Project no. IST-033576

XtreemOS

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

XtreemOS-G for MDs/PDA
D3.6.3

Due date of deliverable: November 30th, 2008
Actual submission date: December 16th, 2008

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

Type: Deliverable
WP number: WP3.6
Task number: T3.6.3

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

Version 1.0 / Last edited by Santiago Prieto / December 16th, 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	24/9/08	Luis Pablo Prieto	TID	Initial template
0.2	13/10/08	Luis Pablo Prieto	TID	Introduction
0.3	22/10/08	TID Team	TID	Credentials chapter
0.4	27/10/08	Jesús Malo, Luis Pablo Prieto	TID, BSC	XtreemFS chapter
0.5	3/11/08	TID Team	TID	Finishing credentials chapter
0.6	4/11/08	Luis Pablo Prieto	TID	Future work chapter
0.7	5/11/08	Luis Pablo Prieto	TID	Execution client chapter
0.8	6/11/08	TID Team	TID	XOSAGA and AIK chapters
0.9	7/11/08	Luis Pablo Prieto	TID	Finishing touches. Ready for internal review
0.98	14/11/08	Luis Pablo Prieto	TID	Added comments from first review
0.99	27/11/08	Luis Pablo Prieto	TID	Added comments from second review and executive summary
1.00	16/12/08	Santiago Prieto	TID	Final version

Reviewers:
Haiyan Yu (ICT), Matthias Hess (NEC)

Tasks related to this deliverable:

Task No.	Task description	Partners involved°
T3.6.3	Implementation and integration of basic services in mobile devices	TID*, BSC

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

Executive Summary

Grid computing and other sibling concepts like metacomputing, virtual organizations (VOs) and, more recently, "cloud computing" have been around for more than two decades. However, they have failed to reach wide acceptance, except among certain scientific communities and, even there, complaints are often heard about the difficulties of using and administrating it, specially in large scale projects. The **XtreemOS** project aims to change this situation by delivering a modified Linux operating system with grid capabilities embedded in it, so that operating inside VOs is transparent for users, easy to manage and scalable to thousands of nodes.

Moreover, it has also been an old aspiration of grid computing to extend its reach, not only to PC computers, but also to any machine capable of providing any kind of computing, storage or interface capabilities, from high performance computing (HPC) clusters to mobile phones. XtreemOS also addresses this aspiration, by the inclusion of several flavours of the operating system for PCs, clusters and mobile devices. **XtreemOS-MD** is the flavour of XtreemOS designed to run in mobile devices like mobile internet devices (MIDs), personal digital assistants (PDAs) or mobile phones, although this software could also be made to run over other kinds of embedded devices such as set-top-boxes or satellites.

In order to realise this vision, XtreemOS-MD has been divided into two layers: a low-level foundation layer (F-layer) which embeds virtual organization concepts into the operating system mechanisms, and a high-level grid services layer (G-layer), which implements and coordinates the interactions among multiple nodes across the VO, to provide distributed computing and storage in a secure manner.

This document closes the development process for the basic version of XtreemOS-MD, by describing the **software modules** that comprise the **G-layer** of XtreemOS-MD. This description includes not only the technical details about the software, but also building and installation instructions for any potential user that would like to try the software. Moreover, it also includes the programming interfaces as well as usage examples for each of the modules.

The modules described in this document are:

- A fully modular and flexible **Credential Acquisition Framework** (CAF), which will be used in XtreemOS-MD to obtain the virtual organization user credentials from the Credential Distribution Authority (CDA), as defined in the XtreemOS security infrastructure. This framework not only is able to connect to XtreemOS CDAs, but can also be customized to access other infrastructures like Kerberos or MyProxy. This module also includes a proxy server for the CDA, which relieves mobile devices from several of the time-consuming operations that public key infrastructures (PKI) impose on a mobile device.

- A **XtreemFS client** for mobile devices, which uses common Linux mechanisms like FUSE and VFS to make files stored in the grid filesystem to appear as local files. It is a porting of the PC flavour XtreemFS client, with additional options suitable for mobile devices enabled (e.g. caching mechanisms) and several bugfixes, which have already been ported back to the PC flavour of XtreemFS.

- An **Application Execution Management** client for mobile devices, which will allow mobile users to launch, monitor and manage jobs running in the virtual organization's resources. This module is based on the C implementation of the XtreemOS Application Toolkit Interface (XATI).

- A version of the XOSAGA API for mobile devices, which will allow mobile grid applications to be developed using a simple C++ programming interface based on the Open Grid Forum (OGF) SAGA standard. Moreover, this will also provide portability for applications programmed with this API, independently of the underlying grid middleware (be it XtreemOS or otherwise).

- A number of tools gathered into an **Application Integration Kit** (AIK) for mobile devices, which eases the integration of XtreemOS mechanisms (such as the aforementioned Credential Acquisition Framework) with new and existing (legacy) applications, providing single sign-on, autoconfiguration and auto-mounting functionalities to end user applications in a transparent way (both for application programmers and for mobile end users).

The development of XtreemOS-MD will from now on concentrate on integrate the different software modules with each other, and with the other layers of the XtreemOS operating system, to deliver a mobile grid operating system that is easy to install and easy to use, even by non-expert users. Thus, the first release of XtreemOS-MD for MIDs/PDAs will be announced some time before the summer of 2009.

Contents

List of Figures

Chapter 1

Introduction

XtreemOS is an open source operating system that integrates grid functionalities, allowing its users to interact inside virtual organizations (VOs) that are secure, highly scalable, and which are easy to setup and easy to use. The final vision would be for users to access the resources of the virtual organization (e.g. CPU, storage) in the same way as they would use their local computer's resources. And by "local computer", we understand not only standard PCs, but also clusters of PCs and even mobile devices.

XtreemOS-MD is the mobile device flavour of XtreemOS which, on the first release of the software, will provide a number of open source packages for Linux PDAs and similar devices (e.g. internet tablets), which will enable those devices to act as XtreemOS grid clients in a transparent and easy way.

In every XtreemOS system, the software components can be grouped into two basic **layers**: a low-level Foundation layer (or F-layer) which is intended to provide operating system entities with support for VO entities (e.g. VO users, as opposed to local users). On top of this layer, the Grid Services layer (or G-layer) provides grid-wide functionalities such as reliable and scalable data management or application execution management.

This document presents the **implementation of the XtreemOS grid services for mobile devices**, more concretely, its basic implementation suitable for running in Linux PDAs. It will describe the functionalities provided by the different software components, detail how to build and install them, and briefly describe the interfaces (e.g. user interfaces, programming interfaces) through which those functionalities can be accessed.

1.1 XtreemOS-MD usage scenarios

In order to better picture what kind of functionality can be expected from the software described in this document, please read the following **typical user scenario**:

> Bob has just started working for an aerospace simulation company (ASC), which is part of a group of companies and universities who

share their resources for conducting simulations more efficiently. This group has been constituted into a virtual organization which we will call “Simulations VO”.

When Bob joins the company, he is automatically registered as a user of that VO, receiving for example a username and password for using the VO’s resources.

On his first week at ASC, Bob has to visit some customers to present the results of the latest simulations, and decide the next steps to be taken. Bob receives from the company a brand new XtreemOS-MD Internet Tablet.

On his way to the airport, he decides to work out some details with his workmate Alice. To their surprise, they discover that some of the needed simulations had not been run.

Bob uses his tablet to start the simulations, which perform complex renderings and obtain 3D video renditions of the simulation. The simulation applications and data are stored in VO servers only accessible to members of Bob’s company with a certain security qualification, and a user/password is asked before granting access. However, the interface for accessing all this grid data is the same as with any other local file or program.

Once the simulations are ended, Bob accesses from his device the (gigabyte-sized) video files which are stored in other VO machines, and visualizes them without having to enter additional credentials. Now that all the simulations have been performed, Bob and Alice can continue preparing the meeting...

1.2 XtreemOS-MD architecture

This first version of the XtreemOS-MD software is structured in a series of open source software packages that should be easily integrated in any modern mobile Linux distribution for PDAs or Internet tablets which use the ARM architecture. Although most of the code is independent of the rest of the software stack, XtreemOS-MD will be packaged for easier installation in two Linux distributions: Maemo [4] and Ångström [1].

The typical archictecture of a XtreemOS-MD software stack is pictured in figure 1.1.

As it has been already mentioned, the current document relates to the G-layer of the software (in yellow in the figure).

1.3 XtreemOS-G modules in mobile devices

The modules described in this document include:

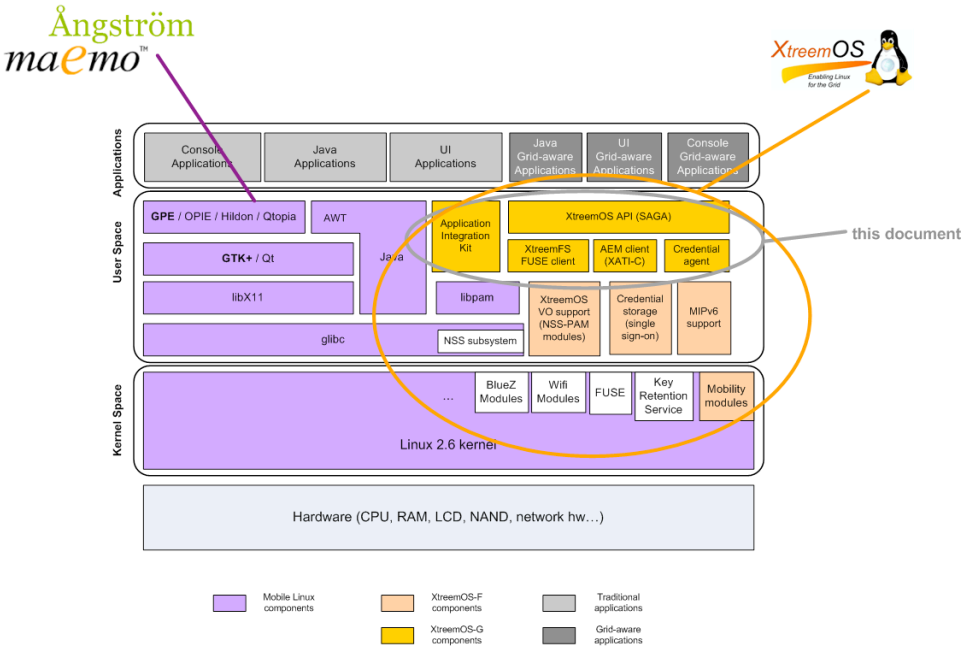

Figure 1.1: XtreemOS-MD Software Architecture

- A version of the XtreemOS API (XOSAGA) for mobile devices. This is the official XtreemOS API for grid applications, which is based on the Open Grid Forum's Simple API for Grid Applications (SAGA) [6]. By using this API, any SAGA-based grid application will be able to run on XtreemOS (regardless of the original grid infrastructure used when it was first designed).
- A credential acquisition agent, which is a modular, extensible infrastructure for obtaining security credentials, either XtreemOS user certificates (obtained from a Credential Distribution Authority [15]) or other kinds of credentials.
- A XtreemFS grid filesystem client [11] for mobile devices, which uses FUSE [3] to make the grid-wide filesystem to appear as any other local filesystem for accessing grid data securely.
- An XtreemOS Application Execution Management (AEM) client [2] for mobile devices, used for launching and managing grid jobs.
- An Application Integration Kit, containing a number of utilities, libraries and scripts that can be used for easily integrating XtreemOS functionality into legacy and new mobile end-user applications.

1.4 Document structure

The rest of the document is structured as follows: chapters 2 through 6 will present the different modules included in the software: the credential acquisition agent (chapter 2), the XtreemFS client (chapter 3), the AEM client (chapter 4), the XOSAGA interface (chapter 5) and the Application Integration Kit (chapter 6). These chapters will cover the user's guide to building, installing and configuring the software, the interfaces for usage and programming, and will also include examples of how to use it.

Afterwards, chapter 7 explains some of the next steps to be taken after the first software release, in order to design and implement the advanced version of the XtreemOS-MD software which, among other things, will add support for smartphones.

Chapter 2

Credential Acquisition Framework for Mobile Devices

In order to securely authenticate and interact with a virtual organization, a XtreemOS user has to obtain adequate credentials which guarantees his identity. In an XtreemOS system, this is done by obtaining a kind of X.509 credentials (called XOS-Certificate, or just XOS-Cert) from a XtreemOS credential distribution service, called Credential Distribution Authority (CDA). Please refer to deliverable D3.5.5 [15] for more information about this process and about the CDA.

In XtreemOS-MD, and using the CDA client program as a starting point, we have developed an extensible, modular framework for obtaining user credentials, which includes the current XOS-Cert request, as well as other enhancements that make this process more adequate to a mobile device environment.

2.1 Main features

By installing this software, an XtreemOS-MD client will be able to:

- Obtain XOS-Cert certificates directly from the CDA for a certain VO, and store them securely in the device, either permanently (hard disk/permanent storage) or in memory (for single sign-on functionalities – see chapter 6).
- Obtain other kinds of certificates, or use other methods for obtaining them (e.g. from MyProxy, Kerberos) and store them securely in the device, either permanently or in memory (for single sign-on functionalities – see chapter 6).
- Obtain the certificates through what we call a CDAProxy (proxy as in mediator, not as in proxy certificate) node, which can also act as a credentials cache, or as a kind of "authentication method converter" (in case the VO or the device support different authentication methods).

- Configure this framework to use one of the many possible authentication methods, with or without using a CDAProxy, and to use different kinds of graphical interfaces to authenticate the user.
- Extend this framework to other authentication methods, or to different graphical interface stacks, by implementing a simple API.

2.2 Software description

The framework for the acquisition of certificates presented here is designed for **flexibility** and **modularity**, so that it can be used even outside the XtreemOS grid system, allowing grid and security implementors to mix and match different processes for interacting with the credential distributor and with the users.

The software is composed of several packages:

libxos-credagent is a modular library for obtaining credentials. It is designed to abstract the request of credentials, and make it independent of the concrete method used for getting them. The different methods have been developed in separate libraries in a *plugin* fashion, and new ones can be easily added, since they are dynamically loaded as needed, in a similar vein as Pluggable Authentication Modules (PAM) do. The current software implementation includes four different modules:

- **xos_credagent_example.so** is a simple example implementation, where the credential is expected to be in the module's configuration file.
- **xos_credagent_runprogram.so** is another simple example, that invokes a custom application, which is expected to return the credential over its standard output (e.g. could be a script that downloads the credential from a web server and uses `cat` to return it).
- **xos_credagent_cdaclient.so** asks for the credential to an XtreemOS Credential Distribution Authority (CDA). This is the typical method to be used in a XtreemOS system.
- **xos_creadgent_readfile.so** expects the credential to be read from a local file. This file can be encrypted, and several ways of interacting with the user are supported (e.g. to confirm that it is the user who wants to read the file, or to ask for a passphrase, etc).

Apart from these modules, the interactions with the user have also been abstracted so that different implementations can be used interchangeably, depending on the device's software and the concrete authentication methods desired. Currently, only one implementation is included:

- **xos_creduiagent_gtk.so** uses the GTK+ graphical interface toolkit for graphical interaction with the user, which is present in all XtreemOS-MD devices.

cdacclient includes the concrete implementation of the protocol to obtain XOS-Certs from an XtreemOS CDA, both as a library and also as a configurable application which uses this library.

cdaproxy implements a proxy (mediator) process, residing in a non-MD node, which can be used to alleviate the load that generating private/public keys pose for limited devices, and which can also double as credentials cache, or even as an authentication system converter, in case the device and the infrastructure use different authentication methods (e.g. Kerberos, MyProxy).

Figure 2.1 shows the relationships among these pieces of software, as well as with other security and VO-related components in XtreemOS-MD (see chapter 6 and D2.3.4 [17] for a more detailed explanation of those modules), providing a fully customizable and extensible security and single sign-on framework suitable for mobile devices.

2.3 System requirements

2.3.1 Hardware requirements

XtreemOS-MD is built to be run in ARM devices (such as PDAs or internet tablets). However, the source code will build and install equally well in standard PC architectures, and in any other architecture that supports the software dependencies below (e.g. MIPS, PowerPC).

2.3.2 Software requirements

This framework has been developed with the design goal of making it integrable with almost any mobile Linux distribution, thus eliminating all but the most indispensable software dependencies. The software dependencies of each package are:

For libxos-credagent:

- GLib2

For cdacclient:

- OpenSSL 0.9.8

For cdaproxy:

- OpenSSL 0.9.8
- GLib2

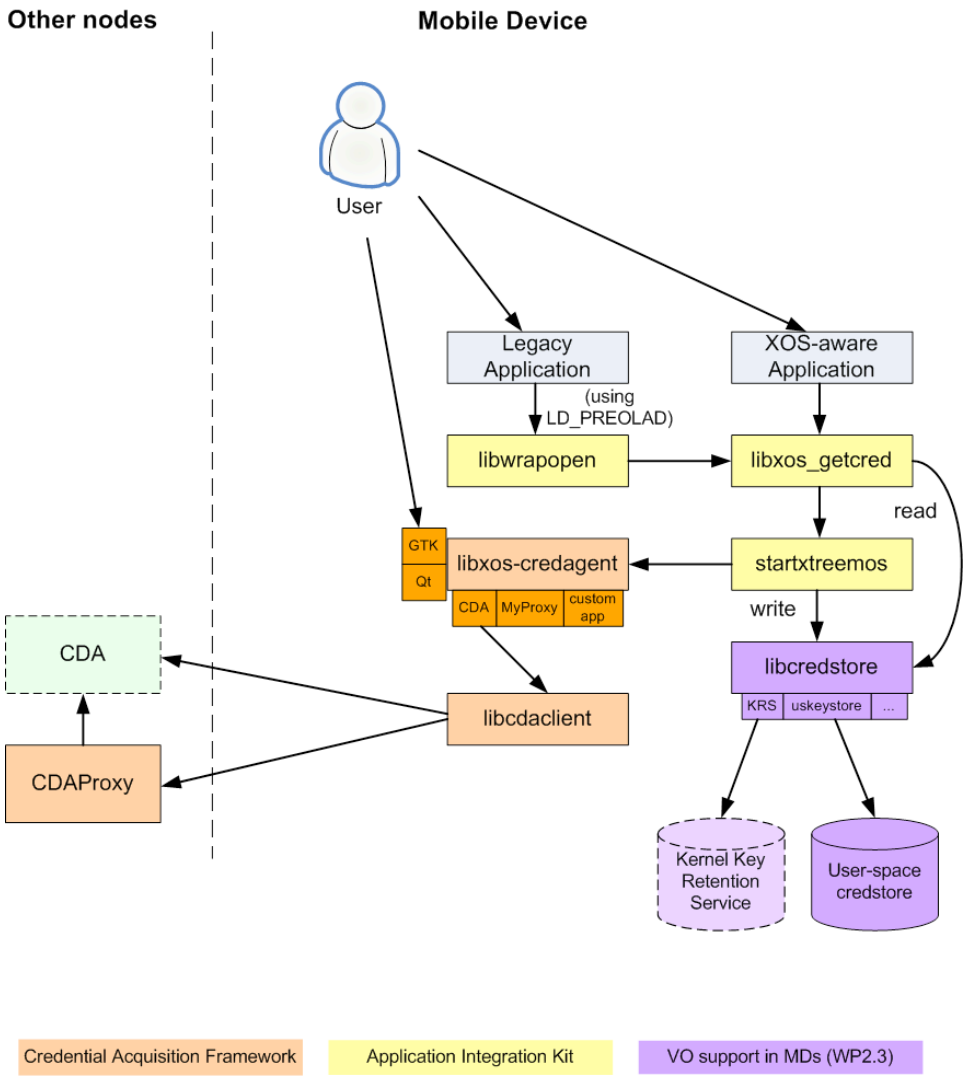

Figure 2.1: XtreemOS credential acquisition + single sign-on framework

For xos_creduiagent_gtk:

- GTK+ 2

For xos_credagent modules:

The software dependencies vary with the different modules in use, but no module uses more than:

- OpenSSL 0.9.8
- cdacclient

2.4 Installation

The process for building and installing the packages that form the framework is straightforward:

1. Download the packages from the XtreemOS SVN repository:

```
svn co
  svn+ssh://user@scm.gforge.inria.fr/svn/xtreemos
  /xtreemos/grid/mobile/caf
```

2. Execute as root: `install.sh`

2.5 Configuration

In order to configure the framework to function in the desired mode, a number of configuration files have to be edited. The following commented examples will help in understanding the way they operate:

For libxos-getcred:

The different configurations for this library (e.g. there can be one configuration for each end user application, groups of applications or just a single configuration for the whole system) are specified in the files `/etc/xos/creds/<configuration>.conf`. An example file can be seen below:

```
[general]
credagent=xos_credagent_cdaclient
creduiagent=xos_creduiagent_gtk
[credagent]
cda_host=192.168.15.45
cda_port=8080
```

```
cda_vo=votid
cda_user=bob
#cda_password=password
#ask_confirm=true
use_ssl_proxy=false
use_proxy=true
#server_pem=server_pem
#ca_pem=ca.pem
cache_file=/tmp/credential.pem
fsuidisuid=true
renew_if_expire_before=36000
```

The format of files is similar to that of ".desktop" files in GNOME and KDE, and are also very similar to the Windows ".ini" files [1].

A configuration file must include at least three sections: `general`, `credagent` and `creduiagent` (the `creduiagent` section may be omitted if the credagent does not use interactions with the user).

[general] section

Inside [general] section, five parameters are available. The only mandatory one is credagent:

- `credagent`: This parameters sets the name of the credential acquisition module to be used. The module must be a similarly named ".so" file present in `/lib/security`.
- `creduiagent`: This parameter sets the name of the user interaction module that must be present in `/lib/security`. This module is optional, since the credential acquisition module may not interact with the user, or may interact with him using a method hardcoded in the module itself.
- `alloweduser`: If this parameter is used, the module may be used only by programs with the UID (real UID, not EUID!) of the username specified. If this parameter is used in combination with `allowedgroup`, members of the group specified in the group allowed are also authorized.
- `allowedgroup`: If this parameter is used, the module may be used only by programs with the GID (real GID, not GUID!) of the specified group. If this parameter is used in combination with `usergroup`, the user specified is also authorized even if it is not a member of the allowed group.
- `fsuid_isuid`: If this parameter is true (the default value is false), then FSUID (the UID used to control the access to files and to set the owner of

[1] For more details about the valid syntax of these files see the Desktop Entry Specification in http://www.freedesktop.org/wiki/Specifications/desktop-entry-spec

the files created by process) is filled with the real UID instead of with the EUID. This option is useful, for example, when the program that invokes `libxos-credagent` is SUID root, but the files should be accessed or written as the user that launched the program. *Note: this variable doesn't change FSGID, which is the EGID.*

[credagent] and [creduiagent] sections

The contents of credagent and creduiagent sections are only used by the credagent and creduiagent modules and may contain arbitrary parameters that only each particular module may interpret. Therefore, information about the values inside credagent and creduiagent sections are described in the documentation of each particular module.

For CDAProxy:

CDAProxy configuration file has the same syntax as GNOME/KDE ".desktop" files. The configuration file has a mandatory section (`[proxy]`) with the general configuration of the proxy (e.g. information about ports, whether to use or not SSL, proxy server certificate etc). Additionally, it has a section for each of the user it supports. For example, if proxy accepts connections from user `bob`, a `[bob]` section should exist in the configuration file. A optional section `[all_users_defaults]` contains parameters that are will be inherited by all users in the different user sections. Finally, the configuration file may have a special user section: `[other_users]`. If this section is present, the configuration is applied to any user that does not have its own user section. If this section is missing, the proxy will deny access to any user that has no user section.

```
[proxy]

# proxy server use SSL in connections with clients?
usessl=false

# TCP port where the proxy listen connections
proxyport=8080

# File with the RSA key and certificate of the proxy
# server. If you want to generate a self-signed certificate,
# run cdaproxy_createproxycert.sh script.
# This parameter is mandatory if proxy server use SSL.
proxypem=server.pem

## For each username accepted by the proxy, you must
## include a section named with the username. In this
```

```
## example, we included the users "bob" and "mary"
##
## all_users_default section contains values that are
## inherited by all users section. Users sections may overwrite
## these parameters.
##
## Optionally, you may include a other_users section. If this
## section is present, it apply to any user not listed
## explicitly. If section is missing only users with a section
## in the configuration file are authorized to use the proxy.
##
## For each user, it's possible to load the credential
## from a local file (attribute type=local) or request
## it to a CDA server (attribute type=cdaserver). In both
## cases, attribute "proxy_password" is optional: if
## present the proxy authenticates user. If omited and
## type=cdaserver, only cdaserver authenticates. If it is
## omitted and type=local, user password is used
## to decrypt the RSA private key.
##
## If type=local, parameter credential_file with the
## path of a PEM file with the RSA key and public
## certificate is mandatory.
## If type=cdaserver, parameters cdahost and cdaport are
## mandatory; proxy_password, cda_user, cda_password,
## servercert and cacert are optative.

[all_users_default]

# Certificate of the server or of the CA that sign the
# server certificate. If this parameter is missing,
# don't check the server certificate (this is insecure but
# useful for debugging).
#servercert=cdacert.pem

# Root certificate of the CA that signs the obtained
# certificate. If this parameter is missing, don't
# check the obtained certificate.
#cacert=cacert.pem

# The CDA server hostname
cdahost=xtreemos-b.esc.rl.ac.uk

# The CDA server port
```

```
cdaport=6730

credential_file=credential.pem

[bob]
# credential is inside credential.pem; the RSA key is
# encrypted with the password of "bob". Proxy check
# password trying decrypting the RSA key. In the
# example file, key is crypted with passphrase "password".
type=local

[mary]
#type=cdaserver
# password to authenticate client in our proxy (if it is
# missing, authenticate only with server)
#proxy_password=1417

# user to authenticate with cda server (not required
# if it's the same as the username on the proxy)
# cda_user=mary

# password to authenticate with cda server (only
# needed if proxy_password is present and the
# password is different)
#cda_password=ax!48591XjzFl

[other_users]
type=local
```

2.6 Interface

libxos-credagent

As mentioned above, `libxos-credagent` is a library to implement a pluggable, modular system to get credentials. The objective of the library is to allow administrators to change the method used by applications to get the credentials to authenticate in a Single Sign-On (SSO) system, without source code modification. The basic interface for this library is:

```
char *xos_credagent_getcred(char *configuration);
```

This function returns the credential using the credagent configuration specified in file `/etc/xos/creds/<configuration>.conf`. The function returns NULL if there is any error (e.g. the user is not authorized to read the credential). The returned credential must be `free`'d by the caller.

Additionally, and in order to read any additional parameters from these configuration files, applications have the following interface available:

```
int xos_setconfigenv(char *config_name,char *section);
```

This functions maps all the parameters of the specified section in environment variables, which can be read using `getenv`. The function returns zero on success.

However, it is recommended that end-user applications do **not** invoke `libxos-credagent` directly, but run a wrapper such as `startxtreemos` (see chapter 6) instead. This kind of application has permission to read the `/etc/xos/creds` directory.

The advised behavior of applications is to try to obtain the credential from a cache (a credential store, e.g. using libcredstore, see D2.3.4 [17]) and then, only if it is not available, to run `startxtreemos` (which in turn invokes `libxos-credagent`). This behaviour is implemented in `libxos-getcred`, see chapter 6.

libcdaclient

`libcdaclient` is a C library developed in order to get a credential from a CDA server or from a CDAProxy. It is used in several applications such as `cdacclient`, `cdaproxy` and in the `xos_credagent_cdaclient.so` module. A program that uses `libcdaclient` must include the header `cdaclient.h` and be linked with the `-lcdaclient` option.

The ordinary method to obtain a credential from a CDA server is:

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

Where the parameters are:

- The `hostname` and `port` of the CDA server (mandatory).
- `CDAcertfilename` (optional): if not `NULL`, it should contain the name of a file containing the server certificate of the CDA server in PEM format, to test that the SSL connection is really with CDA server and not with a rogue server (to prevent a man-in-the-middle attack).
- `RootCAcertfilename` (optional): if not `NULL`, it should be the name of a file containing the root CA certificate (in PEM format) that signs the obtained credential. This is used to verify that the credential is authentic.
- `username` and `password` (mandatory): these are used to authenticate in the CDA server, and in order to indicate the user for whom the credential will be generated.

- `voname` (mandatory): the VO to which the user belongs (in this credential, at least).
- `credential` (mandatory): a pointer to a variable that will be filled with the credential in PEM format. If the `certificate` parameter is also filled in, only the private key will be stored in `credential`. If `certificate` is `NULL`, the certificate will also be included in `credential`.
- `certificate` (optional): the certificate part of the credential. This parameter may be `NULL`, and in that case `credential` will store the private key and the certificate.

The function returns zero on success.
A variant of this function is:

```
int cda_client_defergen(char *hostname,
              int port,char *CDAcertfilename,
              char *rootCAcertfilename, char *username,
              char *password, char *voname,char **credential,
              char **certificate);
```

This function is a variant of previous one. The only difference is that the RSA private key is generated after connecting and authenticating. This variant is useful for the implementation of CDAProxy, where the RSA key is generated by the proxy and the proxy does not authenticate the user but the CDA server. This is done to avoid denial of service attacks.

```
int cda_client_nossl(char *hostname,int port,
                        char *username,char *password,
                        char *voname, char **credential,
                        char **certificate);
```

This function is another variant of the first function, that uses a plain connection with the CDA server instead of a TLS/SSL one. Currently this function is not very useful, since the CDA server only accepts TLS/SSL connections, but could be useful in special cases where the connection with the CDA is trusted.

The `libcdaclient` library also supports **obtaining credentials through a CDAProxy**. In this case, the interface is slightly different:

```
int cda_client_proxy(char *hostname,
              int port,char *Proxycertfilename,
              char *RootCAcertfilename, char *username,
              char *password, char *voname,char **credential,
              char **certificate);
```

The parameters of this function are similar to those in `cda_client()`, but in this case the the `host` and `port` are those of the proxy, not the CDA ones. This function returns zero on success.

A variant of this function, for use when connecting to the CDAProxy through a plain connection (i.e. without TLS/SSL), is also available:

```
int cda_client_proxy_nossl(char *hostname,int port,
                           char *username, char *password,
                           char *voname, char **credential,
                           char **certificate);
```

cdaclient

`cdaclient` is a version of the CDA Client written in C instead of the original Java version. It was written using `libcdaclient` and therefore it inherits all of its features, as CDAProxy support.

This small application prints the credential (RSA private key unencrypted + X509 certificate signed) in PEM format, using standard output. Its usage follows the syntax:

```
cdaclient [options] user@cda_host:cda_port/vo
```

With the following options available:

- -h : shows help message
- -a : ask password using getpass Unix function instead of reading from stdin
- -p : use a proxy
- -n : don't use SSL in connection with proxy
- -c <server_cert.pem>: check server/proxy certificate
- -v <caroot_cert.pem>: verify the obtained certificate

When a credential is requested, it is required to provide the CDA address (`cda_host:cda_port`), the username (`user`) and the Virtual Organization (`vo`; each user may be member of several virtual organizations and each credential is valid only for authenticating users in one VO).

CDAProxy

To launch CDAProxy, simply run `cdaproxy`. CDAProxy admits two parameters, but both are optional:

- `-c <configfile>`: Use `configfile` as the configuration file for the proxy. If this option is missing, configuration file is `/etc/xos/cdaproxy/proxy.conf`.
- `-f`: run in the foreground. By default, `cdaproxy` runs as a daemon in the background. This option is useful for debugging purposes.

Apart from the proxy itself, a number of command line utilities are included in the package:

- `dump_server_chain`: This program receives as parameter a TLS/SSL server address in form `hostname:port`. It connects with the specified server and dumps the server chain certificates to the standard output. This utility can be used to obtain the CDA server root CA certificate, in order to include it in the file referenced by the `servercert` parameter in the proxy configuration.
- `cdaproxy_createproxycert.sh`: This script generates a RSA private key and a self-signed certificate valid for 365 days. The content is saved in `server.pem`. This file may be used as the private key and certificate of the SSL proxy server socket (see `proxypem` parameter in the proxy configuration file).
- `cdaproxy_createtestcred.sh`: This script generates a test credential, with a RSA private key encrypted.

2.7 Usage examples – Testing

Once the software has been built and installed in a mobile device, a small test application can be used to check that everything is working correctly. `xos-getdumpcred` is a very small application that asks for the credential (using `libxos-getcred` – see chapter 6 –, which in turn uses the Credential Acquisition Framework) and displays it on the standard output.

The configuration files used in these examples are located in the `examples/` directory of the package, and should be put into `/etc/xos/creds` for making them effective.

Very simple example

In this case, use the simple.conf configuration file, and just run into your terminal[2]:

```
xos_getdumpcred simple
```

[2]It is recommended to run the examples in a console inside a X-Window desktop, because the `creduiagent` module used in some of the examples uses dialog windows to interact with user.

This example uses the example credagent module (which reads the credential directly from the module's configuration file) and the GTK creduiagent module (thus, a dialog window will appear asking for user permission to read the credential from disk).

If we execute the same command again, the credential will be shown immediately without asking for confirmation, since the credential was already stored in the single sign-on system (also known as the credstore).

Encrypted credential example

In this example, we will use a different configuration in order to read an encrypted credential from `/etc/xos/creds/credential.pem`, and ask the user a password for decrypting it. Thus, copy `readfile.conf` to `/etc/xos/creds/` and run:

```
xos_deletecred # past credentials deleted from credstore
xos_getdumpcred readfile
```

In this case, the dialog window will appear asking for a password to decrypt the credential (which is, by the way, "password"). You will also notice that no other window can be used until the password is entered or the operation is canceled; this is another option that can be set in the configuration file.

Please note that the credential files are not readable by common users, but only through the startxtreemos wrapper application, which libxos-getcred uses (see chapter 6).

Credential from CDA example

To run this example, a CDA server should be available and reachable from the mobile device. Also, a virtual organization must be already created and a user should be part of that VO for this example to work. All these parameters must be used to edit the `/etc/xos/creds/cda.conf` configuration file, which can be taken from the `examples/` directory. If we just run:

```
xos_deletecred # past credentials deleted from credstore
xos_getdumpcred cda
```

In this case, the VO user's password for authenticating with the CDA server will be asked and, if correct, the unencrypted RSA key and X.509 certificate in PEM format will be printed. Alternatively, to examine the certificate, the following command can be used:

```
xos_getdumpcred cda | openssl x509 -noout -text
```

(In this case, the credential should be printed immediately since it was already in the credstore)

Credential from CDAProxy example

In this case, apart from all the requisites of the previous section (CDA server, VO and VO user), a CDAProxy must be running in a fixed node. Please refer to the above sections on how to install and configure a CDAProxy. In this case, we can use the `cdaproxy1.conf` file that is provided, customising it for our concrete data.

```
xos_deletecred # past credentials deleted from credstore
xos_getdumpcred cdaproxy1
```

(If running on a real mobile device, a substantially shorter wait time should be noticed)

Chapter 3

XtreemFS client for mobile devices

As it happens in any other grid system, in XtreemOS it is very important to manage the data of the users, which is distributed across the virtual organization. The ability to store and access all this information in an scalable, reliable and efficient way is the goal of XtreemFS, the grid filesystem that is included within the XtreemOS operating system. With XtreemFS, users can store and access data in remote servers across the VO, with the permissions and policies that such a VO dictate, as if they were any other kind of local files. Please refer to deliverable D3.4.2 [11], or to the XtreemFS website [8] for more detailed information about XtreemFS.

XtreemOS-MD allows access to this grid filesystem through a client access layer, which is a slightly modified version of the original module, to work in ARM architectures and function more efficiently, through the use of cahing mechanisms. In fact, these modifications have been found so useful that they have also been incorporated into the main development line (e.g. for the PC flavour) of the XtreemFS client.

3.1 Main features

By installing this software module, XtreemOS-MD users will be able to:

- Mount and unmount XtreemFS volumes from the mobile device.
- Access files in XtreemFS volumes through the same interface (POSIX) as a user would use to access any other Linux filesystem.
- Store files in an XtreemFS volume, with the consequent guarantees of reliability and scalability that the XtreemFS system implies (see [8]).
- Both access and storage will be done in a secure way, by using XtreemOS certificates, and will be controlled by the virtual organization's policies.

- Improved file access performance thanks to caching mechanisms.

3.2 Software description

This software module is based on XtreemFS's access layer, a client software that uses Filesystem in Userspace (FUSE, [3]) and Linux's Virtual Filesystem (VFS) to make this object-based grid filesystem to look like any other normal, local filesystem. More details into the internal workings of the XtreemFS access layer can be found in other documents like D3.4.1 [10].

The main modifications of the XtreemOS-MD version of this access layer is that it has been gone through an additional stage of testing and development to enable the caching mechanisms that are crucial to obtain acceptable performance in mobile device environments.

This process has also helped to correct uncovered errors and improve the overall stability and performance of the client. In fact, all the changes done in the original code have also been added to the latest release of the PC version of XtreemFS (version 0.10.0).

3.3 System requirements

3.3.1 Hardware requirements

XtreemOS-MD is built to be run in ARM devices (such as PDAs or internet tablets). However, the source code should build and install equally well in standard PC architectures, and in any other architecture that supports the software dependencies below (e.g. MIPS, PowerPC,...).

3.3.2 Software requirements

In order to build the XtreemFS client for mobile devices, the following dependencies must be met:

Build-time dependencies:

- openssl-dev 0.9.8
- gmake 3.81 or higher
- gcc 4.1.2 or higher

Run-time dependencies:

- openssl 0.9.8
- libxml2
- kernel-module-fuse (2.6 or higher kernels)

- fuse-utils
- libfuse2 2.6

3.4 Installation

The process of building and installing this client software in mobile devices is similar to that of the PC flavour. The basic process consists of downloading the code (in a tarball or directly from SVN), building and installing it.

Download the code

You can download the XtreemFS client for mobile devices source code from the XtreemOS website [9]. Alternatively, since the modifications for mobile devices were integrated into the main line of XtreemFS in revision 4392, and this is the version of the code that has been throughly tested, users can download it from the SVN repository:

```
svn co -r4392 \
 svn+ssh://user@scm.gforge.inria.fr/svn/xtreemos/WP3.4/trunk
```

Build and install

To build and install the client, just enter the source code directory and type:

```
make client
```

3.5 Configuration

No additional configuration is necessary for beginning to use the client, since the parameters for connecting to XtreemFS servers are specified in each command. However, before starting to use it, we have to make sure that the FUSE module is loaded:

```
root# modprobe fuse
xtfs_mount -o dirservice=http://DIRSERVICE,\\
volume_url=YOUR_VOLUME,direct_io \\
YOUR_MOUNT_POINT
```

In order to test XtreemFS from mobile devices, users can find a test script in the `bin/` directory: `remote_ALtests`. In order to test XtreemFS:

- Start a XtreemFS installation (OSD, MRC and DS) in a remote (fixed) node.
- In the mobile device, run the `remote_ALtests`, specifying the host where XtreemFS was started.
- Type `test` and wait until the results are shown.

3.6 Interface

The XtreemFS client for mobile devices does not have a specific API in order to access the files in the grid filesystem, since that is precisely the goal of the system, to be transparent and appear to the user as any other local filesystem. This is achieved using Linux's Virtual File System (VFS).

However, there are a number of commands and utilities that are distributed with the client, in order to perform basic operations in the filesystem, such as mounting and unmounting XtreemFS volumes in your mobile device. These commands include:

- **xtfs_mount** is used for mounting XtreemFS volumes:

```
xtfs_mount -o dirservice=http://DIRSERVICE,\\
volume_url=http://remote.mrc.machine/myVolume \\
/mount_point
```

- **xtfs_umount** is used for unmounting XtreemFS volumes:

```
xtfs_umount /mount_point
```

- **xtfs_stat** displays XtreemFS specific file information such as the striping policy and the OSDs where the file resides:

```
xtfs_stat filename.ext
```

For more information, please refer to the XtreemFS documentation [8].

3.7 Usage examples

An example session, running the XtreemFS tests as described above, is depicted below. XtreemFS services are supposed to be installed in a desktop PC (in `/XtreemFS`), and the mobile client is supposed to be built in the mobile device (in `/XtreemFS`).

1. Start the XtreemFS services (in a PC):

```
user@PC:~$ cd /XtreemFS
user@PC:/XtreemFS$ ./start.sh
```

2. Execute the test script in the mobile device

```
user@MD:~$ cd /XtreemFS
user@MD:/XtreemFS$ cd bin
user@MD:/XtreemFS/bin$ ./remote_ALtests --host PC
```

```
Temporary directory for cfg, logs and data is
/tmp/xtreemfs_XXLVPDJc
## making volume x1 (stripesize=128, width=1) in
## mounting volume to /tmp/xtreemfs_XXLVPDJc/mnt/x1
+ set +x
Enter 'test' if you want to run the tests.
Anything else will skip the tests and cleanup.
> + ./xtfs_mount -f -d -o volume_url=http://bscib04/x1
-o direct_io,logfile=/tmp/xtreemfs_XXLVPDJc/x1.log,debug=4
-o caching=1 /tmp/xtreemfs_XXLVPDJc/mnt/x1
```

3. Write `test` and press 'Enter':

```
test
  PID CMD                              RSS    SZ
  *** Starting parallel ddwrite tests ***
==== Writing in volume x1 ====
 --- 10MB, 1 client(s) ---
5.34759 MB/s
 PID CMD                             RSS    SZ
 --- 10MB, 2 client(s) ---
1.19474 MB/s
...
```

During the tests, the output will tell if tests proceeded correctly or not.

Chapter 4

Execution management client for mobile devices

One of the core functionalities of any grid system is the ability to launch and manage the lifecycle of computing processes (be them either batch jobs or interactive applications) across the whole grid or virtual organization, crossing the boundaries not only of users' local machines but also of their administrative domains. In XtreemOS this functionality is provided by the Application Execution Management (AEM) subsystem, which is in charge of. Please refer to previous WP3.3 deliverables (D3.3.2 [13] and D3.3.3-4 [2]) for more information about the AEM.

In the same way, one of the most important functions of XtreemOS-MD is to allow mobile users to do this launching and management of processes from mobile devices. The module described in this chapter is a porting of the C implementation of the XtreemOS Application Toolkit Interface (XATI-C or CXATE), to run in ARM architectures and in the limited computing environment of a mobile device (a PDA/Tablet, in this case).

4.1 Main features

Once this module has been installed in a mobile device, its users will be able to:

- **Manage jobs**
  - **launch**
  - **suspend** (stop execution, leaving the job in memory)
  - **resume**
  - **cancel** (kill the job)
  - **wait** on a job
- A mechanism to **monitor running jobs**. Monitored information includes:

- **launch time**
- **estimated time to end**
- **status** (running, suspended, awaiting execution etc)
- **resource consumption**
- **special notifications**

4.2 Software description

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.

Although native communication with the XOSD is normally performed via Java objects, in order for non-Java applications to access the AEM, a C translation of this XATI interface was created, and is delivered in the form of an static library (`libXATICA.a`). In this case, communication is achieved through XML-formatted messages, for which the parameters are set through a configuration file (`XATICAConfig.conf`).

The main components of this interface are:

- A XML wrapper and parser, used to exchange XML-formatted messages with the XOSD
- A communication module, used to communicate with the XOSD via sockets. The parameters for this connection are retrieved from the aforementioned configuration file.
- A CDA module that manages the user certificates through the XtreemOS CDA server.
- A job execution module, which is used to launch jobs and get the overall execution information.
- A job management module, which is used to perform actions over a determined job.
- A resource management module, used to manage all the available resources

4.3 System requirements

4.3.1 Hardware requirements

XtreemOS-MD is built to be run in ARM devices (such as PDAs or internet tablets). However, the source code should build and install equally well in standard PC architectures, and in any other architecture that supports the software dependencies below (e.g. MIPS, PowerPC,...).

4.3.2 Software requirements

In order to build the execution management client for mobile devices, the following dependencies must be met:

- openssl 0.9.8
- libxml2

4.4 Installation

The building and installation of the execution management client library for mobile devices is very simple:

First, download the code of the client, be it either from the SVN,

```
svn co
 svn+ssh://user@scm.gforge.inria.fr/svn/xtreemos
 /WP3.4/trunk/XATICA
```

or as a source tarball, and decompress it.

Go into the source directory and build it:

```
cd src
make all
make install
```

This process will generate a static library file, `libXATICA.a`, which can then be installed to the path where applications can find it.

4.5 Configuration

The main configuration file for this module is named `XATICAConfig.conf`, and it is also included in the package. This file should be installed in `/.xos`.

```
# Properties File for the client application
# Host address and port where the nearest XOSD can be found
xosdaddress.host=PORTATIL
xosdaddress.port=55000
# Address and port where the client is listening
address.host=PORTATIL
address.port=10001
# Location of the DIXI truststore for SSL
trustStoreSSL=/etc/xos/truststore/certs/dixi_ssl
# Whether to use SSL for connecting to the XOSD or not
useSSL=false
```

Another configuration file that must be installed in `/.xos` is `XATIConfig.conf`.

```
userCertificateFile=::xos::cert
```

Once these files have been adequately edited, and in case the user wants to run some simple tests using this library, some simple test programs can be found in the `test/` directory of the package. These tests can be run just by typing:

```
cd ../tests
make all
./testXATICA
./testXATICA2
./testXATICA3
```

4.6 Interface

Once it is built and installed, the execution management client for mobile devices takes the form of a (static) library, which provides an API for other applications to launch and manage jobs in a XtreemOS virtual organization. The main functions of this API are described below. A complete description of the API is available in the API documentation of the AEM.

Execution management module

```
int getJobsResource( char** returnValue);
```

This function returns a string with the information of the jobs that the user has running at the moment.

```
int createProcess(char* __jobId,char* __JSDL,
              char* __reservationID,char* __resourceID,
              char* __userCtx, int* returnValue);
```

This function is akin to a `fork`, but in the context of XtreemOS. If `JSDLPath` is `NULL`, it will use the provided in `createJob()` (see below). If a `reservationID` is provided, the reservation will be used; otherwise, if `resourceID` is provided, the specified resource will be used. Otherwise, the process will be created locally.

```
int getJobSelf(int __pid, char** returnValue);
```

Returns the JobID of the calling process process, identified by its PID.

Job management module

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

Creates a job in the AEM based on the JSDL description. The job can be automatically scheduled or just created, depending on the value of `startJob`. `jsdlFile` is the job description of the job to be created in JSDL format (the actual contents of the file, not the path to the JSDL file).

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

Starts the scheduling process of a previously created job. `JobId` must be a valid jobId in the system.

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

Applies a control operation to the specified job:

- 0: SUSPEND JOB
- 1: RESUME JOB
- 2: CANCEL JOB

```
int exitJob(char* __jobId,int __exitValue,char* __userCtx );
```

Finishes immediately the specified job (all the processes of the job), with the exit code passed as a parameter.

```
int getJobsInfo(ArrayList __jobIds,int __flags,
              char* __infoLevel,ArrayList __metrics,
              char* __userCtx, char** returnValue);
```

Returns the monitoring information of a list of jobs. `flags` denote the kind of information to return (in mask format):

STATUS return the job/process status

METRICS return all the metrics defined by the job

RESOURCES return the resource information

PERFORMANCE return the performance metrics

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

Returns the monitoring information of a given job.

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

Returns all the jobIDs of the jobs belonging to the given user.

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

Sends a specified event to a job.

```
int jobWait(char* __jobId, int* returnValue,
            char* __userCtx);
```

Blocks the calling process until the indicated job finishes.

Resource management module

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

Retrieves a collection of resources that match the job's resource demands. The resource query is expressed in a JSDL document with its contents in a string.

```
int getResInfo(char* __userCtx, void* returnValue);
```

Returns the monitoring information associated with the resource. Only the users authorised for access to the resource can obtain the information.

```
int getResMetrics(char* __userCtx, void* returnValue);
```

Returns the list of metrics available on that resource. Only the users authorised for access to the resource can obtain the information.

4.7 Usage examples

The following source code exemplifies the use of this execution management client from a mobile application:

```
#include <stdio.h>
#include <stdlib.h>

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

```
#include <XCResMng.h>

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

    /* 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 5

XOSAGA API for mobile devices

In the world of grid computing, currently there exist a great variety of middlewares and architectures which are used to achieve the common vision of distributed execution and storage across a virtual organization. This, together with the complexity of this kind of system, makes the development of applications that use grid computing a daunting task. In XtreemOS, the approach has been to adopt OGF's Simple API for Grid Applications (SAGA, [6]) as the official API for grid application developers, extending it to cover XtreemOS functionalities (and thus, termed XOSAGA). This will have the additional advantage of making existing SAGA-compliant grid applications able to run over a XtreemOS system.

As a gateway to XtreemOS grid's resources, the mobile flavour of XtreemOS will also support C++ SAGA-aware grid applications, so that any of those applications which is able to run in ARM architectures is also capable of working with XtreemOS mobile devices. Due to the limitations of mobile Java Virtual Machines, only the C++ SAGA implementation will be supported in the basic version.

5.1 Main features

At this stage of the development, the XOSAGA interface allows C++ applications to access the following functionality:

- Authenticate on behalf of the user, by means of the XtreemOS credentials system.
- Access files hosted on the XtreemFS filesystem.

5.2 Software description

The package presented here is based on the SAGA C++ Reference Implementation (SAGA++), which is the first complete implementation of the OGF SAGA standard for high-level Grid programming interfaces. As it is well explained in other WP3.1 documents (D3.1.3 [14]), this implementation is divided into:

- A **core engine** which contains a set of common "look and feel" classes, as well as the functional aspects of the engine, dealing with files, jobs, security contexts etc.
- A number of so-called adaptors, which implement each of those functionalities in the concrete grid implementation that lies underneath (in this case, the XtreemOS implementation).

Having this in mind, this package for mobile devices contains:

- The SAGA C++ reference implementation 1.0 (core engine plus a few example adaptors).
- The XtreemFS File adaptor for accessing files in the grid filesystem.
- The XtreemOS Context adaptor, in order to authenticate users that want to use any other XtreemOS servies, using XtreemOS certificates.

For more details on the SAGA C++ reference implementation, please refer to its official documentation [5].

5.3 System requirements

5.3.1 Hardware requirements

XtreemOS-MD is built to be run in ARM devices (such as PDAs or internet tablets). However, the source code should build and install equally well in standard PC architectures, and in any other architecture that supports the software dependencies below (e.g. MIPS, PowerPC,...).

5.3.2 Software requirements

In order to build the mobile XOSAGA API for mobile devices, the following dependencies must be met:

- GNU make
- g++ 3.4.6 or newer
- libboost 1.34.1 or higher version (and all its sub-packages, such as libboost-date-time etc)

5.4 Installation

The installation procedure of the mobile version of XOSAGA is very similar to the PC version one:

First, download the XOSAGA source code, directly from the SVN repository at,

```
svn co
 svn+ssh://user@scm.gforge.inria.fr/svn/xtreemos
 /xtreemos/grid/xosaga/trunk/c++
```

Once inside the source code directory, and if all dependencies are met, the user can just type:

```
./configure --prefix=$(INSTALL_DIR) --with-boost=$(BOOST_DIR)
make
make install
```

Please refer to the SAGA Installation Guide [7] for more information on this subject.

5.5 Configuration

Once it has been installed, the SAGA engine can be configured (please refer to [7] for more information).

First, the following environment variables must be set, in case custom paths were chosen at installation time:

```
export SAGA_LOCATION=/saga/install/path
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$SAGA_LOCATION/lib
```

In order to test the installation, a number of tests are included with the distribution, which can be run by typing:

```
make check
```

The main configuration file for the SAGA engine is `saga.ini`, which contains the information about the location of SAGA and the available adaptors. This file should be already preconfigured on installation, but a simple example is shown below:

```
[info]
ini_path = ${SAGA_LOCATION}/share/saga/

[saga.adaptors.<adaptor_name>]
name = <adaptor_instance_name>
path = <adaptor_path>
enabled = true
```

5.6 Interface

The complete API of SAGA can be accessed at the C++ Reference Implementation section of the SAGA official site [5].

5.7 Usage examples

The following code snippet shows how a simple client application can access grid functionality (accessing a file, in this case) by using the XOSAGA interface:

```
#include <string>
#include <iostream>
#include <saga.h>
int main () {
  try {
    // open a remote file
    saga::file f ("gsiftp://ftp.university.edu/pub/INDEX");
    // read data
    while ( string s = f.read (100) ) {
      std::cout << s;
    }
  } catch (saga::exception e) {
    std::cerr << e.what() << std::endl;
  }
}
```

Chapter 6

XtreemOS-MD Application Integration Kit

In most grid systems, the modification of existing applications to work with grid features, or the development of new applications that use the grid infrastructure is often a difficult subject, especially for application developers. In its efforts to make grid development as easy and transparent as possible, XtreemOS provides the XOSAGA programming interface, for developers of new grid applications, so that they can ignore the details of the myriad of different middlewares that they could use.

XtreemOS-MD takes this goal a step further, by providing a set of tools and libraries that can be used by mobile application developers in order to make the integration of grid features (such as the usage of credentials and single sign-on) very easy for those developers, and even to use these features in existing (legacy) applications, without modifications to their source code. Moreover, these features could also be useful for any desktop application developer, and will probably be available in next releases of the XtreemOS PC/cluster flavour[1].

6.1 Main features

By building and installing the Application Integration Kit described in this section, users (and/or developers) will be able to:

- Obtain credentials transparently to the applications, through a very simple programming interface, from a wide variety of credential acquisition mechanisms, either external (through the Credential Acquisition Framework, see chapter 2) or internal (through the credential storage framework developed in WP2.3 [17]).

[1]NB: Some of the modules described here were first proposed in WP2.3 (see D2.3.4 [17]), as part of the VO support for mobile devices package. However, the evolution of the development of those modules and their application-oriented goals prompted for their separation from the VO support package and their inclusion in WP3.6 as an independent package.

- Have single sign-on capabilities in a transparent way, regardless of the actual implementation of the credential scheme.
- Extensibility of these features to new authentication and credential acquisition schemes.
- Extensibility of these features to different user interface paradigms and GUI software stacks.
- Access to the single sign-on features from legacy applications without modificating their code, by just opening a specially-named file.

6.2 Software description

The Application Integration Kit, together with the Credential Acquisition Framework (see chapter 2) and the credential storage developed in WP2.3 (see D2.3.4 [17]), provide a comprehensive authentication and single sign-on solution for mobile clients, which is:

- Flexible
- Modular
- Extensible
- Easy to use
- Legacy application friendly
- Portable to other architectures and distributions
- Internationalized (support for various languages)

How all these modules interact to provide this single sign-on solution is shown in Figure 6.1:

The figure shows (in yellow) the three building blocks of the Application Integration Framework:

- A simple credential acquisition library (`libxos_getcred`) designed for use by end user applications, that hides all the complexity and means for getting the certificates and provide single sign-on access to them.
- A set of launcher applications (`startxtreemos`) designed to transparently perform all the necessary operations for initializing XtreemOS VO support features in a Linux distribution, under a number of VO and mobile device configurations. `startxtreemos` loads credential in credstore, mounts XtreemFS volume and creates XATICA configuration files at `/.xos` directory.

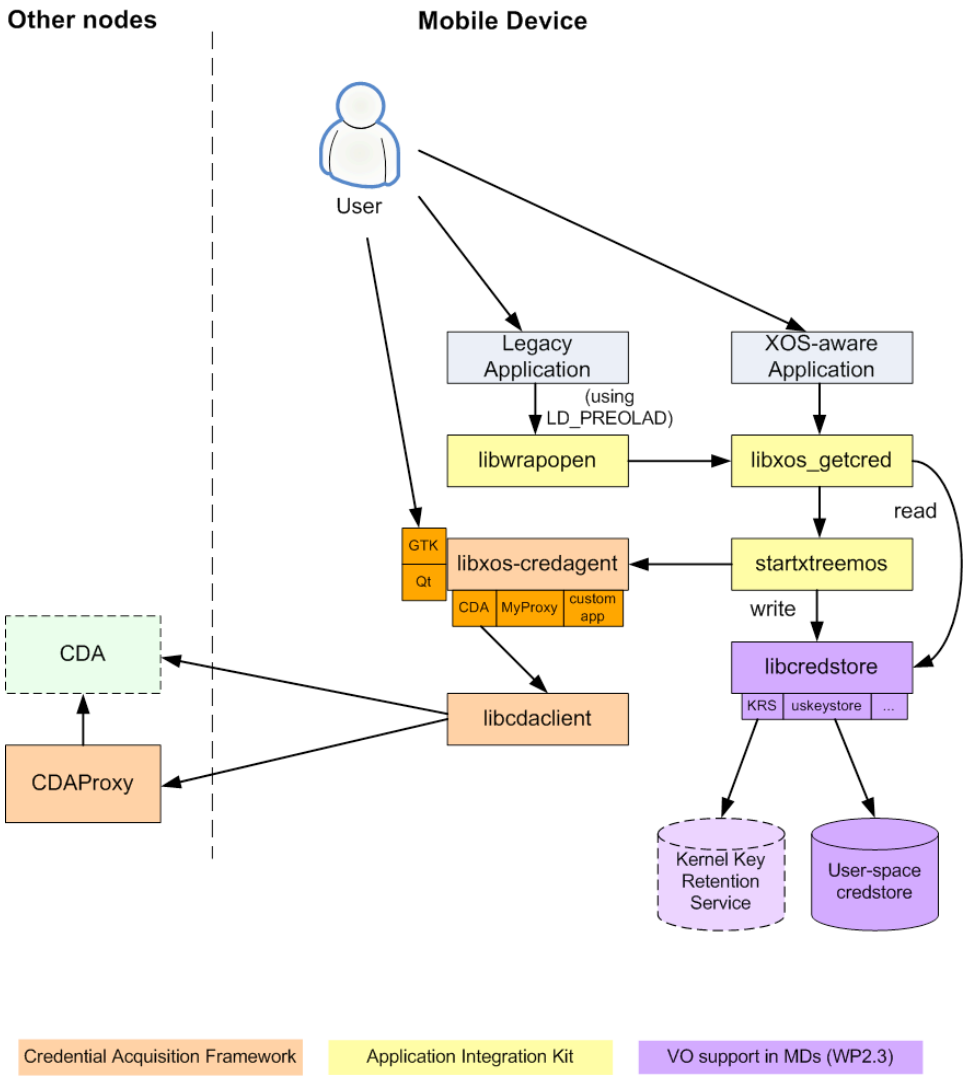

Figure 6.1: XtreemOS credential acquisition + single sign-on framework

  - `startxtreemos` is the simplest implementation, which does not rely on external software and does not isolate grid applications. Ideal for systems that do not support PAM modules (i.e. `libpam`) and do not require special isolation for grid-related applications. `startxtreemos` may be invoked either manually or automatically by `libxos-getcred`.
  - `startxtreemos-ams` provides isolation for grid-related processes by mapping the current user session to a new (temporary) local identity generated on the fly, mapped to the grid identity represented in the XtreemOS credentials. Requires the `xos-nss-pam` and `libpam` packages.

- A way for legacy (XtreemOS-unaware) applications to get access to this credential acquisition and storage system (`libwrapopen`).

The basic workflow of this system is as follows:

1. End user (potentially graphical) applications ask for the credentials of the current user, by using either the high-level API of `libxos_getcred`, or simply by opening a file with the syntax "`::xos:configname`" (which will provoke `libwrapopen` to be executed, which in turn invokes the same `libxos_getcred`).

2. libxos_getcred will invoke libcredstore to obtain the credential from credstore. If credential is found in credstore, it inmmediately returns the credential (whichever the implementation of this credstore is). If the credential was not found in the credstore, libxos_getcred executes the startxtreemos application [2], whose mission is to ensure that credential is obtained and saved in credstore. After running startxtreemos, libxos_getcred invokes again libcredstore to read the credential and return it if available.

3. If startxtreemos is invoked by libxos_getcred, it uses the Credential Acquisition Framework (libxos_credagent) to try to obtain the credentials from whichever means it is configured to use (e.g. from a CDA server, see chapter 2) and returns it. During this process, diverse (potentially graphical) interactions with the user can be needed, for security reasons (e.g. password input or user request authorization).

4. If startxtreemos successfully obtains a credential from libxos_credagent, it stores it in credstore using libcredstore. Then it reads configuration section "xtreemfs" from credagent configuration file and mounts the xtreemFS volume. Finally, it reads configuration section named "xati" and creates configuration files at /.xos directory.

[2] Actually, libxos_getcred searches an executable file named "credagent", but startxtreemos package creates a link to startxtreemos with name credagent. This behavior is because libxos_getcred is independent from XtreemOS: different implementations of "credagent" are possible in other projects.

`Startxtreemos` includes launch-xtfsmount utility. An interesting feature of `startxtreemos` is that it allows a user to mount a specific XtreemFS volume although the user has no privileges to mount a FUSE volume. This is not a security weak, because the volume and mount point are configured by the administrator in the configuration file under `/etc/xos/creds` and the user cannot change it. To achieve this, it is only required that `/dev/fuse` is writable by root and group fuse only.

6.3 System requirements

6.3.1 Hardware requirements

XtreemOS-MD is built to be run in ARM devices (such as PDAs or internet tablets). However, the source code will build and install equally well in standard PC architectures, and in any other architecture that supports the software dependencies below (e.g. MIPS, PowerPC,...).

6.3.2 Software requirements

This Application Integration Kit has been developed with the design goal of making it integrable with almost any mobile Linux distribution, thus eliminating all but the most indispensable software dependencies. The software dependencies of each part of the kit are:

For startxtreemos-ams:

- libpam
- xos-nss-pam and all their dependencies, including OpenSSL 0.9.8

For libwrapopen, libxos-getcred and startxtreemos:

No software dependencies worth noting.

6.4 Installation

The process for building and installing the packages is straightforward:

1. First download the `libcredstore` package from the XtreemOS SVN repository:

```
svn co
  svn+ssh://user@scm.gforge.inria.fr/svn/xtreemos
  /foundation/linux-xos-md/libcredstore
```

2. Execute as root: `make install`
3. After that, download the `aik` packages from the XtreemOS SVN repository:

```
svn co
  svn+ssh://user@scm.gforge.inria.fr/svn/xtreemos
  /xtreemos/grid/mobile/aik
```

4. Execute as root: `install.sh`

6.5 Configuration

`startxtreemos` executable makes use of the same configuration files described in chapter 2 where the `libxos-credagent` was described. The main difference is that `startxtreemos` uses now two more sections: one section with parameters to configure the automount of XtreemFS volume and another section to create the XATI configuration in `/.xos` directory.

6.5.1 `xtreemfs` configuration section

The configuration parameters to mount a XtreemFS volume are included in section [xtreemfs] of credential configuration file:

1. `xtreemfs_server`: XtreemFS server, and optionally the port. E.g. myserver.acme.com:8081.
2. `xtreemfs_dirservice`: XtreemFS directory service server (and optionally, port). If this parameter is omitted, default value is the value of `xtreemfs_server` parameter.
3. `xtreemfs_volume`: XtreemFS volume created with xtfs_mkvol. E.g. xtreemfs_volume=MyVolume.
4. `xtreemfs_mountpoint`: filesystem directory where XtreemFS volume will be mounted. Path my be absolute (starting with /) or relative to home directory user. If a home directory is shared between several users (e.g. /tmp), relative paths may be problematic: to fix this situation, if path starts with `'%'`, this character is replaced by /<user>/.

6.5.2 `xatica` configuration section

The `xatica` configuration section parameters are:

1. `xatica_remoteserver`: The remote XATI server hostname or IP address.

2. `xatica_remoteport`: The TCP port of XATI server.
3. `xatica_localport`: The TCP port where the client listens to notifications from the XATI server.
4. `xatica_localserver`: The IP address of the local server where the client listens to notifications from the XATI server. It is possible to provide `xatica_showmyipserver` and `xatica_showmyipport` instead of this parameter.
5. `xatica_showmipserver, xatica_showmipport`: A remote TCP server that is used to know what is the IP address used by the client to connect to the Internet. This is an alternative to provide `xatica_localserver` parameter.

6.5.3 Configuration file example

```
[general]
credagent=xos_credagent_readfile
creduiagent=xos_creduiagent_gtk

[credagent]
credential_file=/etc/xos/creds/cred.pem
file_is_encrypted=false
ask_user_confirmation=true

[creduiagent]
autokill_after=5
grabserver=true

[xtreemfs]
xtreemfs_server=10.95.38.166
xtreemfs_volume=myVolume
xtreemfs_mountpoint=MyDocs/xtreemfs

[xatica]
xatica_remoteserver=10.95.38.166
xatica_remoteport=55000
#xatica_localserver=192.168.15.82
xatica_localport=10000
xatica_showmyipserver=192.168.15.45
xatica_showmyipport=8080
```

6.6 Interface

6.6.1 Included utilities

Here is a short list of the utilities and commands included in this Application Integration Kit. Please refer to chapter 2 in D2.3.4 [17] for more information.

```
startxtreemos [-c <configuration_name>]
          [-t <seconds_timeout>] [-s]
          [<program_name> [<program_parameters>] | -]
```

This multipurpose program is designed to run one or more of this tasks:

- Loading credentials in the current user credstore or starting a new session credstore and loading credentials in it.
- Running a program or a script read from standart input, which has access to the credstore.
- Setting a credential timeout.

The `-c` option is used to specify the configuration name of the requested credential to use. If no configuration_name is specified, it is assumed that the user would like to use the credential currently present in the credstore; if the credstore is empty, the "default" credential is requested.

The `-t` option is used to set a timeout (expressed in seconds) over the credential. When the timeout expires, the credential is removed. A zero value cancels timeout.

The `-s` option launches a program (or a shell, if no program specified) in a private session, with its own credstore that is destroyed when program/shell ends.

A program to run may be specified, or use "-" parameter to get shell commands from `stdin`. If no program_name or "-" parameter is specified, then a shell is executed (if using "-s") and the `ENV` variable is defined with value `/etc/xos/shrc_xtreemos`. Otherwise, `startxtreemos` only guarantees that a credential is available in the credstore and sets the timeout, if specified.

```
startxtreemos-ams [-c <configuration_name>] [-r]
          [<program_name> [<program_parameters>] | -]
```

This program obtains the credential corresponding to configuration_name (or gets it from the current user credstore if available) and runs the program specified (or a shell if none is specified) with a new UID. This UID is computed using the Account Mapping System rules (see [12]) and the data present in the certificate of the credential.

The `-c` option is used to specify the configuration name of the requested credential to use. If no config_name is specified, it is assumed that the user would like

to use the credential currently present in the credstore; if the credstore is empty, the "default" credential is requested.

The `-r` option replicates the credential in the credstore of the user that invokes `startxtreemos-ams`. The goal of this action is to cache the credential and avoid asking the user again if he prefers running other commands with `startxtreemos-ams` using the same configuration.

A program to run may be specified, or use "-" parameter to get shell commands from `stdin`. If no program_name or "-" parameter is specified, then a shell is executed and `/etc/xos/shrc_xtreemos` is specified in the `ENV` variable. Before the program/shell is executed, current environment is cleared and variables `HOME`, `SHELL`, `PATH`, `USER`, `LOGNAME` and `XOS_ENV` are filled with appropriate values.

```
xos_getdumpcred [<configuration_name>]
```

This program is a usage example of `libxos_getcred`. It dumps the current credential present in the credstore; if the credstore is empty, or a configuration_name is specified and the credential in the credstore is labeled with a different configuration name, `startxtreemos` is invoked to load the new credential in the credstore.

6.6.2 `libxos_getcred` API

This API just includes one function:

```
char * xos_getcred(char *configuration_name);
```

This function returns a credential in PEM format. The API uses `libcredstore` to implement single sign-on: if the credential is stored in the credstore, it is automatically retrieved, but if the credstore is empty or the credential is labelled with other configuration name different from the `configuration_name` parameter, `startxtreemos` (or `startxtreemos-m` depending on the compilation option which was set in the Makefile) is launched (with `configuration_name` as the paremeter, if not NULL) to obtain a credential that is then saved in the credstore.

`xos_getcread` reads the configuration file `/etc/xos/configname_alias` to convert the configuration name from any of its aliases, as defined in that file.

The `configuration_name` parameter can be NULL. In this case, if the credstore is not empty, the credential is accepted without checking the configuration name registered. If the credstore is empty, a new credential is stored with configuration name "default".

In order to compile an application using this library, use the `-lxos_getcred` compilation option. The source code must include `xos_getcred.h`.

6.6.3 `libwrapopen` API

This library does not have a specific API, since transparency is its main objective. In order to use it, an environment variable must be set:

```
export LD_PRELOAD=/usr/lib/libxos_wrapopen.so
```

Once this is done, applications which are not SUID nor SGID will be able to read the credential corresponding to a configuration name whenever they open the file `::xos:configname` or `/::xos:configname`.

Additionally, libwrapopen can be further automatized to load the credential specified in the environment variable `XOS_WRAPOPEN_CONFIGNAME` when a directory listed in environment variable `XOS_WRAPOPEN_DIRS` (a comma separated list) is opened. This feature can be very useful for automounting XtreemFS volumes.

If the application only needs the private key part of the credential then it should open the following file `::xos::key:configname`. However, if the application only needs the certificate part of the credential then it should open the following file `::xos::cert:configname`.

6.7 Usage examples

The `xos_getdumpcred.c` file, used in several examples in chapter 2, is a minimalistic example application which obtains the credential and shows it on the screen. Its source code is shown below:

```
#include <stdio.h>
#include <stdlib.h>
#include "xos_getcred.h"

int main(int argc,char **argv) {
   char* data=xos_getcred(argc==2?argv[1]:NULL);
   if (data) {
     printf("%s\n",data);
     return 0;
   } else exit(-1);
}
```

An example of the use of the libwrapopen library for legacy applications is shown below. Once the LD_PRELOAD environment variable is set, just type the following:

```
cat ::xos:default
```

or

```
mousepad ::xos:simple
```

Chapter 7

Future work

Once the source code of all the elements of the XtreemOS-MD distribution (both those developed in WP2.3 and in WP3.6) is available, the operating system will enter into the final **integration** phase, that will end when the final packaged binaries (both as flashable ROM images and as separate installable packages) are delivered. The public release of this components is expected by the end of February 2009.

From then on, the work of WP3.6 will flow upon two parallel paths:

- A **pure development** path, which will take the basic release as a base, and continue to make bugfixes, modifications and enhancements to it, following the design directives detailed in deliverable D3.6.2 [16], to enhance the software's stability and performance in mobile environments. This development path will also deal with the porting of the software to new platforms such as smarphones.

- A more **research-oriented** path, which will investigate the feasibility of new mobile grid functionalities and advanced services for XtreemOS-MD, such as context awareness or even the ability to run mobile grid services in mobile devices, and using the mobile device as a grid resource, under the constraints imposed by the mobile device hardware. This path will follow a similar approach to the one followed so far, with requirements, design and implementation phases.

The software that was described in this document is just the first of a series of development milestones, which will mark the different **releases** of the XtreemOS-MD software. As mentioned above, the **first** one of these is scheduled for February/March 2009, and a **final** release is expected near the end of the project, in February/March 2010. Intermediate releases (e.g. after 6 months, in September 2009) will be subject to the number and nature of the modifications that are made in the next months.

References

[1] The Ångström Distribution.
http://www.angstrom-distribution.org.

[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 Maemo web site.
http://maemo.org/.

[5] Open Grid Forum (OGF). Saga c++ reference implementation, 2007.
http://saga.cct.lsu.edu.

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

[7] Ole Weidner and Hartmut Kaiser. Saga c++ installation manual, 2008.
https://svn.cct.lsu.edu/repos/saga/trunk/docs/manual/installation_guide/saga_installation_guide.pdf.

[8] XtreemFS official web site.
http://www.xtreemfs.com/.

[9] XtreemOS official web site.
http://www.xtreemos.eu/.

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

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

[12] XtreemOS Consortium. Design and Implementation of Node-level VO Support D2.1.2. Integrated Project, December 2007.

[13] XtreemOS Consortium. Design of the architecture for application execution management in XtreemOS D3.3.2. Integrated Project, June 2007.

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

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

[16] XtreemOS Consortium. Design of basic services for mobile devices D3.6.2. Integrated Project, June 2008.

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