Project no. IST-033576

XtreemOS

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

Design of Basic Services for Mobile Devices

D3.6.2

Due date of deliverable: May 31st, 2008
Actual submission date: May 27th, 2008

Start date of project: June 1st 2006

Type: Deliverable
WP number: WP3.6
Task number: T3.6.2

Responsible institution: Telefónica I+D
Editor & and editor's address: Luis Pablo Prieto
Telefónica I+D
Parque Tecnológico de Boecillo
47151 Boecillo (Valladolid)
SPAIN

Version 1.0 / Last edited by Luis Pablo Prieto / May 22nd, 2008

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

Revision history:

Version	Date	Authors	Institution	Section affected, comments
0.1	13/02/08	Luis Pablo Prieto	TID	Initial template
0.2	26/02/08	Daniel Galindo & Luis Pablo Prieto	TID	Detailed outline of the document
0.3	16/04/08	TID Team	TID	First draft of security and execution chapters
0.4	17/04/08	Luis Pablo Prieto	TID	First draft of API chapter, plus minor formatting changes
0.5	18/04/08	Jesús Malo	BSC	First draft of the data chapter
0.6	21/04/08	TID Team	TID	First draft of the modifications chapter
0.7	22/04/08	Jesús Malo	BSC	Corrections to the data chapter
0.8	22/04/08	Luis Pablo Prieto	TID	First draft of introduction, conclusions, future work and executive summary
0.9	24/04/08	Luis Pablo Prieto	TID	Fixed missing parts, references etc. First complete draft
0.91	06/05/08	Luis Pablo Prieto, Manuel Martín	TID	Spellchecking
0.92	07/05/08	Luis Pablo Prieto, Daniel Galindo	TID	Spellchecking and minor corrections
0.93	16/05/08	Luis Pablo Prieto	TID	Incorporated reviewer's comments. Still need some polishing
1.0	22/05/08	Luis Pablo Prieto	TID	Final state. Ready for submission

Reviewers:
Matthias Hess (NEC), Ian Johnson (STFC)

Tasks related to this deliverable:

Task No.	Task description	Partners involved°
T3.6.2	Design of basic services for mobile devices (XtreemOS-G for MD/PDA)	TID*, BSC

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

Executive Summary

Mobile access to grid services is a research field that has not been yet fully solved, either in scientific or industrial research. Among the most common solutions are the usage of grid portals or grid gateways to translate grid requests from their native form to more efficient and mobile-friendly ad-hoc protocols.

In the XtreemOS project, however, a different approach has been proposed, taking advantage of the latest advances in mobile device capacities and leveraging the scalability and transparency features of XtreemOS, like native VO support, usage of POSIX standards or scalable and highly available services. With this approach, the mobile devices using XtreemOS "speak the same language" as any other grid node, transforming them into first class consumers of grid resources. This also is a much more scalable solution and opens the door to new, grid-transparent use cases that could appeal to the mass market.

In XtreemOS-MD, this goal of universally accessible grid services is achieved through two different software layers: a foundation layer that enriches the operating system with virtual organization support, and a grid services layer which implements the grid services themselves. This document describes the design of a first, basic version of XtreemOS grid services (XtreemOS-G) layer, designed for PDAs. This layer contains the necessary functionality for accessing grid files and launching and managing grid jobs in a secure manner, as well as the necessary interfaces for grid applications to access these features.

This grid software package will be composed of a number of standard XtreemOS modules, which will be ported and modified to run in the restricted PDA platform in a more efficient and usable way. These modules are: a utility for obtaining user certificates (CDA client), a way of submitting and managing jobs in the grid (XATI), a way of accessing grid files and volumes (XtreemFS client) and an application interface for accesing these functions (XOSAGA API). Also, a number of additional mechanisms and utilities have been devised to make the process of using XtreemOS grid features more intuitive and user-friendly.

The design of these components also contemplates the avoidance where possible of Java components in favour of native Linux alternatives, given the current limitations of Java in mobile devices. However, these native components should be nonetheless portable to any other modern Linux mobile platform, so that Linux integrators are attracted to incorporate XtreemOS-MD in their solutions.

Moreover, the work in these modifications has been devised in a way that makes them as generally applicable as possible. Thus, it is expected that many of those modifications and additional utilities will also be applied to other XtreemOS flavours, in order to obtain an even more streamlined and usable grid operating system.

Contents

Glossary

API	Application Programming Interface
AEM	Application Execution Management
CDA	Credential Distribution Authority
FUSE	Filesystem in Userspace
MRC	Metadata and Replica Catalog
OSD	Object Storage Device
PDA	Personal Digital Assistant
PIM	Personal Information Management
POSIX	Portable Operating System Interface
RMS	Replica Management Service
RSA	A kind of private key algorithm
SAGA	Simple API for Grid Applications
VO	Virtual Organization
VOM	Virtual Organization Manager
XATI	XtreemOS Application Toolkit Interface
XATICA	C implementation of the XATI
XtreemOS-F	XtreemOS foundation Layer
XtreemOS-G	XtreemOS Grid services Layer
XtreemFS	XtreemOS FileSystem
XtreemOS-MD	XtreemOS for Mobile Devices
XOSD	XtreemOS Daemon

Chapter 1

Introduction

Mobile access to grid services is a research field that has spawned a number of publications and projects [4, 6, 5], but which has not been yet fully solved. Among the most proposed solutions are the usage of grid portals or grid gateways to translate grid requests from their native (mostly service-oriented) form, to more efficient and mobile-friendly (but mostly *ad-hoc*) protocols.

In the XtreemOS project, however, a different approach has been proposed, which takes advantage of the latest advances in mobile device capacities (e.g. increased CPU and memory), but also leverages the scalability and transparency features of XtreemOS (e.g. native VO support, usage of POSIX standards, scalable and highly available services, etc). In this approach, the mobile device version of XtreemOS (from now on, XtreemOS-MD) "speaks the same language" (i.e. uses the same protocols) as any other grid node, being thus promoted to first class consumers of grid resources. This also provides the advantage of being a much more scalable solution than going through the bottleneck of a limited number of grid gateways/portals, and opens the door to new, grid-transparent use cases that could appeal to the mass market.

In XtreemOS-MD, this goal of universally accessible grid services is achieved through two different software layers: **XtreemOS-F**, a *foundation* layer that enriches the Linux operating system with VO support features, to serve as a base for **XtreemOS-G**, the *grid services* layer that implements the grid functionality itself (execution and data management, security services, application interfaces...).

This document describes the design of a first, basic version of this XtreemOS-G layer, designed for Personal Digital Assistants (PDAs). It includes the necessary modifications that several XtreemOS modules will undergo in order to work (and do it efficiently) in a mobile device's limited executing environment.

1.1 Methodology

During this design process, a number of XtreemOS components have been selected, as they were deemed necessary for the basic features that XtreemOS-MD

has to provide, according to a number of use cases (see D3.6.1 [20]). These components have been revised and a preliminary port to the ARM architecture (both through emulators and with real ARM devices) has been attempted, to *detect* the most obvious obstacles for the execution in a PDA.

After this first analysis was done and solutions to these most pressing problems have been proposed, a second layer of modifications has been designed, to *optimize* the operation of those components under mobile conditions. In this phase, a number of additional needs in the mobile environment have been detected, and *additional software components*, not related to any XtreemOS-specific system, have been proposed to fulfill these needs.

Finally, all this information has been gathered in the present document, where the different components and their modifications are explained, along with their interfaces and a number of usage examples.

1.2 Contents of the document

This document is organized in the following manner:

In chapter 2, a brief overview of the architecture, use cases and derived requirements is provided for completeness, in order to clarify which XtreemOS components are needed and why.

Afterwards, chapters 3, 4, 5 and 6 describe the four main XtreemOS components that will be needed in the mobile version of XtreemOS-G, including their functionality, internal architecture, interfaces and examples of use.

Chapter 7 contains the description of a number of additional components that will be useful in mobile grid environments, but which do not relate directly to any XtreemOS system.

Finally, chapter 8 collects the main conclusions of the document, and chapter 9 highlights the major next steps in the implementation of XtreemOS-G, also pointing out some possible future research paths.

Chapter 2

XtreemOS Grid Services for Mobile Devices

In order to fully understand the scope and motivation of the grid services to be offered by XtreemOS for Mobile Devices, which will be described in the following chapters, we must understand which use cases should be covered by them, and which requirements are to be met. Here we will briefly summarize this information, which is more thoroughly covered in deliverable D3.6.1 [20]. Also, to get a better understanding of how these services fit in the overall picture of the grid operating system for mobile devices, a summary of the architecture of XtreemOS-MD is also presented here (see D3.6.1 [20] and D2.3.3 [17]).

2.1 Use cases and requirements

As a previous step to defining the design and architecture of XtreemOS-MD, and even before the requirements for it could be set, a number of **usage scenarios** for XtreemOS-MD were gathered, analysed and categorised, to ascertain the most suitable implementation order.

These scenarios, which are detailed in previous deliverables, can be roughly divided into two big groups, from the point of view of the user and her degree of knowledge of the underlying grid infrastructure (advanced and optional use cases are presented in *italics*):

Grid-aware use cases: in this group, grid users (e.g. people from corporations or academic institutions that usually work with grid technology) gain mobile access to the XtreemOS grid. As transparency is one of XtreemOS's main design principles, this access will be done in a more transparent way than with conventional grid middleware, but users will be conscious, to a certain extent, that there are concepts like jobs or certificates being manipulated in the background. These scenarios include:

- A user will be able to **login to the XtreemOS Grid**. From then on, the user is enabled to operate into the Grid according to his role and permissions.
- A user will be able to **launch and manage jobs** to be executed in the Grid. During execution, the user will be notified of the job's main events, and she will be able to operate on the job.
- *A mobile user will be able to **manage VOs** from her mobile device, if she has adequate permissions for doing so. This includes adding/deleting users, creating VO subgroups, setting resource policies, etc.*
- *A user will be able to **search and manage grid resources**: getting a list of resources available to her, according to certain parameters given by the user. If she has adequate permissions, she could also define what resources to share in the Grid, as well as monitor its usage.*
- *A user will be able to **manage grid data**, specifying where data will be stored (logically), the access pattern, privacy level, places where data can be stored (physically), required level of consistency and coherence, etc.*
- *The user will be able to **transfer an ongoing grid session** to a nearby device that belongs to the same VO without interruption.*

Grid-transparent use cases: in this group of use cases, users that have no knowledge or interest in the underlying grid technology experience the benefits of XtreemOS in a completely transparent way. These scenarios logically require functionalities that must be provided to fulfill the grid-aware use cases (e.g. basic grid infrastructures) and thus, they build upon the previous ones. In fact, they are even more interesting, since they showcase mass market applications of grid technology. These scenarios include:

- A user will be able to securely communicate with other grid users in the same VO, by using a **grid-integrated instant messaging application**. This application will include XtreemOS authentication and security mechanisms, and will be able to store user preferences and chat history in XtreemFS grid filesystem.
- A user will be able to transparently **store, search and retrieve files from the grid**, stored in the XtreemFS filesystem.
- A user should be able to use **grid-based advanced voice recognition systems**, to make up for the lack of an adequate keyboard interface for entering text.
- *A user will be able to interactively participate in grid applications and workflows, according to the specific user context. This effectively would allow **people to become grid resources**.*

- *A user should be able to establish and interact into **secure Mobile Ad-hoc Networks (MANets)**, by using XtreemOS certificates and security infrastructure, in case a direct access to the Internet is not available.*
- *A user should be able to share and use **voice communication channels as grid resources** (e.g. mobile telephonic communication).*
- *A user should be able to participate in **mobile multiplayer online games** that utilize the XtreemOS technology, in a secure and billable manner.*

***NB:** The grid systems represented by these use cases will not be developed by the XtreemOS project (further than the operating system layers), since developing new grid applications is mostly out of the scope of the project. Two proof-of-concept applications (a job managing application and an instant messaging application) will be implemented in WP4.2 [15].*

Once these use cases were defined and prioritized, a number of **requirements** for the grid services layer of XtreemOS-MD became clear, specially when a number of other requirements stemming from XtreemOS reference applications were added in D4.2.3 [15]. The requirements for the basic version of XtreemOS-MD include [20]:

- The ability to obtain tracing information from grid applications (e.g. execution and resources information) (R3.6.1).
- To provide IPv6 support (R3.6.2).
- The ability to manage grid jobs (R3.6.4).
- The ability to monitor grid jobs (R3.6.5).
- To identify mobile nodes as special grid nodes that do not share resources (R3.6.6).
- The ability to access files in XtreemFS (R3.6.12).
- The ability to mount XtreemFS volumes as any other filesystem (R3.6.13).
- To ensure integrity in filesystem operations from mobile devices (R3.6.14).
- To allow for grid (VO) user authentication (R3.6.17).
- To provide users with single sign-on capabilities (R3.6.18).
- To provide independence between local and grid user accounts (R3.6.19).
- To allow for users to belong to multiple VOs (R3.6.20).
- Means of having confidential communications (R3.6.21).
- To provide a SAGA job management API (R3.6.24).

- To provide a SAGA data management API (R3.6.25).
- To provide a SAGA-based security and VO support API (R3.6.26).
- Java support[1].

To these requirements, we should implicitly add other hardware, network and software requirements for the foundation layer on which XtreemOS-G is based (e.g. usage of a high-end PDA with wireless connectivity, 200+MHz of CPU, 128MB of RAM, additional permantent storage, a modern mobile Linux operating system with virtual organization support etc). These requirements are discussed in more detail in WP2.3 documentation [25, 21].

2.2 XtreemOS-MD Architecture

Having all these use cases and requisites in mind, and also taking into account the structure and implementation of the underlying mobile Linux distribution of choice in XtreemOS-MD (Ångström), a software architecture of an XtreemOS-MD node has been drawn [17]. This architecture is reflected in figure 2.1:

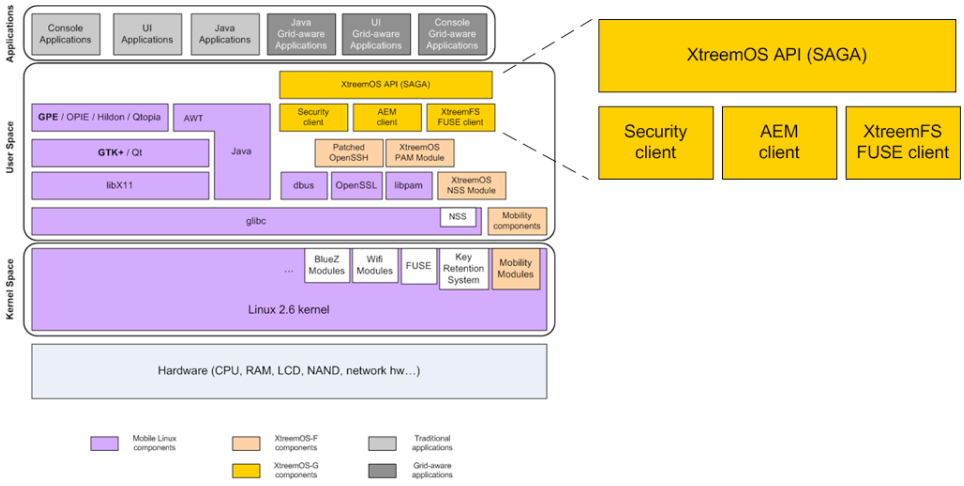

Figure 2.1: XtreemOS-MD Software Architecture

As it can be seen, apart from typical Linux components like the kernel, C libraries or graphical toolkits, XtreemOS-MD also has a number of basic VO support and mobility modules that comprise the foundation layer of XtreemOS-MD. These components provide the necessary OS-level support for transparent terminal mobility (through Mobile IPv6 protocol) and VO support to the upper software layers. They are developed in WP2.3, and a more thoroughly description of them can be seen in D2.3.3 [17].

[1]This requirement is currently being reconsidered, due to poor mobile support for newer Java features.

On top of this foundation layer we have the mobile grid services and functionalities that are the focus of this deliverable, the so-called XtreemOS-G layer for Mobile Devices. This layer provides access to the XtreemOS grid services that are being executed, for the most part, in more powerful (fixed) machines.

This layer is composed of the following modules:

Application Execution Management (AEM) client As we have seen in the use cases, one of the main objectives of mobile grid users is to be able to launch jobs, check their status and get information about available resources.

In the concrete case of XtreemOS, this will be implemented by offering XATI, the interface to contact the XOSD (XtreemOS Daemon, handling resources and jobs) that can be located in any other node. With this interface we can build all needed applications to submit jobs and check the status of resources and jobs.

More concretely, the mobile version of this module will be a slightly modified version of XATICA, the C implementation of the XATI interface, that has been developed in WP3.3 [19].

XtreemFS client Regarding the grid filesystem XtreemFS in this MD version, we only need a mechanism to "mount" a XtreemFS volume and be able to access grid files for both reading and writing. This service, in the standard version, is managed through a FUSE module that receives all file operations and redirects them to the right services in XtreemFS (MRC, OSDs, and RMS) [14, 16]. For mobile devices, a similar library to redirect all these file operations to the XtreemFS services will be implemented.

It is worth noting that allowing other users in the grid to access files located in the MD is not initially planned as a feature of the MD flavour. If these files are to be accessed from outside the device, they should be copied first to a XtreemFS volume.

Security client In order to provide the kind of access needed by the aforementioned use cases, a XtreemOS mobile device node will only need a way to access the Credential Distribution Authority (CDA) to obtain a XtreemOS certificate for the user's grid identity (XOS-Cert). With this XOS-Cert stored in the node (by the VO support components underneath developed in WP 2.3 [17]), the user will be able to access, for example, the AEM and the XtreemFS services.

For this purpose, XtreemOS-MD will have a modified version of the C implementation of the CDA client, developed in WP3.5 [24].

XtreemOS (SAGA) API The XtreemOS API will be based on the Simple API for Grid Applications (SAGA) API, as detailed in the API deliverables [11, 22, 10]. This API is aimed at high-level application developers who don't want to know the fine details about Grid computing or the middleware/infrastructure used, but want to make use of distributed resources through the Grid.

The structure of SAGA is modular, so as to be extended whenever needed. From the use cases, we can infer that a mobile XtreemOS node will have, at least, the following components of the API:

- The Look & Feel API, as it is needed by all the other APIs, or at least the parts of it needed by the components detailed below.
- The SAGA Engine, as it is also a core element, or at least the parts of it needed by the components below.
- The Namespace package and its corresponding XtreemOS Adaptor, as it is referred by all the other packages.
- The Job package and its corresponding XtreemOS adaptor (or a subset of it, as needed by the use cases).
- The File and Logical File packages, and their corresponding XtreemOS adaptors (or the subset needed by the use cases).

Optionally, this API can be extended or stripped down as much as possible, to fit the restrictions of a mobile device environment (e.g. disk space or memory consumption).

Other XtreemOS-MD enhancements Apart from the aforementioned modules, designed specifically to communicate with XtreemOS grid services, a mobile node running XtreemOS could benefit from a number of special modifications that cannot be included in the previous modules, as they have a more transversal nature. These modifications include:

- A way of **starting XtreemOS services and processes on-demand** just when they are needed, in order to save the scarce memory and CPU resources of the device. This can take the form of a kind of "launcher application/wrapper" that checks the existence of the necessary processes and data sets (for example, the presence of a XtreemOS certificate), and launches them if not present.
- A way of **stopping XtreemOS services** in the device when they are not needed anymore, again in order to save resources. Following typical mobile procedures, this can be implemented as a desktop applet that can be used when services are not needed anymore.
- In order to save disk space, it would also be useful to have only the bare minimum of XtreemOS functionality installed in the device. If more functionalities are required, they can be **installed on-the-fly**.

Chapter 3

Common User Management (security client)

Security and security-related mechanisms play a central role in many (if not all) grid systems, and XtreemOS is not different in this regard. As described by WP3.5 [13], there are a number of security areas that are important in typical grid operation:

Authentication Is the user who she claims to be?

Authorisation Should the user be allowed to do what she wants to do?

Confidentiality Communications and data should only be read by authorised users (e.g. by using encryption).

All these processes are needed, either implicitly or explicitly, by *all* the use cases described in chapter 2, although depending on the concrete context of the scenario, some of them could be optional (for example, we could decide that confidentiality is not as important as saving computational resources in our mobile IM application, and thus forego the encryption of communications).

In XtreemOS, confidentiality is attained mostly by standard encryption methods in the communications (using secure protocols like SSL/TLS), and by restricting access to data (e.g. files in the XtreemFS filesystem) only to authorised users.

Authorisation is performed, for the most part, in XtreemOS nodes and servers, which check the credentials presented by the user and applying a number of policies defined by the Virtual Organization to which both the user and the resources she is attempting to use, belong.

Thus, we finally come to *authentication*, the single most important aspect of security that occurs in the client-side of XtreemOS. To demonstrate the identity of the user, in XtreemOS it is used a certificate-based solution, using what is called XOS-Certs (a kind of X.509 certificates), which are issued by a Credential Distribution Authority (CDA) that serves credentials for the VO. Using typical PKI techniques,

any grid entity can check that the certificate is valid (at least for a certain period of time) and thus, that the user is who she claims to be.

In this chapter we will describe the functionalities and interfaces of the main XtreemOS-specific security module which should be present in mobile devices, a modified version of the CDA client. Other mechanisms like encryption for confidentiality are implemented by standard Linux modules such as OpenSSL [8].

3.1 Features and functionalities

XtreemOS users wanting to use grid resources have to **obtain a XOS-Cert** from the CDA, so that PKI methods can be used by nodes and servers to authenticate them as valid VO members. These XOS-Certs are currently implemented as X.509 certificates, with the CDA of the Virtual Organization acting as the primary Certificate Authority (CA), and have a limited time validity (VO-configurable, default value being currently 30 days) [23].

3.2 Design alternatives

CDA client vs. MyProxy When designing a way of getting certificates from a mobile device, two main options were considered:

The most obvious approach was to have in the mobile device a **native CDA client** that directly asks the server for a certificate, implementing the security protocols for doing so that were specified in WP3.5. This option has the advantage of being quite simple, conceptually and implementation-wise. But it also has a disadvantage, which is that certain "heavy" operations (like the generation of random keys), are computed in the mobile device, leading to less-than-optimal response times (e.g. latest evaluations of this operation yielded delays of 2-8 seconds).

Another option was the usage of **MyProxy-like** solutions for the acquisition of proxy certificates. In this kind of implementation, the mobile device obtains a short-term proxy certificate from a MyProxy server [7], who has previously obtained the "real" XOS-Cert from the CDA. This MyProxy would run in a more powerful (e.g. PC) node, thus freeing mobile devices from some of the heavy operations involved in getting the certificate. However, the implementation of a MyProxy solution involves more complicated protocols and the inclusion of a new entity in the security chain (the MyProxy server), and would require careful consideration and a more costly implementation.

After several conversations with WP3.5 members, it was decided to **use a CDA client** in a first phase, and study the feasibility of a MyProxy solution for an advanced stage of the project.

Java issues The work in WP3.6 has been mainly directed in this case to make the CDA client more adequate for a mobile device environment (see "Specific modifications for mobile devices" below).

One of these alternatives has been the choice of **programming language** for the client. Initially, WP3.5 developed a Java prototype of the CDA client. But first evaluations on mobile platforms revealed the poor compatibility of open mobile Java Virtual Machines and libraries with newer Java standards (the Java CDA client required Java 6, while most mobile open source Java libraries like GNU classpath [1] do not even completely support Java 5).

This fact led us, after conversations with WP3.5 members, to use a **C implementation** of the client, provided by WP3.5 and ported and modified by WP3.6.

3.3 Architecture and design

The architecture of this component is really simple. As WP3.5 implemented it, it is a single command-line executable which asks for the certificate to a well-known CDA server, using the username, passphrase and VO identifier that are passed as command line parameters. The protocol roughly follows this sequence:

1. A random private key is generated by the client
2. A certificate request for the CDA is constructed, and later signed using the generated private key
3. A SSL connection to the CDA is established
4. The client authenticates with the server using the provided username and password
5. The client asks for the certificate to the CDA, specifying the VO and passing the certificate request generated in step 2.
6. After the new XOS-Cert is received, the client verifies its validity

Using this implementation as a base, in WP3.6 a number of modifications have been made, which will be described in the following sections.

3.4 Specific modifications for Mobile Devices

As we faced the implementation of the CDA client in mobile devices, and after a few preliminary tests with the software in the XtreemOS-MD development environments, we could foresee a number of aspects that will require modifications to better fit in a mobile environment:

CDA client as a library The CDA client as it was provided by WP3.5 was just a command-line executable. In mobile devices (and probably also in other

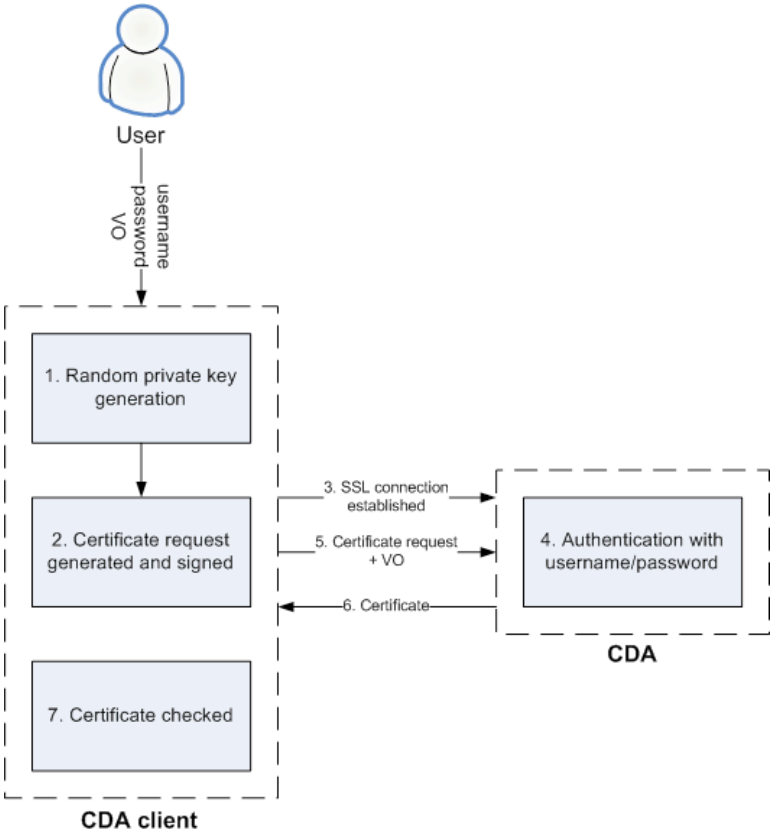

Figure 3.1: CDA protocol

XtreemOS flavours) it would be very useful to have the same functionality in library form, so that it can be used by other applications (such as `startxtreemos`, see chapter 7) or by a Pluggable Authentication Module (PAM).

The idea behind this modification is that the certificate requests can be issued implicitly when they are needed, instead of the user having to invoke the CDA client whenever a new XOS-Cert is needed. In the limited user interface (and possibly, knowledge) that is available to mobile users, this is certainly a must. Examples of this function in use may include slightly modified Telnet/SSH servers, or the kind of grid application launcher/wrapper described in more detail in chapter 7.

Code cleanup/customization for mobile devices A number of slight modifications have been devised in the original code in order to streamline the process of the request, specially for minimising the number of user interactions, disk writes and ciphering operations. Also, the code will provide more configurability about policies regarding the way and timeframe of the disk storage of the certificates.

Optional SSL communications Given the limited amount of processing capacity and memory available in mobile devices, and the short response times (almost real-time) needed in this kind of environments, it would be advisable to allow the deactivation of the SSL communication with the CDA. This would leave the process open to potential eavesdropping, but this can be acceptable for certain scenarios where confidentiality is not a big issue. An example of this would be when communicating with a CDA server or a key-generation proxy (see below) through a secure local network (secure LAN, USB/point-to-point, WPA/EAP Wifi link, etc), or in scenarios where the security is not a concern at all.

This option would be a fundamental change in how credentials are distributed, and would reduce the level of confidence of services which receive an XOS-Certificate from a mobile user. Also, this would imply sending the user's username and password in the clear. Thus, this option should be used with caution by VOs.

Of course, if the VO policies require that the SSL protocol is used for communicating with the CDA, it can reject the requests from this "unsecure client". The client should detect this issue and switch to SSL mode in order to complete the transaction.

Optional Key-generation proxy Another measure aimed at improving response times and minimising resource usage in the device is the usage of a kind of "proxy application" to generate the random private (RSA) key and the certificate request, as these are the most time-consuming steps of the process. This small proxy application would run in any user-owned or VO-owned machine (probably a PC or a laptop), and would relieve the mobile device of performing this chore.

The process of this kind of proxy application would look like this:

1. A SSL connection from the mobile device client to the proxy is established

2. The client passes the username and password needed to authenticate with CDA server onto the proxy
3. The proxy establishes a SSL connection with the CDA server and authenticates using the information provided by the client
4. The client requests the credential to the proxy, specifying the VO
5. The proxy generates a private key and a certificate request, signs the certificate request with the private key and requests a certificate from the CDA specifying the VO.
6. The CDA server returns the certificate to the proxy
7. The proxy returns the private key and the certificate to the client

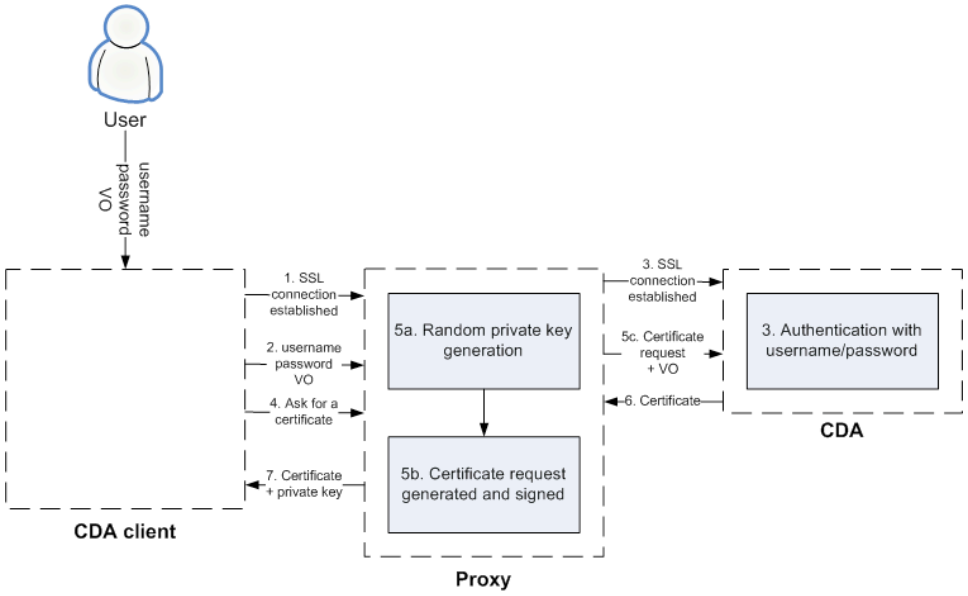

Figure 3.2: (Optional) Proxy-CDA protocol

Alternatively, instead of passing the CDA's username and password from the client to the proxy, the client could use some proxy-specific username and password to access the proxy, which would store the real CDA information associated to this proxy-specific information. This would allow for simpler passwords to be introduced with the limited mobile device keyboard facilities (also, see below).

Usage of PIN instead of passphrases In order to ease the entering of passphrases from the limited keyboards of mobile devices, it could be very useful to allow for users to have a PIN-like (Personal Identification Number) 4-digit password to access the CDA (or the PC proxy, if that option is chosen).

To avoid *brute force* attacks on this PIN, a mechanism should be set up so that, after a limited number of failed attempts to introduce this PIN (e.g. three attempts), this mode of input is not allowed anymore, and in this case the user will have to introduce the full passphrase to authenticate with the CDA or the proxy.

Portability All the aforementioned capabilities shall be implemented with portability in mind, that is, design for easy porting to other flavours of XtreemOS, since the modifications proposed (the usage of the CDA client as a library, the proxying of credentials etc) are worthy additions to other flavours of XtreemOS.

3.5 Interfaces

As it has been already mentioned, there are two main ways of obtaining a credential with the security client: a user can do it directly through a command-line executable (quite improbable, due to interface limitations), or by other applications invoking it (i.e. through a library API). For more advanced uses, WP3.5 will provide lower-level APIs which allow user interaction with the CDA client, e.g. prompting again for credentials if the username/password are rejected.

3.5.1 Command-line interface

The command-line version of the CDA client has the following interface:

```
%./cdaclient <CDA_hostname> <CDA_port> <CDA_PEM_cert> <rootCA_PEM_cert>
```

Where

- `CDA_hostname` and `CDA_port` denote the location and listening port of the CDA.
- `CDA_PEM_cert` is the path and name of the CDA server certificate, which is used to establish SSL connectivity and to verify the validity of the obtained XOS-Cert.
- `rootCA_PEM_cert` is the path and name of the root Certificate Authority that signs the certificate of the CDA.

This command asks for the username, passphrase and Virtual Organization, information which is used by the CDA to identify the user and issue the certificate. If all the information is correct, the output of the command is the XOS-Cert received from the CDA, and verified by the client. Optionally, the certificate can be stored in some permanent storage or in a credential store like the kernel's Key Retention Service (see deliverable D2.3.4 [27]).

3.5.2 Application programming interface (library)

The **main interface** with the CDA client is as follows:

```
int cda_client(char *hostname,
               char *port,char *CDAcertfilename,
               char *RootCAcertfilename, char *username,
               char *password, char *voname,char **credential,
               char **certificate, char verify_cert);
```

Where:

- `hostname` and `port` denote the location and listening port of the CDA.
- `CDAcertfilename` is the path and name of the CDA server certificate, which is used to establish SSL connectivity and to verify the validity of the obtained XOS-Cert.
- `RootCAcertfilename` is the path and name of the root Certificate Authority that signs the certificate of the CDA.
- `username` and `password` are the user's name and password in the CDA server, used to authenticate when connecting to it.
- `voname` is the name of the Virtual Organization for which the user wants the certificate (this is used in case the same user belongs to more than one VO, or the CDA serves more than one VO).
- `credential` is the pointer to the structure where the private key (in PEM format) will be stored.
- `certificate` is the pointer to the structure where the XOS-Cert will be stored, adequately signed by the CDA. If `NULL`, the XOS-Cert and the private key will be stored in `credential` instead.
- `verify_cert` indicates whether the received XOS-Cert should be verified.

This method returns 0 if successful, and other values if there were any errors.

If the optional **key-generating proxy** is implemented (see above), there will be a second method, very similar to this one:

```
int proxy_cda_client(char *hostname,
               char *port,char *CDAcertfilename,
               char *RootCAcertfilename, char *username,
               char *password, char *voname,char **credential,
               char **certificate, char verify_cert);
```

The only difference would be that, instead of connecting to the CDA server, this method would connect to the proxy, that would generate the RSA key and perform the rest of the process in a similar way.

3.6 Examples of use

3.6.1 Command line usage

Although it is unlikely that a mobile user will use this kind of interface, the CDA client in XtreemOS-MD should be able to be executed as a command-line utility:

```
%./cdaclient isegserv.itd.rl.ac.uk 6731 cdacert.pem cacert.pem
```

3.6.2 Application interface (API) usage

Also, mobile user applications can leverage the library API of this module, to obtain XtreemOS certificates from the CDA, like the following example:

```
  char *voname = (char *) malloc( VONAME_MAX );
  char *username = (char *) malloc( USERNAME_MAX);
  char *password = NULL;
  voname = "esvo";
  username = "bob";
  password = "password";
  ...
  char *credential;
  char *certificate;
  int status=cda_client(hostname,port,CDAcertfilename,
   RootCAcertfilename, username, password,
   voname, &credential, &certificate, 0);
  if (!status) {
  /* credential and certificate should be used/stored somewhere */
...

  }
  ...
```

Chapter 4

Common Data Access (XtreemFS client)

Most applications do not make sense if they cannot access data stored in a permanent storage system, and this is also the case of many of the use cases described in chapter 2. In XtreemOS, this permanent storage is offered by XtreemFS, the component that builds a grid-wide filesystem.

XtreemFS architecture is designed with the aim of getting high availability, high performance, parallel and concurrent access to data [14]. These challenging features are achieved by means of a set of scalable and distributed services: OSD, MRC, RMS and XtreemFS client. The Object Storage Devices (OSDs) store the real data. A Metadata and Replica Catalogue (MRC) manages the filesystem metadata. The Replica Management System (RMS) provides mechanisms for automatically improving data access, using striping and replication. Finally, the XtreemFS client provides transparent data access to XtreemOS components and, in general, to any client application.

This XtreemFS client is the component that will be integrated in XtreemOS-MD flavor, managing data accesses from mobile devices. Complex issues such as replica selection, concurrent access, consistency and data replication guarantees will be provided by the remaining components. XtreemFS client will transparently provide those features to XtreemOS-MD users.

Due to mobile devices' characteristics, the reduction of power consumption is a must. Thus, CPU-intensive operations will be requested to XtreemFS services in order to increase the duration of batteries. Improvements in XtreemFS protocols will be also added in order to reduce the amount of communications needed. Therefore, the main complexity of XtreemFS protocols and internal mechanisms will not be included in XtreemOS-MD flavor.

XtreemFS client will provide a POSIX-compliant [9] interface. This interface is suitable for common operations over files such as reading, writing, deletion and querying about file attributes. Users will also be able to define advanced data properties such as striping and replica policies [26].

4.1 Features and functionalities

XtreemFS client will provide the functionality required for accessing XtreemFS. This functionality will include a POSIX-compliant interface for the filesystem and the required operations for volumes, striping and replica management.

The usage of Linux virtual filesystem mechanisms will allow data access from a mount point. Through this mount point, users will be able to perform the whole set of I/O operations such as open, read, write, stat, close and delete with POSIX-compliant system calls. They will be guaranteed independently of the underlying data storage policies. XtreemFS will transparently manage the complexities of striped data access, replica selection, user authentication, permissions and data caching.

Nevertheless, features outside of POSIX such as management of XtreemFS volumes, striping and replica policies will be also provided in the XtreemFS client. Because these features can affect the performance achieved by data accesses, XtreemOS-MD users will be able to tune it to their own requirements or delegate it to the XtreemFS components.

4.2 Design alternatives

There are several complementary alternatives that can be applied to the XtreemFS client. The current design of XtreemFS is based on a distributed set of services that communicate with each other in order to handle data and metadata. These services could be accessed through the Linux VFS interface or by means of a XtreemFS-specific library.

On one hand, the **VFS interface** provides a uniform data access ensuring filesystem semantics in Linux systems. Using this abstraction layer, applications do not need to be aware about the real filesystem storing their data. They just use the POSIX interface for every data access and VFS translates these requests to real operations on the corresponding filesystem. As XtreemFS uses this aproach, XtreemOS applications can access data without modifications. Details about real behavior of the filesystem are hidden by the interface and transparency is totally achieved with POSIX semantics. However, advanced features and innovations which are not already included in POSIX will not be available due to portability and backward compatibility reasons.

On the other hand, accessing XtreemFS services by means of a **library** brings a whole set of possibilities and new functionalities. Applications could be linked to this library making them able to directly set striping policies, replication characteristics or volumes up. The drawback is that applications should be modified for taking advantage of this new funcionality, forcing developers to be aware of internals and semantics of the XtreemFS filesystem.

Since XtreemOS should be able to run legacy applications and standard Linux ones, and XtreemFS should be available for all of them, implementing the **VFS**

interface with POSIX semantics is totally justified. However, advanced features and innovations will be available by a set of tools and libraries which XtreemOS users and specific XtreemOS applications will be able to use.

4.3 Architecture and design

XtreemFS client is a component of the whole XtreemFS architecture, which provides data access to applications. It is responsible for managing communication to the remaining XtreemFS components. In order to be able to process I/O requests without too much delay, XtreemFS client is highly multi-threaded.

At loading time, XtreemFS client registers the filesystem as a FUSE filesystem. Once loaded, filesystem operations coming from the FUSE module are processed and finally translated into requests to the XtreemFS services. More concretely, applications submit their I/O requests through the Linux `glibc`. `glibc` forwards the incoming requests to the kernel VFS layer, that forwards it to the FUSE kernel module. At this point, the FUSE user space library is involved and requests are routed to the XtreemFS client, which will translate them to the proper ones for other XtreemFS entities like MRC or OSD.

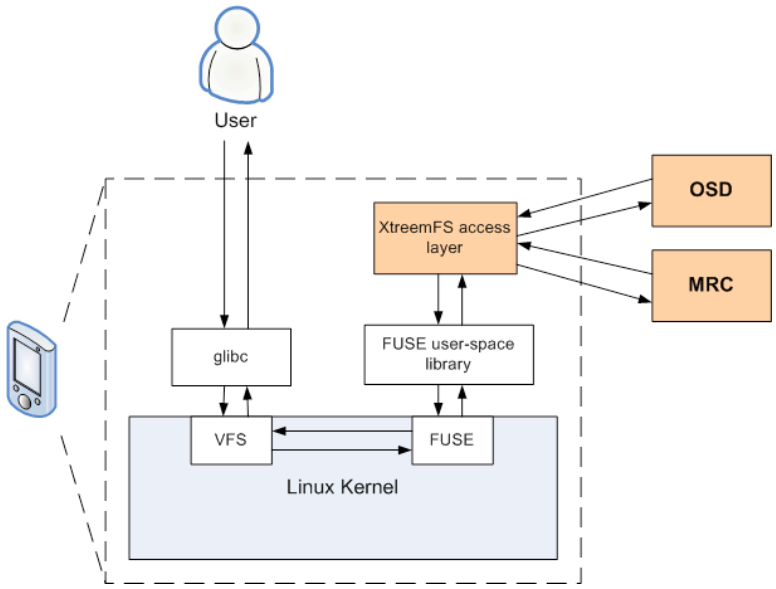

Figure 4.1: XtreemFS client structure

The requests from the FUSE library are processed in several stages. Incoming requests are handled in the FileRW stage, which translates the I/O requests into file object requests. Then the file object stage processes these requests translating them into stripe object requests. Stripe object requests are handled by the stripe object stage, which will finally generate the requests to the right OSDs, i.e. it chooses which OSDs to talk to for a specific stripe object according to the striping policy.

If there is an incoming request from the FUSE library that involves only metadata, the client talks to the MRC. In this case, there is an additional stage that handles all detailed aspects of the MRC communication.

4.4 Specific modifications for Mobile Devices

XtreemFS client requires FUSE [3] to work. FUSE is a part of most modern Linux kernels, which allows the integration of filesystems at user level. This is achieved by a module of the kernel that should be loaded before mounting any filesystem based on it. Therefore, in order to get XtreemFS client working, FUSE's module should be loaded and its utilities installed in the mobile device.

Secure communication is a requirement in mobile devices. Data will usually be transmitted by wireless networks and it will potentially contain personal credentials, permissions and confidential and private user data. For this reason, XtreemFS uses SSL as transport layer, being OpenSSL [8] the chosen implementation of SSL in XtreemFS.

Since XtreemFS is a distributed set of services, some protocols among them had to be defined as well. These protocols have complex fields that require parsing them before the requested operations begin. XtreemFS performs part of this parsing using the JSON library for it.

Beyond dependencies with other software, there also exist specific issues to mobile devices such as battery duration, available bandwidth, frequent disconnections and non-reliable networks, that XtreemFS client will have to face.

Some modifications to XtreemFS client are expected to improve the access from mobile devices to XtreemFS. For instance, **prefetching of data** and **persistent caches** will be basic modifications that could improve the existing features. However, other advanced improvements such as exclusiveness of data accesses or generation of hints about users and data location will be also considered in order to reduce latencies and communication requirements.

Besides own optimizations of XtreemFS client for mobile devices, some optimizations in the whole XtreemFS architecture could improve the mobility experience. They include processing of metadata sent from services to client for reducing computation, requesting of data placement in neighbor nodes, request delegation to proxies, adaptive encryption levels, non-permanent connections and addition of checking, compression and recovery codes for data transmissions. All of them are very advanced features that could improve the big set of current characteristics of XtreemFS.

Regarding the internals of XtreemFS client, some portability issues have to be solved before getting a usable XtreemOS-MD release. On one hand, **atomic operations** are platform dependent code that will have to be changed by the proper ones for the ARM architecture. On the other hand, issues related to **code optimization** of XtreemFS client's internal libraries will also have to be solved.

4.5 Interfaces

XtreemFS client provides a POSIX compliant interface for file access operations. This interface transparently performs the communication to the remaining XtreemFS components. Internally, XtreemFS client perfoms the connections to the distributed services of XtreemFS (OSD, MRC and RMS) which are done via HTTP over SSL for data transmission.

XtreemFS client also provides a set of utilities for volume management. Every file stored in XtreemFS is stored in a volume. These utilities cover the functionality for creation, listing and deletion of volumes (`mkvol`, `lsvol` and `rmvol` respectively).

In order to establish the mount point for XtreemFS, the `xtreemfs` command is also provided. This command mounts XtreemFS in the mount point (path) specified by the users, allowing them to set some XtreemFS performance options up.

4.6 Examples of use

These are some examples of use of the XtreemFS client features. For more information, please refer to deliverable D3.4.2 [16].

4.6.1 Creation of a volume

The creation of a volume is the first step that should be done prior to any other XtreemFS operation. A volume is created with the *mkvol* command:

```
% mkvol -a 2 http://myhost:32636/MyVolume
```

This will create a volume called `MyVolume` in `myhost` (there is a Directory Service listening in port `32636` in `myhost`)

4.6.2 Mounting XtreemFS

In order to perform any operation in XtreemFS, the filesystem should be mounted in a mount point. This is done by the *xtreemfs* command:

```
% xtreemfs -o volume_url=http://myhost:32636/MyVolume,\
    direct_io /mntpnt
```

This will mount the volume `MyVolume` stored in the host `myhost` on the mount point `mntpnt`. The mount option specifies that it will be done with direct input and output.

4.6.3 Unmounting XtreemFS

Once every operation over XtreemFS was performed, users could want to unmount the filesystem. This can be done with the command *fusermount*. This command is an utility of FUSE package and allows unprivileged users to unmount FUSE-based filesystems.

```
% fusermount -u /mntpnt
```

It will flush caches and unmount the mount point `mntpnt`.

4.6.4 Deletion of a volume

Volumes can be deleted with the *rmvol* command:

```
% rmvol http://myhost:32636/MyVolume
```

It will delete the volume called `MyVolume` stored in host `myhost`

Chapter 5

Application Execution (AEM client)

A set of functionalities that is an almost definitive trait of any grid system, is the ability to run and manage processes (or jobs) in remote nodes of the grid. Moreover, this need also appears in many of the use cases described in chapter 2, be them grid-aware or grid-transparent (e.g. advanced voice recognition systems, etc) ones.

Thus, a mobile grid user will most likely need some way to connect to XtreemOS AEM service to launch, control and monitor the status of jobs.

5.1 Features and functionalities

As it can be derived from the use cases described in chapter 2 and deliverable D3.6.1 [20], the job-related functionalities that mobile users are most likely to need are:

- Full **job management**: **launch**, **suspend** (stop execution, leaving the job in memory), **resume**, **cancel** (kill the job) and **wait**.
- A mechanism to **monitor running jobs**. Monitored information must include **launch time**, **deemed time to end**, **status** (running, suspended, awaiting execution etc), **resource consumption** and **special notifications**.
- Implicitly, XtreemOS-MD nodes must identify themselves with the resource management services as special nodes, which would mean a very limited storage and computation capacity.

Of course, apart from these functional requirements, there are a number of non-functional requirements that must be met by this (and any other) XtreemOS-MD module, like **reduced memory and disk footprint**, computational **efficiency**, or the ability to run in a limited **executing environment** (i.e. availability of certain libraries or daemons).

5.2 Design alternatives

XATI vs. XOSD To offer all the aforementioned services, there were two different options. The first (and probably simplest) one, was just to offer **XATI**, the interface to contact the XOSD (the AEM daemon handling resources and jobs) that can be located in any other XtreemOS node. With this interface we could build all needed applications to submit jobs and check status of resources and jobs.

The second option would be to have a reduced version of this **XOSD daemon** in the mobile device. In this case only some services should be part of the MD version of the daemon: `jScheduler` and `jController`. The `jScheduler` would allow to submit jobs and would put all scheduling and negotiation overhead in the mobile device. The `jController` would take care of the jobs submitted from this device (this service would have fault tolerance characteristics, and thus residing in a MD would not be a technical problem).

But, since having the `jController` in the MD would pose several performance problems, specially if the MD gets disconnected very frequently, and also because of sheer runtime efficiency in the device, it has been decided that, at least on a first phase, just the XATI woud reside on the mobile node, leaving the implementation of a customized XOSD for mobiles to a later phase, specially if more execution-related features are deemed interesting.

The Java issue In the standard flavour of XtreemOS, job management tasks and issues are managed by the Application Execution Management (AEM). Current implementations of all the components of the **AEM were written in Java**, concretely needing the version 6 of the SDK.

However, Java support in mobile devices using the ARM architecture is lagging behind his PC counterpart: there is no official Sun JDK for this architecture, and the support of open source alternatives (like, for example GNU classpath project [1]) that have been ported to the ARM architecture do not even completely support JDK version 1.5. Moreover, the mobile-specific J2ME flavour has some licensing issues and does not provide with many of the standard JDK features, even on its most complete profiles.

So, after considering a complete reworking of the XATI interface, it was decided that it would be more efficient to have a **C language implementation of XATI**, both in developing time and also in runtime. This C implementation, named XATICA, was already expected in WP3.3 [2], and was provided by WP3.3 developers, to be ported and modified by WP3.6.

5.3 Architecture and design

As it is already detailed in deliverable D3.3.3/4 [2], the job-related requests from clients are issued from through the XATI interface to the XOSD, a distributed fault-tolerant daemon that resides in every XtreemOS computing node (see figure 5.1).

Jobs are defined in a JSDL file that is stored either locally or in the XtreemFS; this file will be read before launching, and the adequate call will be sent to the XOSD, as it can be seen in figure 5.1:

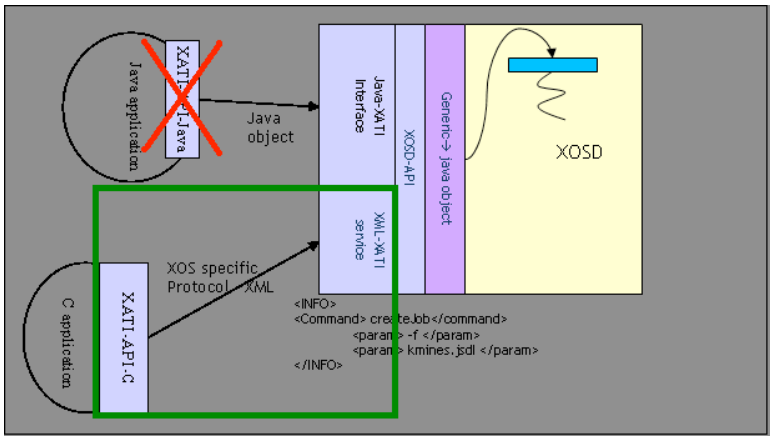

Figure 5.1: The XATI interface to the AEM

Native communication with the XOSD is normally performed via Java objects, but not all applications are written in Java. This is the reason why XATICA was created: it is the C translation of XATI, making the communication with the XOSD available through XML-formatted messages [2]. Application developers just have to statically link with the library ***libXATICA.a*** and set the parameters of the configuration file *XATICAConfig.conf* to the correct values and the communication with the XOSD is immediately available.

Although there is not a specific document describing XATI or XATICA yet, the following main components can be identified from the source code:

XML wrapper and parser in order to exchange XML-formatted messages with the XOSD. Associated files: *XATICACommon.*, XATICAXMLParser.**.

Communication module to communicate with the XOSD via sockets. The communication parameters are retrieved from the configuration file mentioned above. Associated files: *XATICACommunication.*, XATICAProperties.**.

CDA module that manages user certificates in order to communicate with the CDA. Associated files: *XCCDAMng.**.

Jobs execution module to execute jobs and get overall execution information. Associated files: *XCExecMng.**.

Job management module that allows to perform several operations over a concrete job. Associated files: *XCJobMng.**.

Resource management to manage all the available resources. Associated files: *XCResMng.**.

5.4 Specific modifications for Mobile Devices

Implementation Once a C implementation of the XATI interface was provided by WP3.3, a series of preliminary tests were conducted on the Development Environment of XtreemOS-MD (a QEMU-ARM emulator), to test its efficiency and overall load on a more restricted ARM machine. These tests revealed that XATICA (the name for this XATI C implementation) is lightweight enough to be run without problems in a mobile device. Since it does not have any important library dependencies either, we concluded that the XATI module will be included in XtreemOS-MD **virtually unmodified** (except for any transversal, generic modifications such as those described in chapter 7).

Command line utilities It is true that there is still missing some command line utilities like the AEM console that is present in the Java version of the software, although it is more likely to be used as testing utilities, rather than by real users, due to the usability problems of command-line interfaces in mobile devices. These utilities should be included anyway, and will have to be implemented either by WP3.3 or by WP3.6.

Graphical user interface In the same direction, it would be most useful for real mobile users to have a graphical interface application for managing (launching, controlling and monitoring) jobs, using this XATI module. Luckily, such an application is already being developed in WP4.2. This application has been labelled JobMA (standing for Job Management Application, see D4.2.1 [12]), and will probably be included in the official XtreemOS-MD distribution, once XATICA is integrated into it.

5.5 Interfaces

5.5.1 Command line interface

This module should include some form of command line interface (e.g. a console application, like `xconsole`, the shell-like utility used in the Java version in WP3.3) for sending requests to the AEM using XATICA. The interface for this application could look like the following[1]:

```
Xsub -f <jobdefinition.jsdl>
```

Used for job submission. The xsub command creates and runs a job defined by a JSDL file, waits until the job ID is returned and prints it onscreen.

[1] This is mostly taken from the Java version in deliverable D3.3.3/4 [2], and is reproduced here for completeness.

```
Xps -j <jobID>
Xps -a
```

Shows the info for the job with id jobID, or shows the info for all the submitted jobs

```
Xwait <jobID>
```

Waits for the finalizations of a specific job specified by jobID

```
Xkill <event> <jobID>
```

Sends the specified event (currently UNIX signals) to the job specified by jobID

```
xrs -a
xrs -f <resourcerequirements.jsdl>
```

Shows all the resources available, or a list of available resources that fit into the JSDL requirements

```
xrs
-cpu <architectureName>
-numcpu <cpuCountInterval>
-ghz <cpuSpeedIntervalGHz>
-os <osName>
-ram <RAMSizeIntervalGB>
```

Shows the list of resources that fit the query expressed in the command line. At least one of `cpu`, `numcpu`, `ghz`, `os` or `ram` has to be provided. The values of `cpuCountInterval`, `cpuSpeedInterval` and `RAMSizeInterval` can be either a single value (e.g. 1.5) or an interval (e.g. 1.5-3).

```
xmonctr -node <address> -m
xmonctr -a
```

Shows the list of attributes (e.g. memory, CPU, disk usage etc) that are being monitored at the specified node. The address has the form `<IP>:<port>`, and the port can be obtained using the `-a` variant of the command.

```
xmonctr -node <address> -i
```

Shows the details on the node by listing the values of the monitored resource attributes. The address has the form `<IP>:<port>`.

5.5.2 Application programming interface (library)

The main usefulness of this module is to be linked from other applications (or from other libraries like the XOSAGA connector, see chapter 6) so that they can communicate with the AEM. The interface offered to applications should include at least the following[2]:

```
int createJob(char* __jsdlFile,char __startJob,
              char* __reservationID, char** returnValue);
```

Creates a job based on a job definition (`jsdlFile` parameter). If `startJob` is true, the job is created and submitted for execution. Otherwise, it is only created.

```
int runJob(char* __jobId,char* __reservationID, int* returnValue);
```

Starts the execution of the `jobId`.

```
int jobControl(char* __jobId,int __ctrOp, int* returnValue);
```

Performs an operation (e.g. CANCEL) on the job.

```
int sendEvent(char* __jobId,int __signal,int __operation );
```

Send the specified event (currently, only UNIX signals) to the `jobID`.

```
int getJobsUser(char* __userId, char** returnValue);
```

Returns the list of jobIDs of jobs from a user existing in the system.

```
int getJobInfo(char* __jobId,int __flags,
              char* __infoLevel, char** returnValue);
```

Returns the attributes associated with the `jobID`.

```
int getResources(char* __query,int __howMany, void* returnValue);
```

Returns the resources that match the provided resource query.

```
int getResInfo( void* returnValue);
```

Returns the information of the node. The returned value consists of the resource attributes currently selected and their values.

```
int getResMetrics( void* returnValue);
```

Retrieves a list of the node's attributes that are currently being monitored.

[2]This is very similar to the Java version in deliverable D3.3.3/4 [2], and has been taken from the code of XATICA.

5.6 Examples of use

5.6.1 Command-line utilities

As it has been already mentioned, a number of command-line utilities to interact with the AEM should be shipped with XtreemOS-MD, even just for testing purposes. An example interaction could be:

```
%vi example.jsdl
%xsub -f example.jsdl
801929a6-9ac1-4f1b-8c18-0267958a0a4a has been submitted
%xps -j 801929a6-9ac1-4f1b-8c18-0267958a0a4a
```

5.6.2 Graphic interface

Although a user application for managing jobs is outside the scope of WP3.6 (as it is already being done with the JobMA application from WP4.2), using the programming interface of the component described in this chapter, it should be possible to write graphic applications that could look like this one:

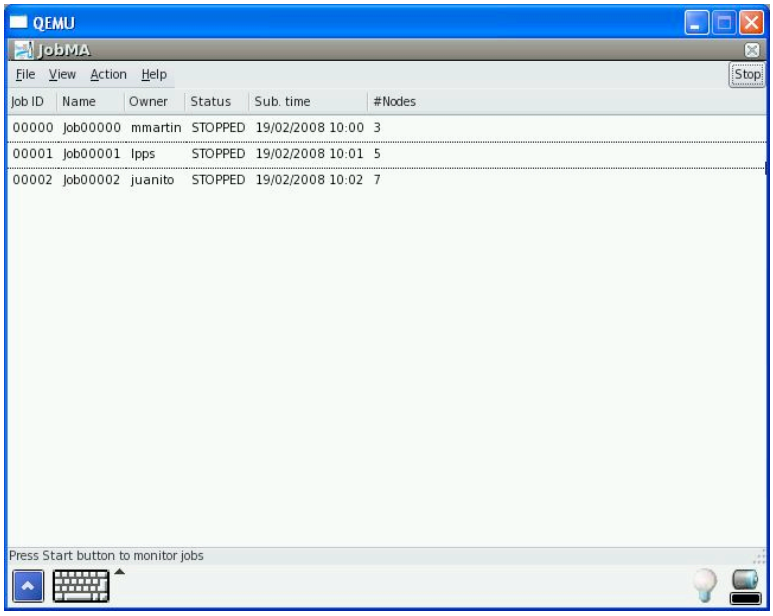

Figure 5.2: Example of graphic job management application

5.6.3 Application interface (API) usage

The functionality provided by the mobile version of XATICA is specifically designed to be integrated in user applications (like the aforementioned JobMA). Here we can see how the code of such an application could look like:

```
#include <XCExecMng.h>
#include <XCJobMng.h>
#include <XCResMng.h>

int main (int argc, char **argv)
{
    /* variable definition */
    char *job_id; /* store job ID */
    char *jsdl_file = "/home/xtreemos/myjob.jsdl";
            /* file containing the job definition */
    char *reservation_id = NULL; /* no reservations required */
    char start_job = TRUE; /* start job once created */
    int exit_value = 0; /* job's exit value when finished */

    /* job creation and starting */
    createJob (jsdl_file, start_job, reservation_id, &job_id);

    /* print job ID */
    printf ("Job ID = %s\n", job_id);

    /* immediately finish job */
    exitJob (job_id, exit_value);

    return 0;
}
```

Chapter 6

XOSAGA for Mobile Devices (API)

Although it is not required explicitly by any of the use cases mentioned in chapter 2, the XtreemOS implementation of OGF's Simple API for Grid Applications (XOSAGA, for short) is an important component of the overall XtreemOS architecture. This API, further described in WP3.1 [18], provides a common high-level API for any grid application, regardless of the middleware or operating system that lies beneath, be it Globus, gLite, XtreemOS, or any other for which there exists a SAGA adaptor.

Thus, XOSAGA will provide any grid developer (in this case, mobile grid developer) with a way of coding portable grid client applications that will be able to run, in this case, in a mobile device with the XtreemOS-MD OS.

6.1 Features and functionalities

The features that this "mobile XOSAGA" component must cover are basically the implementation of the SAGA standard (or, more concretely, the portions of it that provide client access to grid resources). According to the SAGA specification [10], this includes:

Non-functional areas include

- security and session management
- *permission management*
- asynchronous operations
- monitoring
- asynchronous notifications
- *attribute management*
- I/O buffer management

Functional areas include

- job submission and management
- management of namespaces
- file I/O
- *replica management*
- *streaming*
- *remote procedure calls*
- service discovery
- *message exchange*
- storage of application level information
- *database access and integration*
- *checkpoint management and recovery*

In the above list, *italized* text denotes optional functionalities from the point of view of a mobile grid client like XtreemOS-MD basic version, derived from the use cases detailed in D3.6.1 [20].

6.2 Design alternatives

SAGA-Java vs. SAGA-C++ implementation Currently there exist two reference implementations of the SAGA standard (which include not only the core SAGA engine, but also some simple adaptors – see below), one in Java and another one in C++.

At the beginning of the project the Java option seemed more attractive, since the mobile applications from WP4.2 were going to be coded in Java, and also because of Java's natural coding ease. However, there are a number of facts that have made us decide against this way in favor of its C++ counterpart:

- The Java reference implementation (as well as the development of the correspondent SAGA adaptors for XtreemOS) seems to lag a bit behind the C++ code, which seems more stable and will be more timely released.
- The minimum requirements of the SAGA-Java implementation state that at least JDK1.5 is needed but, as it has been already mentioned, the support for this JDK in ARM architectures is only partial.

To sum up, the basic version of XtreemOS-G for mobile devices should include a **C++ implementation** of the SAGA engine and XtreemOS adaptors, although we do not preclude the inclusion of the Java version in the advanced version of the distribution.

6.3 Architecture and design

The internal architecture of the SAGA interface is better explained in WP3.1 deliverables [22, 18], but its general structure can be seen in figure 6.1:

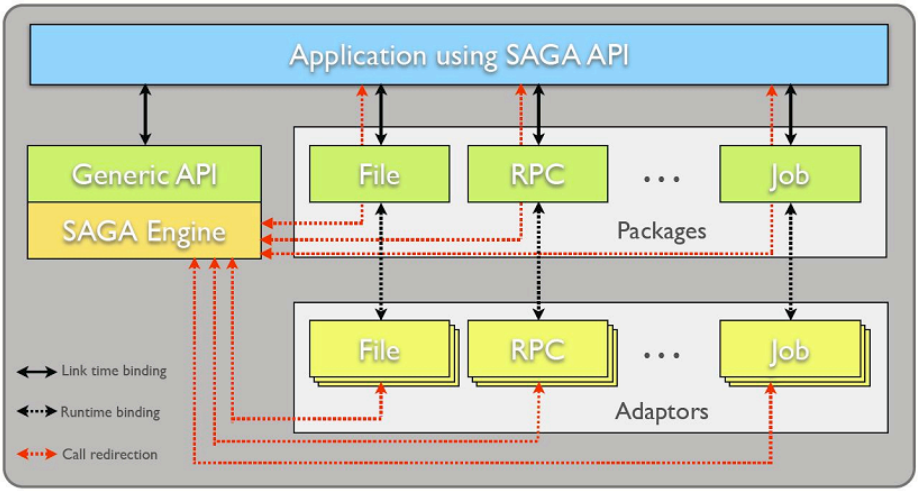

Figure 6.1: SAGA architecture

As it can be seen in the figure, the SAGA API is composed of a core *SAGA engine*, which contains all the common classes and interfaces, and offers what is called the SAGA "look & feel", a generic API that defines how application programmers will have to operate with the SAGA interface.

Apart from this non-functional features, SAGA comprises a number of separated modules, called *packages*, which provide areas of functionality that grid applications normally need, like job management, data/file management etc. These general functionalities are translated to the concrete APIs of the underlying grid software (e.g. Globus Toolkit, or XtreemOS) by a number of grid implementation-specific *adaptors*, which cover each of those areas of functionality. These specific adaptors are bound to the application in runtime.

6.4 Specific modifications for Mobile Devices

Modifications for building in mobile environments In a first porting attempt to a mobile device environment, a number of minor changes had to be made for the SAGA engine to build successfully. It is also probable that additional changes will be needed for the building and execution of the XtreemOS-specific SAGA adaptors (which were still not available as of this writing).

Unneeded packages and adaptors One of the most striking facts that have been revealed during the first experiments with XOSAGA in mobile devices was its important disk footprint (around 120 MB). Due to disk space restrictions on mobile devices, it is very important to streamline the mobile XOSAGA installation. All the unneeded SAGA adaptors (Globus GridFTP, GRAM, etc) which come with the default installation of SAGA should be deleted in the mobile version.

In the same manner, since XtreemOS-MD does not necessarily implement the functionality of all available SAGA packages, the uncovered functionality should also be stripped down from this mobile version. The same applies for documentation, examples and any other material that is not strictly needed for runtime usage of XOSAGA.

Modularization It is also very important for the mobile device version of XOSAGA to take advantage of the modular properties of SAGA: since it is possible to divide XtreemOS functionalities into groups that can be installed independently (see also "On-demand installation" in chapter 7), the corresponding XOSAGA packages and adaptors should also be separately installable, to save permanent storage space.

6.5 Interfaces

Since this component is in itself an API, this should be the most important section of the chapter. However, since the API is very extensive, and is better explained in other XtreemOS and OGF documentation [22, 18, 10], the reader is encouraged to refer to those sources for information about XOSAGA's interfaces.

6.6 Examples of use

Again, the reader is encouraged to refer to WP3.1 and SAGA's documentation [22, 18, 10] for detailed information on the usage of this API in applications. A short snippet of application code is shown below, which copies a file from a source location to a target location:

```
#include <string>
#include <saga/saga.hpp>
void copy_file(std::string source_url, std::string target_url)
{
  try {
    saga::file f(source_url);
    f.copy(target_url);
  } catch (saga::exception const &e) {
    std::cerr << e.what() << std::endl;
  }
}
```

Chapter 7

Other Enhancements for Mobile Devices

Apart from the XtreemOS-specific modules described so far, there are still a number of additional modifications that could prove very useful for a grid operating system like XtreemOS-MD, but do not relate specifically to other XtreemOS system or feature.

These enhancements mostly relate to the special limitations that can be found in a mobile device, ranging from a limited user interface (making command line utilities mostly useless) to the difficulty of installing new software, or reduced disk and memory capacities. To aid in coping with these handicaps, three additional utilities have been designed:

On-the-fly installation of most XtreemOS grid services and modules. In order to save disk space, only the bare minimum of functionality will be installed by default, and additional XtreemOS functionality will be installed only when needed.

On-demand startup of grid services and daemons. In order to save memory and to make grid operations more transparent for mobile users, an XtreemOS launcher/wrapper will be implemented, that takes care of starting all the necessary processes and daemons (e.g. for obtaining a XOS-cert if none is available). This also has the side benefit of allowing for faster device startup times, as it is quite possible that grid functionality won't be needed every time the device is powered on.

Stopper applet that allows users to stop all grid functions once they are not needed anymore, freeing up the device's resources.

Taking into account that these modifications are not (or interact with) XtreemOS-specific components, the structure of this chapter has been changed just to describe their functionality and implementation. Moreover, many of the implementation

details are very much reliant on the underlying Linux distribution. In our effort to remain as portable among Linux distributions as possible, we will describe them in the most flexible way, leaving the implementation details for following deliverables (i.e. implementation stage of the modules).

7.1 On-the-fly installation

Functionality The scarcity of disk space is a very well known limitation of mobile devices, that can be ameliorated by the usage of flash cards (CF, SD/MMC, ...). However, many mobile operating systems do not allow the installation of system software in these cards. Thus, it is very important to optimize the usage of permanent storage in the device.

A way of doing this is by deferring the installation of most XtreemOS components until they are really needed. The idea is to have only a minimum core installed by default in every device, and doing what we could call "just-in-time installation" of the needed XtreemOS components as they are called for. This kind of behaviour would benefit greatly from some kind of launcher application that checks the availability of the different XtreemOS modules and installs them if they're not present (see "on-demand startup", below).

Optionally, an analogous mechanism can be devised for **uninstalling** the software if it is not needed anymore (for example, if it has not been used for a predetermined lapse of time), to reclaim storage space.

Implementation One of the main drawbacks of mobile devices has been historically the "closedness" of their software, as mobile manufacturers are very reluctant to let users tamper with the installed software, even in Linux-based devices[1].

However, this trend is slowly receding, with the increasing presence of devices that allow for user-installable software (e.g. Nokia tablets, OpenMoko project and, to a certain extent, Google's Android and Apple's iPhone). Some of these devices use in one way or the other the concept of virtual machine or virtualization, by using Java or pseudo-Java VMs (Dalvik), while others leave it entirely open or use standard Linux methods (e.g. administrative operation using `sudo`).

In XtreemOS-MD, following the philosophy of adopting Linux practices and transparency, the latter approach is preferred, but this feature is bound to be **dependent on the underlying distribution and device's** policies. XtreemOS-MD should offer at least an example of the auto-installation procedure for Ångström (most probably using sudo or other application that has `setuid` permission). Optionally, an analogous method can be supplied for other distributions like Nokia's Maemo.

In any case, this feature should be used from the "on-demand startup" module (see below), that performs preliminary tasks before any grid operation takes place.

[1] In some cases, such as mobile phones, this is also due to legal reasons, because certain wireless protocols should not be changed.

7.2 On-demand startup

Functionality Another important limitation of this kind of mobile devices is the amount of available RAM memory. Unlike the case of a typical desktop PC, the addition of a number of daemons and processes on every device startup would leave the device handicapped for normal operation (not only because of memory usage, since more CPU would be used, and startup time would also increase). Moreover, mobile devices are very often used for "personal" kinds of operations (personal information management, calendar/agenda, note-taking, etc) instead of "work" operations (like it could be working with grid resources), so those daemons and processes are less likely to be useful in every user session.

In this scenario, it is readily apparent that a mechanism for starting those processes just when they are needed would be very useful. This utility would be executed previously to every grid operation, and it should check that the necessary daemons have been started, and that the necessary data for grid operation (e.g. the XtreemOS certificate) is present in the system. If not, this utility would take care of starting them transparently.

Optionally, if the necessary software is not even installed, it would invoke the installation of the software packages before trying to start any processes.

Implementation For the on-demand startup of grid services in the mobile device, several implementation strategies could be followed. Probably the least intrusive and most flexible way of doing it is by providing a XtreemOS-specific **application launcher** (a kind of wrapper) that checks for the required processes, and starts them if they are not running (or if the certificates for authentication are not available). Optionally, this launcher can check for the installation of the necessary XtreemOS-specific software, and install it on-demand if it is not available (see "on-demand installation", above).

For this implementation to be really useful, the installation process of any grid application should include in its startup shortcut (e.g. in the main menu of the graphical environment) a call to this launcher, for example:

```
startxtreemos mygridapp parameter1 parameter2 parameter3
```

7.3 Stopper applet

Functionality It is rather obvious that, if having the processes running from startup is a waste of resources, having them running once grid operations are not needed anymore will be equally wasteful. Thus, a way of stopping all these processes and reclaiming their memory would be a very welcome addition to the operating system.

As we will see in the next section, there are several ways of implementing this, but the functionality is the same: to have a way of stopping all grid-related functions, to go back to normal, PIM-like usage.

Implementation For the task of stopping all XtreemOS activity in order to free resources, a number of alternatives have been evaluated. The most transparent way of doing it for users would be to automatically stop the processes when they are not used for a certain period of time. However, the calibration of this timeout is difficult, since mobile usage of the grid can vary wildly, and has not been studied thoroughly so far.

On the contrary, for the basic version of XtreemOSMD, a more direct approach has been taken. The user should state explicitly that he is not using the grid for long period of time, thus stopping all the processes that are not needed anymore. To make this scenario user-friendly, this should take the form of a graphical applet, that can be seen at all times when the device is in "grid mode", and that the user can click to return to "non-grid mode". The path for a more automatical solution is left open until a more detailed knowledge of mobile grid usage patterns is available.

The concrete implementation of this applet would largely be **dependent on the desktop interface** (not only user interface, but also programming interface) that the operating system provides. Thus, XtreemOS-MD will develop an example applet for the GPE (GPE Palmtop Environment) desktop available in Ångström and, optionally, examples for other desktops like OPIE or Hildon (the one available in Maemo).

Chapter 8

Conclusions

In this document we have reviewed the first, basic design of the components of XtreemOS-G for mobile devices. As it was derived from previous deliverables [20], the needed functionality of this layer shall include:

- A way of obtaining XtreemOS certificates (XOS-Cert) for user operation. In XtreemOS, this functionality is provided by the Credential Distribution Authority (CDA) client.
- A way of accessing, mounting and modifying grid files from a mobile device. As XtreemOS provides the XtreemFS filesystem for grid data management, the XtreemFS client should also be included.
- Mechanisms for launching, managing and monitoring jobs that are being executed in the grid. In XtreemOS, this feature is offered through the Application Execution Management (AEM) system, using XATI as the interface for accessing it. Thus, an implementation of XATI should be included.
- A standard way for grid applications to access all the aforementioned functionalities. XtreemOS will follow in this regard the Simple API for Grid Applications standard proposed by OGF. Thus, an implementation of SAGA has to be included in XtreemOS-MD.

The document has presented these software modules, alongside additional modifications that should be performed on them to make them more efficient and useful in a mobile environment. Also, a number of usability and performance concerns have been raised, and additional software components have been proposed to tackle those concerns.

In these modifications, the XtreemOS-MD team has always tried to make the modifications as generally applicable as possible, and thus it is expected that many of these modifications will be applied also to other XtreemOS flavours (at least as options), in order to obtain an even more streamlined grid operating system.

It is also worth noting that, although the first prototypes of XtreemOS systems are for the most part written in Java, in mobile devices this approach is not so recommendable because of the limitations of mobile Java Virtual Machines, and also because their use is not so widespread as in the PC world (with the exception of the very limited J2ME version). Thus, the first version of all the aforementioned components should be attempted in native languages like C/C++.

Chapter 9

Future Work

9.1 Next steps: implementation of XtreemOS-G for mobile devices

After the design phase, the realization of this design has to be completed, building on top of the implementation of the mobile version of the XtreemOS-F layer, and integrating with all the other XtreemOS subsystems that populate the other flavours of XtreemOS.

This implementation work has in fact already begun with the first pre-alpha portings of the necessary modules to ARM architecture, which took place during the writing of this document. From now on, the modifications to those modules and the additional components proposed in this document will be developed. This development, of course, will have to comply with the project-wide development guidelines that have been defined, with regard to code quality, documentation, unit testing etc.

In parallel, it is possible that these modifications are backported to the PC and cluster flavours of the components, as many of them have a more general applicability as ways of making the software more efficient and usable.

9.2 Research prospects

Also, during the design process, a number of ideas and research prospects have been detected, and there are a number of advanced use cases that should be catered for. These research paths will have to be evaluated in future deliverables, to assess their feasibility, specially since the advanced version of XtreemOS is directed towards an even more restrictive market: mobile phones.

These research paths can include:

- The ability to execute grid applications and jobs inside mobile devices, thus providing their resources to the grid as any other node.

- The inclusion of non-traditional grid resources, like specialized devices, connectivity or users themselves, as shareable grid resources.
- The embedding of context information not only in mobile grid applications, but in the whole framework of XtreemOS.
- The ability to manage virtual organizations from mobile devices.
- The ability to manage resources (add/remove, modify policies etc) from a mobile device.
- The ability to move grid sessions and applications seamlessly among XtreemOS machines and devices.
- The integration of grids and mobile ad-hoc networks.
- The execution of componentized applications in the XtreemOS grid, using frontend components in mobile devices.

References

[1] GNU Classpath Project.
http://www.gnu.org/software/classpath/.

[2] XtreemOS Consortium. Merge of deliverables D3.3.3 (Basic services for application submission, control and checkpointing) and D3.3.4 (Basic service for resource selection, allocation and monitoring) - Deliverable Number D3.3.3-4. Integrated Project, December 2007.

[3] Filesystem in Userspace.
http://fuse.sourceforge.net.

[4] Official GridLab web site.
http://www.gridlab.org/.

[5] Tao Guan, Ed Zaluska, and David De Roure. A grid service infrastructure for mobile devices. In *Proceedings of the First International Conference on Semantics, Knowledge and Grid*, page 42, 2005.

[6] Torben Knerr. Mobile Access to Grid Web Services, July 2006. Presentation slides.
http://mobdev.tknerr.de/wp-content/uploads/2006/09/mobilegridaccess.pdf.

[7] MyProxy Credential Management Framework (official site).
http://grid.ncsa.uiuc.edu/myproxy/ca/.

[8] OpenSSL: The Open Source toolkit for SSL/TLS.
http://www.openssl.org.

[9] The Single Unix Specification, version 3.
http://www.unix.org/version3.

[10] SAGA :: A Simple API for Grid Applications (SAGA official website).
http://saga.cct.lsu.edu/.

[11] XtreemOS Consortium. First Draft Specification of Programming Interfaces. Integrated Project, December 2006.

[12] XtreemOS Consortium. Requirements Capture and Use Case Scenarios D4.2.1. Integrated Project, December 2006.

[13] XtreemOS Consortium. State-of-the-Art in trust and security for OS and Grids D3.5.1. Integrated Project, December 2006.

[14] XtreemOS Consortium. The XtreemOS File System - Requirements and Reference Architecture D3.4.1. Integrated Project, December 2006.

[15] XtreemOS Consortium. Application References, Requirements, Use Cases and Experiments D4.2.3. Integrated Project, July 2007.

[16] XtreemOS Consortium. Basic XtreemFS object-based file system and basic Object Sharing Service D3.4.2. Integrated Project, December 2007.

[17] XtreemOS Consortium. Design of a Basic Linux Version for Mobile Devices D2.3.3. Integrated Project, December 2007.

[18] XtreemOS Consortium. First Prototype of XtreemOS Runtime Engine D3.1.3. Integrated Project, December 2007.

[19] XtreemOS Consortium. Implementation of the basic services for job submission, control and monitoring D3.3.3. Integrated Project, December 2007.

[20] XtreemOS Consortium. Requirements and Specification of Basic Services for Mobile Devices D3.6.1. Integrated Project, December 2007.

[21] XtreemOS Consortium. Requirements and Specifications of a Basic Linux Version for Mobile Devices D2.3.2. Integrated Project, June 2007.

[22] XtreemOS Consortium. Second draft specification of programming interfaces D3.1.1. Integrated Project, December 2007.

[23] XtreemOS Consortium. Second Draft Specification of XtreemOS Security Services D3.5.4. Integrated Project, December 2007.

[24] XtreemOS Consortium. Security Services prototype month 18 D3.5.5. Integrated Project, December 2007.

[25] XtreemOS Consortium. Virtual Organization Basic Requirements and Specifications for Mobile Devices D2.3.1. Integrated Project, February 2007.

[26] XtreemOS Consortium. Design Report for Advanced XtreemFS and OSS Features D3.4.3. Integrated Project, June 2008.

[27] XtreemOS Consortium. Linux-XOS for MDs/PDA D2.3.4. Integrated Project, June 2008.

Appendix A

Specification Lists

A.1 Software dependencies

This section includes a list of dependencies that have been already detected in the design phase of XtreemOS-G layer for mobile devices. Only the packages that may not be covered in a mobile Linux environment out-of-the-box are listed here.

A.1.1 Security client

As described in chapter 3, a client for obtaining XtreemOS certificates (XOS-Cert) is needed in XtreemOS-MD.

The software dependencies of this module are:

- openssl 0.9.8

A.1.2 XtreemFS

As described in chapter 4, the only part of the XtreemFS needed in mobile devices is the access layer, in order to access the files and mount XtreemFS volumes.

The XtreemFS client has the following dependencies: openssl-dev libxml2-dev kernel-module-fuse fuse-utils libfuse-dev libfuse2

- openssl
- libxml2
- kernel-module-fuse
- fuse-utils
- libfuse2

A.1.3 Application Execution Manager (AEM)

As described in chapter 5, the only part of the AEM needed in mobile devices is the client, in order to access the remote server (daemon XOSD). This client is XATICA, a XATI C implementation.

The only additional packages needed to build and use XATICA are:

- libxml2

A.1.4 XtreemOS API (XOSAGA)

As described in chapter 6, XtreemOS-MD needs an implementation of the SAGA engine and adaptors for C++, in order for SAGA grid applications to be executed in mobile devices.

The SAGA C++ engine has the following dependencies:

- g++
- libc6
- pkgconfig
- libboost 1.33
- sqlite3
- openssl
- xmlrpcpp

A.1.5 Additional components

Some dependencies that can be foreseen for the additional software components described in chapter 7 include:

- xosmd-vosupport
- xos-nss-pam