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 MD/MP
D3.6.6

Due date of deliverable: March 31st, 2010
Actual submission date: April 19th, 2010

Start date of project: June 1st 2006

Type: Deliverable
WP number: WP3.6
Task number: T3.6.6

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 Alvaro Martínez / April 15th, 2010

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	12/02/10	Telefónica I+D	Telefónica I+D	Document created
0.2	25/02/10	Telefónica I+D	Telefónica I+D	First contributions
0.3	22/03/10	Telefónica I+D	Telefónica I+D	Draft version for internal review
1.0	15/04/10	Telefónica I+D	Telefónica I+D	Final version

Reviewers:
Massimo Coppola (CNR), Jan Stender (ZIB)

Tasks related to this deliverable:

Task No.	Task description	Partners involved°
T3.6.6	Implementation and optimization of advanced services in mobile devices (XtreemOS-G for MD/MP)	TID*, BSC

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

Executive Summary

The XtreemOS project aims at integrating native Grid functionalities into Linux operating systems and addressing the heterogeneity of current Grid computing technology, from clusters to mobile devices (MDs). The objective of this deliverable is the implementation of the advanced G-layer for the XtreemOS Mobile Device flavour, which includes PDAs like Nokia N8x0 and mobile phones like Nokia N900. For this implementation, and following the specifications given in D3.6.4 [7] and the design explained in D3.6.5 [4], we have worked on different lines, firstly updating XtreemOS-MD G-layer to make it compatible with the last versions of XtreemFS and AEM services.

The inclusion of "resource sharing" capabilities is probably the most important change respect to the G-layer basic version, converting the mobile device in not just a simple Grid client, but also becoming a special node of the Grid, sharing resources like the I/O devices and network access and also sharing data files on demand. For this new data sharing functionality, and in order not to compromise the XtreemFS security model, a special OSD (called *OSDProxy*) has been added to the Grid architecture, acting as a proxy between the non-trusted mobile device and the rest of the trusted infrastructure.

Also the smartphones support offered in this advanced version it's a very interesting feature, opening the access to XtreemOS-MD to mobile phones like the Nokia N900 (based on Maemo 5), and not just limiting it to PDAs based on Maemo 4 as it was the case with the basic version.

Other additional features have been incorporated, like the modification of AEM in the mobile to convert it in a pure-client, or the development of a new security plug-in supporting PAM *(Pluggable Authentication Modules)*. The latter allows a better integration with other SSO systems, and it also offers additional authentication mechanisms, like Bluetooth pairing, etc. whose client part were already implemented by layer F.

Finally, note that even if we have initially worked on the Nokia N900 support, following the priorization criteria identified in deliverable D2.3.6 [5], the XtreemOS Mobile Device flavour software is also portable to other Linux-based platforms like Ubuntu (to be used on netbooks for example) and others.

Contents

List of Figures

List of Tables

Glossary

ARM	Advanced RISC Machine
API	Application Programming Interface
CDA	Credential Distribution Authority
CN	Common Name
DIR	XtreemFS Directory Service
EGID	Effective Group Identifier
EUID	Effective User Identifier
GID	Group Identifier
GPS	Global Positioning System
MD	Mobile Device
MRC	XtreemFS Metadada and Replica Catalog
OSD	XtreemFS Object Storage Device
PAM	Pluggable Authentication Modules
PDA	Personal Digital Assistants
RSD	Resource Sharing Daemon
SRDS	Scalable Resource Discovery Service
SSH	Secure Shell
SSL	Secure Socket Layer
SSO	Single Sign On
TCP	Transport Control Protocol
TLS	Transport Layer Security

UID	User Identifier
UUI	Universally Unique Identifier
VO	Virtual Organization
WP	Work Package
XtreemOS-MD	XtreemOS for Mobile Devices

Chapter 1

Introduction

This document presents the implementation of the advanced version of XtreemOS-MD services layer (G-layer). This advanced version, respect to the first version previously released, includes support for mobile phones like the Nokia N900 and also includes new functionality like the resource sharing (completely new in the G-layer advanced version) and additional features in Security, AEM and XtreemFS services. The software could also be ported by 3rd parties to other Linux-based mobile device platforms (MobLin, LiMo, Android, etc.) as already stated in D2.3.6 [5].

The resource sharing functionality, which was started in the F-layer with the implementation of the resource sharing daemon, has been now completed in the G-layer, offering a set of capabilities including data sharing and I/O and network device sharing.

Apart from the smartphone support and the resource sharing new feature, some improvements related to Security, AEM and XtreemFS has been implemented for this G-layer advanced version, and will be documented in following chapters. As a quick summary:

- AEM server part has been removed (now it's a pure client) but an additional monitoring callback has been included to keep the monitoring feature present in previous versions without needing asynchronous notifications. This way, the previously port opened on the MDs is no more needed. Also, the communication with SRDS in order to publish the resources shared by the mobile devices has been implemented.

- An special OSD (OSDProxy) has been implemented to act as a proxy between the non-trusted mobile device and the rest of the trusted infrastructure, so that the XtreemFS security model is not compromised when the mobile device is sharing files thanks to the new on-demand file sharing feature offered.

- A new plug-in architecture to support PAM has been also included and the CDAProxy has been consequently modified. This way we solve the server

part for the new authentication mechanisms and SSO integration client part which was already implemented in layer F.

1.1 Document structure

The document is structured as follows:

- Chapter 2 provides an overview of the architecture and the advanced services implemented for mobile devices that will be detailed in the following chapters.
- Then, Chapter 3 deals with the implementation of the modifications related to AEM, especially focusing on the new monitoring callback and API to publish the shared resources on the SRDS
- Next, Chapter 4 focus on Data Management and the OSDProxy provided to permit the data sharing functionality. A special section for the Vivaldi algorithm is also included here.
- Chapter 5 deals with VO management and security, emphasizing the PAM support and the new plug-in architecture to support it
- Chapter 6 analyzes in detail the Resource Sharing feature, describing separately the different kinds of shared resources offered: on-demand file sharing, GPS sharing (as a particular case of I/O devices sharing) and network sharing.
- We finalize with Chapter 7 by giving the final conclusions to WP3.6 and more concretely to this deliverable.

Chapter 2

Overview of the advanced services implementation for MDs

This chapter introduces a high level vision of the advanced services implementation for mobile devices. We first present a comprehensive overview of the requirements and specifications already identified in previous deliverables and then we will show the high-level architecture of the final XtreemOS-MD advanced version. Later chapters will address the specific implementation issues, service by service.

2.1 Design review

2.1.1 AEM service

Monitoring with local buffers and callbacks. The monitoring with buffering with the metrics is an important component in AEM. It provides an easy way to store and check metrics values reducing the overhead and overload of checking it every x seconds. Callbacks provides a way to receive notifications of events (like job finished). In the callbacks case the C-XATI implementation introduced changes over the Java version.

Resource reservations. Resource reservations can be automatic or manual, in both cases no modification is needed

Dependency trees. Dependency trees provide relations between jobs. User can check all dependency tree in xps or other commands.

2.1.2 Data management service

Vivaldi. The Vivaldi algorithm is now supported in the mobile devices, being possible to locate the MDs in the Vivaldi space, and also using the Vivaldi coordinates to obtain the XtreemFS closest replica to the MD.

On demand file uploading and transparent file sharing. Those new features are possible thanks to the possibility of off-line mode operation and the data sharing capability offered by the new Resource Sharing module

2.1.3 VO management and security

Integration of legacy SSO with CDAProxy. The CDAProxy has been modified to support the new plug-in architecture provided to support PAM. The client part os this feature was already covered by the F-layer advanced version.

2.1.4 Resource Sharing

The advanced version of F-layer developed the bases for this resource sharing capability. G-layer focuses on the proxy clients developed for the different kinds of resource sharing (and running on trusted nodes), using the APIs provided by the resource sharing daemon.

Data sharing. Offering the local files stored in the mobile device as resources accessible in the Grid.

I/O Device sharing. And concretely sharing the GPS (if available in the MD) to other Grid users.

Network sharing. To allow sharing the MD's 3G connection to other XtreemOS users.

2.2 Final architecture

Figure 2.1 shows the architecture of the advanced version of Xtreemos-MD. In comparison with the basic version, already presented in deliverable D3.6.3 [3], the main architecture changes are the addition of the context-awareness and resource sharing modules. Context-awareness was fully developed in the F-layer, and explained in D2.3.7 [6], whereas resource sharing was partially developed in F-layer (and also explained in D2.3.7) and completed in this G-layer.

The following chapters provide the specific implementation details grouped by the main services: AEM, XtreemFS, Security and Resource Sharing.

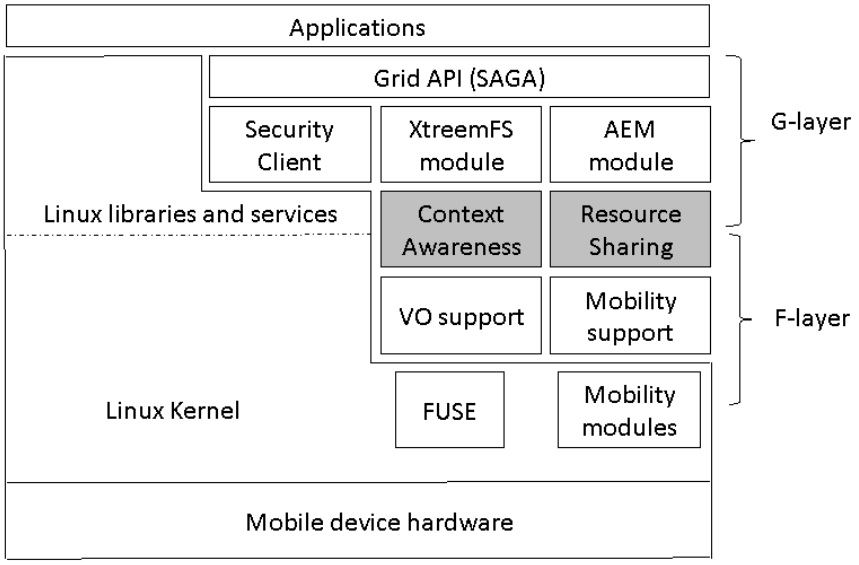

Figure 2.1: XtreemOS-MD advanced version architecture

Chapter 3

AEM service

3.1 Introduction

AEM is the component that provides the execution and management of jobs in XtreemOS. More details of the AEM architecture can be found on deliverable D3.3.7 [8].

3.2 General architecture

The general architecture of AEM includes an XOSD daemon hosting a set of distributed services. For example, in a core node we can find JobManager, and ReservationManager. In a resource node, we can find ExecMng and the AllocationManager. Clients and interfaces to access the nodes are automatically generated from the code. XATI and C-XATI are the generated client interfaces, for Java and C clients.

Some of the modifications included for mobile devices are the elimination of the server side (open an extra port) becoming a pure client. Mobile devices greatly benefit of this change. Regarding callbacks, this pure client change produced that the users can't get notifications if they are not connected. To solve this, the client can fork and do the addMonitoringCallbackMD call and wait until the event triggers.

On the other hand, the AEM offers a new functionality to support the mobile resource sharing (Connectivity and GPS information):

- Publishing the mobile resource information on SRDS system through XATICA. The mobile device sends its resource information (status, GPS position, availability...) in XML to AEM that process the information and sends it to the SRDS system in a transparent way for the mobile device.

Other features are ported directly to mobile devices without changes.

3.3 API definition

The specific API definitions for mobile devices are the next ones:

```
int addMonitoringCallbackMD(const char* __jobId,
    const MetricEvent __metricEvent,const char* __userCtx,
    int* returnValue);

public Integer pushMDinfo(String Info)
public String retrieveMDinfo(String Query)
```

3.4 Implementation

Client implementation of C-XATI for mobile devices does not allow the server behavior it had in previous releases and thus the Grid cannot send asynchronous events to it. To overcome this effect while keeping all the specified functionalities, a new function has been added to the API. This is the aforementioned `addMonitoringCallbackMD`.

The functionality of this service invocation is the same as the previously available `addMonitoringCallback`. The only difference is that this new one blocks the client until the callback is triggered, while the previous one returned immediately allowing the definition of a callback function.

The proposed way to use this mechanism is through a new thread at the client side per each desired callback. This new thread would just set the callback, which blocks it until triggered, and invoke the actual callback function immediately afterwards.

At server side there is no blocking: DIXI decouples requests from replies in a manner that the implementation can delay the reply of a service method invocation just by storing the context of that request. This has been already used in other method implementations such as the `jobWait` call, which blocks the client until job finalization.

3.5 Installation and usage

Monitoring is used through XCJobMng.h C-XATI lib and MD publishing info is used through XCResMng.h. We use ResMng as it is easier for the mobile devices to use ResMng that an extra header (for SRDS). Also development is easier this way as all C-XATI services are created in the WP3.3 part of the code.

Chapter 4

Data management service

4.1 Introduction

XtreemOS-MD on-demand data sharing allows mobile devices to share files with other users, taking advantage of XtreemFS features, like replication for instance, but not compromising the XtreemFS security model, which specifies that *OSD* servers should run only on trusted nodes. To this purpose, a special software running in the mobile device notifies a proxy each time a new file is available to share; this file is then uploaded to XtreemFS, being *OSD Proxy* responsible for it. The first time a fragment of the file is required, the proxy contacts the mobile device to retrieve that fragment and then replicate it (avoiding the need to repeat the operation if the fragment is requested in the future). This chapter is dedicated to the client part of the on-demand data sharing feature, while the server side will be explained in chapter 6.

4.2 General architecture

The architecture of client-side data sharing is already described in deliverable [6]. Both `Resource Sharing Daemon` and `libxos_notifydirchange.so` library, detailed in Data Sharing section of that deliverable, cover an important part of the implementation.

Client side implementation of data-sharing involves the creation of two new modules in G-Layer:

- A inetd-like service, that receives a request for a file fragment and sends the data to *OSD Proxy*.
- A sharing module that is responsible of detecting changes in a user configurable shared folder and contact with *OSD Proxy*.

4.3 API definition

There are not public API with this software.

4.4 Implementation

Inetd-like service implementation is simple because most details are implemented by Resource Sharing Daemon: it only parses a request for a fragment specified by file name, offset and length and writes to standard output the data preceded by the length. On the other hand, data sharing module implementation is more complex and needs a more detailed explanation.

When data sharing module starts, it first must compare the sharing folder with a hidden folder where the XtreemFS shared volume is mounted. This step is required, as users may share a folder that was not initially empty. It is required also because files and directories may be created, deleted or moved by the user while data-sharing is not enabled. This initial synchronization compares recursively both folders; if a set of files and/or folders are in sharing folder but not in XtreemFS shared volume, module contacts with *OSD Proxy* to replicate the filesystem structure and share the new files. If files and/or folders are available in XtreemFS shared volume but not in sharing folder, module contacts with *OSD Proxy* to remove the missing contents. Finally, it some file is different between both folders, file is first removed and then added again: file changes are checked comparing the last modification date.

Once the initial synchronization ends, the data-sharing module detects the possible modifications in the filesystem using libxos_notifydirchange.so library. The following situations are monitored:

- A file is closed after a write operation: this event occurs when copying a file to the shared folder and when creating a new file directly in the shared folder (after closing the file and setting the privileges, as XtreemFS only supports replication of read-only files). In this case, a notification is sent to the *OSD Proxy* indicating that a new file is shared. If the file was already uploaded to the OSDProxy, and was modified locally by the file owner, the *OSD Proxy* is responsible for firstly deleting the file and then adding it again.

- A file or folder is moved into the sharing folder. This event implies a similar interaction with the *OSD Proxy* than the previous case.

- A file or folder is deleted locally: a notification is sent to the *OSD Proxy*, to remove the file stored in the XtreemFS. If a folder is removed, there will one notification for each file included in it.

- A file or folder is moved out of the sharing folder. A notification is sent to the *OSD Proxy* to remove the concrete file from the distributed file system. The

process will be recursively done for folders containing subfolders or more than one file.

- A file or folder is renamed or moved inside the sharing folder. A notification is sent to the *OSD Proxy*, which should then request the real path file to the mobile device (as the *OSD Proxy*, differently from ordinary *OSD*s, stores the filenames and not just the file IDs.

4.5 Installation and usage

Configuration is saved in file `/etc/xos/sharingservices/filesharing.conf`. The following is a example of configuration file:

```
[proxy]

host=10.95.42.89
port=42643
#uuid=7ebea9ea-14e5-45c4-b3fc-5f9b02810319

[persona]
uid\_service=-1
gid\_service=-1
uid\_module=-1
gid\_module=-1

[availability]
enable=true
disable\_if\_not\_wifi=false
disable\_if\_battery\_less\_than=0
#disable\_if\_status\_is

[approval]

ask\_request\_approval=false
notify\_new\_request=false

[config]

sharingdir=xossharing
sharingdir\_mount=.xtreemfs\_shared\_sync
```

All sections except *config* are common between all resource sharing modules and are already documented in *Data Sharing* section of deliverable [6].

All module specific parameters in section *config* are intended only for integrators and deployers, as alternative to modify the source code. It is not expected that user modifies these values and therefore configuration tool hides them. Parameter *sharingdir* is the sharing folder and *sharingdir_mount* is the folder where the XtreemFS sharing volume is mounted. This volume is mounting automatically by *startxtreemos* tool.

To install the service manually, only is needed to run `make install`. However, XtreemOS-MD offers this software packaged and user only need to install it as any other native application. For example, in Nokia N800 or Ubuntu is sufficient to run `apt-get install xosrsd-filesharing` to install this module with all its dependencies, including a XtreemOS-MD minimall instalation. This package is also part of the metapackage *xosmd-full*.

Data sharing service is started with Resource Sharing Daemon. A configuration tool allow users to set what resource sharing modules are enabled.

4.6 Vivaldi feature

Vivaldi is a light-weight algorithm, developed by MIT, that assigns a position in a coordinate space to every node of a network, so the distance between the coordinates of two nodes predicts the real communication latency between them. This mechanism eventually enables predicting the latency between two given nodes with a simply mathematical calculation, with no network overhead associated.

According to the algorithm requirements, the nodes keep contacting themselves periodically in order to re-adjust their positions, so any possible change in the network may be reflected. For each re-adjustment, a node contacts one of its peers, gets its coordinates and accordingly modifies its own position, so eventually the euclidean distance between both points is equal to the measurable round trip time delay.

4.6.1 Vivaldi in the OSDs

Every OSD runs a dedicated stage that keeps iterating indefinitely in order to periodically recalculate its position. Thereafter, it is granted that the provided coordinates keep "consistent" according to the generated space. Moreover, they all keep a cached list of OSDs from which they may choose a peer to recalculate against. To fill that list and to keep it up to date, an OSD contacts periodically the DS (the Directory Service is a XtreemFS service that, among other data, keeps track of every existent OSD) and asks it to select randomly N OSDs. This way they are able to get a set of reference nodes, while the system remains perfectly scalable. Furthermore, this allows the nodes to avoid having to detect when an OSD goes offline, since it is the own DS which notices that kind of event and consequently reflects it in its list of available OSDs.

4.6.2 Gathering the coordinates

According to the XtreemFS architecture, in the OSD, the Heart-Beat-Thread stage uploads to the DS some information of its corresponding device every certain period of time. Among other data, the node includes its remaining free space, the load of its processors or its coordinates and its local error. Consequently, this is the first time the coordinates of every OSD in the system are gathered at the same distributed service all together.

Next, the MRC obtains, through its own service OSD-Status-Manager, all the information contained in the DS about every registered OSD. Thus, since the first transmission is done, the MRC has all the information known about the currently available OSDs. It then can use that data to process the requests received from the clients.

4.6.3 Vivaldi in the clients

In general terms, a client works in a very similar way to an OSD. It runs a dedicated service that recalculates periodically its coordinates and tries to minimize the algorithm prediction error. But there is one main difference between clients and OSDs: clients have no influence over the position of any other node, as they are not tracked by any system service. Every OSD is registered in the DS and hence the rest of nodes can contact it when recalculating their position. It currently influences the rest of nodes. However, the DS keeps no track of any client, so they cannot be contacted by other nodes, since they remain invisible to them. Therefore, the clients can be considered as pure observers, that seek their optimal position in the coordinate space, but whose presence does not affect the rest of nodes at any point.

XtreemFS is supposed to be executed on an environment where OSDs keep running during long periods of time. In this situation and depending on the characteristics of the network, they may need more or less time but, eventually, they converge to a stabilized state where the global prediction error is reasonably low. Thus, when the clients join, they get into a consistent coordinate space. Such a space has a set of firmly fixed nodes that may serve as landmarks, so clients are able to find an appropriate position with only a few iterations.

In addition, for situations where the clients must be well placed since the very first iterations, a mechanism to initialize the position of this kind of nodes has been provided.

```
xtfs\_vivaldi --fast-initialization <dir host>[:port] \
 <path to Vivaldi coordinates output file>
```

Using the option *–fast-initialization* when executing *xtfs_vivaldi*, the service ignores the coordinates contained in the coordinates file and tries to determine a new set. In order to get an approximate location, the client first measures the RTT against 5 OSDs randomly chosen, and takes the coordinates of the closest one,

according to the real latencies. Next, it makes the most of the coordinates already obtained and adjusts its position 5 times in a row, one for each pair of coordinates. It is important to remark that the coordinates of the closest node (those used to initialize the client's position) are always left for the end of this process, so they really contribute with significant information. Otherwise, the client could adjust its position for the first time against the closest node, so the coordinates would be the same, and eventually Vivaldi would move them away in a random direction.

According to the performed evaluations, when using "fast initialization", the client is able to find its right position in less than 2 hours, which is less than the half of the time required on average, when using regular initialization.

This mechanism is specially useful when dealing with mobile devices, which are expected to drastically modify its geographical position with a higher frequency than other types of device. In these cases, the client's coordinates can be manually reinitialized and thus the global performance does not get so deteriorated.

Chapter 5

VO management and security

5.1 Introduction

In [4], the main design decision about security in G-Layer was PAM (Plugable Authentication Modules) support in CDAProxy. This support is important to provide a better integration with the SSO implementation of the user's enterprise system (or with the SSO of the user's ISP). Thanks to PAM support in the CDAProxy, mobile device users can use their enterprise username and password instead of the CDA ones.

PAM support in CDAProxy is also required for some security enhancements obtained from [6], as Bluetooth pairing support and PIN based authentication. Client-side of both functionalities (basically involving *credagent* and *creduiagent* modules) were previously implemented in the Foundation Layer, but they also require special support in server side beyond PAM integration in CDAProxy.

PAM support is a very powerful feature, but it is also complex to configure and requires a software infrastructure that is not available in all platforms, specially in mobile devices. Usually the CDAProxy runs in a PC and not in mobile devices, but some scenarios are possible where the CDAProxy could run in a mobile device; for example as a personal, mobile and secure store of credentials or as a method to implement parental control: a child mobile phone needs to obtain a credential from the CDAProxy, that is running in the mobile device of his parent, and the CDAProxy request confirmation.

To make optional the PAM support in the CDAProxy, and also to achieve a higher portability, a new plug-in architecture has been implemented. PAM support is provided by a plug-in with a very simplified API hiding the PAM complexity. The PAM support may be enabled or disabled in a per account basis.

5.2 General architecture

The logic of the new plug-in starts just after reading the user and password from the corresponding request. The plug-in authenticates the user and three scenarios

are possible in case of success:

1. The plug-in just authenticates the user: in that case, the CDAProxy logic continues, evaluating the configuration (e.g. checking a local file credential or connecting to CDA server).

2. The plug-in maps to a new password: this is the case when the CDAProxy uses PAM to authenticate users against their enterprise's SSO (e.g. a LDAP server) and then uses a secret password to authenticate against the CDA server. In this situation, the CDAProxy logic continues evaluating the configuration, but using the password returned by the plug-in instead of the password provided by the user.

3. The plug-in gets the credential by itself, doing all the rest of the work

5.2.1 Architecture of PAM plug-in

The plug-in uses internally a PAM implementation, but hiding its internal details and just exposing a public API that will be later detailed in section 5.3.It's interesting anyway to quickly review the PAM principles.

PAM architecture provides an API for applications, a configuration file for administrators and an API for module programmers. When an application invokes the PAM API it passes an application name that is used to select the modules to run according a configuration file with pattern
`/etc/pam.d/<application_name>`.

There are three types of PAM modules:

1. Authentication modules: modules required to verify the identity of the user (for example, by asking username and password)

2. Authorization modules (account modules in PAM terminology): modules that, once verified the users' identity, determine if user access should be granted.

3. Session modules: modules which can execute arbitrary code when logging in/out. In the CDAProxy plug-in architecture, these modules could retrieve the final credential from the CDAServer or return a password to be used to connect to the CDAServer (nothing would be done if PAM configuration is only used for authentication and authorization purposes).

Actually, there is also a fourth module type: password modules, which are used to modify the user's security token. This type is not considered as it's used only by special tools to modify the password (like the *passwd* command) and is not intended to usual applications.

The CDAProxy does not need to know anything about modules and PAM complexity, and therefore the plug-in exports a simplified API with only one function.

This function invokes the PAM API to authenticate and authorize, returning, depending on the type of session module, the CDA credential, the CDA password or an authorization to use a local credential.

Note that PAM authentication modules may support authentication methods involving an interaction with the user different than the usual request for username and password. Interaction with users occurs really in client-side: this implies that, in order to support new methods of interaction, some modifications are required in *credagent* and *creduiagen*t modules. This is for example the case with Bluetooth pairing and PIN based authentication.

5.2.2 Architecture of PIN based authentication

A possible security flaw of PIN based authentication, when using an enterprise proxy instead of a personal proxy, is that storing passwords is necessary, to send them to the authentication server once PIN-authenticated against the proxy. To minimize this security risks, maximum isolation between the proxy and the software that checks the PIN and returns the password or locks the password after three faults, is required. PAM modules runs in the same memory address space than the invoking application. Therefore, as a security enhancement instead of implementing PIN authentication in the PAM module, the module only does a fork & exec to run the real code, a utility named *pin_authentication*.

5.3 API definition

The plug-in API is very simple:

```
int cdaproxy_plugin(char *appname,char *user,
                    char *security_token,
                    char **cda_user,
                    char **cda_password,
                    char **credential);
```

The function returns 0 on success, otherwise a negative error code:

- -1: Authentication error
- -2: Authorization error
- -3: Internal/misconfiguration error
- Appname parameter is used by libpam to choose the configuration file (and therefore the modules to apply) as was explained before.
- User parameter must contain the username to authenticate and authorize.

- Security_token parameter must be filled with the secret used to authenticate the user. It may be for example a password or a PIN. Parameter may be NULL if authentication is external (e.g. Bluetooth pairing authentication).
- Cda_user parameter is a pointer to a string that may be filled with the username to use for CDA authentication. If it is not filled with NULL, is responsibility of the caller to free the pointer.
- Cda_password parameter is a pointer to a string that may be filled with the password to use for CDA authentication. If it is not filled with NULL, is responsability of the caller to free the pointer.
- Credential parameter is a pointer to a string that may be filled with the credential to return to the remote client. If it is not filled with NULL, is responsability of the caller to free the pointer.

If there is not a special session module, parameters *cda_user*, *cda_password* and *credential* are filled with *NULL*: in this case PAM modules only authenticate and authorize users and the CDAProxy must know how to obtain the credential. A second possibility is that the session module fills the *cda_user* and*cda_password* parameters with values and the *credential* with *NULL*: in this case the CDAProxy must contact the CDA to obtain the credential. Finally, a session module may fill the *credential* with the XOSCert while the *cda_user* and *cda_password* will be NULL.

5.4 Implementation

The CDAProxy was updated to support plug-ins. The CDAProxy uses *dlopen* and loads modules from `/usr/lib/xos/`. The CDAProxy uses a hash table to avoid *dlopen*'ing multiple times.

PAM Plug-in is implemented in file `cdaproxy_plugin_pam.c`. The code invokes the different PAM API functions and is aware of username changes.

A set of PAM modules and a utility named *pin_authentication* have been written to implement the server-side of PIN authentication.

5.5 Installation and usage

A new project cdaproxy_pam with the source code of the plug-in was created. This project only requires libpam to build. The software is packaged for the supported platforms, and installing *cdaproxy_pam* package is sufficient to install also *cdaproxy*.

To enable PAM support to authenticate all users, edit `/etc/xos/cdaproxy.conf` and set `use_plugin=true` and `plugin_name=PAM` in *all_users_default* section. To enable/disable PAM support in a user-by-user basis, insert *use_plugin* pa-

rameter inside the specific user section. If *plugin_config* key is present, it is used as *appname* paremeter of cdaproxy_plugin method, otherwise *cdaproxy* is used.

Chapter 6

Resource sharing

6.1 File sharing

6.1.1 Introduction

File sharing between mobile devices and the Grid demands new capabilities to the XtreemFS. The OSDProxy is a new element integrated within the XtreemFS that provides those new capabilities. In this section, we cover the general architecture and implementation of the different elements that shape the file sharing system, as well as its installation and configuration.

6.1.2 General architecture

The OSDProxy is another OSD in the XtreemFS distributed file system, but with the particularity of making the mobile device data accessible for the Grid, thanks to the integration of three new software modules.

- Mobile communications
- Shared files database
- Shared volumes manager

6.1.2.1 Mobile communications

The *mobile communications* software module offer an API to exchange sharing requests between mobile devices and the OSDProxy. The mobile device can execute commands for sharing files and manage them, and the OSDProxy requests the shared files data to mobile devices.

6.1.2.2 Shared files database

The *shared files database* software module is a local database using BabuDB that contains all the references about the files shared by any mobile device.

6.1.2.3 Shared volumes manager

The *shared volumes manager* software module is in charge of discovering the user's volume in the Grid and also managing the shared files as files stored in the XtreemFS.

6.1.3 API definition

The API offered by the mobile communications module is bidirectional, offering to mobiles devices a series of "sharing requests" and offering to the OSDProxy itself the "data requests" that are sent to the mobile device.

6.1.3.1 Sharing requests

The OSDProxy is always ready to accept connections in the configured port for sharing requests. The mobile device can ask for a new connection sending a cookie with its UUID. When the cookie is correct, the OSDProxy establish a new socket with the mobile device, being ready for processing the possible sharing requests (commands) from mobile device.

Those requests are Big-Endian binary streams starting with a command code followed by the parameters required in each case. The command code is always the first byte of the request and it can contains five different values identifying the different commands supported: share file, remove shared file, make directory, remove directory and move.

1. **Share file command** This is the main command of this API, allowing mobile devices to share files in the Grid. These files cannot be modified while they remain as shared files. The code for the share command is "0" and the structure of the stream includes three parameters, the shared file reference at first time, followed by the shared file size in bytes, and a timestamp. The shared file reference is composed by two fields. Two bytes to code the shared file name length in bytes and a byte array with the file path. The file path is always referred to the local shared folder in the mobile device what represents the user volume in the grid. The shared file size (in bytes) field is coded with four bytes. The timestamp is referred to the mobile device local date and is coded with eight bytes.

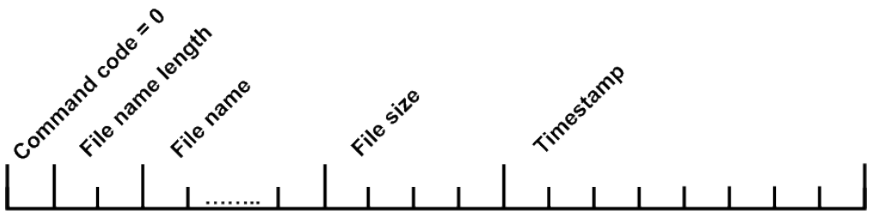

Figure 6.1: Share file command

2. **Remove file command** The remove command is only allowed for the mobile devices that have ever shared files in the grid, and it makes possible the removal of those files from the grid. The code for the remove command is "1" and it's followed by just one parameter, which is the shared file reference coded as the previous command.

Figure 6.2: Remove file command

3. **Make directory command** This command makes possible to organize the shared files in different folders within the mobile device shared folder and, at the same time, in the Grid user's volume). The code for this command is "2" and it's followed by two parameters. The first one is the folder reference that is composed by two fields, representing the file share reference: two bytes encoding the folder name length followed by a byte array with the folder path referred to the local shared folder. The last parameter is a timestamp that is referred to the mobile device local date and that is coded in eight bytes.

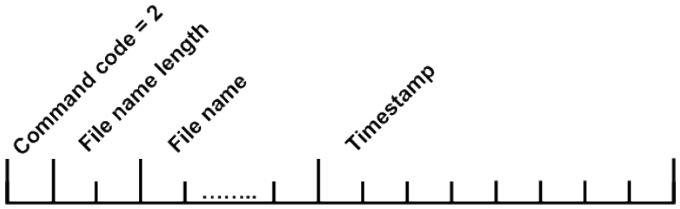

Figure 6.3: Make directory command

4. **Remove directory command** This command is used to remove folders and their content. The code for this command is "3" and it's followed by the same parameters than the "Make directory" command.

5. **Move command** This command allows users to rearrange their shared files in the different folders, and also their content. The code for this command is "4" and it's followed by three parameters: the source, the target and a timestamp. The source and the target may be a reference to a shared file reference or to any folder. The timestamp is referred to the mobile device local date and is coded in eight bytes.

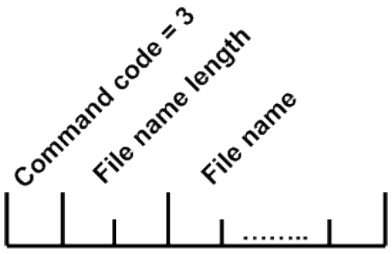

Figure 6.4: Remove directory command

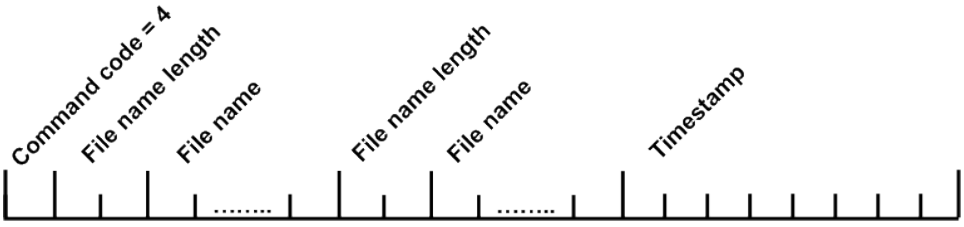

Figure 6.5: Move command

6.1.3.2 Data requests

When the OSDProxy needs the shared files data, it asks for a new connection to the file owner (the mobile device) sending a cookie with the mobile device UUID for authentication. If the cookie is correct, a new socket is established and then the OSDProxy sends a Big-Endian binary stream to the mobile device, containing the following fields:

1. Offset. Four bytes coding the position within the file where the required data block starts
2. Length. Four bytes coding the size of the required data block in bytes
3. Shared file reference. Two bytes to code the shared file name length in bytes and a byte array with the file path. The file path is referred to the local shared folder in the mobile device

When the mobile device receives a data request, it looks for the file and reads the requested data. The mobile device replies sending the data in a binary stream, starting with a "0" code or a value different than zero in case of a read error. Only when the code is zero the OSDProxy reads the file data from the stream.

6.1.4 Implementation

The OSDProxy is a Java6 program, like the standard OSDs, which provides its own library, `OSDProxy.jar`. This new program keeps the same dependencies than the original OSD and includes two new dependencies:

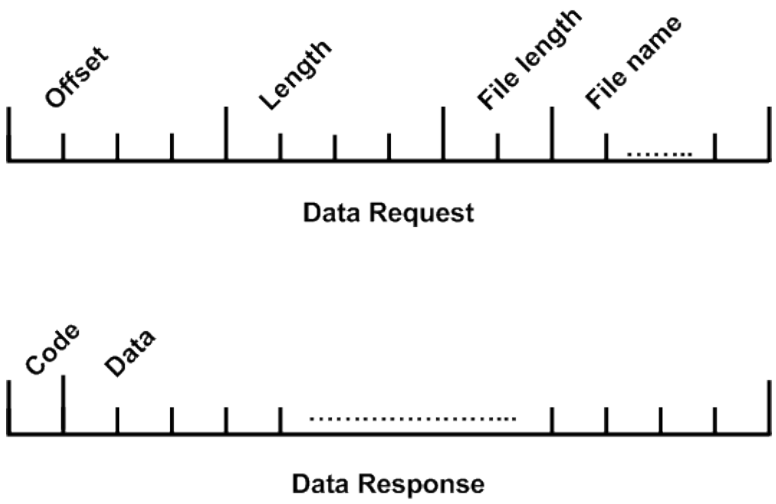

Figure 6.6: Data request

- *BabuDB*. As the MRC and DIR, the OSDProxy needs a local database and uses the same release than them.
- *BerkeleyDB*. The OSDProxy uses the java interface of BerkeleyDB release 4.8.24 to query about mobile devices cookies and ports before requesting the shared files data.

6.1.4.1 Mobile communications

A server socket provided by Java6 JDK, opened in the configured port of *localhost*, is the resource used to accept the sharing requests. It is always open since the process start, and it accepts new connections from mobile devices, starting a new thread for each one, which will be in charge of processing every request received over it. When the process receives data requests for shared files, and if the shared file has not been uploaded yet, it requests a new connection using a standard socket opened in the port registered by the mobile device cookie. If the mobile device accepts the connection and sends data, the process writes the data in the configured disk space. Then, it passes the control to the OSD standard procedure. If the mobile device does not accept the connection or a transfer error occurs, an IOExcepcion is thrown to the OSD standard procedure.

6.1.4.2 Shared files database

The OSDProxy implements a local database, using BabuDB, to store the shared files references and the folders used to organize them. This information is indexed using a record where the key is the fileID provided by XtreemFS, containing the following fields:

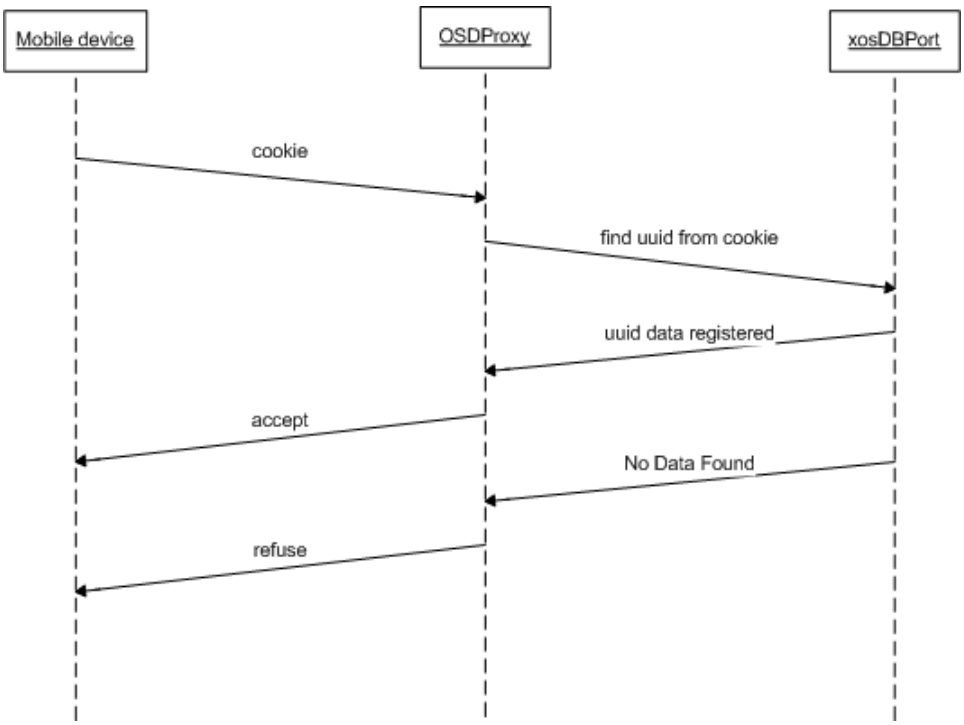

Figure 6.7: New connection for sharing requests

- Type Three different items can be indexed. A shared file, a folder or a *home*. The *home* represents one mobile device and in this case the key is provided by the cookie and contains the mobile device UUID.
- Name Contains a folder name or shared file name. It's empty when the type is *home*.
- ParentKey Contains the key of the *home* or folder which group it.

6.1.4.3 Shared volumes manager

The manager is an object that implements the sharing commands by using the MRC API and a user's volume discovering procedure. Using the DIR API xtreemfs_service_get_by_name method, provided by the *OSDRequestDispacher*, the user's volume discovering procedure looks for the MRC which manage those volumes.

Striping Policy The process uses the same striping policy for every volume

- Type. STRIPING_POLICY_RAID0
- Width. 1, because only the OSDProxy can store shared files
- Size. The same size configured in the volume if it is less or equal to the striping policy size configured in OSDProxy properties

The reason why the OSDProxy fixes its own striping policy is to take control over the OSD which stores the shared file objects, and to restrict the size of the objects in order to speed up the data requests between the mobile device and the process.

Sharing commands The OSDProxy uses the MRC API methods in order to execute the different commands over users' volumes in the Grid. The *share file* command shown in figure 6.8 is the main one (and the more complex).

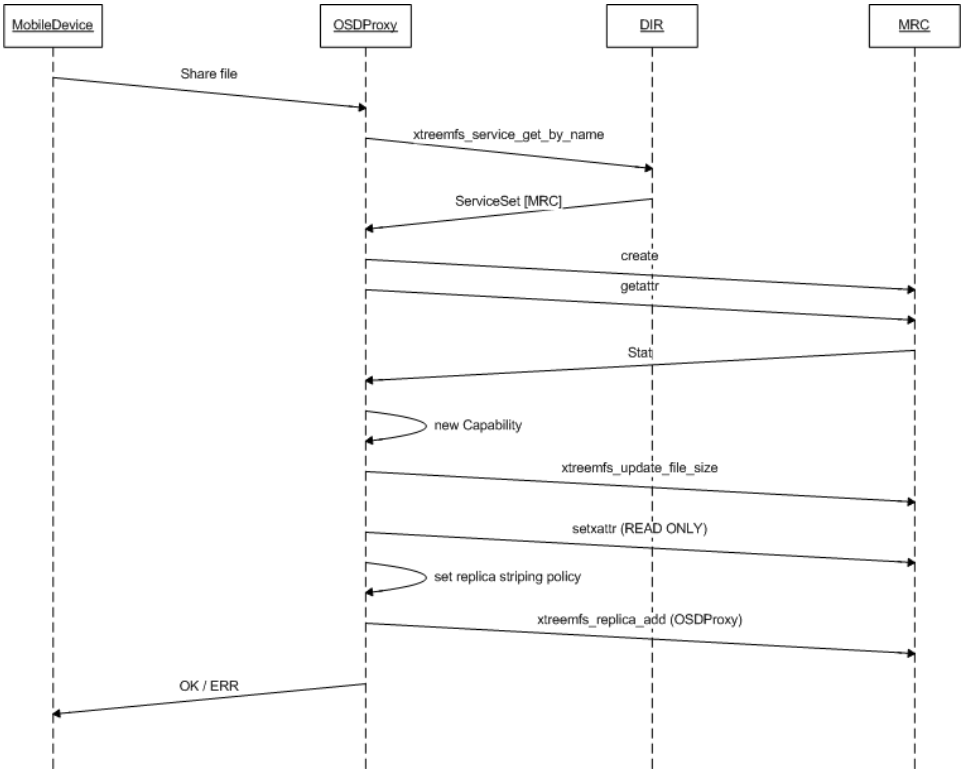

Figure 6.8: Share file command

6.1.5 Installation and configuration

The installation of OSDProxy is resolved by a rpm package which contains the software, configuration file and start/stop script. The administrator needs to install the package to deploy the OSDProxy in the same locations than the other XtreemFS components.

6.1.5.1 Configuration

The data sharing feature can be easily configured through the specific graphical configuration tool provided, explained in section 6.3. The OSDProxy properties

file is *osdProxyconfig.properties*, including some modifications to the original OSD properties file:

Sharing requests properties

- ds.request_port Defines the local port where the OSDProxy listen for new connections from mobile devices
- ds.volume_prefix Defines the prefix used to compound the name of the users volumes in the grid
- ds.stripe_size Defines the maximum striping size applied by the OSDPoxy in shared files
- ds.max_timeout Defines the maximum waiting time for data requests in miliseconds
- ds.max_attempts Defines the maximum number of attempts for data requests when waiting time expires

BerkeleyDB configuration

- xosPort.db_home Defines the absolute path where is placed the *xos_register_port* database
- xosPort.db_file Defines the file name of the *xos_register_port* database
- xosPort.db_name Defines the name of the *xos_register_port* database

BabuDB configuration It uses the same properties than the MRC and DIR.

6.2 I/O device sharing

6.2.1 Introduction

XtreemOS-MD offers the possibility to share I/O devices and network connectivity with other XtreemOS users [4]. In this section we cover the general architecture and the implementation of the different elements that shape the resource sharing system for connectivity and I/O devices, including as well an example of use and a complete explanation about its installation and configuration.

6.2.2 General architecture

Given the different particularities of the input/output devices in mobile devices, the architecture of the system has been divided in three different subsystems:

- Publishing Subsystem
- 3G Connectivity Sharing Subsystem
- GPS Sharing Subsystem

6.2.2.1 Publishing Subsystem

This subsystem is in charge of collecting information from the mobile devices, sending it to the SRDS system through AEM. This subsystem is composed of a client part and a server part.

The client part is based on the `I/O sharing module`, which invokes the XOS-MD contextAwareness API [6] to get information about the GPS position, accurate date, connectivity situation and sharing resources availability (resources information). The data is formatted as an XML document following the schema detailed in section 6.2.3, which is then sent to SRDS using the XATICA standard function (`pushMDinfo`). Mobile devices update the info stored by SRDS each ten minutes.

The sever part of this Publishing subsystem consist basically of a wrapper for the XATICA SRDS API, in order to send information from mobile devices to SRDS in a transparent way.

The figure 6.9 shows the general architecture of the Publishing subsystem implemented in XtreemOS-MD

6.2.2.2 3G Connectivity Sharing Subsystem

The main goal of the Connectivity Sharing Subsystem is to facilitate the sharing connectivity to other XtreemOS users (connected from either mobile devices or PC's). Most of the recent smartphones offer 3G connectivity to Internet, which is consider as one of the resources to share in the XtreemOS Grid. This subsystem

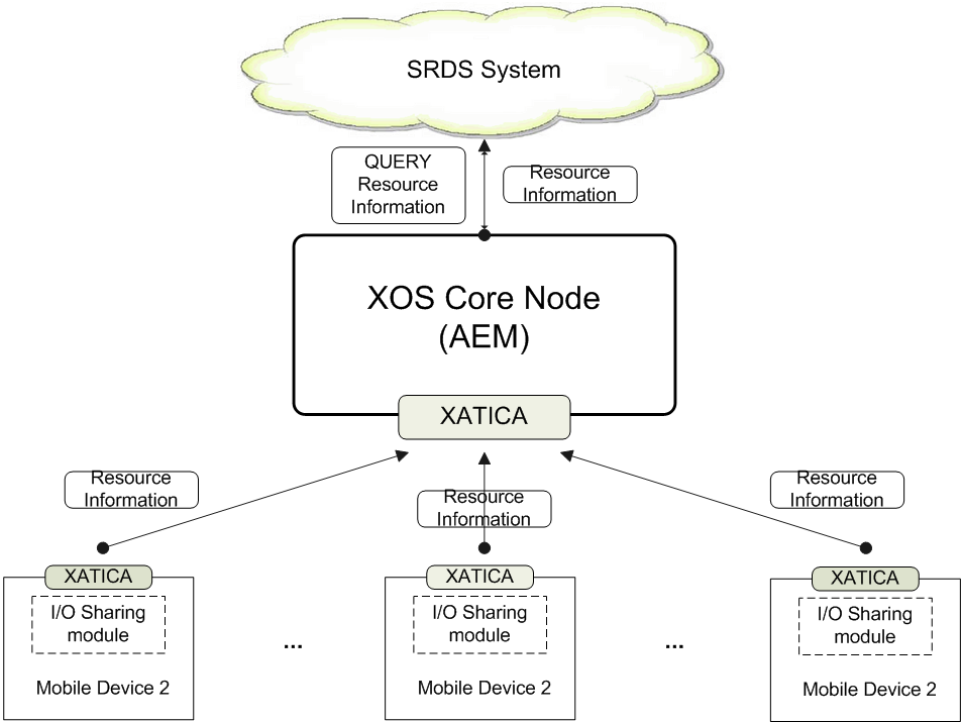

Figure 6.9: System for publishing information on SRDS

permits XtreemOS users to enjoy web traffic from devices without direct connectivity to the Web.

The OpenVPN networking platform [1] is used to share the 3G connectivity. OpenVPN works as a server in the mobile devices, controlling the number of connections, handling certificates and forwarding the web traffic. The OpenVPN, available for Maemo 4.2 (Diablo)[2] and Maemo 5 (Fremantle)[2], is controlled by the I/O sharing module. It manages the 3G connectivity resource in the MD, determining when the connectivity is shared and publishing its availability on the SRDS system.

To access to a "3G" shared resource, its necessary to be signed and authenticated inside a XtreemOS VO (XtreemOS certificates are used to control the access to the 3G resources) and to request a resource with 3G connectivity to the Grid. AEM (Resource manager) consults the SRDS system and returns the selected mobile device to share its connectivity. The information supplied by the SRDS (via AEM) to the user is the IP address and port where the OpenVPN running on the mobile is listening to "connectivity requests". It's necessary as well to have a openVPN client installed, which will be used to send the corresponding "connectivity request" to the mobile device acting as 3G provider. When the OpenVPN accepts the sharing request, a connection between the 3G provider and the claimant client is established for the web traffic.

The routing tables of the mobile device acting as provider are modified, and NAT forwarding is activated to manage the new connections and to allow the web traffic in a transparent way for the client.

6.2.2.3 GPS Sharing Subsystem

Currently, a lot of mobile devices are provided of a GPS device, which can be then considered as a resource to share in the Grid, in the XtreemOS context, allowing XtreemOS users to get accurate dates, GPS coordinates and to track paths of mobile devices.

This information is provided to the XtreemOS users as a standard metric available in the monitoring system of the AEM. Consequently the GPS resource is shown to the interested users or services completely integrated with the XtreemOS Monitoring system. On the other hand, mobile devices send periodically information about their position, date and followed paths using the XATICA API.

A service (user) that wants to know the GPS position of a mobile device or to track the path followed must send a job to the Grid with a reservation for a mobile node, requesting the desired metrics related to GPS using the standard procedure. The requested metrics are returned to the service together with other standard metrics (for example, in order to save them in the XtreemFS user's volume).

The figure 6.10 shows the GPS sharing subsystem inside XtreemOS-MD, based on events and metrics of the Monitoring system.

6.2.3 API definition

6.2.3.1 Publishing Subsystem API

The section presents a detailed description of the Publishing Subsystem API, which is invoked to communicate the mobile device with the SRDS system. Table 6.1 summarizes the different operations, parameters, results and status codes returned:

Name	Parameters	Result
pushMDinfo	(String) info	none
retrieveMDinfo	(String) query	(String) nodeInformation

Table 6.1: Publishing SubSystem API operations

6.2.3.2 pushMDinfo

Permits Mobile devices to push the resource information on the SRDS system.

Parameters

info: a string in XML format 6.11 with the resource information of the mobile device.

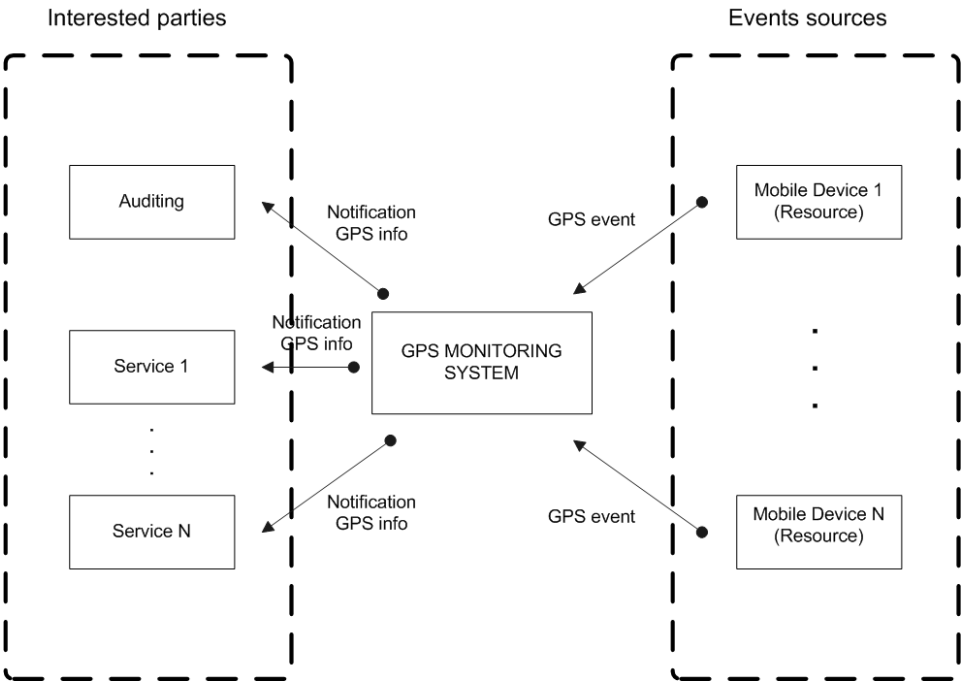

Figure 6.10: Schema of GPS information as metric values

Result

None

6.2.3.3 retrieveMDinfo

This method returns the selected Mobile device fulfilling the premises of the parameter *query*.

Parameters

query: a string in XML format 6.13 with the premises to select a concrete mobile device.

Result

result: a string with the contact information of the selected mobile device.

6.2.3.4 GPS Sharing Subsystem API

Table 6.2 summarizes the different operations, parameters, results and status codes returned by the GPS Sharing Subsystem API.

Name	Parameters	Result
pushGPSInfo	(String) info	none

Table 6.2: GPS sharing SubSystem API operations

6.2.3.5 pushGPSinfo

Permits Mobile devices to send the resource information related to GPS to the AEM.

Parameters

info: a string in XML format 6.2.3.6 with the resource information of the mobile device.

Result

None

6.2.3.6 info data format

The schema of the XML document for publish messages of resources on the SRDS system is presented in the figure 6.11

An example of a possible publish message is presented in the figure 6.12

6.2.3.7 query data format

The schema of the XML document for queries resources on the SRDS system is presented in the figure 6.13

An example of a query message is presented in the figure 6.14

6.2.4 Implementation

6.2.4.1 Publishing parameters Implementation

The I/O sharing module is a program written in C language that starts when the mobile device is switched on. It is in charge of collecting the resource information through the context awareness API, formatting it in XML format following the schema shown in section 6.2.3.6 and sending it as a a XML string to the SRDS using the XATICA method *pushMDInfo*. The procces is repeated each 10 minutes in order to keep updated information on the SRDS system. Each publish message contains a timestamp to inform when the information was collected. Note that the SRDS system considers obsoleted the information published more than 30 minutes ago, that is the reason to resend the mobile resources infromation from time to time, even if the data have not changed.

Given that the module uses the ContextAwareness API, the program is linked with the dynamic library `libCONTEXT` installed with the XtreemOS-MD distribution.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
   <xs:element name="XtreemOS">
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="MobileNode" maxOccurs="unbounded"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="MobileNode">
      <xs:complexType><xs:sequence>
            <xs:element ref="TimeStamp"/>
            <xs:element ref="ID"/>
            <xs:element ref="EnabledForGrid"/>
            <xs:element ref="GPS"/>
            <xs:element ref="G3"/>
         </xs:sequence></xs:complexType>
   </xs:element>
   <xs:element name="TimeStamp" type="xs:string"/>
   <xs:element name="ID" type="xs:NCName"/>
   <xs:element name="EnabledForGrid" type="xs:boolean"/>
   <xs:element name="GPS">
      <xs:complexType>
         <xs:sequence><xs:element ref="GPSdata" minOccurs="0"/></xs:sequence>
         <xs:attribute name="capability" type="xs:boolean" use="required"/>
         <xs:attribute name="enabled" type="xs:boolean" use="required"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="GPSdata">
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="GPStime"/><xs:element ref="GPSlat"/><xs:element ref="GPSlong"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="GPStime" type="xs:string"/>
   <xs:element name="GPSlat" type="xs:decimal"/>
   <xs:element name="GPSlong" type="xs:decimal"/>
   <xs:element name="G3">
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="G3data" minOccurs="0"/>
         </xs:sequence>
         <xs:attribute name="capability" type="xs:boolean" use="required"/>
         <xs:attribute name="enabled" type="xs:boolean" use="required"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="G3data">
      <xs:complexType>
         <xs:sequence><xs:element ref="providerIP"/><xs:element ref="providerPort"/></xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="providerIP" type="xs:NMTOKEN"/>
   <xs:element name="providerPort" type="xs:integer"/>
</xs:schema>
```

Figure 6.11: Schema for publish message in XML format

```
<?xml version="1.0" encoding="UTF-8"?>
  <XtreemOS>
   <MobileNode>
    <TimeStamp>2010-03-11 16:10:53.968</TimeStamp>
    <ID>md-xxxx0000</ID>
    <EnabledForGrid>true</EnabledForGrid>
    <GPS capability="true" enabled="true">
     <GPSdata>
      <GPStime>2010-03-11 16:10:53.968</GPStime>
      <GPSlat>30.0</GPSlat>
      <GPSlong>40.5</GPSlong>
     </GPSdata>
    </GPS>
    <G3 capability="true" enabled="true">
     <G3data>
      <providerIP>123.45.67.3</providerIP>
      <providerPort>1145</providerPort>
     </G3data>
    </G3>
   </MobileNode>
  </XtreemOS>
```

Figure 6.12: Example of publish messages

6.2.4.2 3G Connectivity Sharing Implementation

The 3G connectivity sharing is based on:

- OpenVPN software, to create manage and terminate communications (VPN) between the resource mobile device (as server) and the possible clients (other mobile devices or PC's). In addition, the module is in charge of modifying the routing tables to enable the forwarding of the web traffic through the mobile device that works as a connectivity provider.
- the I/O sharing module that controls the OpenVPN starting/stopping and current status, as well as the number of open connections and possible incidents related to the VPN.

The `I/O sharing module` contains the functions *start_sharing(), stop_sharing(), monitor_sharing()* that perform the operations mentioned above. The maximum number of sharing connections supported is defined by the user in the configuration files and it is controlled by the module.

Some modules have also been added to the kernel, taking into account that some of the smartphones compatible with XtreemOS do not support NAT forwarding:

- ipt_MASQUERADE.ko
- ipt_NETMAP.ko
- iptable_nat.ko
- nf_conntrack.ko
- nf_conntrack_ipv4.ko

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
   <xs:element name="Query">
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="TIMESTAMP" minOccurs="0"/>
            <xs:element ref="GPS" minOccurs="0"/>
            <xs:element ref="G3" minOccurs="0"/>
            <xs:element ref="GRID" minOccurs="0"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="GPS">
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="Capability"/><xs:element ref="Enabled" minOccurs="0"/>
            <xs:element ref="Lat" minOccurs="0"/><xs:element ref="Long" minOccurs="0"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="TIMESTAMP">
      <xs:complexType>
         <xs:sequence><xs:element ref="TSMin"/><xs:element ref="TSMax"/></xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="Lat">
      <xs:complexType>
         <xs:sequence><xs:element ref="Min"/><xs:element ref="Max"/></xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="Long">
      <xs:complexType>
         <xs:sequence><xs:element ref="Min"/><xs:element ref="Max"/></xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="G3">
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="Capability"/><xs:element ref="Enabled" minOccurs="0"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="GRID">
      <xs:complexType>
         <xs:sequence><xs:element ref="Enabled"/></xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="Capability" type="xs:boolean"/>
   <xs:element name="Enabled" type="xs:boolean"/>
   <xs:element name="Min" type="xs:decimal"/>
   <xs:element name="Max" type="xs:decimal"/>
   <xs:element name="TSMin" type="xs:string"/>
   <xs:element name="TSMax" type="xs:string"/>
</xs:schema>
```

Figure 6.13: Schema for queries in XML format

```
<?xml version="1.0" encoding="UTF-8"?>
   <Query>
     <TIMESTAMP>
      <TSMin>2010-03-11 16:10:53.961</TSMin>
      <TSMax>2010-03-11 16:12:53.961</TSMax>
      </TIMESTAMP>
      <GPS>
        <Capability>true</Capability>
        <Enabled>true</Enabled>
        <Lat>
          <Min>10.0</Min>
          <Max>20.0</Max>
        </Lat>
        <Long>
          <Min>0.0</Min>
          <Max>20.0</Max>
        </Long>
      </GPS>
      <G3>
        <Capability>true</Capability>
        <Enabled>false</Enabled>
      </G3>
      <GRID>
        <Enabled>false</Enabled>
      </GRID>
    </Query>
```

Figure 6.14: Example of queries

- nf_nat.ko

Furthermore, the `IPTABLES` command included in Maemo and Openmoko plantfomrs does not support NAT forwarding so this command has been recompiled for ARM architecture and included in the new XtreemOS-MD distribution.

The figure 6.15 shows the sequence of messages taking part in the 3G sharing process. First the `I/O sharing module` publishes the resource information through the `pushMDInfo()` method and the instance of AEM forwards the message to the SRDS system.

When a XtreemOS client needs a 3G connection, a reservation for a resource with 3G is sent to the SRDS, returning back the contact information (IP address and port) of a resource fulfilling the client requirements. This information is used to invoke the OpenVPN running on the Mobile resource. Once the client contacts with the Mobile resource, the control of the operation is transferred to the OpenVPN software.

6.2.4.3 GPS Sharing Implementation

Following the general premises of the implementations for Mobile devices, GPS sharing mechanisms tries to avoid opening ports in the mobile device for security reasons. Due to the fact that an open port is necessary to listen to GPS requests from clients and to return the requested information, a solution has been implemented to avoid the possible security problems of opening ports. To this purpose, a plug-in has been included in the AEM system as a new AEM service. This AEM plug-in is in charge of receiving and storing periodic updates of GPS positions from mobile devices through `pushGPSInfo()`, as well as receiving requests from AEM to

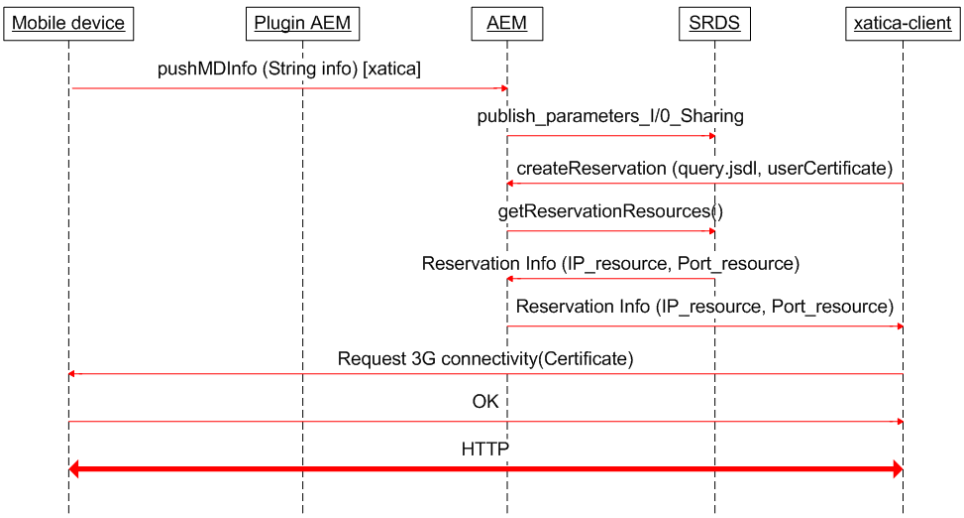

Figure 6.15: 3G Sharing Schema

get the GPS information as a metric for its monitoring tasks. When the plug-in receives this kind of requests, it collects and sends the information invoking the `addMetric` and `setMetricValue` methods of the AEM API.

The complete sequence of message is shown in the figure 6.16

This solution fulfills the security premises for MDs but modifies the guidelines of sharing mobile device resources: the implemented GPS sharing mechanism is a passive process (the MD is updating the information without any client request) instead of offering the GPS information on demand (in an asynchronous way). However, the selected solution allows the integration of GPS information as a metric into the resources metrics set available in the Monitoring System of the AEM, which will be offered like other classical resources (RAM, CPU, disk, etc.)

The implementation of the GPS Sharing Subsystem can be divided in two components:

- Server (Plugin AEM)
- Client

Server side: Plugin AEM The AEM process includes a new plug-in for receiving the GPS location from mobile devices and storing and offering it to any job in the GRID. The plug-in exports the API for mobile devices: `shareGPS`, `updateGPSPosition` and `unshareGPS`. For each mobile device, and while sharing its GPS, the plug-in stores the location and offers an API to access that location data to any job in the GRID through the AEM, using the methods: `subscribeToGPS` and `unsubscribeToGPS`.

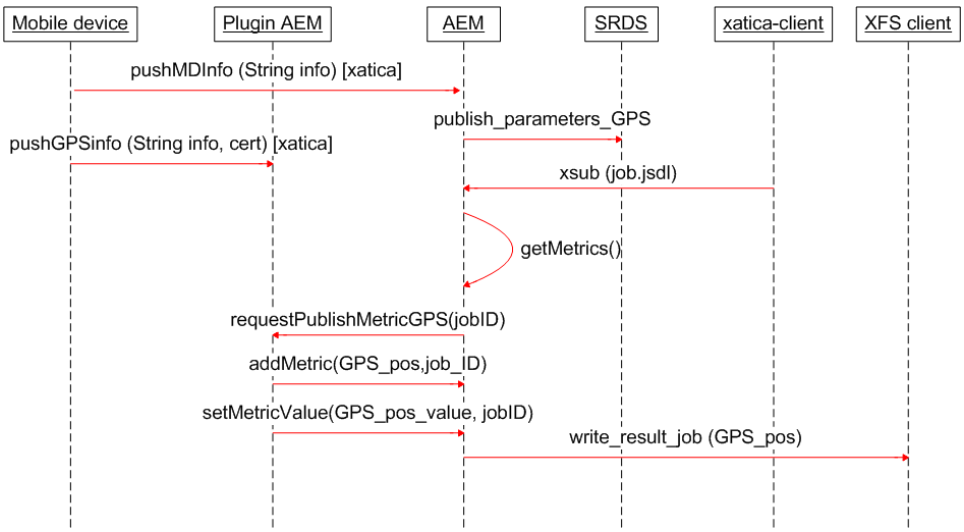

Figure 6.16: GPS Sharing Schema

Client side The client part of GPS sharing is integrated into the `I/O sharing module`. The main goal of the client part is to update the GPS information (in format 6.2.3.6) to the plug-in AEM using the method `pushGPSInfo()` . This process is similar to the other functions of the `I/O sharing module`.

6.2.5 Installation and usage

6.2.5.1 Installation

Client side The implementation of the I/O sharing module is created and delivered in the form of an executable which is offered with the XtreemOS-MD version as a `deb` package for Maemo, although it is possible to install the module manually, executing the following commands:

- Maemo devices:

  ```
  #dpkg -i moduleSharingIO-1.0.deb
  ```

  This command installs and configures the module in the system.

Server side MDEventProxy is an AEM service and as such is packaged along with the rest of services in the third release.

6.2.5.2 usage

Client side The I/O sharing module works in a transparent way for the user, but the configurations and options can be modified from the configuration tool, which is detailed in section 6.3.

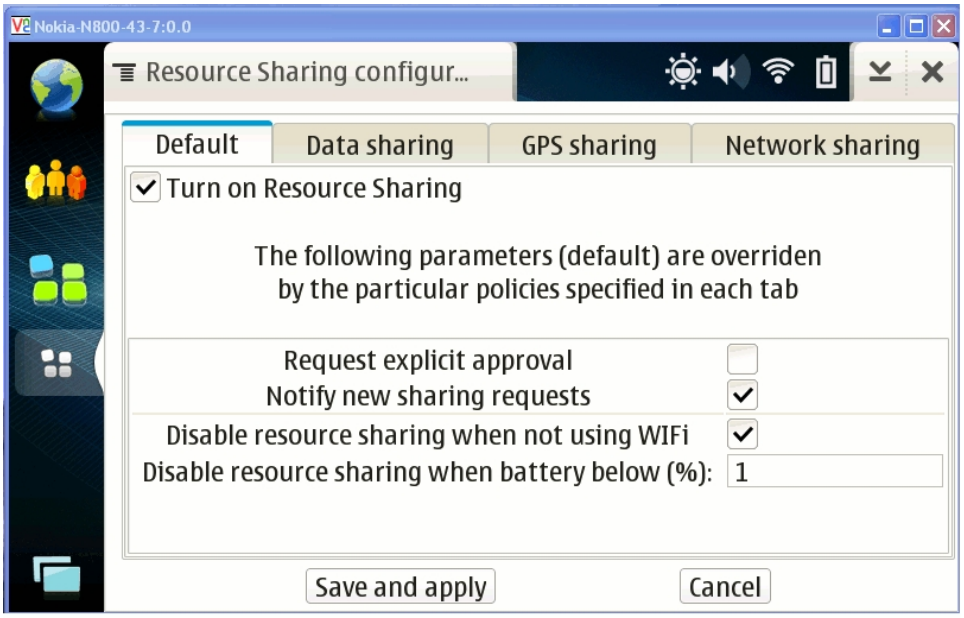

Figure 6.17: Resource sharing configuration tool

Server side In order to use the MDEventProxy, there must be at least one instance of it running in the Virtual Organization. To enable it just check that the following file exists in the appropriate node and has the enabled property set to true.

`/etc/xos/config/xosd_stages/MDEventProxy.stage`

6.3 Resource sharing configuration tool

6.3.1 Introduction

As mobile device users are not necessarily "computer-experts", the modification of a textual configuration file could be supposed as a not so trivial task. Graphical tools are much more usable for those kind of devices, so that, the resource sharing package includes a tool that is installed and accessible from the device's application menu, in order to facilitate the possible configuration modifications of the resource sharing features.

6.3.2 Configurable parameters

The resource sharing configuration tools allows a quick configuration of the main parameters of the resource sharing daemon and the different sharing modules (data, I/O and network). The graphical interface, as shown in figure 6.17, is composed of different tabs:

Default Includes the "Turn on Resource Sharing" check button, which allows enabling/disabling the whole resource sharing feature (when enabled, just the different enabled sharing modules will be active anyway) and the general parameters that are considered if the different sharing modules don't specify a different policy overriding the default one. The following list of parameters is available as well from the other sharing module tabs, just by clicking on the correspondent "Show specific parameters" button.

- Request explicit approval: to force the user's approval of any request to access to the shared resources
- Notify new sharing requests: to show a notification on the device screen when the shared resource is being accessed
- Disable resource when not using Wifi: to allow disabling the resource sharing feature when not connected via Wifi (as the user could be connected with a non-free access...)
- Disable resource when battery below (%): to disable the resource sharing feature in order to save battery

Data sharing

- Enable Data Sharing: to enable/disable the data sharing module
- Shared folder: to specify the local folder containing the files that will be shared to the rest of Grid users.
- Data sharing proxy IP: IP of the proxy module used to facilitate the data sharing with the rest of the Grid
- Data sharing proxy port: port where the mentioned proxy is listening

GPS sharing This is a particularization of the I/O device sharing, as for the moment the GPS is the only device considered (other possibilities could be the onboard camera, micro, etc.)

- Enable GPS Sharing: to enable/disable the GPS sharing module
- GPS sharing proxy IP: IP of the proxy module used to facilitate the GPS sharing with the rest of the Grid
- GPS sharing proxy port: port where the mentioned proxy is listening

Network sharing

- Enable Network Sharing: to enable/disable the network sharing module
- Max. number of simultaneous connections: to limit the number of users that can be accessing the shared network interfaces simultaneously
- Network sharing proxy IP: IP of the proxy module used to facilitate the network sharing with the rest of the Grid
- Network sharing proxy port: port where the mentioned proxy is listening

Chapter 7

Conclusions

In this document we have reviewed the implementation of the advanced version of XtreemOS-MD G-layer. The main new features, compared to the basic version, covered by this implementation are:

- Support for smartphones bsaed on Maemo 5, like the Nokia N900, keeping the previous support for PDAs based on Maemo 4, like the Nokia N8x0 family
- Data management features: using Vivaldi to select replicas in XtreemFS and providing an OSDProxy to permit the data sharing feature without compromising the XtreemFS security model.
- AEM improvements related to call-back monitoring (server part removed, the notifications are no more based on asynchronous events) and related as well to the publication in the SRDS of the MD resources shared
- Security improvements related to the PAM support, allowing the integration with legacy SSOs as well as opening the range for new authentication mechamisms.
- Resource sharing capabilities allowing the mobile device to share data on demand (acting like a fake OSD), and to share the access to the I/O devices and network access provided by the mobile terminal.

It is especially relevant the addition of resource sharing capabilities to XtreemOS-MD, as it makes possible the inclusion of mobile devices as special resource nodes of the Grid, not being the MDs limited to pure Grid clients, as they were with the XtreemOS-MD basic version. As the mobile device is not a conventional resource node of the Grid (taking into account the XtreemOS security model), it cannot be considered a trusted node and that is the reason to include a special “trusted proxy” in the Grid, which is in charge of the direct communication with the mobile devices sharing resources.

This new XtreemOS-G advanced version is currently being tested and integrated with the existing XtreemOS components. Together with the F-layer it will be packaged, in cooperation with WP4.1, for PDAs (Nokia N8x0, like the basic version) and also for smartphones (like Nokia N900). It will be ported to Ubuntu (for netbooks) and additionally, 3rd parties could port it to other Linux-based mobile device platforms (MobLin, LiMo, Android, etc.)

As a final conclusion for the work done in WP3.6, we should remark the evolution experimented from the basic version to the final advanced version implemented, supporting not only the last version of Security, AEM and XtreemFS XtreemOS services, but also converting the mobile device in a special resource node thanks to the inclusion of the resource sharing feature. Those implementations open the door for the development of new applications in the domain of mobile devices, extending enormously their capabilities with the potential offered by the Grid.

References

[1] OpenVPN - Open Source VPN.
`http://openvpn.net/`.

[2] OpenVPN for Maemo5.
`http://maemo.org/downloads/product/Maemo5/openvpn/`.

[3] XtreemOS Consortium. XtreemOS-G for MDs/PDA D3.6.3. Integrated Project, December 2008.

[4] XtreemOS Consortium. Design of advanced services for mobile devices D3.6.5. Integrated Project, December 2009.

[5] XtreemOS Consortium. Design of an advanced Linux version for mobile devices, D2.3.6. Integrated Project, October 2009.

[6] XtreemOS Consortium. Linux-XOS for MD/MP, D2.3.7. Integrated Project, December 2009.

[7] XtreemOS Consortium. Requirements and specification of advanced services for mobile devices D3.6.4. Integrated Project, May 2009.

[8] XtreemOS Consortium. Advanced AEM prototype D3.3.7. Integrated Project, March 2010.