In XtreemOS, VO attributes are issued by the VO manager and signed by the VO manager as well. For compatibility reasons, signed VO attributes are embedded in the an attribute certificate [7] as VOMS attribute certificates. However, it should be pointed out that the attributes in a XOS-Cert is potentially much richer than those in a VOMS certificate. Similar to VOMS, the XOS-Cert's attribute certificate is stored in the standard extension part of a XOS-Cert.

Here is the format of a XOS-Cert:

- version (v3)
- serial number
- signature (signing algorithm using the VO manager)
  - algorithm identifier
  - parameters
- issuer's name (i.e. the VO manager's DN)
- validity period
  - not before date/time

  - not after date/time

- subject's name (i.e. the user's globally unique ID)
- subject's public key info
  - algorithm identifier (aDH - authentication using Diffie-Hellman algorithm)
  - parameters (i.e. g, and n)
  - public key value
- issuer unique identifier (optional)
- subject unique identifier (optional)
- extensions (XtreemOS attribute certificate)
- signature (of the entire certificate, including extensions)

7.2 XOS-Cert - the Attribute Certificate Part

Since some parts of XOS-Cert have been described in the previous section, this section focuses on describing the content of the XOS attribute certificate. Here is the format and the attribute certificate part of the XOS-Cert.

- version
- holder (corresponding to the subject name in the XOS-Cert)
- issuer (corresponding to the issuer name in the XOS-Cert)
- signature (signing algorithm using the VO manager)
- serial number
- validity period
- authorization attributes
- extensions (XtreemOS VO attributes)

Here is a list of VO attributes that may go into the extension part of the XOS attribute certificate.

- GlobalPrimaryUserName

- GlobalPrimaryVOName
- GlobalPrimaryGroupName
- GlobalPrimaryRoleName
- a list of GlobalSecondaryUserNames
- a list of GlobalSecondaryVONames
- a list of GlobalSecondaryGroupNames
- a list of GlobalSecondaryRoleNames
- a list of extra GlobalUserNames, GlobalVONames, GlobalGroupNames, and GlobalRoleNames

References

[1] XtreemOS Consortium. *D3.2.1 - Design of an Infrastructure for highly-available and scalable Grid services*, December 2006.

[2] XtreemOS Consortium. *D3.3.1 - Requirements and specification of XtreemOS services for application execution management*, December 2006.

[3] XtreemOS Consortium. *D3.5.2 - Security Requirements for a Grid-based OS*, December 2006.

[4] D. Engert, L. Pearlman, M. Thompson, S. Tuecke, and V. Welch. Rfc 3820 - internet x.509 public key infrastructure (pki) proxy certificate profile. 2004.

[5] W. Diffie and M. E. Hellman. *New Directions in Cryptography*. IEEE Transactions on Information Theory IT-22 (6): 644-654 Nov. 1976.

[6] R. Housley, W. Polk, W. Ford, and D. Solo, *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*, RFC 3280, April 2002.

[7] S. Farrell and R. Housley, *RFC 3281: An Internet Attribute Certificate Profile for Authorization*. April 2002.