Project no. IST-033576

XtreemOS

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

Third Specification of Security and VO Services

D3.5.11

Due date of deliverable: November 30^{th}, 2008
Actual submission date: January 15^{th}, 2009

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

Type: Deliverable
WP number: WP3.5
Task number: T3.5.2/T3.5.3

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

Version 1.0 / Last edited by Erica Y. Yang / January 14^{th}, 2009

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)	$\surd$

Revision history:

Version	Date	Authors	Institution	Section affected, comments
0.01	03/10/08	Erica Y. Yang	STFC	draft outline
0.02	23/10/08	Aleš Černivec	XLAB	VOPS
0.03	23/10/08	Matej Artač	XLAB	RCA
0.1	12/11/08	Erica Y. Yang	STFC	background
0.2	03/12/08	Erica Y. Yang	STFC	VO management
0.3	08/12/08	Erica Y. Yang	STFC	user management
0.4	10/12/08	Erica Y. Yang	STFC	trust model
0.5	15/12/08	Erica Y. Yang	STFC	certificate management
0.6	16/12/08	Erica Y. Yang	STFC	XVOMS section
0.7	17/12/08	Erica Y. Yang	STFC	revisited RCA and VOPS sections
0.8	17/12/08	Erica Y. Yang	STFC	introduction, executive summary, conclusions
0.9	06/01/09	Erica Y. Yang	STFC	incoporated Luis's comments
0.9.1	10/01/09	Erica Y. Yang	STFC	Luis's remaining comments: reworked XVOMS, VOPS
0.9.2	12/01/09	Erica Y. Yang	STFC	Thilo comment: rework VOPS upcoming features, and minor typo corrections
0.9.3	12/01/09	Erica Y. Yang	STFC	Thilo comment: rework introduction
0.9.4	12/01/09	Erica Y. Yang	STFC	Thilo comment: rework exec summary
0.9.5	13/01/09	Matej Artač, Aleš Černivec	XLAB	rework VOPS and RCA
0.9.6	13/01/09	Erica Y. Yang	STFC	finishing touches
1.0	14/01/09	Erica Y. Yang	STFC	final

Reviewers:
Thilo Kielmann (VUA) and Luis Pablo Prieto (TID)

Tasks related to this deliverable:

Task No.	Task description	Partners involved°
T3.5.2	Specification of XtreemOS Security Services	STFC*, SAP, XLAB, ICT, ULM, INRIA
T3.5.3	Autonomic Security Policy Management and Enforcement	STFC*, SAP, XLAB, ICT, ULM, INRIA

°This task list may not be equivalent to the list of partners contributing as authors to the deliverable
*Task leader

Contents

Glossary

AEM	Application Execution Management
CDA	Credential Distribution Authority
CA	Certification Authority
GGID	Global Group IDentifier
GUID	Global User IDentifier
GVID	Global VO IDentifier
NLP	Node Level Policy
PDP	Policy Decision Point
PKI	Public Key Infrastructure
RCA	Resource Certification Authority
TCB	Trust Computing Base
VOM	Virtual Organization Management
VOPS	Virtual Organization Policy Service
VOLife	Virtual Organization Lifecycle service
XtreemFS	XtreemOS File System
XVOMS	XtreemOS Virtual Organization Management Service
XOSD	XtreemOS Daemon

List of Figures

List of Tables

Executive Summary

This deliverable is the third edition of the specification of security and VO services for the XtreemOS project. It aims to revisit and consolidate the foundation and development roadmap after the first release of the XtreemOS prototype system.

Although the first release has made some progress towards making security and VOs more managable than conventional Grid middleware systems, the system deployment in our testbed has revealed a few critical issues that we will have to address if the expectation of XtreemOS users (including end users, developers, system adminstrators) is to be met. These issues include a) the complexity of managing and distributing certificates, b) the lack of a clear definition of the trust model underpinning XtreemOS, c) static realisation of VOs in the first release, d) the missing links between the implemented VO and security components (from WPs 2.1 and 3.5), and, last but not least, e) a detailed definition of what a XtreemOS system is composed from.

It should be pointed out that some of these issues, namely a) and c), are not unique to XtreemOS: they also exist in the all the main stream Grid middleware systems. They are more pressing to our project because users will have higher expectations on XtreemOS, a Grid *operating system* featured by its support for VOs and better usability (in terms of ease of use, compact system administration and deployment). This deliverable makes extra efforts to address these issues. We have therefore focused on two important aspects of our work: revisiting the key concepts and design decisions that underly our system and giving an outlook into the future direction of our development.

This deliverable contributes to the state of the art Grid computing research in four different ways. For the first time in the project, we present *the XtreemOS trust model*, which is fundamentally a cross-certified hierarchical PKI trust model. Different from other conventional PKI trust models, our model is established upon the existence of one or more *online* Certification Authorities (CAs) in an XtreemOS system. Second, by leveraging the online CAs, this trust model allows us to simplify, and more importantly, automate the management and distribution of (root) CA certificates in our system, which is often a stumbling block for large scale deployment of PKI based systems. Such automation can significantly enhances the capability of XtreemOS being deployed as a large scale distributed system and reducing the level of manual and/or offline operations involved in configuring widely distributed resources (this is a must with conventional Grid systems). Third, we present a set of key protocols underlying this trust model, describe its relationship with our security and VO services, and outline their connections with the implementation we have delivered. Finally, this document also clarifies and revisits the following fundamental questions to the project: what is a XtreemOS system, what are the entities that a XtreemOS system involves, and what are the

VOs in a XtreemOS system.

As part of the overal system implementation roadmap, this deliverable finally presents a preliminary implementation roadmap of our security and VO services. in the upcoming release. We provide an update of the architectual relationship between these services, and describe a set of advanced features that will appear in the next release.

1 Introduction

This deliverable is the third edition of a specification of security and VO services for the XtreemOS project. It is delivered after the first public release of XtreemOS. After two years into the project, we are now standing at a point where we have designed and developed a range of services that have been integrated into the public release and we have also accumulated a level of understanding of the strength and limitations of the services we delivered.

The previous edition of this deliverable focused on presenting an overall security architecture of XtreemOS, and that has been used as the foundation leading to the the first release. However, the actual experience and knowledge accumulated along the production (especially development, deployment and testing) to the first release has given us many new insights and further understandings into the limitations of implemented services as well as the (hidden) strengths of the XtreemOS security infrastructure.

Therefore, this time, we revisit some of the design decisions, most importantly, trying to understand the cause of the problems we have experienced during the course of in the system deployment of the first release and present solutions on how they can be addressed in the upcoming releases.

1.1 Brief Recap of D3.5.4

The previous edition of this deliverable (i.e. D3.5.4) was published a year ago. It was a preliminary design document aiming to set the foundation for the WP3.5 implementation work in 2008. Over time, especially along with the indepth design and development following the publication of D3.5.4 and throughout year 2008, our understandings on the various topics discussed in D3.5.4 have significantly been advanced and have undergone a continously (still ongoing) evolution. This outcomes of this process has been reflected in the contents being produced for this current deliverable - D3.5.11.

Although D3.5.11 is entitled as ”Third Specification of Security and VO Services”, it actually is more than just a specification. It includes conventional topics which are covered under a specification (i.e. service descriptions and upcoming features). But it also presents the fundamental concepts and design decisions that have evolved since the production of D3.5.4. These concepts, decisions, as well as the rationale behind these decisions are valuable outcomes resulted from extensive involvements in the implementation, deployment, and experimentations led to the first XtreemOS release, especially on the topics related to security and VO management. As they are ”fundamental and representing the latest development”, we see them as important foundations for the upcoming development and releases for the work in the WPs and that in the project as a whole.

D3.5.4 vs. D3.5.11 D3.5.4 included the presentation of the following technical topics:

1. entities for secure VO management
2. XtreemOS Security Architecture
3. technical discussion and evaluation aspects of the security architecture

Expansions Since D3.5.4, topic 1. has evolved significantly. In fact, through a careful examination across the security services delivered in the first release, we have concluded that a revisit and consoliation of the major concepts and design decisions are urgently needed.

This is mainly due to two major limitations with the previous editions of this deliverable (i.e. D3.5.3 and D3.5.4): a) there is a lack of systematic presentation of what the VOs in XtreemOS are and what are their relationships with the XtreemOS system; and b) certificate management, a key foundation to XtreemOS security, has not been clearly described. Addressing limitation a) allows us to understand the rationale behind the deisgn decisions regarding Grid-level and node-level services. Tackling limitation b) paves the way to understand how XtreemOS security infrastructure differs from that provided by conventional Grid middleware and how we can avoid some major limitations (especially in terms of certificate management) delivered in the first release and also present in the conventional Grid middleware systems.

Having answers to both will certainly help readers (including developers, perspective users from XtreemOS consortium and outside) to appreciate and understand the novelty of the security and VO features that will be delivered in the upcoming releases.

Topic 1. in D3.5.4 has been largely expanded in D3.5.11 along three lines:

- `(revised) background materials to XtreemOS VO and security design`: this includes the description of entities in a XtreemOS operating system, credentials for each entity, actors related to our system, and usage scope of each type of credentials.
- `VO management in XtreemOS`: this section examines what is a XtreemOS system and how does the VO and security components relate to this system, and what are the (security & VO)software layers included in this system. This section also helps to understand how XtreemOS differs from a conventional operating system and how do these key differences drive us to key XtreemOS security design decisions.
- `Certificate management in XtreemOS`: this section presents the XtreemOS trust model and explains how this model differs from a conventional PKI trust model. More importantly, it highlights the key limitation

(in terms of security) of certificate management in the first release. It explains why it is possible and how to leverage the new features brought by the XtreemOS trust model to offer novel security features that are not possible with convnentional Grid middleware systems. This section also discusses what features we can borrow from conventional Grid middleware systems and how.

Revisions and New Materials The topic 2. - XtreemOS Security Architecture of D3.5.4 has been significantly revised in D3.5.11 (although under a different title). The revisions are two folds. First, we include the description and discussion of security services as they were delivered in the first release. Some of these services (e.g. RCA and VOLife - a web frontend of XVOMS) are simply newly developed after the publication of D3.5.4. Second, a consolidated architecture of XVOMS (including three previously separated security components: XVOMS, CDA, VOLife) is presented in D3.5.11. This provides a timely illustration of what are the relationships between these components as they were presented in the first release and will likely continue to be in the upcoming releases.

In addition, the upcoming features of each of the security services (XVOMS, VOPS, and RCA) are described. For each, we also described how these services relate to the services being developed in other WPs.

A Companion Document Regarding topic 3., a separate companion deliverable - D3.5.10 "Security Evaluation" covers the evaluation aspects of this deliverable. More specifically, D3.5.10 complements D3.5.11 by providing a quantitative and qualitative evaluation of the security and VO services delivered in the first release and a discussion of the materials presented in the current deliverable. Please refer to D3.5.10 for the discussion of the design and implementation work conducted by WP3.5.

2 Basic Concepts

Fundamentally, XtreemOS is a distributed operating system aiming to provide a single abstraction of physical hardware and software services offered by a collection of standalone Linux operating systems so that they can function collaboratively to support the utilization of computational and storage resources regardless of the geographical location of users or machines. In order to achieve that, two layers of software, namely *(Grid-wide) system services and (foundation-level) node services*, are *directly integrated* into Linux operating systems (e.g. Linux PC, Linux Single System Image, and embedded Linux distributions) so that such software programs can work *natively* alongside with the existing Linux kernel and system programs to facilitate the coordination among the conventional Linux operating systems.

Figure 1 shows the layered software stacks of a computer system that runs XtreemOS as its operating system. A detailed digram consisting of the major functionalities offered by each layer will be presented at Figure 3. According to [2], an XtreemOS operating system consists of three layers of software, XtreemOS Grid layer, XtreemOS Foundation layer, and standard Linux operating systems, all packaged and integrated to function as a single operating system. The design and development work of the project focuses on former two layers; whilst the packaging work occurs between the XtreemOS Grid/Foundation layers and the standard Linux distributions.

This chapter examines the design principles of the added software layers: system and node services and describes their relationships with the underlying Linux system programs and the applications and tools sitting on top of XtreemOS. The emphasis of this examination focuses on the software stacks, including Grid-wide system services and node-level core services, to support VOs and the enabling security mechanisms.

2.1 Background

A major function of XtreemOS is to hide the complexity of distributed resources dynamically aggregated from cross-domain services providers and ensure the *transparency* of using such a distributed operating system from users. For example, just like with a standalone operating system, once a user is registered with the XtreemOS system, it should be conceptually the same for the user to use resources from any machines that the system is composed of, regardless of whether such resources have been added to the system recently or for a long time. For system administrators, XtreemOS administrative tools should also ensure the managibility of XtreemOS regardless of how many VOs they machines are involved in the past, now, and future. This section looks at a range of concepts that make XtreemOS

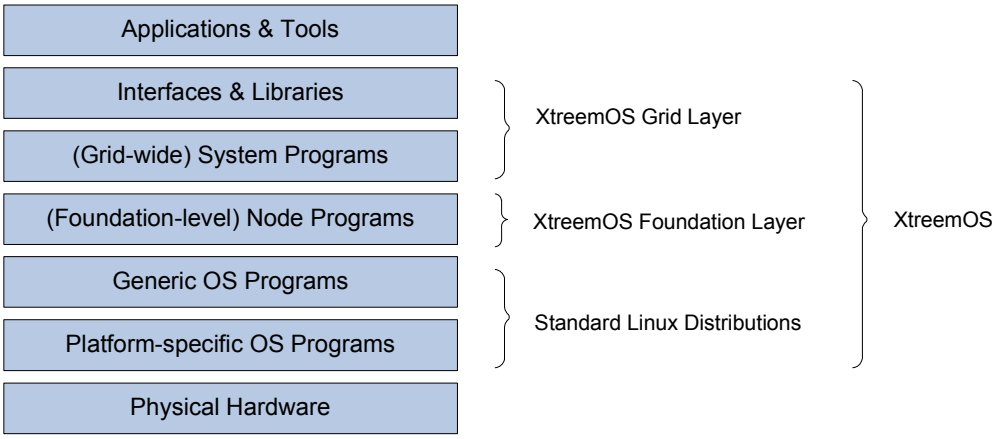

Figure 1: The structure of a computer system that runs XtreemOS.

different from conventional operating systems and how XtreemOS accommodates these concepts in the design of its software stacks.

2.1.1 Entities

One of the key features of XtreemOS is its support for VOs spanning across the entire operating system. This system consists of two types of entities: global level entities, that is, *global entities*, and (conventional) operating system level entities, that is, *local entities*.

Local (OS-level) Entities XtreemOS is built upon conventional (standalone) Linux boxes, which are not aware of global entities. In Linux, there are two types of local (OS) entities: OS users and OS resources (files and processes). Local entities exist in a OS (local) namespace and are identified by local identifiers within a standalone operating system. In Linux, users are identified by a **UID** (user id), files are identified by an **inode** number, which is a data structure which contains information about a file in a Linux/Unix operating system, and processes are identified by a **PID** (process id).

Global Entities Global entities persist in a global namespace and are identified by a global identifier (an attribute of a global entity). Examples of global entities are: users, nodes (machines), services, and VOs. *However, only two types of global entities, users and nodes, are associated with a X.509 public key certificate*

[16]. The X.509 certificate standard was chosen as credential format because it is being used widely in production Grids and there are many mature technologies manipulating certificates.

Other entities (e.g. VOs and services) which also persist in the global namespace and have a global identifier associated with them. But, these entities do not have a certificate associated with them. In fact, we aim to avoid using certificates in the system as much as possible.

Having too many certificates in the system could lead to a system management disaster when it comes to deployment and management of the system. In addition, creating certificates typically involves an off-line (this is the standard practice today) certification authority and often involves humans in the loop (e.g. submitting certification request and installing certificates). This implication hinders our aim to support dynamic VOs, as they are often created in a very short time and are short-lived. Thus, for the above reasons, we decide to reduce the number of certificates in the system as much as possible. Consequently, the other global entities (i.e. VOs and services) are no longer configured with certificates. Their identifiers will appear as attributes of a certificate-attached global entity.

Dealing with Global and Local Entites Global entities are associated with one or more attributes, including a global identifier. A global identifier allows a global entity to be recognize globally across the underlying collection of Linux operating *systems*. However, the concurrent presence of global and local entities naturally leads to two questions.

The first question is how to manage identifiers so that they are globally unique. Without the guarantee of global uniqueness of identifiers, users sharing the same identifier are able to access each other's files/jobs without being detected. Given that users and resources are often originated from different administrative domains, it is not realistic to assume there is a centralized control on the assignment and management of identifiers (this is the assumption in the first release). Further, because there are different validity scopes in XtreemOS (see Section 2.1.4): identifiers need to preserve the uniqueness within each and every valid scope it belongs to during the lifetime of the corresponding scope.

Next, it comes the question of how to manage attributes of a global entity so that the XtreemOS system can perform consistent access control across the board. This can be a tricky issue as attributes can be associated with users or machines of varied time period. For instance, because VOs can be created dynamically, a user's association with VOs, especially with dynamic VOs, can vary from time to time, depending on the lifetime of the VOs.

In order to manage the entire XtreemOS system consisting of global and local entities, the key challenges can be summarized as follows. First, XtreemOS

should provide *a set of system-wide (global) services to serve all the Linux boxes* that the system is composed from. The purposes of these global services are to manage global identity, attribute, policy, and VO membership, and provide a global governance of resource usage.

Second, *a new set of node-level system services* should also be provided to work alongside the existing native OS services and to coordinate with the global services. To that end, these OS-level services are: a) to provide the *bi-directional mappings* between a global user and a local user, a global resource (e.g. file) and a local resource (e.g. file); b) to allow OS-level services to recognize and make use of global attributes in its handling of local resources and providing varied quality of services guarantees, and c) to corporately work with global services to enforce policies to govern the usage of resources on a global local scale.

2.1.2 Credentials

Apart from the information (e.g. DN) embedded in its public key certificate, a global entity basically has two types of credentials: identifiers and attributes. Both are held in its attribute certificates. Put it simply, a credential is a piece of information about a subject; whilst a certificate is a mechanism to carry credentials.

Identifiers In XtreemOS, an **identity** is a piece of human-readable information about an entity; whilst an **identifier** is a computer processable piece of information about an entity. Specifically, there are four types of identifiers in a XtreemOS system: Global User Identifier (GUID), Global Virtual organization Identifier (GVID), Global Group Identifier (GGID) and Global Node Identifier (GNID). Users and nodes are associated with a X.509 public key certificate and their identity is represented by the Distinguished Name (DN) field in the certificate. In other words, the DN field in a certificate is the identity of an entity and this entity can be users or nodes.

Distinguished Name (DN) In a PKI world, the DN in a certificate is the unique way to identify an entity. A DN field consists of the following information:

- Country
- Locality (or city)
- State (or province)
- Organization
- Organization Unit
- Common Name

Once the DN is assigned to a global entity, the binding between the DN and the entity should remain the same *throughout the lifetime of the entity*. For instance, the DN of a user's or a node's certificate should remain the same regardless how many VOs they register with and how many different attributes they may have in each VO. However, in XtreemOS, the DN is not being used by the distributed operating system as an identifier to handle global entities' identity. The following paragraph explains the rationale behind this decision. Section 2.4.3 describes how the DN field is determined in a XtreemOS system.

Why Not Use DN as a Global Identifier? Although the DN field in a certificate is a useful source of information about an entity and it is unique across the system, it is often descriptive and generally too long to be used as a unique identifier for the Linux operating system: conventional operating systems are not designed to recognize long descriptive names. In the traditional Linux (e.g. libc-5), the POSIX compliant usernames is only 8-byte long. In the recent versions of Linux (libc-6), the length of Linux usernames has been extended to 32 bytes [14].

Format Format wise, a XtreemOS identity certificate, called XOS-Cert, has no difference to a standard X509 identity certificate [16].

Attributes Global entities can have one or more attribute certificates associated with them. An *attribute* is a property that *a global entity* can have. Attributes can be bounded to an entity with a specified lifetime and a scope of valid usage(s). Sometimes, such usage(s) are referred as policies. Typically, these policies restrict the context where one (or more) attribute can be used with a global entity.

Instead of using the full DN field directly as a global identifier for a global entity, the XtreemOS mapping process (see [3] for details) uses *a 32-byte alpha-numeric identifier* to identify users and map them into operating system level usernames. The format of a global identifier is comprised of two parts (for additional details, see Section 2.4.4):

```
/<16-byte credential issuer's IP address>/
<16-byte host-wide unique ID>
```

The separators (i.e. forward slashes and dashes) are for illustration purpose: they are not being counted as part of the identifier. An example of such is: /130.246.76.73/ff841bf7-d6c543d7-9ad7-d54ad8c6cb91. This example uses a hypothetical IPv4 address (i.e. 130.246.76.73), which only occupies 4 bytes of the space. However, because 16 bytes are allocated, the first half of the identifier can accommodate 16-byte IPv6 addresses. This is perhaps needed for XtreemOS as the adoptation of IPv6 protocols in WP3.2's proposed design and development.

Readers might wonder why don't we use the more descriptive DNS hostnames to replace the IP addresses. The major reason is that DNS hostnames can be up to 255 bytes long, which makes it practically not feasible for being used as a candidte of Linux usernames.

The remaining 16 bytes are for host-wide unique IDs (i.e. ff841bf7-d6c543d7-9ad7-d54ad8c6cb91). Because it is unique within the issuing host, the host administrator can choose how this is implemented locally. In the first release of XtreemOS, we have chosen the widely used the UUID standard [14] to implement the host-wide unique IDs.

To summarize, the DN of a global entity is the identity of the entity in a PKI world but not the global identifier being used by the XtreemOS distributed operating system to recognize a global entity. In our system, the global identifier is being treated as *an attribute* of the entity.

Format Attributes can be stored as (non-standard) extensions in a X.509 identity certificate, thus forming an integral part of a certificate. However, in practice, this solution could restrict the usage of certificates because attributes in the extensions may have different validity period and scope than those of the identity of an entity (as an entity's identity has a lifetime association with the lifetime of the entity). Therefore, in XtreemOS, the attributes of a global entity are distributed via *attribute certificate* [6]. An attribute certificate is a mechanism storing a set of attributes of the same validity period of a global entity. An attribute certificate is associated with an identity certificate through sharing of a common DN field. In practice, a global entity may have multiple attribute certificates, each associated with its identity certificate with a varied set of attributes.

Types of Attributes There are two types of *attributes* that users can have in his/her attribute certificate.

1. Identifiers
    - **Global User IDentifier (GUID)**: a user's global identifier (unique within a XtreemOS system) Typically, a user is associated with one GUID in the system. However, there is at least one exceptional case (described in Section 2.4.4).
    - **Global Vo IDentifier (GVID)**: a user's VO association (unique within a XtreemOS system) A user can be associated with multiple GVIDs.
    - **Global Group IDentifier (GGID)**: a user's global group association (unique within a XtreemOS system) A user can be associated with multiple GVIDs.
2. VO Attributes

- **Group**: a user's group association within a VO (unique within a VO)
- **Role**: a user's role in a group (unique within a VO's group or unique within a VO: the former means a *hierarchical structure* between groups and roles within a VO; while the latter indicates a *flat structure* between groups and roles within a VO.)
- **Capability**: a list of capabilities that a user has over certain objects (unique within a VO)

Machine attribute certificates can contain the following attributes:

- **Global Node IDentifier (GNID)**: a machine/node's global identifier (unique within an XtreemOS system)
- **Global Vo IDentifier (GVID)**: a node's VO association (unique within an XtreemOS system)
- **Service**: services running on the node for a VO

2.1.3 Actors

So far, we have discussed how the system assigns identity and attributes to global entities. However, what we have not discussed are the actors involved in the process. So, who are the people involved in generating, managing, distributing, using, and removing the credentials? Who run the services? More specifically, the following questions are formulated to be investigated in this section:

- Who manages the lifecycle of the identities and attributes?
- Who uses the attributes?
- Who runs the services?

Types of Actors A XtreemOS system consists of: VOs, system services, and resource nodes. Subsequently, three actors are involved in managing a XtreemOS system: resource owners, system administrators, and VO owners. In addition, we have users who interact with the XtreemOS system by using services from the system. Here is a brief summary of the type of actors that a XtreemOS system involves:

- **users**, *including software (i.e. other systems) and humans*, who interact with a XtreemOS system by receiving services from the system. When a user is a software system, the user interacts with the XtreemOS system through Application Programming Interfaces (APIs). When a user is a human being, s/he interacts with the XtreemOS system through interactive interfaces, such as web interfaces.

- **VO owners** typically are humans, but can also be software programs, who create, manage and terminate a VO. The creator of a VO becomes the owner of the VO. A VO owner can also grant its ownership privileges to others. So, there can be more than one owner of a VO.
- **resource administrators (resource admins)**, either a system administrator from a participating organization (managing a group of machines from that organization) or an individual (managing his/her own machine). There can be more than one resource administrator from an administrative domain. This actor manages resource provision to the system, for example, by setting resource usage policies with respect to each VO it participates in.
- **system administrators (system admins)**, *a group of individuals* who govern the global operations of the entire XtreemOS system by operating Grid-wide system services for the system. There can be more than one system administrators in a system: they manage the system collectively. However, it should be emphasised that it is resource administrators who have the final say on the resource provision of their machines.

Responsibilities of the Actors

Users A *VO user* has the following responsibilities:

1. configure his/her machine to allow interactions with XtreemOS Grid-wide system services.
2. register with the system as a user: this also includes registration with VOs.

VO Owners A VO owner is responsible for the following tasks:

1. create a VO
2. determine what attributes (e.g. role and group) are supported in the VO, and later, if necessary, modify these attributes
3. register a user into the VO and assign appropriate attributes accordingly upon the request from the user
4. modify (add, change) the attributes (e.g. role) of users, when needed.
5. register/remove resources upon the request from a resource administrator of a participating organization
6. remove users and resources from a VO
7. specify and managing (add, remove, change) VO policies to govern coordinated use of resources among users
8. report VO-wide resource usage
9. terminate the VO

Resource Administrators The responsibilities of a resource administrator are as follows:

1. obtain a specification of the resource (physical and abstract resources) provision
2. decide to which VOs this resource is (or is not) being contributed
3. register/de-register with the VO owner (or its delegates) with the specification
4. specify node policies
5. configure nodes according to the instructions from VO owners

System Administrators A system administrator has the following responsibilities:

1. configure and run Grid-wide system services, including setting up policies and certificates for these services.

Who creates the identities and identifiers? We have discussed the format and requirements of identities and attributes (except the VO attributes: groups, roles, and capabilities, they have to be set by VO owners.) It should be pointed out that it is not the actors described above, but the XtreemOS Grid-wide system services (software programs), who are responsible for creating the identities and attributes. These programs implement the guidelines described in Section 2.4 to set the values.

2.1.4 Usage Scopes

Identifiers and attributes for global entities can be used within different scopes. In XtreemOS, there are four usage scopes where identifiers and attributes of global entities can be used (processed by XtreemOS node services). They are:

- node scope: only use within a standalone XtreemOS Linux box
- VO scope: only use within all the nodes belonging to a VO
- system (global) scope: use throughout an XtreemOS system
- XtreemFS scope: only use within XtreemFS

These scopes are only valid within the defined context. The identifiers and attributes become meaningless when the context is changed. For example, even if two VOs, VO_1 and VO_2, both have the concept of groups, the groups in VO_1 are invalid in VO_2, and vice versa. Table 1 summarises the relationship between the attributes and usage scopes.

Attribute	**Created by(when)**	**Processed by what services**	**Validity**
GVID	a Grid-wide system service called XVOMS (when a VO is created)	node	VO
GUID	a Grid-wide system service called XVOMS (when a user is successfully registered with the system)	XtreemFS, node	system
GGID	a Grid-wide system service called XVOMS (when a global group is successfully created in the system)	XtreemFS	system
Group(s) in a VO	VO owner (when a VO is created or modified)	node+VO	VO
Role in a group of a VO	VO owner (when a VO is created or modified)	node+VO	VO
Capability in a VO	VO owner (when a VO is created or modified)	node+VO	VO
GNID	a Grid-wide system service called RCA (when a resource node is successfully registered in the system)	node	system
Service	resource admin (when a service is added to a resource node)	node	system

Table 1: Usage Scopes of the Attributes

A user and a node can be associated with more than one VO. When more than one GVIDs are associated with a user or a node, the first of such is the primary GVID and the subsequent ones are the secondary GVIDs. Similarly, a user can associate with multiple GGIDs. The first GGID is the primary GGID, and the subsequent ones will be treated as secondary GGIDs.

2.2 VO Management

XtreemOS is not a conventional distributed operating system. It emphasis on the support of *large-scale dynamic collaboration and coordination* among heterogenous resource providers who are not subject to centralized administrative control. One key concept underpinning such systems is referred as *Virtual Organization (VO)* [7] in the Grid community. Let us first look at what is a XtreemOS system, then proceed to describe the VOs that are building in the system.

2.2.1 A XtreemOS System

A **XtreemOS system** is an entity that consists of three parts: a set of **machines** from one or more participants (e.g. individuals and/or physical organizations) offering resources through a set of foundation-level node services (abbr. node services), a set of **Grid-wide system services** (abbr. system services), and a set of **VOs** to support cross-machine resource sharing and logical isolation of resource usage within the system. Typically, neither the particpants nor the resources they managed are subject to centralized administrative control. The resources include computation and storage resources, software services, and data. The system services and node services coordinate the machines to make them appear like a single machine. The VOs consist a subset (or a complete set) of the machines in the system to facilitate resource sharing among the participants. Figure 2 illustrates a XtreemOS system, which consists of three parts: VOs, system services, and machines.

A **user** of a XtreemOS system is defined as another system, including *humans* or separated autonomous *software systems* that interacts with the current system through a set of well-defined *interfaces*. If a user is a human, the interface is a *human-computer interface*, such as a GUI. If not, the interface is a set of *Application Programming Interfaces* (APIs) allowing programs from the other system to use the resources within the XtreemOS system.

As illustrated in the figure, the machines, system services, and VOs are managed by different people from different administrative domains: the machines are owned by resource owners and managed by resource administrators, the system services are administered by system administrators, and the VOs are owned and

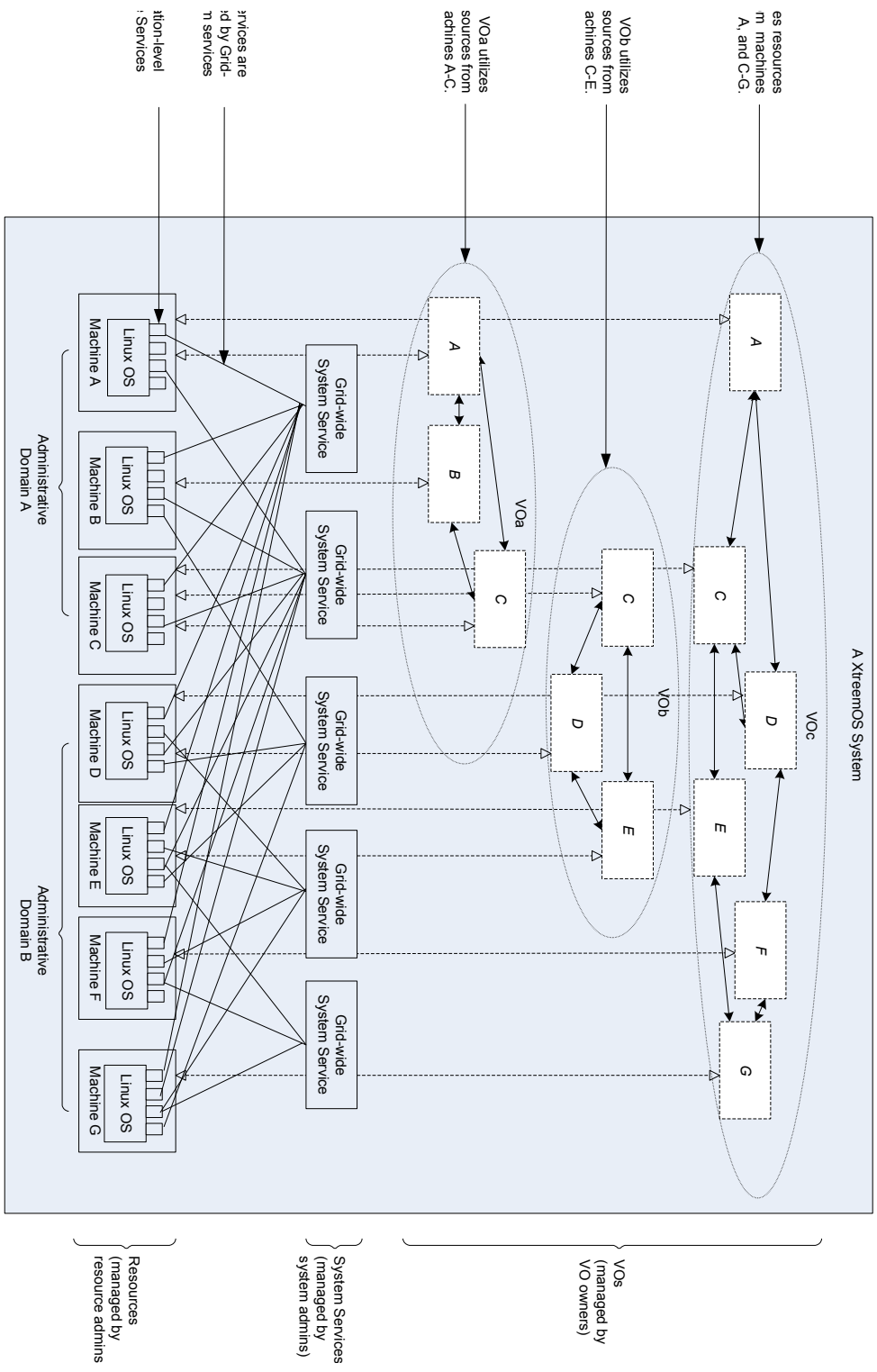

Figure 2: Components of a XtreemOS system.

managed by VO owners. There is *no central management authority* to govern the operation of the entire system.

The VOs can overlap with each other. In the illustration, three VOs, VO_a, VO_b, VO_c, share the physical infrastructure via the system. All three VOs share machine C; VO_a and VO_b share machine C; VO_b and VO_c share machines C, D, and E; VO_a and VO_c share machines A and C. Because of the infrastructure are managed by two distinct administrative domains, VO_b and VO_c are actually sharing the resources across the domains seamlessly via the XtreemOS system.

2.2.2 VOs in XtreemOS

A **Virtual Organization (VO)** is typically a temporary or permanent coalition of *geographically dispersed and autonomously governed (independent)* **participants**, including individuals and/or organizations, who agree to share **resources** so that users can consume the resources in the system to fulfill their tasks (e.g. running jobs, sharing applications, accessing data).

VOs's Properties VOs are typically characterized by the following properties:

1. the resources are geographically *distributed*;
2. the resources are *autonomously* governed by the participants themselves;
3. the lifetime of the collaborations can be *short-termed or long-term*; and
4. the coalition among participants the can be *static or dynamic*.

Without these characteristics, we can use conventional solutions from decades of distributed systems research to address the resource sharing and collaboration problem. So, let us briefly elaborate what these properties imply in the design of our VO approach.

Property 1) means that when running a VO-based system (such as XtreemOS), one has to take firewall issues into account because the resources do not normally reside in a single administrative domains. From a practical point of view, conventional network protocols, such as TCP/IP, may not be the best protocol for such a large scale distributed system because non-conventional ports need to be opened on organizational firewalls to allow traffic rather than TCP/IP.

Conventional distributed systems are generally under a single administration. But, a VO-based distributed system aggregates resources from *independent and autonomous* participants, among which, there may not be any administrative relationship in existence. Therefore, when designing a XtreemOS system, it is important to accommodate the flexibility of allowing participants to choose to whom they would like to share their resources and how. Consequently, forcing all the participants to trust a single centralized authority to manage the entire system not only does not scale but could also creates a single point of failure in the system.

For the short-term VOs mentioned in Property 3), it is mandatory that they can be managed and operated by the system to the same flexibility as the long-term ones. However, the challenge here is that a scalable approach has to be adopted to allow the system to accommodate such VOs because of their short lifetime.

Now, let us discuss Property 4). A static coalition means that, through the lifetime of a VO, the membership of participants are fixed, or type, quality, and/or quantity of the resources offered by the participants are fixed, or both. A dynamic coalition is the opposition of a static coalition, meaning that, through the lifetime of a VO, the membership of participants can be ever changing (participants are allowed to come and go as wish), or the type, quality, and/or quantity of the resources offered by the participants are not fixed (participants are allowed to change its offering of resources as wish), or both.

Challenges in Designing a VO-based System These important, yet closely related, properties differentiate a *VO-based distributed system*, such as XtreemOS, from conventional distributed system, consequently leading to a range of fundamental challenges in the design of such a system. These requirements are: a) the system has to support resource federation from different administrative domains, meaning that there is *no centralized authority* over the ownership of the resources; and b) the system has to support addition and removal of resources while maintaining the overall transparency of such operations from users and shielding the complexity of managing such operations from system and resource administrators. As there is not a single authority over all the resources, it is unrealistic to assume that the system will be governed by a single administrative authority. Therefore, requirement a) essentially means that we have to support a decentralized management infrastructure allowing the concurrent presence of multiple autonomously managed authorities in the system.

The quality of the solutions to requirement b) largely depends on the degree of support for dynamic VOs in the system. A dynamic VO is defined as follows: a dynamic VO is a VO that is either created at runtime from the resources provided by the system, to fulfill the tasks specified by the users.

Although forming from aggregated distributed resources, XtreemOS needs to behave like a single computer to users, who should be shielded, as much as possible, from the complexity of managing and using such resources. This has been the aim of conventional distributed operating systems, which is sometimes referred as meta-computers, such as Legion [8]. However, to support dynamic collaboration and coordination, XtreemOS has to support the *establishment and termination* of a fast and often short-lived relationship among resource providers. This breaks an important assumption that conventional operating systems are based upon. Thus, it is a non-trivial task for XtreemOS to manage distributed resources transparently.

2.2.3 VO Software Layers

XtreemOS is a distributed operating system to support VOs in the next generation Grids. VO-related software plays an important role in the entire system software stacks. Figure 3 focuses on the VO-related software layers and depicts their relationship with the entire XtreemOS software stacks. This figure is a detailed expansion of the same figure presented in Figure 1.

Above the physical hardware, there are six layers of software in the stack. They are:

1. Applications & tools: software developed on top of the XtreemOS operating system.
2. Interface & libraries: APIs and system software libraries from the XtreemOS system.
3. Grid-wide system programs/services: global services operated by system administrators.
4. Foundation-level node programs/services: node-wide services operated by resource administrators managing the resource node.
5. Conventional OS system programs/services: common OS services, such as process, file, and memory management services, running at a conventional Linux OS.
6. Linux distribution: typical Linux distributions, including Linux PC, cluster, and mobile distributions, that XtreemOS OS targets.

As illustrated, Figure 3 emphasis on VO and security related functionalities, which are classified into two categories:

- Grid-wide system programs, which provides functionalities related to operating the distributed operating system globally.
- node-level programs, which provides functionalities operated alongside with the native Linux system programs to support the distributed operating system.

Grid-wide System Functionalities A range of Grid-wide VO-related system functionalities, independent of those provided by standalone Linux boxes, are needed to coordinate the standalone Linux machines. These system functionalities are classified into the following categories:

- Identity management: managing globally unique identifiers for the global entities so that they can be uniquely identified uniformly across the entire system regardless of the fact that nodes could be constantly added and removed from the system.

Figure 3: How VO and security components fit into the overall XtreemOS software stacks.

- Attribute management: managing attributes for the global entities so that resource administrators can enforce policies based on these attributes discretionarily.
- VO lifecycle management: managing the three phases of VO lifecycle: creation, evolution and termination to make sure that VOs can be recognized and handled regardless of their lifetime (short-lived or long-termed).
- VO membership management: managing the VO membership of users and resources so that (nodes and global) services can enforce VO-based access control.
- VO policy management: managing and enforcing VO-wide policies on resource usage.

Node-level Functionalities The node-level VO-related functionalities added to a conventional Linux box to allow it behave as part of a XtreemOS distributed operating system are:

- Identity mapping: dealing with the logical mapping from a global identifier to a local entity. Note that this process varies on individual nodes depending on its local strategy and policies.
- (Bidirectional) name resolution: providing bidirectional name resolution between a global identifier and a local identifier. This functionality is often used by node services before it talks to Gird-wide system services.
- Authentication: authenticating global entities based on the credentials issued by Grid-wide system services.
- Authorization: authorizing the access from a global entity based on the credentials issued by Grid-wide system services.
- Credential management: managing credentials for global entities in a node so that credentials cannot be accessed by authorized entities.
- Node policy management: managing node policies on VO-based resource usage, identity mapping strategies, trust relationship with VO management authorities, credential issuing authorities.
- Session Management: managing sessions for global entities.

2.2.4 VOHost: a VO Hosting System

All the above functionalities are implemented by the services of **VOHost**, a VO hosting sub-system, an integral part of the XtreemOS system. VOHost consists of two parts: a) Grid-level services that implement the Grid-wide system functionalities described above; and b) node-level services that implement the node-level functionalities mentioned previously. Collectively, these services provide infrastructural support to enable users to access resources in a secure, accountable, and

coordinated manner in a multi-VO distributed computing environment. VOHost differs from existing VO management tools by supporting

- secure VO and credential management for supporting diverse VO models.
- *online* credential distribution to users and resource nodes.
- resource certification to allow dynamic resource admission to, and exclusion from, a VO.
- flexible support of various operational features of VOs (e.g. cross-VO sharing, dynamic VOs).
- VO-level policy management to govern resource utilization within a VO.

A VOHost system can host multiple VOs created statically or dynamically (created at runtime). It is a system jointly operated by three actors: VO owners, system administrators, and resource administrators. For a description of these actors, please refer to Section 2.1.3. A user can register with multiple VOs hosted by one VOHost system. Within a VOHost system, multiple independently administered services, offering the same collection of functionalities, can coexist to manage the lifecycle of different VOs.

Grid-wide VO Services We have designed a range of services to implement the Grid-wide system functionalities described above. These services are XVOMS, VOPS, and RCA. The first version of their implementation are incorporated in the first release of XtreemOS. The design of these services, except VOLife and RCA, has been presented in the previous edition of this specification. Therefore, to be self-contained, we shall briefly summarize the major functionalities of these services here.

- *XtreemOS VO membership service (XVOMS) - managed by system admin and VO owner:* this service consists of five software components: registration manager, credential distribution authority, VO manager, XVOMS native interface, and XVOMS web interface (i.e. VOLife). XVOMS follows the rules described in Section 2.1 to specify identifiers and attributes of the global entities.
  - `registration manager`: allows users to register themselves with the system and resource administrators to register resources with the system.
  - `VO manager`: allows VO owners to register, modify, and remove VO members of the VOs they own, as well as manage VO attributes.
  - `Credential Distribution Authority (CDA)`: issues security tokens, in the form of X.509 certificates, to end users. Such a token binds a user's DN to a set of credentials (e.g. VO attributes, VO identifiers, and other XtreemOS specific attributes - for XtreemFS).

  - XVOMS native interface: provides native APIs to programs who want to access XVOMS functionalities through native APIs.
  - XVOMS web interface - VO Lifecycle service (VOLife): provides a web frontend to the XVOMS functionalities through a browswer interface.

- *Resource Certification Authority (RCA) - managed by resource admin:* this service allows resource administrators to certify the specification of resources before they are provided to VOs. This provides a trustworthy way (via cryptographic means) to allow the resource administrator to check whether resource specification is genuine. Each participating organization (or domain) runs a RCA service, which is managed by the resource administrator from that organization.
- *VO Policy Service (VOPS) - managed by system admin and VO owner:* this service provides policy decision point to allow the policy checking and enforcement at a VO level. VO owners specify policies for the VOs they own and this service acts as a VO-level policy decision and enforcement points *at a VO-level across all the VO nodes in the system*. *On each node*, there are also policy decision and enforcement points for VO policies, which are specified by the resource administrator of that node to control how their resources should be utilized by VOs. Altogether, the VO-level policies and node-level VO policies are called VO policies.

Additional Notes There are two other security services, namely IDS - Identity Service and AttrS - Attribute Service, specified in the previous edition of this specification. Their main purpose is to support *integration and interoperability* with *existing* identity and attribute authorities. As this specification is focusing on specifying and detailing new services, we shall not discuss them hereafter.

In addition, although VOHost offers the above services, when it comes to implementation, each of these services does not have to be implemented as a standalone server.

Node-level VO Services A number of node-level services have been designed and implemented to support node-level VO functionalities. They are:

- *Pluggable Authentication Module (PAM) - managed by resource admin:* allows the flexibility of integrating different authentication schemes (e.g. password, certificate, public key) into a high-level application programming interface (API) so that program implementation can be freed from any specific authentication scheme.

- *Name Service Switch (NSS) - managed by resource admin:* allows Linux configuration databases to be provided other than the standard sources, including local files (for example: `/etc/passwd, /etc/group, /etc/hosts`), LDAP, and others.
- *Kernel Key Retention Services (KKRS) - managed by resource admin:* offers a way to cache authentication data (e.g. cryptographic keys, certificates, cross-domain user mappings) in the kernel so that the kernel can improve the efficiency of accessing these security tokens and also delegate important operations to the user-space processes. It should be noted that, in the first release, the KKRS has now been substituted for a more generic credential store in WP2.1/WP2.3, to take into account systems (like mobile devices) which do not include the KKRS in the kernel.
- *Account Mapping Service (AMS) - managed by resource admin:* local user accounts are allocated dynamically on a resource node to match the actual global users exploiting that node.
- *Node-level Policy Service(NPS) - managed by resource admin:* specifies account mapping rules governing how a Grid user should be mapped to a local user, VO policies specifying which VOs (or what types of VOs) the node will admit, how much node resources (memory, file space) will be allocated to a VO user.

The first three services (PAM, NSS, KKRS) are present in the standard Linux distributions whilst the AMS was developed in XtreemOS WP2.1 to allow flexible creation/deletion of user accounts in Linux OS. Together with the support of these services, the authentication, authorisation, and session management of Grid users are made transparent to the local OS users. For further details of their design, refer to a previous WP2.1 deliverable [3].

2.3 User Management

This section addresses an important issue: how users are managed in XtreemOS, a Grid-wide distributed operating system.

User Account Management XtreemOS is a distributed operating system, which needs a global strategy of managing users so that they can be not only uniformly recognized globally but also be able to cope with the concurrent presence of multiple user management authorities in the system. Like the conventional Linux/Unix based file systems, XtreemFS itself does not manage users for XtreemOS. Instead, it relies on the user management facility provided by the XtreemOS VO Membership Service (XVOMS) to manage users's identity and attributes globally. A user

is identified by a Global User Identifier (**GUID**). Two distinguished GUIDs represent two different users. These GUIDs, like its counterpart UIDs in Linux, provide a unique way to identify users across the entire system. GUIDs are allocated to a user by XVOMS upon its registration. The GUIDs are globally unique even when multiple systems are present.

It should be pointed out that it is possible for a user to register with a XVOMS (thus, giving a GUID) without being associated with any VOs. This is supported in the current XVOMS implementation. A user can register with one XVOMS multiple times or with multiple XVOMS, thus acquiring distinct GUIDs. However, in those cases, the user will be treated as different users, each identified by a GUID.

User Credential Management In addition to GUIDs, Global Virtual Organization Identifiers (**GVIDs**) and Global Group Identifiers (**GGIDs**) are also useful credentials in the system. Similarly, these IDs are also managed by the XVOMS. A GVID, generated when a VO is created, is associated with a user (i.e. the correspon ding GUID), thus effectively making the user the owner or a normal user of the VO. Such an association can occur while the VO is created or after.

GGID is also generated by the XVOMS upon the *creation of a new group in a VO*. A GGID is associated with a user when a user joins a group in a VO. In the current design, the relationship between a VO group and a GGID is one-to-one. This is a VO model being supported in the first XtreemOS release. Putting it simply, a VO model is the type of attributes that a VO supports. For example, the EGEE VO membership service (VOMS) supports four types of vo attributes: role, group, subgroup, and capability. A VO owner can choose which attributes his VO supports.

In the future releases, GGIDs can be associated with other attributes under different VO models. For example, it is possible to associate a GGID with a role of a group within a VO. That is, a GGID is uniquely mapped to a VO role within a VO group of a VO. Depending on the VO model, furthermore, it is even possible to associate a user (i.e. GUID) with different GGIDs, one for each combination of attributes in various VO models.

2.4 Certificate Management

Grid Security Infrastructure (GSI) [15] are by far the most popular security technology adopted by the Grid computing community world-wide as the defacto standard of managing Grid certificates. It is also an important building block for the Virtual Organization Membership Service (VOMS) [1], an attribute management and distribution authority previously developed by the EDG-DataGrid

project[12] and currently maintained by the EGEE project[13].

GSI relies on public key cryptography and the existence of PKI[17] to support mutual authentication, secure communication, delegation of authority, and single sign-on in a Grid system. Before using any GSI-based technologies, the end entities in a Grid, typically including users, machines, services, containers, need to perform two steps: a) obtain a public key certificate from a Certification Authority (CA); and b) install trusted root and intermediate CA certificates in their trust anchor[1]. In today's practice, both steps are performed ***offline***: CA certificates are obtained through *off-the-band channels*, typically via trusted emails or secure web sites and the trust anchor is configured *manually*. As humans are involved extensively in these processes, such operations are error-prone, time-consuming, and complicated for users and system administrators in practice.

XtreemOS' security infrastructure is also based on PKI, requiring global entities (i.e. users and machines) to have end entity certificates[2] so that the system can also support mutual authentication, secure communication, delegation of authority and single sign-on in a large scale distributed environment. However, XtreemOS adopts a novel approach to manage its security infrastructure, especially the certificates, by leveraging the benefits of XtreemOS Grid-wide security services to automate the above steps. Although users and machine administrators still need to perform both steps described above, acquiring certificates are automated through introducing a set of novel and secure **Distributed Certificate Management (DCM)** protocols.

The main aim of this section is to present a set of Distributed Certificate Management (DCM) protocols for managing XtreemOS certificates. To lay down the foundation, we shall first describe a trust model that a XtreemOS system is based upon. We shall then describe a set of secure DCM protocols for obtaining CA and end entity certificates. Along with the protocols, four types of certificates used in the system will be discussed, they are: a user's identitity and attribute certificates, and a machine's identity and attribute certificates.

2.4.1 Trust Model

Trust is defined as the *confidence* that an entity A is able to believe the authenticity of the information presented by another entity B. The notion of trust can be extended to a multi-party scenario, but we are not going to explore in this context. There are many means through which one can establish the trust of the other, for

[1] A trust anchor specifies the key stores that contains trusted root and intermediate CA certificates.

[2] An End Entity Certificate (EEC) is a certificate [9] belonging to a non-CA entity. Taken from: http://www.globus.org/toolkit/docs/3.2/gsi/key/glossary.html

example, through personal recommendations, advertisements, ratings, comments, and reputations from trusted sources.

We are particularly interested in measuring trust through cryptographic means: permitting an entity to use computer programs to cryptographically verify the information given. If everything is all right, the trust on the information is established. Otherwise, if things go wrong, the trust is broken. So, when a verification is successfully performed, we say that A trusts B. Otherwise, we describe that as A does not trust B. Although trust can be measured by different means, leading to levels of trust between entities. Due to the use of cryptographic means, we only have two possible outcomes of the verification: either A trusts B, or not.

In today's production Grid environment, such as TeraGrid[3] or EGEE[4], the common approach to build a trust infrastructure is based on upon PKI technologies, which require all the end entities (non-CA entities) in the system, including users, machines, services, containers, to have public key certificate installed and configured before any users can utilize the Grids securely. In this section, we will present the **XtreemOS Trust Model** with emphasis on how to boostrap the trust relationships between users and machines in a Grid environment based on a variant of the classic PKI Trust model. This new trust model solves the usual deployment problems with offline management and processing of end entity and CA certificates in PKI.

PKI Trust Model As with many production Grid software, we have adopted a classic PKI trust model - a hierarchical PKI based trust model as the base of our security infrastructure to establish trust among global entities in the first release of XtreemOS. Let us first look into PKI trust models. Then, we shall describe a range of challenging deployment problems we have experienced in using the PKI trust model.

PKI Trust Models There are mainly two types of trust models used in a PKI world: a) **a hierarchical trust model**, also called a subordinated hierarchy trust model, where CAs are organized as a root CA and subordinate CAs. The structure of the trust model is like a tree with a single root at the top with branches and leaves laid underneath it; and b) **a cross-certified hierarchical trust model**, also called a cross-certified mesh trust model, where there coexists of multiple hierarchical CAs with their root CAs cross certified each other. Here, the structure is like a forest with multiple trees, each has its own root on the top. There are variants of these models, such as hybrid trust models and bridge trust models [10].

[3] http://www.teragrid.org/

[4] http://egee1.eu-egee.org/test

Among these models, the hierarchical trust model is the most fundamental one because it is the basis for all the other models. As illustrated in Figure 4, in order to establish trust among end entities, such as machines and users, it is important that all the end entities and and subordinate CAs obtain and install a trusted copy of the root CA's public key certificate. This is often done offline through a manual process. Once the trusted root CA certificate is in place, end entities and subordinate CAs can obtain the certificates of the other subordinate CAs offline or online securely.

Regardless of which trust models to be used, they all suffer from a range of deployment and usability problems in practice. These problems often deter the wide spread use of PKI technologies in real world end user applications, which perhaps the reason why the basic password based authentication remains to be the most popular.

Deployment Problems with PKI From an operational point of view, PKI trust models deliver a poor quality of managibility and usability in an actual system that involves a large amount of machines and users. The problems can be classified into the following categories:

- *Certificate Request:* in order to acquire certificates, users and machine owners need to manually create certificate signing requests and send them to be signed by certificate authorities. This is often a cumbersome process for normal users and resource owners, especially when a large number of resources are involved.
- *Trust Anchor Maintainence:* users and machine administrators need to proactively involve in the process of deciding who to trust in a distributed environment by answering questions such as: what root CA certificates should I trust? This is often not an intuitive question to answer.
- *Certificate Installation:* once the trustworthiness of root CA certificates is established, users and machine owners need to install them manually. However, this process demands a fair amount of understanding of the system (e.g. designated location of the private keys, certificates, which commands to use etc.). Again, this is not intuitive for everyone.
- *Machine or Service Migration:* A machine or service certificate is typically bound the machine (machine name or IP) or the service (service name or machine IP) where it is installed. However, it is not that uncommon that machines gain a new IP or are renamed. Similarlly, services can be redeployed to a different environment (e.g. a container) on a different machine (different IP or machine name). Both will cause the security exception in

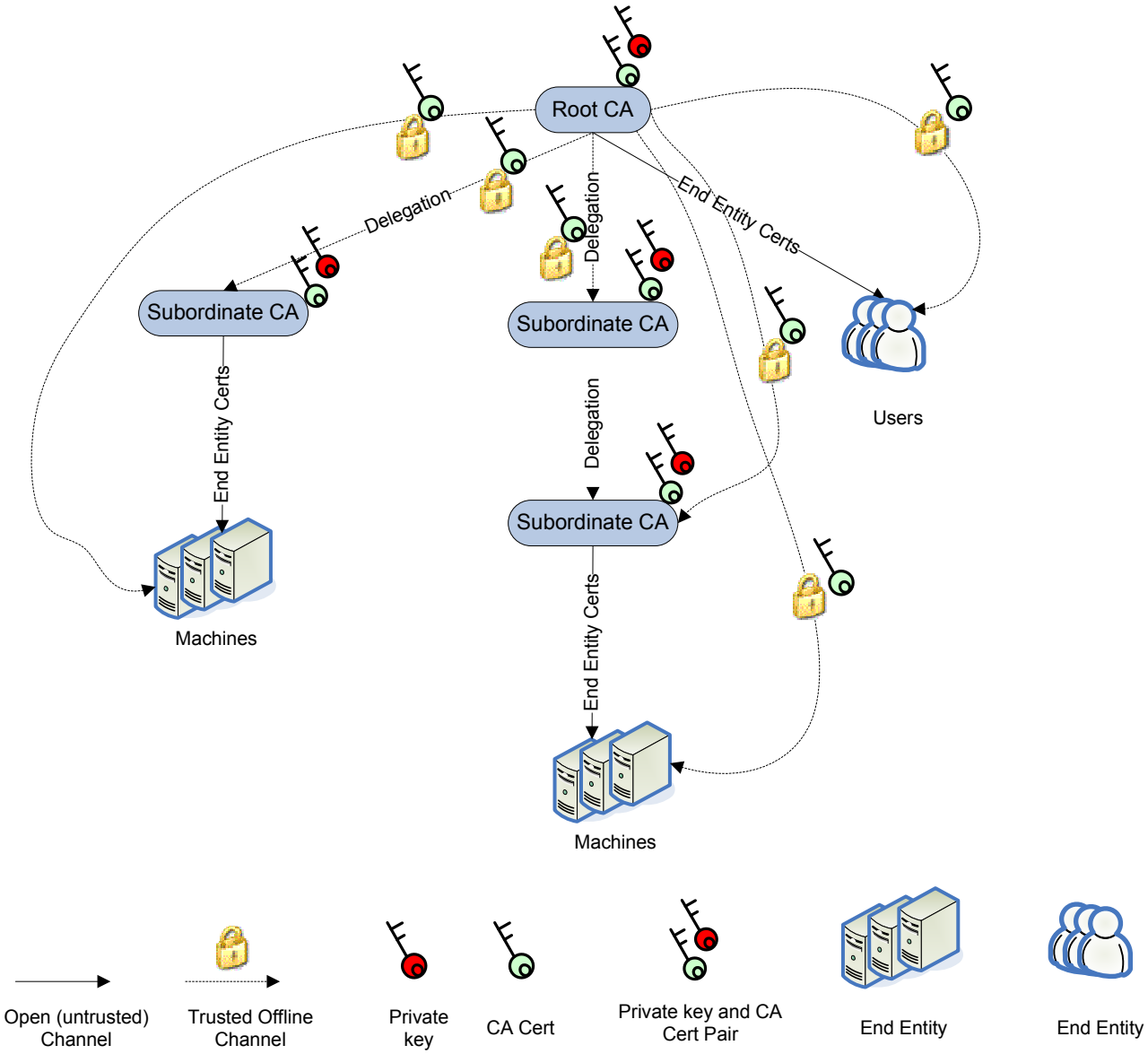

Figure 4: A hierarchical trust model of PKI

the SSL/HTTPS post-handshake stage[5].

All these problems are an issue for XtreemOS because it is a system consisting of a fair number of Grid-wide system services facilitating the coordiations within the system and of a large number of dynamically added end entities. As the number of services increases, the amount of work involving certificate signing requests, certificate installation and configuration, trust anchor configuration, and support for the continuous evolvement of the system will become tedious for users and machine owners: most of them demand human involvement and manual handling of offline processe, which can be error-prone, requiring both caution and care by users.

XtreemOS Trust Model

The Challenges Our aim is to build up the trust between a user and a machine in the system by allowing them, *through trusted online means, to obtain end entity certificates and root CA certificates*. To avoid the management problems with PKI, we are facing the following requirements in trying to design a new solution to manage the trust within the system:

- *Managing Trust Anchors Manually by Users:* we try to avoid asking users to maintain trust anchors manually: this can base on the established trust means. With the conventional PKI based trust model, the offline manual installation procedure is a way to ensure the authenticity of CA certificates. Therefore, under the conventional model, it is compulsory to have such a step. However, this can be error-prone and tedious process for the users. Also, it can cause managibility problems when the number of root CAs increases as root CA certificates need to be manually updated. So, avoiding asking users to maintain trust anchors manually can solve the following problems: 1) The users do not need to worry about the errors with offline root CA certificate installation and configuration; and 2) There is no need for users nor resource administrators to generate certificate signing requests either: system could handle such steps automatically for them.
- *Installing Trust Anchors by Resource Administrators:* we also try to avoid asking resource administrators to install trust anchors manually to avoid the same types of problems.
- *User Certificates:* we should not require users to install certificates and corresponding private keys, thus avoiding problems with managing certificates themselves.

[5] See, http://groups.google.com/group/mozilla.support.firefox/browse_thread/thread/9370ee21bae498ad

- *Machine Certificates:* we also try to avoid asking resource administrators to install certificates and corresponding sprivate keys, also avoiding problems of managing certificates themselves.
- *User Registration:* Apart from registering with a XVOMS, users are not required to register with any other services.
- *Machine Registration:* Apart from registering with a RCA, machines are not required to register with any other services.

XtreemOS Trust Model As illustrated in Figure 5, the XtreemOS trust model is fundamentally a cross-certified hierarchical PKI trust model. At the top level of the model, there are a list of *cross-certified root CAs*, which are implemented by the Certificate Distribution Authority (CDA) component of XVOMS. The CDA is responsible for directly *disseminating* end entity certificates to users. Note that a CDA only distributes certificates but not manages them. That is the responsibility of the other components within XVOMS. Underneath each root CA is a list of *subordinate CAs*, which is implemented by a standalone service called Resource Certification Authority (RCA). A RCA service *manages and disseminates* certificates for machines within an administrative domain. So far, our model has no difference to the PKI model.

Now, without loss of generality, let us focus on a special case where there is only one CDA/XVOMS in the system so that we can have an indept understanding of the system. This case is illustrated in Figure 6.

XVOMS facilitates three functionalities: registration management, credential distribution, and VO management. These functionalities are implemented by three *software components* respectively: registration manager, CDA (Credential Distribution Authority), and VO manager. The registration manager is responsible for registering users and RCA services (not machines). The CDA component is a root CA (there could be several of them) in the system which issues users with identity and attribute certificates. Different from conventional offline CAs, the CDA is operated online. The RCA (Resource Certification Authority) service is a standalone service operating a subordinate CA of the root CA. The RCA service is also an online CA which issues certificates (identity and attributes) on behalf of the CDA to machines.

The following is a list of steps for building up the XtreemOS trust model.

1. *XVOMS Certificate:* The XVOMS has a self-signed CA certificate, representing a root CA certificate in the system. The private counterpart of this certificate is used by the CDA component to sign end entity certificates for users and a subordinate CA certificate for RCAs. Note that there can be multiple root CAs currently present in the system.

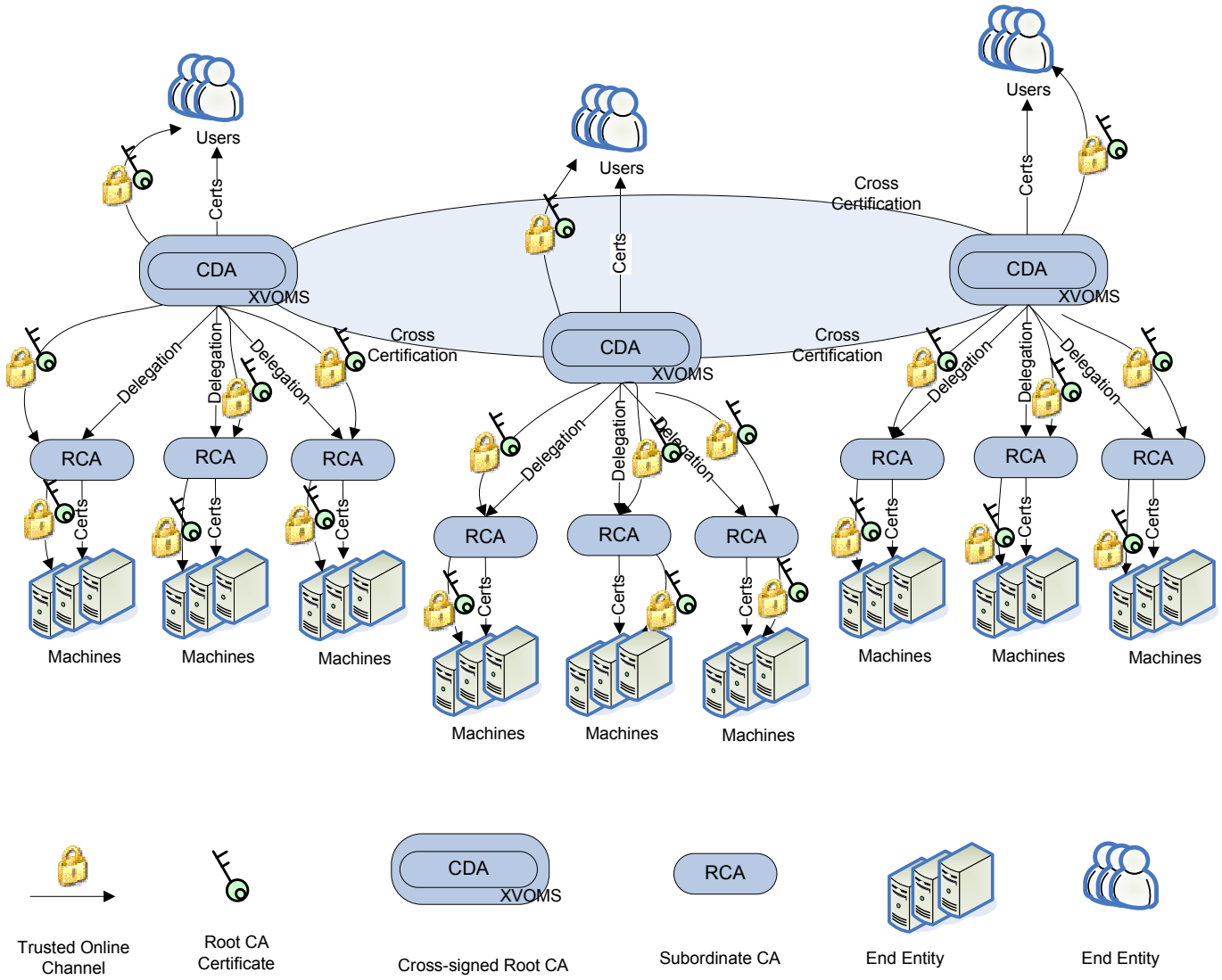

Figure 5: The XtreemOS trust model

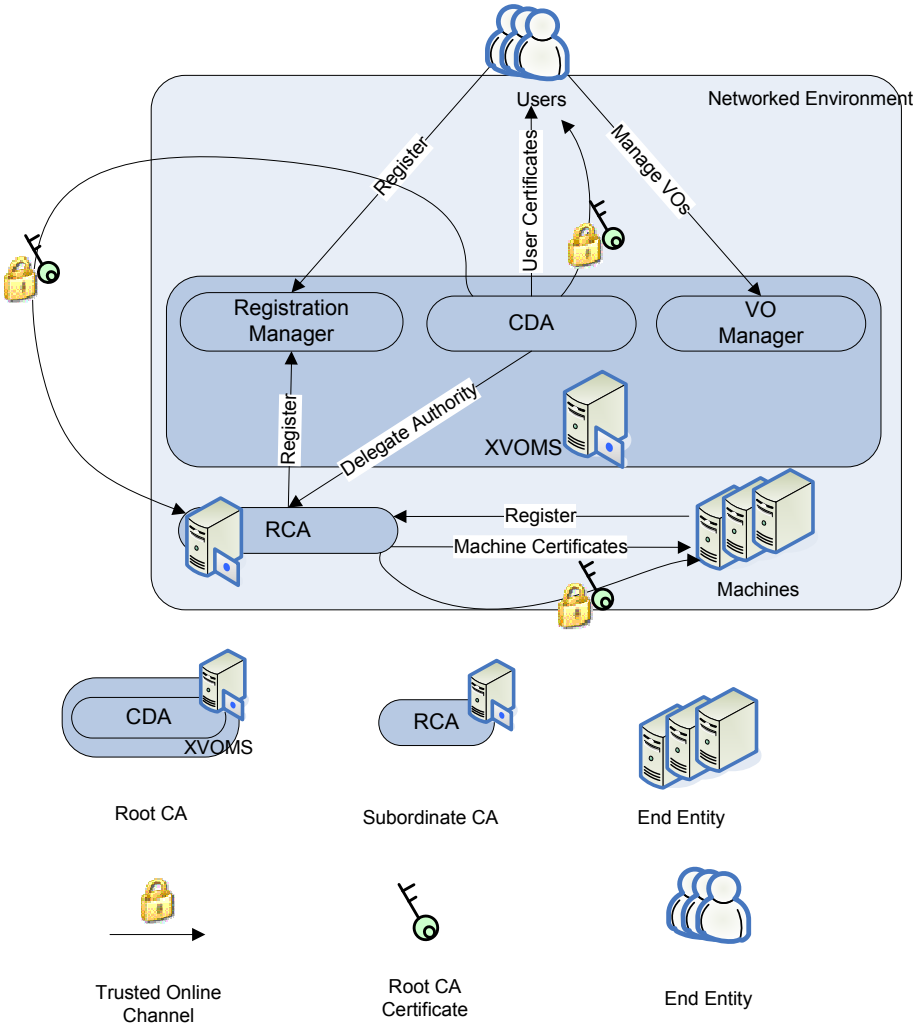

Figure 6: The relationship between XVOMS, CDA, and RCA in a networked environment.

2. *User Registration with XVOMS:* Users are registered with the XVOMS: each user shares a secret (i.e. password) with the XVOMS. This password allows a user to obtain a copy of the XVOMS' public key certificate securely through many well-established password based mutual authentication protocols[6]. Unlike the operation of an offline CA, this shared password allows a user to mutually authenticate with the XVOMS ***without the need to have any pre-installed certificate and private key pair***. This mutual authentication channel is used for two tasks: a) to securely (encrypted and mutually authenticated) upload a Certificate Signing Request (CSR) from a user to the XVOMS so that the latter can securely bind the CSR to the user; and b) to allow a user to securely obtain a *trust copy* of the root CA certificate. Once both tasks are completed, the user can verify the authenticity of the certificate given by the CDA.
3. *RCA Registration with XVOMS:* Each RCA in the system is registered with one XVOMS in the system by the resource administrator of that RCA. As part of the registration process, each RCA is given a shared secret (i.e. password) with the XVOMS. Again, this password allows the RCA to mutually authenticate with the XVOMS ***without having any pre-installed certificate and/or private key pair***. Similarly, this mutual authentication channel is used for two tasks: a) to securely (encrypted and mutually authenticated) upload a Certificate Signing Request (CSR) from a RCA to the XVOMS so that the latter can securely bind the CSR to the RCA; and b) to allow a RCA to securely obtain a *trust copy* of the root CA certificate. Once both tasks are completed, the RCA can verify the authenticity of the certificate given by the XVOMS.
4. *RCA Certificate:* Because a RCA is a subordiante CA of a XVOMS, its certifiate needs to be signed by the XVOMS's certificate.
5. *Machine Registration with RCA:* Each machine needs to register with at least one RCA in the system so that it can obtain identity and attribute certificates from the RCA.

Advantages Following this trust model, user management is cleanly separated from resource management through a XVOMS service. Because of the separation, the addition or removal of users will not impose significant performance and configuration impacts on resource management in a VO, and vice versa.

The key difference between the PKI trust model and that of XtreemOS resides in the process that the trust model is set up. In the former, trusted root CA certificates are distributed through *offline* means whilst in the latter, these certificates are disseminated through online protocols. That is, different from other conven-

[6]For explaination, see http://en.wikipedia.org/wiki/Challenge-response_authentication

tional PKI trust models, our model is established upon the existence of one or more *online* certification authorities in an XtreemOS system.

2.4.2 Root CA Certificates

The rationale for adopting passwords in various interactions (e.g. between a XVOMS server and a user, and between a RCA server and a machine) is to simplify the certificate deployment and management issues related to set up PKI. When SSL-based mutual authentication is required, two communicating parties involved in the protocol need to acquire the root CA certificate and any intermediate CA certificates through offline processes before any sort of SSL protocol (e.g. socket-based SSL or https) can proceed. Although this is a standard practice in today's Grid world, it is still an practical issue for system administrators.

In this section, we shall first present a simplified password-based protocol to establish a secure[7] (encrypted and mutually authenticated) channel between two communicating parties. This protocol is the foundation for our system to get rid of the offline process of getting trusted root CA certificates, simplifying the deployment of our system in practice.

Notations The following is a list of notations used in the rest of this chapter:

- X: a XVOMS server.
- U: a registered user of X.
- R: a RCA server.
- M: a registered machine of R.
- A_U: network address of the client program running on behalf of U
- v: default validity period of an attribute certificate issued by C
- V: default validity period of a public key (identity) certificate issued by X
- $GUID_U$: U's global user identifier
- K_{XY}: a share secret (password) between X and Y, it is equivalent to K_{XY}.
- PK_A: A's public key.
- SK_A: A's corresponding private (secret) key.
- N_U: a cryptographic nonce[8] generated by U
- CSR_U: a certificate signing request from U.
- <M>SK_A: a message M signed by A's private key.
- $\{M\}K_{XY}$: a message M encrypted by K_{XY}.
- $CredReq_U$: a credential request generated by U
- $Cred_U$: a set of credentials requested by U.

[7] At some point, this term needs to be formally defined.

[8] See: http://en.wikipedia.org/wiki/Cryptographic_nonce

- <$Cert_U$>SK_A: an end entity identity certificate signed by A's private key.
- <$Cert_X$>SK_X: a self-signed root CA certificate.
- $A \rightarrow B$: sending a message from A to B via an insecure open network
- $[A \rightarrow B]$: sending a message from A to B via a *secure* (encrypted and mutually authenticated) channel over an insecure open network

A Simplified Password-based Mutual Authentication Protocol It is evident that these interactions have to be mutually authenticated to ensure the communicating parties are conversing with an authentic party. Therefore, for completeness, a simplfied password-based mutual authentication protocol between a client Cli and a server $Serv$ is presented as follows with the aim to pave ways for the implementation. Upon successful execution of the protocol, two goals are achieved: a) both parties are mutually authenticated; b) a secret session key is established for the parties to securely communicate with each other.

It should be noted that the above protocol is just one of the many password-based mutual authentication protocols[9] that are available in the literature.

1. $Serv \rightarrow Cli$: a unique $Serv$ generated challenge value sc
2. $Cli \rightarrow Serv$: a unique Cli generated challenge value cc and cr, where cr=hash($cc + sc + password$)
3. $Serv \rightarrow Cli$: sr, where sr=hash($sc + cc + password$)

The protocol commences when $Serv$ generates a sufficiently random and unique challenge value, called a server generated challegne sc. Also in step 1.), sc is sent to Cli in clear text.

In 2.), when Cli receives the value sc, he also generates a sufficiently random and unique challenge value, called a client generated challenge cc and calculates a client response cr where $cr = hash(cc + sc + password)$ locally. If the password is stored as a hashed value at $Serv$, $password$ should be replaced by the hashed value of $password$. Also, the calculation is done locally: the password is not transmitted over the network in any form. Together with cc, cr is transmitted over the network to $Serv$.

In 3.), upon receiving cr in clear text, $Serv$ computes a server response called a server generated response sr and ensures cr is equal to sr. If they are equal, the client is authenticated. Otherwise, the client is *not* authenticated and the protocol stops $Serv$ then sends sr, in clear text, to Cli, who then compares sr and cr. If they are equal, the server is also authenticated, leading to the mutual authenticated state of both parties. Otherwise, the server is *not* authenticated.

[9] http://en.wikipedia.org/wiki/Challenge-response_authentication

Upon successful execution of this protocol, a session key $hash(cc + sc + password)$ is also agreed between $Serv$ and Cli so that they can communicate using this session key afterwards.

Secure Channels in the Trust Model In Figure 6, there exists three secure channels: between U and X, between X and R, and between M and R. Informally, a *secure channel* is defined as a channel encrypted using the shared secret between two mutually authenticated parties. Therefore, no third party can eavesdrop the messages exchanged in this channel. A *mutually authenticated* channel is defined as a channel that a party can verify the identity of the other and vice versa. Because U and X share a secret - U's password, they can together execute one of the many password-based mutual authentication protocols to establish a secure and mutually authenticated channel between them. This also applies to the communication between X and R, and that between M and R.

The Protocol over the Secure Channel Between Users and XVOMS Over the secure channels described above, message authenticity can be verified and its confidentiality can be guaranteed. At this stage, neither U nor R has been issued any end entity certificate or a root CA certificate. At this stage, all they know about each other is the shared secret (i.e. password) between U and X, and between R and X.

Specifically, over the secure channel between U and X, the following messages can be exchanged :

1. $[U \rightarrow X]$: a certificate signing request CSR_U
2. $[X \rightarrow U]$: an end entity identity certificate <$Cert_U$>SK_X of U signed by X's private key and a XVOMS's certificate (a self-signed root CA certificate) <$Cert_X$>SK_X

This protocol commences after U successfully registers with X, meaning that U has a shared secret (i.e. his password K_{UX}) with X. This allows U and X to establish a secure channel between them. U can start to generate a key pair locally, where the public key is then be used to generate a certificate signing request CSR_U. Normally, U needs to specify the DN information in the certificate while generating a CSR. However, this is not required here because the DN information will be automatically populated by X following the convention specified in Section 2.4.3. Over the secure channel, U sends over the CSR_U to X. Because the channel is mutually authenticated, X can safely *bind* this CSR_U to the user, establishing the binding between the public key embedded in the CSR and the user's identity. Once the CSR is accepted by X, the user can request *(short-term) proxy certificates* from X repeatitively.

Also over the secure channel, the next message is from X back to U, where two pieces of information are sent: an end entity certificate <$Cert_U$>SK_X for U signed by X's private key, and a XVOMS's self-signed certificate <$Cert_X$>SK_X. Because the channel is secure, U gets an authentic copy of X's certificate, which will be installed locally by the user to his machine. U's certificate <$Cert_U$>SK_X can then be verified by <$Cert_X$>SK_X, if needed.

The Protocol over the Secure Channel Between RCAs and XVOMS The channel between RCAs and XVOMS can also be secured using the password-based mutual authentication protocol. However, the messages exchanged over this channel is of the same nature as those presented in the protocol between users and XVOMS. Only for completeness and self-containment, we include the protocol over the secure channel between RCAs and XVOMS.

At this stage, R has not been issued any end entity certificate or a root CA certificate. Between R and X, there exists a shared secret (i.e. password). Therefore, over the secure channel between R and X, the following messages can be exchanged :

1. [$R \rightarrow X$]: a certificate signing request CSR_R
2. [$X \rightarrow R$]: an end entity certificate <$Cert_R$>SK_X of R signed by X's private key and a XVOMS's certificate (a self-signed root CA certificate) <$Cert_X$>SK_X

This protocol commences after R successfully registers with X, meaning that R has a shared secret (i.e. his password K_{RX} with X. This allows R and X to establish a secure channel between them. R can start to generate a key pair locally, where the public key is then be used to generate a certificate signing request CSR_R. Normally, R needs to specify the DN information in the certificate while generating a CSR. However, this is not required here because the DN information will be automatically populated by X following the convention specified in Section 2.4.3. Over the secure channel, R sends over the CSR_R to X. Because the channel is mutually authenticated, X can safely *bind* this CSR_R to the RCA, establishing the binding between the public key embedded in the CSR and the RCA's identity.

Also over the secure channel, the next message is from X back to R, where two pieces of information are sent: an end entity certificate <$Cert_R$>SK_X for R signed by X's private key, and a XVOMS's self-signed certificate <$Cert_X$>SK_X. Because the channel is secure, R gets an authentic copy of X's certificate, which will be installed locally by the resource administrator onto the machine. R's certificate <$Cert_R$>SK_X can then be verified by <$Cert_X$>SK_X, if needed.

The Protocol over the Secure Channel Between Machines and RCA As illustrated in Figure 6, machines need to register with at least a RCA securely. Because machines are operated within the same administrative domain as a RCA, the problem of establishing of secure channels between machines and the RCA is considered to be resolved. Thus, the root CA certificate can be securely transfered to the machines via this channel.

2.4.3 User and Machine Identity Certificates

There are three ways that a global entity (e.g. a user, a VO, a node or a service) can be added into a XtreemOS system: they can be either added via a XtreemOS VO Management Service (XVOMS, see Section 3.1), via a Resource Certification Authority (RCA, see Section 3.3), or through importing them from an existing Grid system.

In the former case, *the Country, Locality, State, Organization, Organization Unit subfields in the DN field is the same as those properties of the XVOMS server or the RCA server and the Common Name is set to a host-wide unique identifier given by a credential issuer, which is the XVOMS server for user identity certificates and the RCA server for machine identity certificates.* Host-wide unique identifiers are unique within the scope of the issuing server. Because they are host-wide valid, it is not necessary to ensure their global uniqueness across the system.

In the latter case, where users and machines already have certificates assigned by a third-party trusted authority, such as an offline certification authority. In this case, to maintain the backward compatibility with legacy Grid systems, the DN of such certificates *will not be replaced* so that these certificates can still be used within their existing Grid systems.

2.4.4 User Attribute Certificates

A user can simultaneously hold multiple attribute certificates, *one per VO*. For example, when a user registers with multiple VOs, he will be associated with multiple GVIDs. Similarly, a user can register with multiple global groups, and associate with multiple GGIDs. Within a VO, a user can concurrently join different groups and have multiple roles.

Global Attributes' Value Format Table 2 summarizes the type of attributes for the global entities. GUIDs, GVIDs, and GGIDs are global attributes for users and they are are the concatenation of two 16-byte strings (separated by a forward slash):

Global Entities	has PKC	has AC(s)	Attributes
User	yes	yes	GUID(s), GVID(s), GGID(s), Group, Role, Capability
Machine	yes	yes	GVID(s), Services
VO	no	no	N/A
Service	no	no	N/A

Table 2: Certificates for Users and Machines (PKC: Public Key Certificates, AC: Attribute Certificate), a user and a node can concurrently belong to multiple VOs, thus simultaneously holding multiple ACs.

```
/<16-byte credential issuer's IP address>/<16-byte
host-wide unique ID>
```

A GNID, the identifier for a machine, is in the same format.

```
/<16-byte credential issuer's IP address>/<16-byte
host-wide unique ID>
```

However, the credential issuer for GNIDs is typically not the same as the credential issuer for GUIDs, GVIDs, and GGIDs. For GNIDs, the credential issuer is the corresponding RCA server. Whilst, the credentials issuer for GUIDs, GVIDs, and GGIDs is the corresponding XVOMS server.

Guaranteeing Global Uniqueness Ensuring the global uniqueness of the attributes of global entities is important because they provide the fundamental support for the correct working of an XtreemOS system. Without such a guarantee, users may be able to access each other's files without being noticed and/or control each other's jobs without being detected.

However, in practice, an XtreemOS system is typically an aggregation of resources from different administrative domains. Demanding participating organizations to rely on one single centrally controlled server to ensure the global uniqueness is not likely to go far. And it is not scalable either. Thus, we have to allow decentralized management of the system through the support of *concurrent presence of multiple* (user and machine) **Credential Issuer** and **Credential Distributor**. Putting it simply, a credential issuer manages the lifecycle of credentials. From an implementation point of view, the service implements the functionalities of a credential issuer is XVOMS. A CDA (for users) is responsible for distributing credentials. Therefore, they act as credential distributors in the system. Note that a RCA is also a Credential Issuer for machines.

To accommodate for multiple credential issuers in the system while ensuring the global uniqueness, we require all global IDs (i.e. GUID, GVID, GNID, and GGID) start from a globally unique string <ID issuer's hostname> and is followed by a (ID issuer's) host-wide unique string. The uniqueness of the former is guaranteed by the DNS system, a well-known trusted third party, and the latter by host self-checking mechanisms.

An hypothetical example of a user attribute certificate is showed in Figure 7. The first three attributes (i.e. GUID, GVID, and GGID) are in the plural form: a user can have multiple GUIDs, belong to multiple VOs (thus having multiple GVIDs), and belong to multiple global groups. Typically, a user has only GUID, which has an one-to-one relationship with the DN in its public key certificate.

However, our design can also allow for the situation that a user has multiple GUIDs. The fact that a user can have multiple GUIDs does not conflict with the global uniqueness of the ID itself. These multiple GUIDs still correspond to the same DN of the public key certificate of a user. This design simply gives receiving nodes the flexibility to handle the more complex situation that a process (on a receiving node) can change the GUID according to the execution state and environment. For instance, during an installation process of a standard SAP business application, eight default user accounts, with varied privileges, are needed to install the application. It is a standard application installation as it is often installed on a standalone machine. In a large-scale data centre scenario where applications are needed to be installed remotely and on-the-fly, these accounts are often created on-demand by the installation procedure by a privileged process. For the details of this scenario, please refer to the 2nd specification [4]. To cope with such a scenario, it is possible to reserve a range of GUIDs specifically for installation processes on the XVOMS server and these GUIDs have predefined one-to-one mappings with the original eight IDs that are required by the applications. The GUIDs and the mappings are globally unique. At a result, the application can be installed uniformly across the entire data centre.

Structure between Groups and Roles The relationship between groups and roles can either be hierarchical or flat. In a hierarchical setting (as shown in Figure 7), roles are further divisions within a group of a VO. Otherwise (as shown in Figure 8), VO roles are independent from VO groups: they are equal.

Groups and Roles' Value Format Groups and roles are only valid within the context of a VO. They are not standalone attributes for global entities. Therefore, they are always presented as VO attributes. In an attribute certificate, VO attributes appear as a sub-attribute of a GVID.

For a VO group, its value format is:

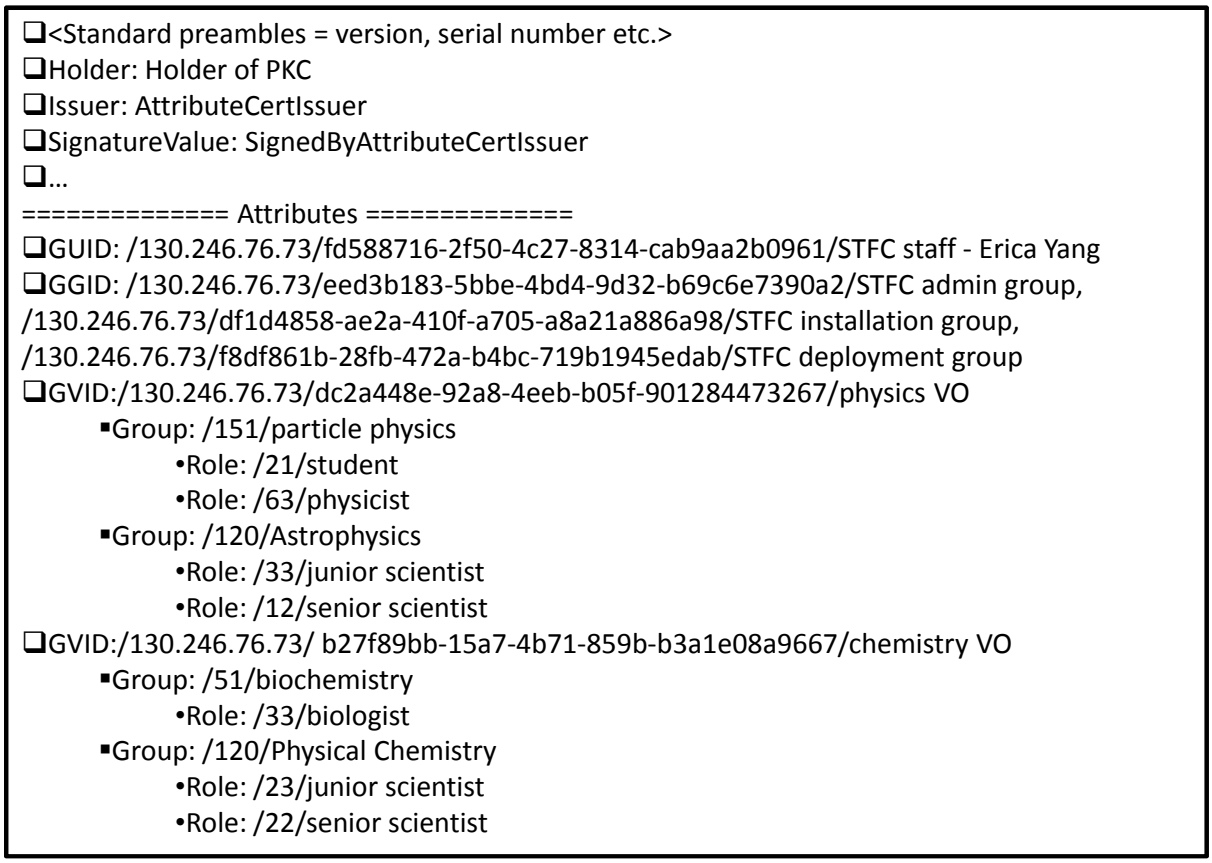

Figure 7: A hypothetical user attribute certificate with a hierarchical structure between groups and roles

❑<Standard preambles = version, serial number etc.>
❑Holder: Holder of PKC
❑Issuer: AttributeCertIssuer
❑SignatureValue: SignedByAttributeCertIssuer
❑…
============== Attributes ==============
❑GUID:/130.246.76.73/fd588716-2f50-4c27-8314-cab9aa2b0961/STFC staff - Erica Yang
❑GGID: /130.246.76.73/eed3b183-5bbe-4bd4-9d32-b69c6e7390a2/STFC admin group,
/130.246.76.73/df1d4858-ae2a-410f-a705-a8a21a886a98/STFC installation group,
/130.246.76.73/f8df861b-28fb-472a-b4bc-719b1945edab/STFC deployment group
❑GVID: /130.246.76.73/dc2a448e-92a8-4eeb-b05f-901284473267/physics VO
- ▪Group: /151/particle physics
- ▪Role: /21/student
- ▪Role: /63/physicist
- ▪Group: /120/Astrophysics
- ▪Role: /33/junior scientist
- ▪Role: /12/senior scientist

❑GVID:/130.246.76.73/ b27f89bb-15a7-4b71-859b-b3a1e08a9667/chemistry VO
- ▪Group: /51/biochemistry
- •Role: /33/biologist
- ▪Group: /120/Physical Chemistry
- •Role: /23/junior scientist
- •Role: /22/senior scientist

Figure 8: A hypothetical user attribute certificate with a flat structure between groups and roles

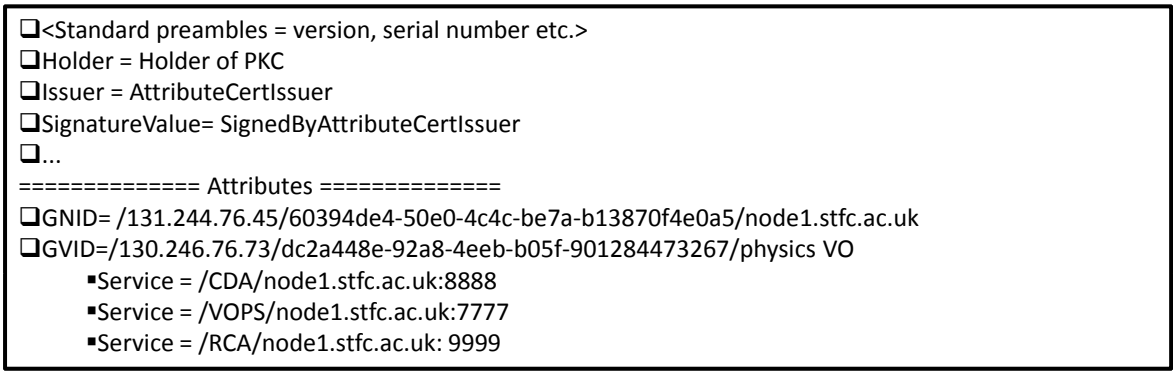

Figure 9: A hypothetical machine attribute certificate

```
/<the group's unique ID within the VO>/[description]
```

For roles, the situation is slightly complex. When VO roles are completely independently from VO groups, the value format for a VO role is:

```
/<the role's unique ID within the VO>/[description]
```

However, when VO roles are bound to VO groups as a subdivision to VO groups, its value format is:

```
/<the role's unique ID within the VO group>/[description]
```

2.4.5 Machine Attribute Certificates

Similarly, a node can also simultaneously hold multiple attribute certificates, which are also *one per VO*.

An hypothetical example of a machine attribute certificate is showed in Figure 9.

Services' Value Format A node could provide one or more services to a VO by embedding its services information in a node's attribute certificate. Of course, a node may also choose not to constrain itself to any specific VOs, thus allowing its services being used by anybody, with or without associations with VOs. This is entirely down to the local policy set by the node administrator. That leads to two possible outcomes in terms of the format of a node's attribute certificate.

Possibility one is the case that services are valid within the context of a VO. This is shown in Figure 9, where the services (i.e. CDA, VOPS, and RCA) are

presented in the context of the VO (/130.246.76.73/dc2a448e-92a8-4eeb-b05f-901284473267/physics VO).

The second possibility is that services are run independent of VOs. In this case, services are not bound to any VOs, thus they are available to anyone.

A service's value format is:

```
/<service type>/<hostname of a service>:<port>
```

2.4.6 Delegation and Single Sign-On

As explained in Sections 2.4.1, XtreemOS adopts the conventional PKI hierarchical based trust model. Therefore, from an end user perspective, it seems straightforward for our system to use the GSI technologies [15], including proxy certificate [16] to support delegation of authority and single sign-on.

Given that there already exists large scale production Grid infrastructure being widely deployed and maintained in scientific communities, the support for GSI would allow the maximum competibility and interoperatibility with existing large scale Grid infrastructure, such as EGEE and TeraGrid, in the world. In return, such support will maximize the success of the XtreemOS project as a whole.

It should be noted that GSI is an umbrella term grouping a range of public key cryptography based security technologies, including the following

1. single-sign on using proxy certificates
2. delegation using proxy certificates
3. secure communication
4. mutual authentication
5. securing private keys

Some of them (namely 3., 4., and 5.) are "mature and old" technologies meaning that they existed before GSI was invented. The term GSI simply covers these technologies for convenience. What is unique to GSI (namely a GSI unique invention) is technologies 1. and 2. As the foundation of the XtreemOS trust model is based upon PKI and all end entities (users and nodes) in the system receive end entity public key certificate, the introduction and use of GSI in XtreemOS is straight-forward. For clarification, the first release of XtreemOS has already incoporated support for technologies 3., 4., 5. through the use of SSL, HTTPS, and configuration files (specifying the password for private keys) in the security services.

3 Services

This chapter updates the design of the security and VO services offered by WP3.5. The services presented here complement the description given in Section 2.2.4.

3.1 XVOMS

3.1.1 Brief Introduction

X-VOMS is an advanced Virtual Organisation (VO) management service for supporting secure and flexible collaborations and resource sharing among people, projects and organisations. It is written in Java and is backed by a (Hibernate-based) X-VOMS database schema. X-VOMS provides three major functionalities: registration management of users and resources, VO and credential management, and credential distribution. It provides two types of interfaces: XVOMS native APIs (which programs can invoke directly) and VOlife web interfaces (which users can directly interact with through a web browser).

X-VOMS's novel design allows it go beyond the support for focused scientific collaborations and resource sharing among scientists and across scientific institutions. It allows flexible management of VO lifecycle: VOs can be set up and demolished manually and on-the-fly by runtime applications. It supports a powerful VO management structure: users can join multiple VOs under different types of VO models; and credentials from all (or a selection of) registered VOs can be obtained via API calls. In addition to managing users and VOs, X-VOMS can also be used to manage resources. This makes it possible to construct an end-to-end secure VO environment.

3.1.2 Major Components

Figure 10 illustrates the architecture of XVOMS, which consists of the following major *software components*:

- *Registration Manager:* manages users and RCAs registration into the system.
- *VO Manager:* maintains VO lifecycles and credentials across the system.
- *Credential Distribution Authority (CDA):* distributes signed credentials, including identity and attributes to users and to RCAs. The XVOMS's private key is used by the CDA component to sign the certificates issued by the XVOMS.
- *XVOMS DB:* stores three types of information: the registration details of users and RCAs, including their certificates, VOs, and VO association details of users and RCAs.

- *Web Interface:* XVOMS comes with a web interface, called **VOLife**, which provides a web frontend, which wraps around the XVOMS functionalities (Registration management, VO management, and CDA) to allow human users to use a web browser to interact with XVOMS.
- *native XVOMS interface:* XVOMS also provides native APIs allowing *software programs* to directly invoke XVOMS functionalities.

As XVOMS is responsible for credential management, distribution, and registration management, readers are strongly recommended to read through Section 2.4 to fully understand and appreciate the design philosophy of XVOMS and the underlying protocols that XVOMS uses.

3.1.3 Interactions with Other Security Services

A XVOMS server is operated by a system administrator of a XtreemOS system. It also interacts with VO owners, resource administrators, and (system) users. There are typically three stages of interactions (in a typical *order of event occurence*) in the system: interactions with registration manager, those with CDA, and those with VO manager.

The initial contact with a XtreemOS system starts by a user or a resource administrator of a RCA registration with a XVOMS that the system operates. This is depicted by the interaction with the registration manager in Figure 10. As a result of a successful registration, the user or the RCA is added to the XVOMS DB. Also, a shared secret is established between the user and the XVOMS server, and between the resource administrator and the XVOMS server.

The second type of interactions occurs between CDA and the corresponding software clients acting on behalf of a user (i.e. user client), and CDA and the corresponding software clients acting on behalf of the resource administrator (i.e. RCA server).

The former allows the user to upload CSRs to the XVOMS and download the XVOMS's certificate and a user certificate. The user client also installs the certificates to the designated location on the client machine. A similar type of interactions occur between the RCA servers and XVOMS.

The final type of interactions occur after the end entity and root CA certificates are in place. At this stage, the interactions can be done over a standard SSL/https channel. Users, VO owners, and resource administrators can interact with the VO manager to manage VOs. Updated VO information can also be pushed to RCA servers.

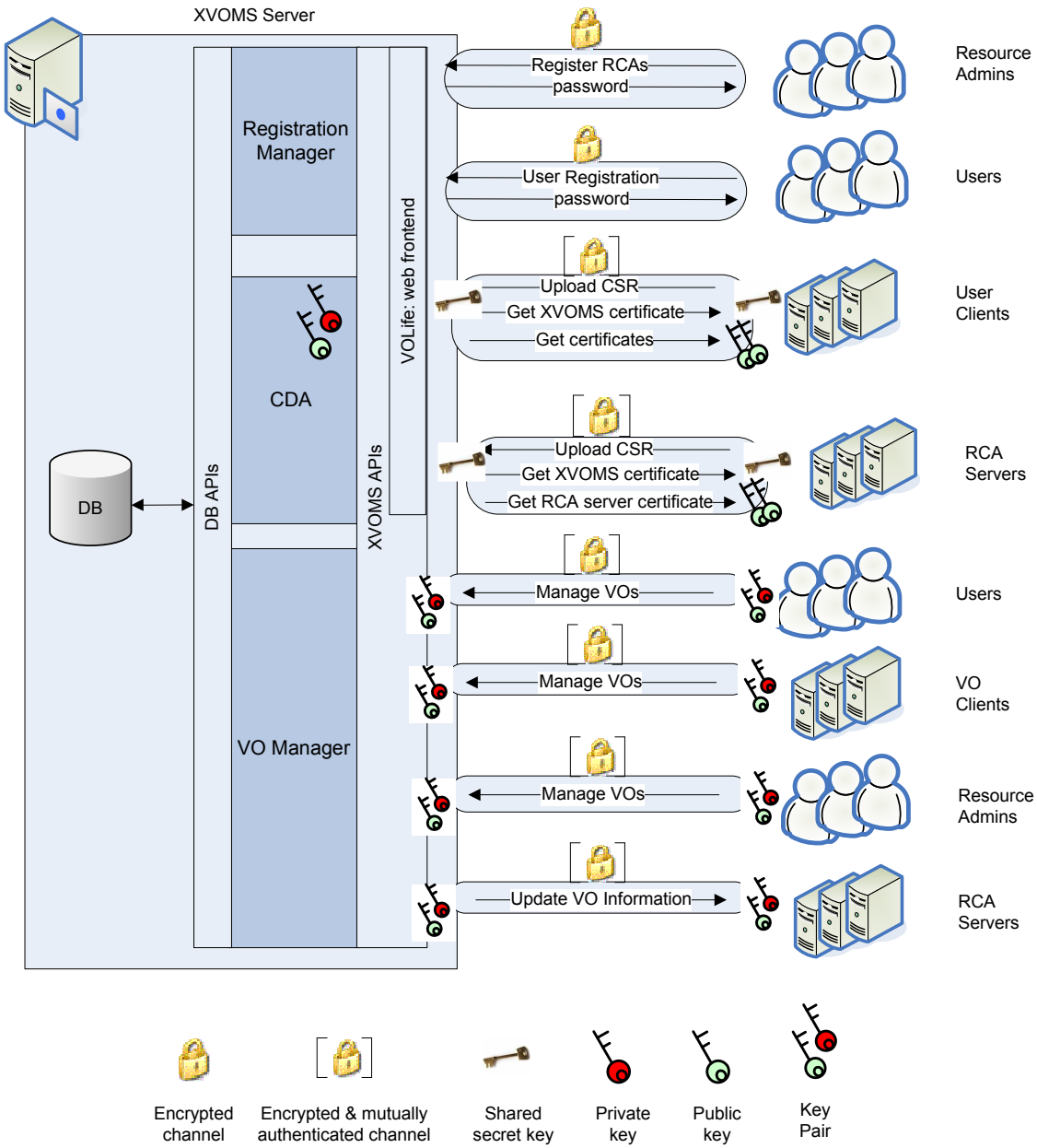

Figure 10: XVOMS architecture

3.1.4 Supporting XtreemOS Application Execution

AEM does not manage users. It relies on XVOMS to register users and manage their credentials. AEM does not directly interact with XVOMS. However, all user and machine certificates acquired via XVOMS protocols (presented in Section 2.4.2) are used in AEM.

For example, user identity certificates are used for delegation, single sign-on, secure communication and authentication, all of which are essential for secure job submission and execution. Machine identity certificates are used for authentication and secure communication. Users' attribute certificates are used to support VO-level and node-level authorisation, account mapping and dynamic account creation. Finally, machines' attribute certificates are used to perform checking of whether the machines belong to a particular VO.

3.1.5 Supporting XtreemOS File Management

XtreemFS does not manage users. It relies on XVOMS to register users and manage their credentials. XtreemFS does not directly interact with XVOMS. However, all user and machine certificates acquired via XVOMS protocols (presented in Section 2.4.2) are used in XtreemFS.

For example, user identity certificates are used for secure communication and authentication, all of which are essential for secure file accessing. Machine identity certificates are used for authentication and secure communication. Users' attribute certificates can be used to support VO-level and node-level authorisation, account mapping, dynamic account creation, and bi-directional mapping between global and local entities.

3.1.6 Supporting MD

Parallel to this deliverable, the mobile device work package (WP3.6) is producing a credential management framework for mobile devices. This work is described in a separate deliverable D3.6.3 - XtreemOS-G for MDs/PDA. For example, the usage of proxy machines to get user certificates could be useful to be incorporated or be used to extend the capabilities of the CDA component within XVOMS. Further investigations and evaluation are required to fully appreciate the implication of this framework to XVOMS.

3.1.7 Supporting WP3.2's Services

At this stage, no interactions are perceived with the WP3.2's services of XtreemOS.

3.1.8 Features for the Next Release

The first XtreemOS release suffers from two major limitations:

1. the trust boostrapping of the system is done offline. Root CA Certificates have to be disstributed offline before any communication commences. Putting it simply, to get a certificate from XVOMS/CDA, users need to obtain the root CA certificate (i.e. the CDA certificate) offline and mannually install them.
2. only one static VO is supported throughout the system. Static means that the identity of the VO and VO attributes are known, through offline means (word-of-mouth), to all services in the system.

Therefore, on the top of our agenda is to provide support to the XtreemOS trust model presented in Section 2.4. The full support of which could lead us resolve the problem related to the first limitation and partially pave the way to provide a scalable solution to the problem described in the second.

3.2 VOPS server

3.2.1 Brief Introduction

VOPS is a **core-level service**. It is a server primarily intended serving requests and forwarding answers from/to resource discovery services and digitally signs its decisions before forwarding responses back to services.

VOPS enforces user requests against VO level policies for gaining access to specific resource nodes. It is a standalone security service which provides Policy Administration Point (PAP), Policy Information Point (PIP) and Policy Decision Point (PDP) to other XtreemOS services (e.g. Application Execution Manager). VOPS acts as a authorization service which authorizes an entity to permit or deny access to a resource by enforcing additional constraints.

Primary intention of having an entity acting as a policy service is to support coordinated access control over VO resources by offering VO level policy decision point (PDP). Such access control can be performed on individual resources and can be used as access control to a group of resources/services sharing certain characteristics. VO-level policies and node-level policies form a hierarchical access control framework that can be tuned to achieve various degrees of control to resource usage within a VO.

3.2.2 Major Components

This section describes management components which are provided by the VOPS. As illustrated in Figure 11, there are four components related to a general policy

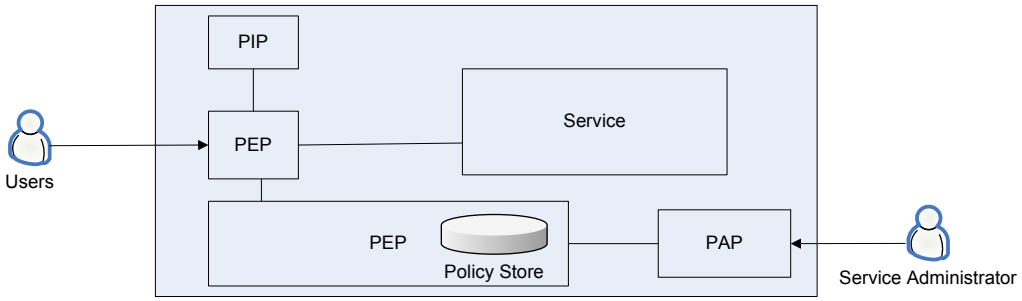

Figure 11: A generic policy model describing the relationship between PIP, PDP, PEP, and PAP

model:

- `PEP - Policy Enforcement Point`: this is where a user initiates a request, which contains information about the user, such as certificates, attributes, and context, targetted resource(s), and actions that the user wants to perform over the resource(s).
- `PDP - Policy Decision Point`: a user request is subject to the decision made by PDP, which can be a integral part of a service (as it currently stands in XtreemOS) or an independent trusted third party outside a service providing general policy decisions for all services. A `PDP` is often attached to a policy store which provides all the policies that the `PDP` is used to make the decisions.
- `PIP - Policy Information Point`: upon receiving a user request, `PDP` can refer to `PIP` for further information, such as additional attributes about the user, to make decisions.
- `PAP - Policy Administration Point`: this is a point where a service administrator addes/modified the policy store.

Specifically, in XtreemOS, PIP is an entity which acts as a source of attribute values. If we are considering different kind of attributes such as static or dynamic attributes, we need to have a way to obtain these by contacting appropriate service e.g. SRDS (Scalable Resource Discovery System) for static/dynamic attributes or VO Management service for certificate attributes. Currently only static attributes

are taking part in requests against policies. These static attributes are extracted from user and resource VO attribute certificates. VOPS PIP contacts RCA Client service to obtain resource VO attribute certificates. User certificate and JSDL document, which is used to describe resource requirements and also participates in the request, have to be provided as parameters of the VOPS command.

Here is the list of attributes, which are used in a request against policies. If some of these attributes are not provided or could not be extracted from certificates or JSDL document, request is constructed using just a subset of these.

Attributes obtained from user certificate When the user submits a job to AEM, she also needs to provide the user certificate containing the user's VO credentials. VOPS reads the following attributes from the user certificate.

```
DnIdAtCommonName
DnIdAtCountryName
DnIdAtLocalityName
DnIdAtStateOrProvinceName
DnIdAtOrganizationName
DnIdAtOrganizationalUnitName
ExtensionsRole
ExtensionsGroup
ExtensionsSubgroup
ExtensionsGlobalUserId
ExtensionsGlobalPrimaryGroupName
ExtensionsGlobalPrimaryVOName
```

Attributes obtained from resource's VO attribute certificate Resource's VO attribute certificate is obtained by querying RCA Client service.

```
DnIdAtCommonName
DnIdAtCountryName
DnIdAtLocalityName
DnIdAtStateOrProvinceName
DnIdAtOrganizationName
DnIdAtOrganizationalUnitName
ExtensionsCPUSpeed
ExtensionsCPUCount
ExtensionsMemorySize
ExtensionsService
ExtensionsVO
```

Attributes obtained from JSDL description JSDL is provided when executing XATI command through AEM.

```
JsdlResourcesOSTypeName
JsdlResourcesIndividualCPUSpeed
JsdlResourcesIndividualCPUCountExact
JsdlResourcesIndividualCPUCountLowerBoundedRange
JsdlResourcesIndividualCPUCountUpperBoundedRange
JsdlResourcesIndividualMemorySizeLowerBoundedRange
JsdlResourcesIndividualMemorySizeUpperBoundedRang
JsdlResourcesNetworkProto
JsdlResourcesNetworkNetmask
JsdlResourcesNetworkPorts
JsdlResourcesResourceCountExact
```

PIP constructs request comprised of upper elements in a key-value pairs. Values are obtained from certificates or JSDL specification. Attributes with prefix *Extensions* are obtained from X509 certificate attribute extensions.

PDP is an entity which acts as a decision point, where applicable policies are evaluated and authorization decision is made. Input to the PDP are attributes and policies which apply to subject requesting a decision. Decision point is called by other services like Application Execution Manager's ResMng.

PAP is used as an administration point, where policies can be initially created, maintained and eventually removed. PAP acts as a source of policies which are used in PDP when evaluating a request.

VOPS policy storage Currently policies are stored as XML files under location as configured in configuration files. There is a module implemented to manage these policies (PAP) and will be extended to use database, see section 3.2.8.

3.2.3 Interactions with Other Security Services

VOPS uses the DIXI framework [5] as a communication bus to interact with other services, security and otherwise. Services querying VOPS should have access to VOPS public certificate to be able to check authenticity of its answers. Currently one AEM service is interacting directly with VOPS - **ResMng**.

It is important that if VOPS is to enforce policies over user queries, **RCA client** must run on resource node which is considered in the query. VOPS needs to access RCA client service to obtain resource certificates from which VO attributes are read and used in the query.

Interactions with other services in the current implementation are presented in figure 12.

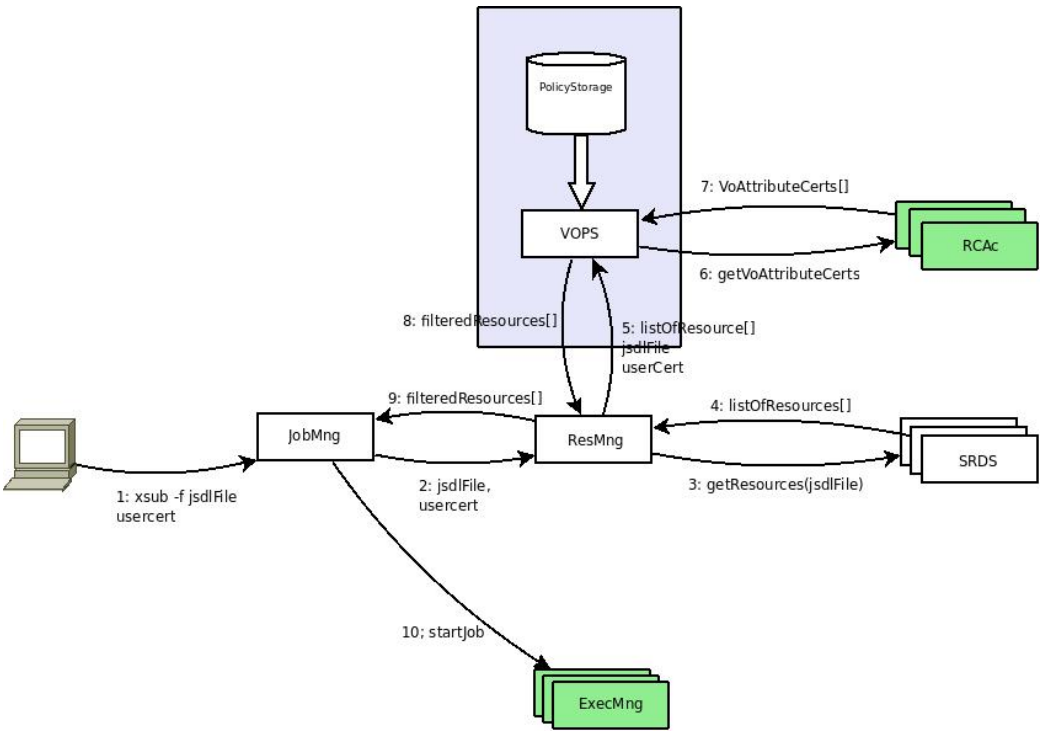

Figure 12: VOPS Interactions with other services.

3.2.4 Supporting XtreemOS Application Execution

VOPS provides easy access point to decision point for Application Execution Management services. Currently AEM's contact point for resource selection, **ResMng**, contacts VOPS for policy decisions and these decisions are forwarded towards AEM's **JobMng** service.

3.2.5 Supporting XtreemOS File Management

Currently VOPS and XtreemFS do not interact. However, VOPS could provide decision point for XtreemOS File System to perform policy enforcement over user's actions to enforce rights on VO level, e.g. *u*ser can not use storage if he is over his quota on VO level.

3.2.6 Supporting MD

Currently, VOPS does not use any service provided by WP3.6.

3.2.7 Supporting WP3.2's Services

Currently VOPS does not contact any of the services provided by WP3.2, nor does provide any interaction directly.

3.2.8 Features for the Next Release

A number of features are planned to be incorporated in the upcoming release. These are described as follows.

Structured Organization of Policy Store We shall support **eXist** database [11] which will enable easier policy filtering and management. With incorporating **eXist** as XML database engine and using more standard communication methods (such as JSON, since DIXI will provide JSON gateway) flexibility of policy languages will be increased and VOLife web application will be able to manage policies easily through e.g. XQuery statements. EXist has also simple backup/restore procedures which can be easily triggered from VOPS. During backup, eXist exports the contents of its database (as standard XML files) to a hierarchy of directories on the hard drive. This hierarchy is organized according to the organization of collections in the database. This way the whole structure of XACML policies will be backed up and could be restored when administrator wishes to run maintenance procedures (or if automatically maintenance will be scheduled to run periodically).

Incorporation of Dynamic VO Information In the current release, most of the information being taken into VOPS PDP is grabbed statically. This is largely due to the static nature of the VOs being implemented in the first release. However, as VOs become dynamic (as it will do in the upcoming releases), For example, VOPS could also take into account the dynamic aspects of VOs, including the dynamic addition and removal of resource nodes through the duration of executing a job. Such dynamism occurs when advanced job execution (such as job migration) is required.

Therefore, in order to support dynamic VOs, the integration of VOPS with other services will be completed: Scalable Resource Discovery System (SRDS), VOLife Web Application, support for decisions based on AFW attributes, which can be incorporated into JSDL (protocol, IP, port ranges) will also be available. To support SRDS (WP3.2), filtered policies will be filtered out from policy storage based on JSDL and user certificate (without information about resource). User policy will be formed (with rules applying to user) and will be forwarded to SRDS and help with resource discovery. This way static and dynamic attributes will already be taken into account when filtering resource nodes and constructing potential list of resources, which will be eventually sent into VOPS PDP for final policy enforcement. Final VO level decisions will be made, taking into account accounting and monitoring services information (policies on resource usage will be enforced via monitoring services).

Interaction with accounting service Since monitoring and auditing design will be developed in close conjunction with WP3.3 AEM development, AEM services will inform of the temporary and historical status of the jobs. This information will be used to enable more general VO level decisions such as: user can not consume more than specific amount of CPU time or consume more than specific amount of disk usage in the VO he is part of.

Advanced Access Control Support There is also an issue with access control in VOPS. In the current implementation access control is enforced in VOPS by confining requestors to be VO owners and/or resource administrator roles (based on their identity certificates). Users (administrators) with these roles are permitted to execute PAP commands (if access control is enabled through configuration file, after which VOPS must be restarted). However, in order for access control to be more flexible and consistent throughout the system, it chould be done through the use of native XVOMS access control mechanisms.

Dealing with Policy Conflicts Policy conflict detection tools in STFC (developed under EchoGrid project) could also be used to validate (new) policies and rules when they are being added into the policy store.

3.3 RCA

3.3.1 Brief Introduction

Resource Certification Authority (RCA) is a security service that provides a base to bootstrap the trust of resource nodes in XtreemOS. Organizations can provide computational resources to allow users to exploit the computational capabilities offered by their machines. In XtreemOS, the resource exploitation is facilitated by the Application Execution Management (AEM) services. The services need a decentralised way to ensure that the target resource nodes are trustworthy. RCA issues machine credentials that peer nodes and services can check.

3.3.2 Major Components

In logical terms, the RCA is a core-level service which issues machine certificates upon request. However, the life cycle of a machine certificate and the whole mechanism of gaining the possibility to have a certificate issued lead to implementing RCA as an **RCA Server** service, **RCA database**, and an **RCA client** service.

RCA Server. This is the service that provides the essential functionality of the resource certification. The service runs core-level, and its main purpose is to

receive clients' requests for issuing certificates, check the validity of the requestee, and sign the certificates. The RCA Server comprises of the following:

- The RCA server logic.
- The front-end for both the RCA server logic and the RCA database implemented for DIXI [5].
- The service certificate signed by the organisation's root certification authority or another authority with the organisation's root CA in the signature chain.

RCA DB. This is an implementational unit which stores the list of the resources and their states. The possible states of each resource is as follows:

- Unregistered. This is a record of a resource and its details as published by the resource administrator.
- Registered. This record describes the resource and also lets the RCA Server know that the resource is trustworthy and can be issued machine certificates.
- The membership of the resource in VO. A resource can take part in any number of VOs, as long as it has a status Registered.

We exposed the RCA DB functionality using the RCA Server' DIXI service front-end.

RCA Client. The RCA Server is a stand-alone unit, and in principle the users could interact with it manually or using small programs. We provide the RCA Client as a DIXI service that runs on each node and eases the administration steps required when using the RCA. Its functionality includes the following:

- Creation of unique private keys and certification requests.
- Obtaining the details of the node from AEM's Resource Monitor service. This saves the resource administrator the tedious collection and entering the resource's details.
- Communication with the RCA Server service to send the registration application, registration confirmations, and reception of the sign certification.
- Saving, examining and installing the machine certificates.

3.3.3 Usage of RCA

The RCA issues machine certificates, encrypted using a so called machine password. This means that in principle the resource administrator is never prompted for the password needed to access the private key part of the machine certificate. However, if an admin would want a higher level of security, she would need to enter the password at each boot time or service start-up. The cycle of the initial sending the resource's details, registring the resource, requesting for certificate and storing it could therefore be done automatically. However, this would not ensure trust and security in any way. Therefore we want to have people with authority taking part within this loop. Figure 13 shows a diagram with the interaction between actors the components of the RCA, illustrating the usage of the RCA. As we can see, this includes

- **The resource administrator**, who is the person in charge of a node (a machine) that is capable of providing services for job execution in the XtreemOS. Before the node can take part in a VO, it needs to be registered with the RCA and obtain the machine certificates. This actor decides whether a machine can be used for running jobs in any of the VOs the machine has been included in.
- **The administrator** here is a person of an authority within the organisation that runs the RCA. The administrator is in charge of checking the applications for including resources in any potential VOs, e.g., by checking the resource administrator's background and the resource's usage within the organisation and outside XtreemOS's VOs.
- **The VO owner** can browse the list of resources registered by the resource administrators and approved by the administrator, and invite resources to take part in the VOs. The VO administrator can also remove the resources from VOs.

Figure 14 shows the usage timeline with the interaction between administrators and the services. The figure shows that once the administrator confirms the registration of a resource with the RCA DB, a VO administrator can find the resource on the list of the registered resources and use the entry to be added to a VO.

Certificates. RCA issues two types of certificates:

- **Resource identity certificate.** This certificate is a standard X.509 V3 certificate. In its extensions it contains the resource attribute values. The certificate can be used for identifying the resource, to authenticate services

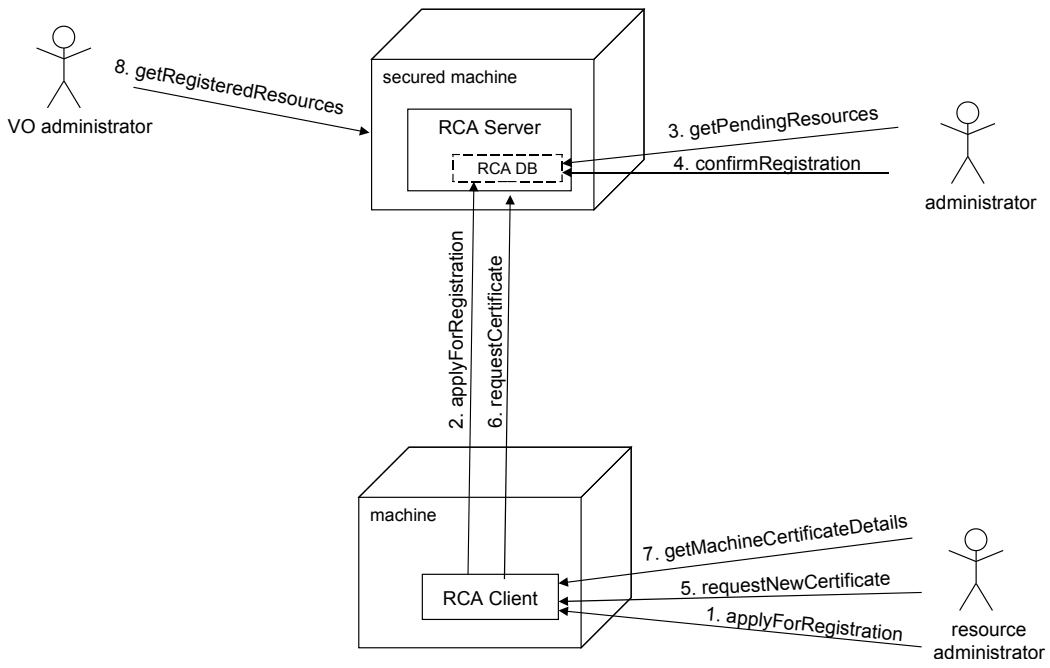

Figure 13: Interactions between resource admin and RCA server

running on the node, or to use for encryption, e.g., in an SSL communication session. The duration of the certificate can be set to a short time to ensure the security levels according to the policies of the organisation.

- **Resource's VO attribute certificate.** This certificate contains the resource's identity descriptors and an identifier of one VO that the resource is a member of. The resource's services prove the resource's trustworthiness within a VO. Each resource can have multiple VO attribute certificates.

3.3.4 Interactions with Other Security Services

The focus of the RCA Server is to provide targeted nodes with the resource identity certificates and other attribute certificates, if the resource is eligible to obtain the certificates. This means that in the current implementation there is no interaction with any other security services.

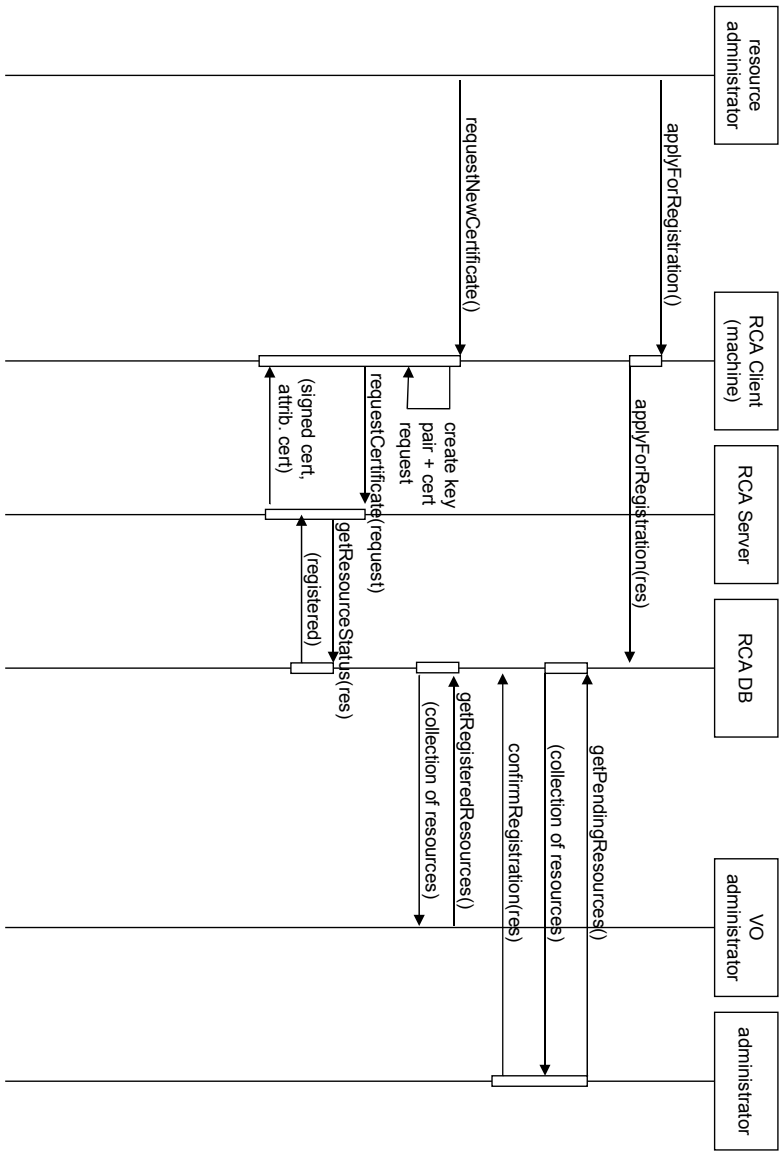

Figure 14: Interactions between a machine and RCA server

In the process of resource discovery and selection, VOPS needs to check the authenticity of the candidate resources as provided from ADS. To do this, the VOPS service interacts with the RCA Client service, which has an access to the

node's public key. In particular, the VOPS needs to check whether the node had been certified for the use in the VO that the user has selected for running a job. RCA Client passes the resource's VO attribute certificate as credentials in the VO.

3.3.5 Supporting XtreemOS Application Execution

The AEM's execution benefits from the RCA's services because RCA provides the following:

- **Resource identity certificate** can be used for encrypting DIXI communication and, indirectly, provides the AEM the means to secure sensitive data sent in the service calls, and to authenticate peer node.
- **Resource's VO attribute cerificates**, when placed in the RCA Client's search path, enable the node's taking part in the job execution for the VO. If the resource administrator would like to prevent further job execution for a particular VO, she only needs to remove the respective certificate from its installed path.

3.3.6 Supporting XtreemOS File Management

RCA and XtreemFS currently do not interact. However, XtreemFS can, however, use the certificates issued by the RCA server, to encrypt the communication using SSL and to authenticate nodes.

3.3.7 Supporting MD

Mobile devices, by design, are client nodes, while the RCA is targeted for the worker nodes. We have therefore not forseen any support for the MD at this point.

3.3.8 Supporting WP3.2's Services

RCA is currently a VO-level service which does not use any services from the WP3.2.

3.3.9 Features for the Next Release

In the first release, there was no communication between RCA (DB) and X-VOMS, partially because X-VOMS has not yet exposed its interfaces to other services. Having the information on resources and their membership within RCA DB was a good and quick solution which worked well for a proof-of-concept prototype. However, as the development of X-VOMS progresses, we can forsee merging of the RCA DB with the X-VOMS, or use ADS's services.

4 Conclusions

This deliverable represents the outcome of an ongoing process of consolidating the design decisions we have made for the first release, digesting the lessons we have learned from delivering the first release, and presenting a preliminary roadmap thinking we have done for the next release of XtreemOS security and VO services. These ideas will be validated in the next phase of our implementation.

This work is still ongoing. In the time to come, we will also plan to continue to revise the security architecture presented in the last edition of this specification, especially by incoporating the design decisions we shall be making for the next release.

References

[1] Andrea Ceccanti. The VOMS Architecture, 2008. https://twiki.cnaf.infn.it/cgi-bin/twiki/view/VOMS/WebArchitecture.

[2] XtreemOS Consortium. *D3.1.7 - The First Version of System Architecture*. Work Package 3.1, December 2007.

[3] XtreemOS Consortium. Design and implementation of node-level vo support. In *XtreemOS public deliverables - D2.1.2*. Work Package 2.1, November 2007.

[4] XtreemOS Consortium. Second specification of security services. In *XtreemOS public deliverables - D3.5.4*. Work Package 3.5, November 2007.

[5] XtreemOS Consortium. Revised system architecture. In *XtreemOS public deliverables - D3.1.7*. Work Package 3.1, Dec 2008.

[6] S. Farell and R. Housley. Rfc 3281 - an internet attribute certificate profile for authorization. IETF, April 2002.

[7] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid Enabling Scalable Virtual Organizations. In *International J. Supercomputer Applications*, 2001.

[8] A. Grimshaw, M. Lewis, A. Ferrari, and J. Karpovich. Architectural Support for Extensibility and Autonomy in Wide-Area Distributed Object Systems, 1998. http://legion.virginia.edu/papers/CS-98-12.pdf.

[9] R. Housley, W. Polk, W. Ford, and D. Solo. Rfc 3280 - internet x.509 public key infrastructure certificate and certificate revocation list (crl) profile, April 2002.

[10] John Linn. Trust Models and Management in Public-Key Infrastructures, 2000. http://citeseer.ist.psu.edu/386363.html.

[11] Wolfgang Meier. eXist - Open Source Native XML Database. http://exist.sourceforge.net/.

[12] The EU DataGrid Project Partners. The EU DataGrid project website. http://eu-datagrid.web.cern.ch/eu-datagrid/.

[13] The EU EGEE Project Partners. The EU EGEE project website. http://public.eu-egee.org.

[14] The people of linux admin@vger.rutgers.edu. Linux Administrators FAQ List, Q 6.17. http://www.tigerteam.net/linuxgroup/linux-admin-FAQ/Linux-Admin-FAQ-6.html.

[15] The Globus Team. Overview of the Grid Security Infrastructure, 2008. http://www.globus.org/security/overview.html.

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

[17] Wikipedia. The PKI Wiki Page. http://en.wikipedia.org/wiki/Public_key_infrastructure.