Project no. IST-033576

XtreemOS

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

Linux-XOS for MD/MP
D2.3.7

Due date of deliverable: November 30th, 2009
Actual submission date: December 10th, 2009

Start date of project: June 1st 2006

Type: Deliverable
WP number: WP2.3
Task number: T2.3.7

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

Version 0.5 / Last edited by Santiago Prieto / December 10th, 2009

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

Revision history:

Version	Date	Authors	Institution	Section affected, comments
0.1	14/10/09	Telefónica I+D	Telefónica I+D	Document created
0.2	02/11/09	Telefónica I+D	Telefónica I+D	Intermediate version
0.3	26/11/09	Telefónica I+D	Telefónica I+D	Ready for internal review
0.4	4/12/09	Telefónica I+D	Telefónica I+D	Draft version after first internal review
0.5	10/12/09	Telefónica I+D	Telefónica I+D	Final version

Reviewers:
Massimo Coppola (CNR), Toni Cortés (BSC)

Tasks related to this deliverable:

Task No.	Task description	Partners involved°
T2.3.7	Implementation of an advanced Linux version for mobile devices (Linux-XOS for MD/MP)	TID*, INRIA

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

Executive Summary

After having released the first version of XtreemOS-MD software, available for Maemo-based PDAs like the Nokia N8x0 series, this deliverable presents the work done on the implementation of the advanced XtreemOS-MD Foundation layer (F-layer). This work has been organized into the following branches:

- On one side, as there is a new release of XtreemOS including several new features, we have updated the XtreemOS-MD F-layer to be compatible with them. This means additional enhancements regarding security and VO support.
- Several enhancements in the configuration and installation process have been implemented, taking into account the experience with XtreemOS-MD release 1.0. That includes also a tool to ease the manual configuration of the system (everything was automatic in release 1.0, very convenient for end users, but closed the door to later modifications on the configuration).
- As detected during the design phase, it would be useful to have knowledge of some context parameters related to mobile devices, like the remaining battery, GPS status and position, network connectivity, etc. These kind of parameters could be used for example to determine if the device status may be considered as "online" or "offline" for concrete services. The specific use of the context information is left to other parts of the software (G-layer for instance), but the support to provide context awareness information have already been included in the Foundation layer, which this document is focused on.
- One of the most important modifications from the previous XtreemOS-MD release to this advanced version is the inclusion of the resource sharing module into the mobile device. This resource sharing module, analyzed in deliverable D2.3.6 [11] is implemented partially on the F layer, concretely for giving support to G layer to implement services like data sharing, or to open the possibility to share the input/output and network devices.
- Finally, this new implementation of the XtreemOS-MD F-layer has been adapted to be supported not only by PDAs, but also by smartphones. Following the criteria to prioritize the smartphone platforms supported identified in deliverable D2.3.6 [11], we have initially worked on the support of the Neo FreeRunner terminal. But the software is portable to other Linux-based platforms and that could be done by the XtreemOS' developers community.

Contents

List of Figures

List of Tables

Glossary

ARM	Advanced RISC Machine
API	Application Programming Interface
CN	Common Name
EGID	Effective Group Identifier
EUID	Effective User Identifier
GID	Group Identifier
GPS	Global Positioning System
MD	Mobile Device
PAM	Pluggable Authentication Modules
PDA	Personal Digital Assistants
RSD	Resource Sharing Daemon
SSH	Secure Shell
SSL	Secure Socket Layer
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 Foundation layer (F-layer). This advanced version, respect to the the first version previously released, includes support for smartphones, like the OpenMoko platform (and its model Neo FreeNunner) and also includes additional features regarding VO management and security. The software could also be ported by 3rd parties to other Linux-based mobile device platforms (LiMo, Android, etc.) as already stated in [11], where a deep analysis on the current Linux-based platforms where made: *"we are ensuring that XtreemOS-MD advanced version will be available for dissemination to the developers community and thus we are leveraging the adoption of XtreemOS-MD to more OS and device manufacturers, which are the key players for a success story of XtreemOS as a whole"*.

In addition, this version provides support for a new advanced feature that allows sharing of mobile device resources to the Grid, thus converting the mobile device from a Grid client into a special Grid resource. For this purpose, the context awareness module has been completed and the resource sharing module designed in D2.3.6 has been partially implemented (as part of the implementation of this module also belongs to the services layer and will be implemented by WP3.6).

Finally, some improvements in the installation and configuration process have been implemented, so that keeping the ease achieved by the basic version, it would be still possible to modify the initial configuration. A graphical tool designed specifically for this purpose allows the user to select the adequate configuration.

For a better comprehension, we recommend to review the XtreemOS-MD architecture, described in detail in [11] and [10], and shown in figure 1.1 for convenience.

1.1 Document structure

The document is structured as follows:

Chapter 2 deals with the new features concerning VO management and security, from the use of the new VOLife web interface, to the new credential modules

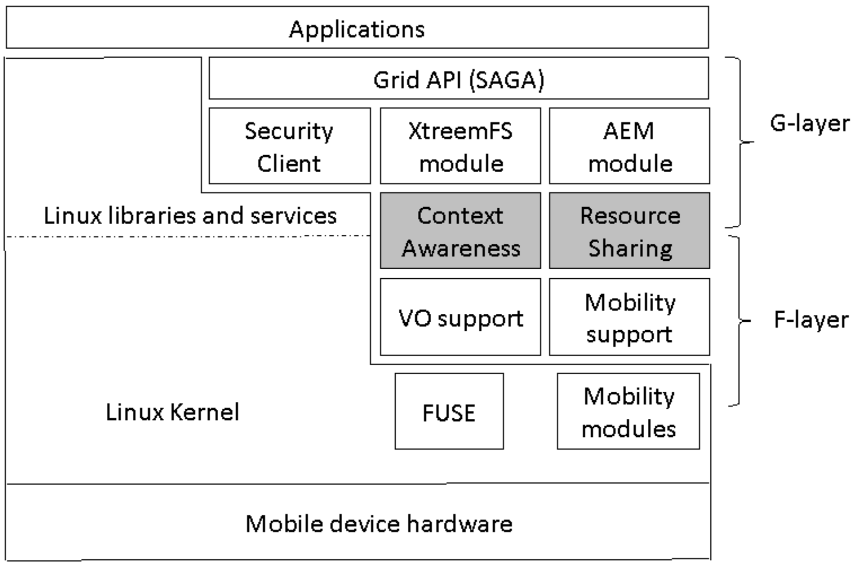


Figure 1.1: XtreemOS-MD general architecture

provided. In chapter 3 the implementation and the API of the new module for resource sharing is explained, at least the part that corresponds to layer F, as this module is part of F and G layers. Then, chapter 4 is focused on the implementation of the context awareness support, and the API offered to upper levels to access to the context information (mode, GPS, etc.).

Next, chapter 5 details the implemented enhancements concerning installation and configuration, and the new graphical tool provided to select a concrete configuration, useful for end-users as well. Finally, chapter 6 outlines the conclusions and next steps to achieve a final version of the whole Xtreemos-MD software (including both F and G layers).

Chapter 2

VO management and security

XtreemOS-MD F-layer basic version already provided a rich security infrastructure to support a secure access to the Grid, implementing a modular, pluggable SSO solution, which allowed users forgetting about credential administration. VO Management through VOLife web interface was also supported by the basic version.

In this advanced version, apart from the smartphone support for every functionality, some new features have been included concerning VO management and security aspects.

2.1 Security enhancements

2.1.1 User interface enhancements for security

User interface enhancements take into account the interface limitations inherent to mobile devices, such as the lack of a physical keyboard in some cases or at least their lack of comfortability. This means that entering a long password for authenticating the user could be a not-so-easy task. To overcome this limitations we have implemented the following enhancements:

- PIN-based authentication: We have included a PIN-based authentication system, where the mobile device user will just need to introduce a PIN, a short number, that will be sent to the CDAProxy. The CDAProxy keeps a table matching user+PIN with the corresponding full password in the Grid, which will be used to continue with the authentication process. Additionally, the CDAProxy manages a system to limit the number of failed access tries (the table that matches PINs and passwords also provides a field for storing the number of consecutive PIN entry errors). After the third one, the PIN is invalidated and the user needs to introduce the full password from his mobile device in order to get authenticated in the Grid. To obtain a new PIN, the user will need to contact the administrator.

- Bluetooth pairing support: As an alternative, we are also developing a solution for an authentication process based on Bluetooth pairing. This is useful when a Bluetooth enabled PC runs a CDAProxy and a mobile device establishes a bluetooth pairing with it. With the appropriate modifications to the CDAProxy this condition may be detected and then, just by establishing a Bluetooth connection with the PC, the CDAProxy will consider authenticated the user and it will launch the authentication process into the Grid. Of course, this is useful for devices located under the bluetooth distance range. However, the pairing is only necessary when establishing the session. In order to implement this feature, the CDAProxy shall support PAM modules. As this is an ongoing development for the CDAProxy at G-layer (being carried out by WP3.6), this feature will not be available until the release of the G-layer software.

2.1.2 Credential modules and cryptography

The basic version of XtreemOS-MD already provided a modular architecture for the authentication process. This architecture was based on two different types of modules: *credagent* modules, to obtain the credential, and *creduiagent* modules, to interact with the user when needed (to request a password for example).

In this new advanced version, we have implemented new *credagent* and *creduiagent* modules. With this new modules, a password may be requested to the user by other means (e.g. Blueotooth pairing or any other type of delegated authentication), acting like a kind of remote *creduiagent* module. Furthermore, it would be possible to implement parental control solutions. Among the solutions considered during the design phase, the implementation of the new *credagent*/*creduiagent* modules has been based on the use of *libcurl* libray.

On the other hand, it has also been provided an access control by group to the credential store (more concretely to the *libcredstore* library), so that just designated applications will be allowed to gain access to the *credstore*. To implement this access control, an additional method has been included in the `libcredstore` library, in order to deny the access to non-members of a special group:

```
int (*protect_credstore)();
```

If this function is invoked, the *credstore* daemon will deny the access to processes with a EGID different to its own. This function does not return any direct result. It only returns a status code to inform whether the method worked properly (value 0) or generated an internal error (value -1).

2.2 VO Management from MDs

The redesigned VO management Web interfaces [12] (the VO Web Frontend interface offered by XVOMS, and the RCAWeb Frontend interface to the RCA Server) are still supported by the target mobile devices, and concretely by smartphones with even more limitations regarding screen size and resolution compared to the PDAs. No additional implementation work is needed for VO management from MDs.

Chapter 3

Resource sharing

3.1 Introduction

Resource sharing in mobile devices is one of the biggest challenges of XtreemOS-MD advanced implementation. Resource sharing services will be implemented in the G layer, but the F layer provides a common foundation for all these services through a daemon, the Resource Sharing Daemon (RSD), and a modular architecture.

This layer also provides the functionality for detecting the moment when files are moved to a shared directory, which will be used by the data sharing service.

3.1.1 Security and Network Model

In XtreemOS PC flavor, just the nodes with a valid RCA certificate can share resources. This policy is not acceptable with XtreemOS-MD, as RCA certificates guarantee that the machine is a trusted node and mobile devices are normally not trusted nodes. Therefore, in XtreemOS-MD resource sharing, only CDA certificates are used. When interacting with services that are designed to run in trusted nodes (e.g. XtreemFS), XtreemOS-MD will use a proxy software running in a trusted node. This proxy software will authenticate XtreemOS-MD node using its CDA credential and will guarantee to other nodes that the mobile device doesn't break the VO security. For example, in data sharing service, a proxy will guarantee that the mobile device doesn't impersonate other users.

As RSD needs access to user credentials, invocation of *libxos_getcred* is needed. The daemon is invoked automatically when *startxtreemos* is successfully started.

It's worth noting that RSD is designed for mobile devices, which are personal, mono-user devices. In a multiuser device, the configuration tools runs as root just for controlling system settings, while the configuration tools for user preferences runs with the identity of the user. However, when there is only one user account, it's more secure to consider the user preferences as system settings, and to protect these preferences in files which are not writable by the user. This is for example the security model of Maemo.

Resource sharing implies that a service running in mobile devices must respond to requests from clients of the VO. Two models are possible:

1. There is a proxy running in a trusted node; clients connect to this proxy and the proxy redirects the request to the corresponding mobile device node.
2. Clients connects directly to mobile device node.

The first model has been selected as it presents some advantages:

- As the proxy runs in a trusted resource, it is much more secure.
- It is more friendly with NAT, network or software interruptions and IP address changes. This is because mobile devices may establish a persistent connection with the proxy to receive the requests, instead of creating a listen socket, that requires to open a port and a static (or at least stable) IP address.
- It implies less resources in mobile device side: it is not necessary to authenticate each request nor establish a connection per request.
- It allows integration with the Grid delegation mechanism (currently being designed), as the authentication could be done against a trusted node using *xos-ssh*.
- It guarantees anonymity between client requesting the service and node providing it.
- It allows integration with Grid services like accounting for instance.

Theoretically, the second model presents the advantage that a central server is not required. However, direct connection does not prevent the need of a central server, because some kind of service is needed to search nodes conforming to client request. If clients connect directly to a specific IP address instead of searching suitable nodes for a specific need, this would be a client-server architecture and not a Grid one.

3.1.1.1 Use of SSH-XOS

There are several methods to establish a secure channel between a resource sharing service and the proxy running in a trusted node.

A well known option is based on the creation of a TLS connection using the CDA credential as authenticator. One advantage of TLS is the session caching: the handshake is avoided when a new connection is established, even if the client IP address changes.

Another option is based on the use of SSH-XOS. The main advantage presented is that this solution is used in other parts of XtreemOS (e.g. AEM) and a new SSO based on SSH-XOS is also expected for the final version. SSH-XOS is based in OpenSSH, which provides “Control Master“, a mechanism that allows creating new channels inside an already established SSH connection. Another strength of SSH is that it implements port-forwarding, a mechanism to support multiple incoming connections in a scenario with NAT, using only one outgoing connection and not needing to open input ports in the intermediate router.

SSH-XOS uses *libxos_getcred* to obtain the credential. This means that the resource sharing is somehow integrated with the XtreemOS-MD SSO. Just one connection is needed to provide the resource sharing and if this connection is broken, it is restarted without disturbing the user as the credential is cached in the *credstore*.

The general idea is the creation of a pair of TCP redirections between the trusted node where the proxy runs and the mobile device where the resource sharing daemon runs:

- One redirection goes from the proxy to the mobile devices. The proxy connects with the mobile device when a request to a shared resource arrives (e.g. another user reads a file shared by the mobile device). This port redirection is created with `OpenSSH -R` option.
- The other redirection goes from mobile devices to the proxy. The mobile device connects to the proxy when there is a change related to the resource availability (e.g. when a new file is shared). This second port redirection is created with `OpenSSH -L` option.

When each part connects to the redirected server port, it first writes a shared cookie for authentication. Even if the use of local ports prevents from unauthorized remote connections, this step is required to avoid connections to a local server from any local process. The cookies are generated using `uuidgen` with `-r` option, so that the UUID are generated with 122 random bits, enough to avoid collisions [8].

3.1.1.2 Ports allocation

There is a problem when creating a redirection with `OpenSSH -R` option: a port in the proxy node must be previously reserved, because the command will fail if the port is already in use. Each supported mobile device needs a different port, therefore, static allocation is not an option. To solve this problem, the proxy node runs the *xos_register_port* utility, which firstly allocates a free port (using *listen* with an unbound socket and *getsockname* to discover the port number) and then releases the obtained port to make it available once again before returning the port number to the mobile device.

Note that a rare race condition may occur if the port is allocated by other process after the utility releases the port but before SSH redirection is created. To avoid this, *xos_register_port* creates an internal connection to the port and closes the connection before releasing the port. This way, the socket will be in `TIME_WAIT` state for several seconds, not being available so immediately.

When the SSH connections ends, the ports are released and recycled automatically by the kernel and they become available again after the `TIME_WAIT` timer expires.

The *xos_register_port* utility is responsible for storing the port in a database as well. This database is used by the proxy to know how to contact the user's mobile device.

3.1.2 Daemon Responsibilities

RSD provides the following features:

- Establishment of a secure, authenticated connection to create bidirectional channels between the remote proxy and the mobile sharing service. RSD will reconnect if the connection is lost. For temporal unavailabilities, it will wait until a D-BUS notification related with the connection availability is received.
- For each incoming proxy request from the proxy, the RSD will invoke the associated service and will pass the open socket to it. This way, the daemon makes the resource sharing service development easier: services are implemented as *inetd* servers and resource sharing daemon may run services as isolated, with the requested EUID and EGID, according to the manifest included in the module package.
- Enforcement of user preferences concerning incoming requests' approval: automatic approval, notification to the user or request user authorization.
- The RSD is in charge of guaranteeing that just the services desired by the user are running. Each module must include a declaration about the information that is shared and its implications: users can check this information and enable or disable the module accordingly to their preferences.
- Enforcement of user preferences concerning service availability and context awareness. For example, users may establish that resource sharing service will be disabled while network connection is 3G or GPRS and enabled if it is free Wifi. Others context criteria are availability and battery status.
- The RSD will restart a resource sharing module that terminates unexpectedly.

3.2 General architecture

The general architecture of resource sharing is shown in figure 3.1.

RSD provides the infrastructure for sharing services through a modular architecture. For each resource sharing service, RSD interacts with two components:

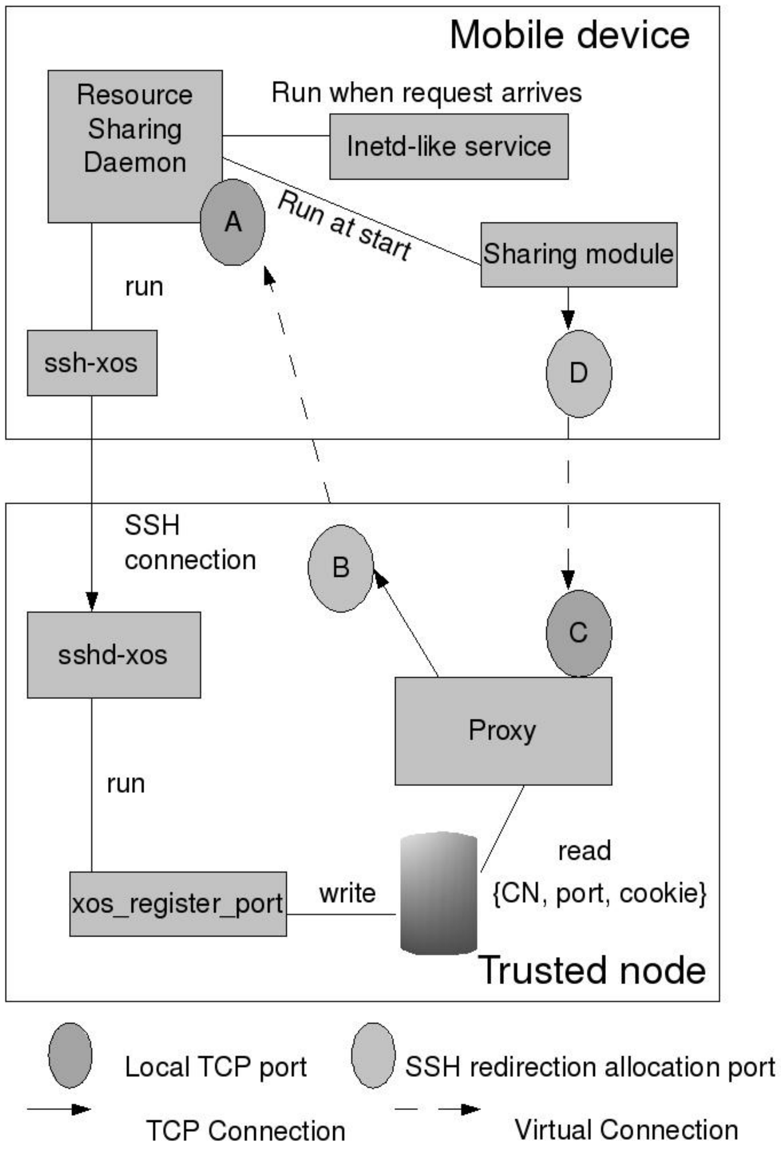


Figure 3.1: Resource sharing architecture

1. the *inetd* service, which is invoked when a request arrives from proxy to port A: the proxy really connects with its local port B, that is redirected by SSH to mobile node port A.

2. the sharing module, which is invoked when the RSD starts if the user has enabled it. This module is responsible for the proxy interaction when resource sharing configuration changes (e.g. a user shares a new file). This module interacts with the proxy through a connection to local port D, which in turn is redirected to proxy's node C using the SSH tunnel.

RSD uses SSH-XOS to obtain port B and register it in the port database used by the proxy. The port database stores two types of information:

1. CN and *machine-id*. This information is indexed by cookie. The proxy queries for this record when a new request from a sharing module running in a mobile device arrives.

2. Port and cookie information. This information is indexed by the CN (Common Name) of the user, machine ID and the name of the service. The proxy queries for this record when it needs to know how to connect with the sharing service running in the mobile device. *Xos_register_port* queries this record before creating a new entry to delete the old CN register indexed by cookie.

Machine-id is just a string to distinguish among several devices of the same user. That is, this ID must be unique only at user level. It's the equivalent of ”resource“ string in XMPP addresses [3].

Port database also stores the last modified date for each record, used for periodical removal of unused entries (e.g. users not already active). Port database is implemented using Berkeley DB.

For simplicity, the diagram shows just one proxy and one sharing service, the architecture supports multiple proxies and services (e.g. a file sharing service and a network sharing service).

3.2.1 How RSD works

The Resource Sharing Daemon is launched automatically by *startxtreemos* if the credential configuration file includes a *start_resource_sharing* parameter in the *[general]* section but it may be launched manually also. When started, it first scans the sharing modules under */usr/share/xos/sharingservices* folder and checks the configuration files in */etc/xos/sharingservices/* to know which modules are enabled.

RSD uses XOS-SSH to communicate with servers running in a trusted node and accepts service requests from them. Multiple channels are embedded in only one authenticated connection with the node, using the OpenSSH Control Master mechanism.

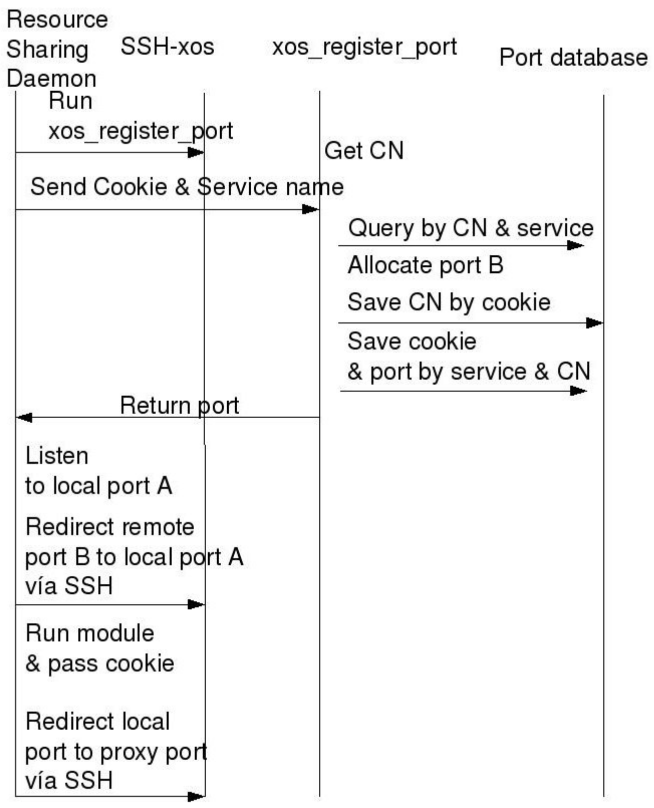


Figure 3.2: RSD starting sequence

The RSD starting process is described in figure 3.2. These are the involved steps:

1. RSD uses SSH-XOS to run *xos_register_port* in the trusted node where the proxy runs. The device generates a cookie (using *uuidgen*) and passes it to *xos_register_port* together with the service name and a machine-id.
2. *xos_register_port* gets CN from user, invoking *getpwuid(getuid())*.
3. *xos_register_port* queries the database by CN, machine-id and service name, to obtain port and cookie. Then it uses the cookie to remove the old entry indexed by cookie.
4. *xos_register_port* allocates a port that is not in use. This port (named B in diagram) will be used by the proxy to connect with the sharing service running in the mobile device. Then, it frees the port in `TIME_WAIT` state to be available for SSH redirection.
5. *xos_register_port* stores in the port database the CN and machine-id using the cookie as key. It also stores the port and cookie using the service name, the CN and the machine-id as key. The last entry is used for communications from the proxy to the sharing service in the mobile device; the former is used for communications from the mobile device to the proxy.
6. *xos_register_port* returns the allocated port B to RSD.
7. RSD allocates a local port A bound to 127.0.0.1 IP address (that is, this server in 127.0.0.1:A is not directly accessible from network without a redirector).
8. RSD creates a redirection from port B in the remote node to local port A, using `SSH-XOS -R` option. This implies that when the proxy running in a trusted node connects to its local B port, the connection is redirected through a SSH secure channel to the mobile device service port A.
9. RSD finds a free local port D
10. RSD uses `-L` option of SSH-XOS to create a redirection from local port D to the remote port C. Proxy server listens in port C the connections from mobile devices (this connections are used for example to inform that a new resource is shared).
11. RSD starts the local sharing module and passes to it the port D and the cookie.

A configuration tool is provided to allow users enabling and disabling services. This *set-euid* tool modifies the configuration file under */etc/xos/sharingservices* and notifies the resource sharing daemon using `SIGHUP` signal. The daemon reloads then the configuration and starts and stops services accordingly. The configuration tool reads a manifest about the module in */usr/lib/xos/sharingservices* with information that will help the user to decide whether enabling or disabling the module.

The configuration tool is also used to configure the specific parameters of each service (e.g. the folder to share in data sharing). This task is delegated in a configuration module that must go with the sharing module.

3.2.2 Use case analysis: user shares a new resource

Figure 3.3 shows the communication between the different involved parts when a user shares a new resource (e.g. a file):

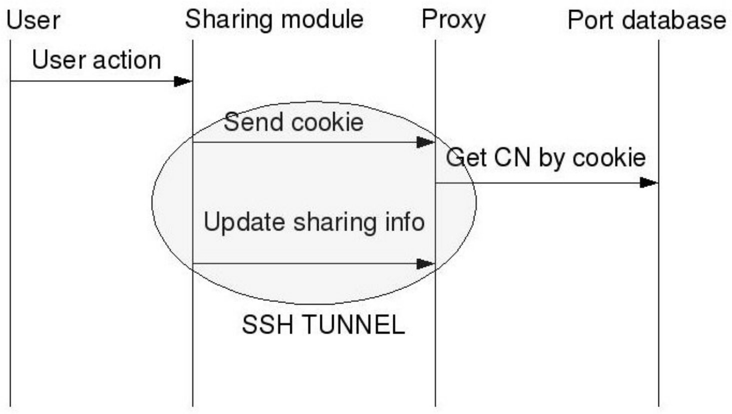


Figure 3.3: sharing a resource use case

1. The client publishes a new resource as available (e.g. a photograph is saved in shared directory). Another example is that a user authorizes now to share its network connection.
2. The sharing module reacts to the user event contacting the proxy. It uses a local port that is redirected by SSH-XOS to the proxy port. When the connection is established, the first data written is the cookie.
3. The proxy module reads the cookie and queries the port database to obtain the CN and machine-id.
4. The sharing module sends the updated information about resource sharing and the proxy uses it to update its information (e.g. service discovery).

3.2.3 Use case analysis: a remote client accesses a resource shared by the mobile device

Figure 3.4 shows the involved communication when a remote client requests access to a resource (e.g. a file).

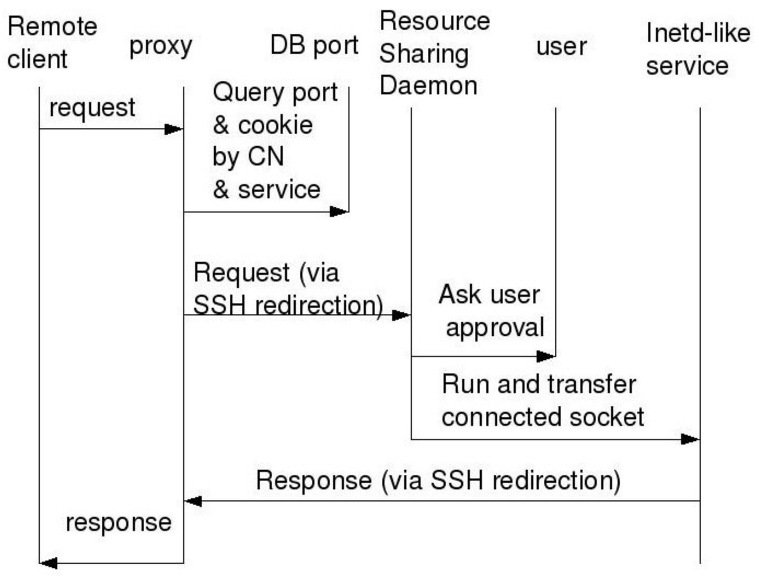

Remote client
proxy
DB port
Resource Sharing Daemon
user
Inetd-like service
request
Query port & cookie by CN & service
Request (via SSH redirection)
Ask user approval
Run and transfer connected socket
Response (via SSH redirection)
response

Figure 3.4: Accessing a resource use case

1. The remote client connects with the proxy and sends the request.
2. The proxy determines the CN and machine-id of the mobile device where a suitable resource that satisfies the request is available; then it queries the port database by CN, machine-id and service name (the proxy name) to obtain the port to connect with the service and the cookie to authenticate.
3. the proxy connects with the service using a local port, which is redirected by SSH-XOS to the mobile device service. the proxy writes first the cookie and then the request (of course the request sent by the proxy may be different that the request received from client).
4. The Resource Sharing Daemon, listening to the port, receives the proxy request; it only accepts it if the cookie is correct. Then, it checks if the service is enabled according to the configuration (e.g. verify battery level) and asks the user for approval (if no automatic confirmation is set). Then, it does a fork, invokes `close(0), close(1), dup(sock), dup(sock)` to redirect the socket to standard input/output and exec the *inetd-like* service.
5. The *inetd-like* service runs the needed code to provide the service, and sends the response back to the proxy.
6. The proxy processes the reply from service, and sends the response to the client.

It is possible that the RSD or *inetd-like* service rejects the request or even that the proxy cannot connect to the obtained proxy because the mobile device is not connected any longer. In that cases, the proxy will try with a different node which also fits the request (if any).

3.3 API definition

3.3.1 User interface library

RSD may need to interact with the user (e.g. to notify when a new request arrives). User interface code is isolated in library *libxos_sharing_gui.so* to make it more easy to port. This library implements the following interface:

```
void show_error_message(char *message);
void show_message(char *message);
void notify_new_request(char *message);
int confirm_new_request(char *message);
```

The third function is used to notify that a new request has arrived and the last one is used to inform that a new request has arrived and to wait for the user's confirmation: the call returns 0 if the user authorizes the request, -1 otherwise.

3.3.2 Module programmer's reference

The *inetd-like* service is implemented as a program that uses the standard input to read the data from proxy and the standard output to write data to proxy.

A sharing module is implemented as a shared object that implements this method:

```
void start(int port,char *cookie,int restarted);
```

where

- `Port` parameter is the TCP port to use to connect with proxy.
- `Cookie` parameter is the string to write to authenticate with proxy.
- `restarted` parameter is a flag that is 0 when module is started normally, 1 when module is started after it unexpectedly ends.

The module receives the `SIGTERM` signal if the user disables the module while it is running and `SIGHUP` signal if the configuration is changed and the module must be reloaded.

To terminate, the module uses *exit(0)*; otherwise the RDS will try to restart it automatically.

If the service needs a specific configuration, a second module must be provided to interact with the user and change the configuration file. The module name should be the same than the service module, but with ".gui" suffix (e.g. if the service module is file *gpssharing.so*, the configuration module file must be *gpssharing.gui.so*). The module must implement only this function:

```
int run_ui_config();
```

This function returns 0 if the configuration changed and the service module must reload the configuration or -1 if the configuration is finally unchanged.

3.3.2.1 Module manifest

Each service module provides a manifest file (a file with the name of the service and ".inf" as extension) with the following information:

- Description: the service description to be displayed to the user.
- Considerations: human-readable text with any security, privacy or performance degradation implications of the service.
- *euid-inetd*: the required EUID to run the *inetd-like* service.
- *egid-inetd*: the required EGID to run the *inetd-like* service.
- *euid-module*: the required EUID to run the sharing module.
- *euid-module*: the required EGID to run the sharing module.

3.4 Implementation

3.4.1 Data sharing foundation code

Data sharing service needs a mechanism to detect when the user wants to share a new file. A good point is that if user chooses to mark a folder as shared, then each new file added to the directory will be shared. This is similar to NFS and SMB behavior, but with XtreemFS advantages: file will be replicated automatically while it is accessed.

How to detect that a new file is added to the shared folder? Neither a periodic polling, nor a manual synchronization are good solutions. A better one is based on using *inotify* kernel support.

A library (*libxos_notifydirchange.so*) using *inotify* to detect when a file created in the shared folder is closed after a write operation is provided. This library changes the file-mode to read-only (currently replica mechanism of XtreemFS requires it) and notifies the event to layer G. It also detects when a file is moved or removed from the shared folder, to act consequently. The only function provided is:

```
void shared_directory_changed(const char *dir,
                void (*callback(char *,int)));
```

Parameter *dir* is the path of the shared folder to monitor. The second argument is a pointer function to the callback that is invoked when a change in the shared folder is detected. The first parameter of this callback function is the full path of the file and the second indicates if a new file is shared (value 1) or removed (value 0). This function never returns.

3.4.2 Network sharing foundation code

There is no specific foundation support in F layer for network sharing. Code will be implemented in layer G.

3.4.3 GPS sharing foundation code

Base functionality to module is provided through context-awareness API, which is described in chapter 4.

3.5 Configuration, installation and use

3.5.1 Resource Sharing User Configuration

The Resource Sharing Daemon reads a configuration file for each module in
/etc/xos/sharingservices/<servicename>.conf

Users has full control about what resource sharing services are running, using section *availability*. Users also have control about the approval or notification when a new request client arrives, using section *approval*.

Configuration file example:

```
[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;
```

Other service specific sections may appear as well.

The following algorithm is checked to decide if module is started:

1. If *enable* is false, the module is not started.
2. If *disable_if_not_wifi* is true and the network connectivity is not through WiFi (e.g. mobile is connected using UMTS or GPRS), the module is not started.

3. If *disable_if_battery_less_than* is defined and not 0, and battery level is under that percentage, the module is not started.
4. If *disable_if_status_is* is defined and user status matches one of the comma-separed values of the parameter, the module is not started; otherwise, the module is started.

The following algorithm is used when a new request for a sharing service arrives:

1. If *ask_request_approval* is true, a user's confirmation is required when a new request arrive; the request will be rejected if the user doesn't authorize it.
2. If *notify_new_request* is true and *ask_request_approval* is false, the user will be notified that a new request arrived, but manual approval won't be needed.

3.5.2 Installation and Usage

Resource sharing daemon has the following dependencies:

- *glib2*
- *libdbus* (optional, but required for receiving asynchronous notifications)
- *lidb*
- *SSH-XOS* (client only). This program requires OpenSSL >= 0.9.8g.

Actually, users don't need the installation instructions. The binaries will be installed automatically (together with the corresponding layer G software), like the other components of XtreemOS-MD.

Chapter 4

Context awareness

4.1 Introduction

First of all, let's consider the special characteristics that distinguish mobile devices from other devices (PC's, clusters) such as: limited battery autonomy, more probability of network connection losses and possibility of suffering changes on IP address. The minor computational power and minor storage capacity of mobile devices will not be considered as important differences in this section, because XtreemOS-MD allows compensating these disadvantages getting the computational power and the storage capability of the Grid through AEM and XtreemFS.

Also, the high possibility of location changes is another differential factor when thinking on mobile devices.

This particularities of mobile devices are taken into account by XtreemOS-MD to achieve a complete integration with the XtreemOS architecture.

Following these design premises, XtreemOS-MD will permit users to exploit all features and services provided by XtreemOS obviating the mobile device disadvantages (battery, connection,...) and taking advantage of mobility capability among others.

The main task of this Application Programming Interface (API) [11], is to permit to 3rd applications (Grid services) to decide to change the operation mode depending on the mobile device context, e.g.: *if the percentage of the battery is under 10%, the mobile device will automatically disconnect from the Grid.*

4.2 General Architecture

The API for Context Awareness has been designed and developed as a set of functions belonging to a static C library called `libCONTEXT.a`. This type of standard implementation allows an easy porting of the API implementation to different platforms: Maemo [6], Openmoko[7], etc.

This API offers context information of the mobile focusing on battery, network connection and GPS position. But the information offered could be extended with new features just by writing new functions and integrating them in the library provided. The XtreemOS-MD target devices offer the D-BUS [2] system, which is a message bus system used by applications and libraries to communicate with a different one, using a simple inter-process communication (IPC).

To get the context information, the implementation of the library uses the D-BUS interfaces to communicate with the Hardware Abstraction Layer [5] (HAL).

The architecture schema for API Context Awareness is shown in the figure 4.1.

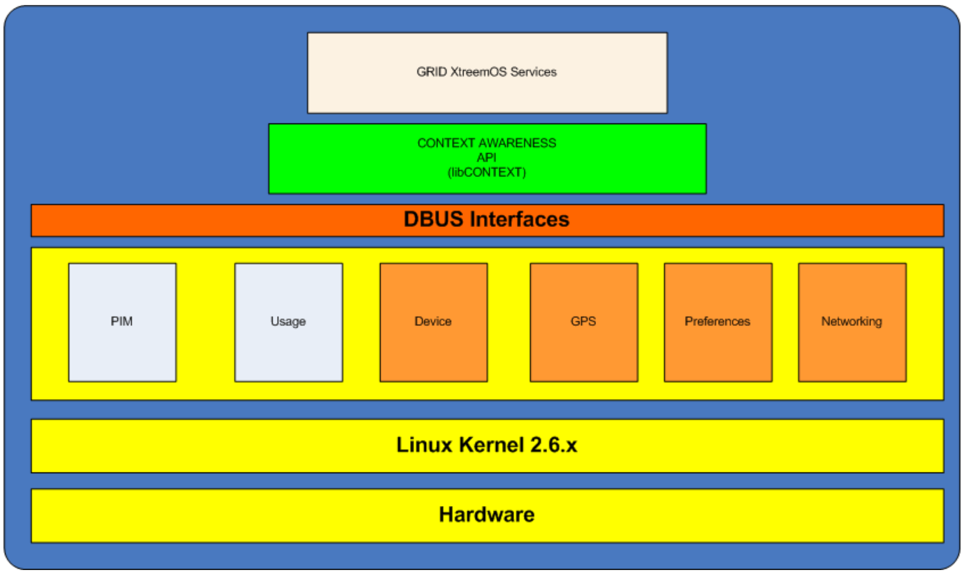


Figure 4.1: General architecture

The `libCONTEXT` library invokes the appropriated method of the D-BUS interface depending on the required type of information, and gets the information as a D-BUS message. When the `libCONTEXT` receives the message, it processes the message and formats its content following the Context Awareness API description. Finally, the library sends the required information to the application (Grid service).

4.3 API definition

This section contains a detailed description of the Context Awareness API including operations, parameters, operations results and status codes returned. All enabled operations of the Context Awareness API are summarized in the table 4.1.

Name	Parameters	Result
getLevelBattery	-	(string) batteryData
getNetworkAccess	-	(int) networkType
getGPSPosition	-	(string) positionData
startGPSTrace	-	(int) statusCode
startGPSTrace	period	(int) statusCode
stopGPSTrace	-	(int)statusCode
getGPSTrace	-	(string) collectionPositions
getProfile	-	(string) profile
getStatus	-	(int) mobile status

Table 4.1: Context Awareness API operations

4.3.1 getLevelBattery

Permits 3rd applications to get the current battery level of the device in percentage %.

Parameters

None

Result

The method returns a string with the value in percentage of the current battery level.

4.3.2 getNetworkAccess

Permits 3rd applications to get the type of network that is active and used by the device to connect to the network.

Parameters

None

Result

The method returns an integer number with a value that indicates the type of primary connection being used as specified in Table 4.2.

4.3.3 getGPSPosition

This method returns the current date and GPS position of the mobile device.

Parameters

connection	return value
no connection	0
WiFi	1
USB networking	2
3G	3

Table 4.2: return code for NetworkAccess

None

Result

The method provides the current date and GPS position in a string with a well-known format:

```
YYYY MM DD; HH MM SS;  HHH MM.M LongHemisphere; HHH MM.M
LatHemisphere
```

Where:

- `YYYY MM DD`

  Current date with four-digit numbers for years, followed by months and days.

- `HH MM SS`

  Current time with numbers for hours, followed by minutes and seconds.

- `HHH MM.M`

  HHH represents hours and MM.M represents minutes for latitude/longitude.

- `LongHemisphere`

  This word indicates if the longitude is measuring a distance east or west of the Prime Meridian. A value of "E" indicates east and "W" indicates west.

- `LatHemisphere`

  This word indicates if the latitude is measuring a distance north or south of the equator. A value of "N" indicates north and "S" indicates south.

4.3.4 startGPSTrace

When a 3rd application invokes this method, the Context Awareness library gets the GPS position each 2 minutes and saves it in a log file. The format used is similar to the one defined in the previous section.

Parameters

None

Result

This method does not return any direct result, but the GPS information is saved in a log file. It only returns a status code in order to inform whether the method worked properly (value 0) or generated an internal error (value -1).

4.3.5 startGPSTrace

When a 3rd application invokes this method, the Context Awareness library samples the GPS position according to the `period` parameter and saves the samples in a log file. The format used is similar to the one defined at the previous section.

Parameters

period: Value in minutes for the time period between two consecutive requests for GPS positions.

Result

This method does not return any direct result, but the GPS information is saved in a log file. It only returns a status code in order to inform whether the method worked properly (value 0) or generated an internal error (value -1).

4.3.6 stopGPSTrace

This method of the API must be invoked after invoking the `startGPSTrace` method. This method stops the process of saving GPS positions.

Parameters

None

Result

This method does not return any direct result. It only returns a status code in order to inform whether the method worked properly (value 0) or generated an internal error (value -1).

4.3.7 getGPSTrace

This function returns the sampled GPS positions of the device that were saved in a log files since the `startGPSTrace` was invoked until this time.

Parameters None

Result

The function provides the saved dates and GPS positions, ordered by date, in files of strings with a well-known format (for more information check: **??**)

4.3.8 getProfile

Permits 3rd applications to get the current status or active profile of the device. For example: `silent`, `vibrate`, `meeting`, `normal`, etc.

Parameters

None

Result

The function returns a string with the name of the current status or active profile of the device:

- Default
- Vibrate
- Ring
- Silent

4.3.9 getStatus

Permits 3rd applications to get the status (*offline* / *online*) of the device to decide if a Grid Service can be invoked or not.

Parameters

None

Result

The possible responses of this method is shown in the table 4.3

Status	return value
offline	0
online	1

Table 4.3: return code for Status

4.4 Implementation

This section includes a detailed description of the implementation in the different functions of the Context Awareness API for OpenMoko and Nokia N8x0.

4.4.1 Battery information

OpenMoko implementation

The information about the current battery level of the device is provided by the D-BUS interfaces with a specific method: `org.freesmartphone.Device.PowerControl.GetPower`, so the `libCONTEXT` invokes this method to get the info, and processes the returned data generating a string with the remaining percentage of battery. The process is shown in the figure 4.2

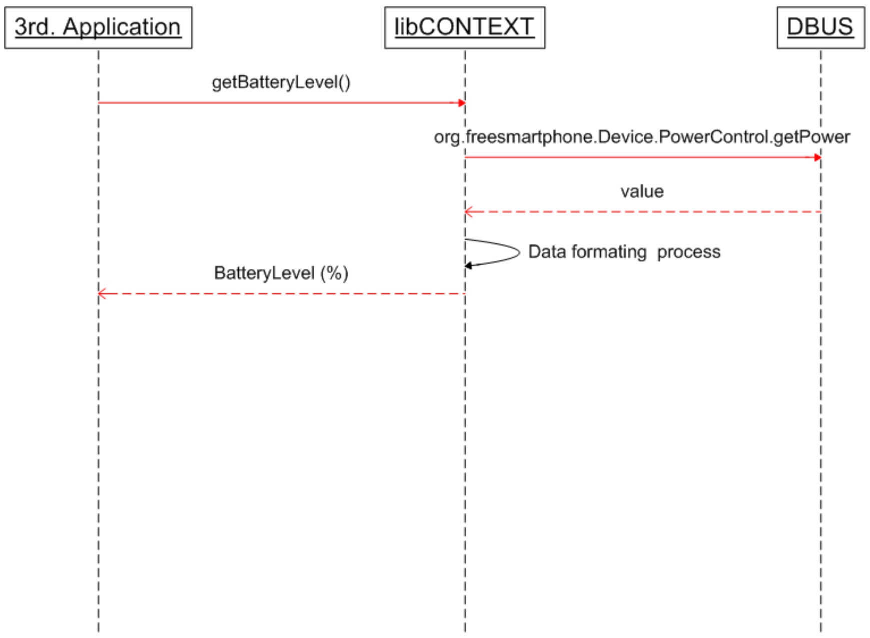


Figure 4.2: getBatteryLevel operation

Nokia implementation

The implementation for Maemo devices follows the same schema as OpenMoko, but invoking a different D-BUS method to get the battery information, in this case: `org.freedesktop.Hal.devices.bme.battery .charge_level.percentage`. Moreover, the processing and response formatting are different but the response of the `libCONTEXT` to the applications is similar to the Openmoko implementation.

4.4.2 Network information

OpenMoko implementation

The information about the type of network used by the device to connect to the network is not available using the OpenMoko D-BUS interface. So the `libCONTEXT` implementation must use another way to get the information: OpenMoko distribution keeps the status of their different network interfaces in a text file at `/sys/class/net/` directory with different values: "down", "up" and "dormant". The method, in charge of returning the type of connection, inspects the status value of the different networks and detects which one is used to connect to the network. If various interfaces are active, the method will decide among them, following this precedence order: Wifi, USB-networking, GPRS, and 3G.

Nokia implementation

The responsible for network connectivity in the Nokia N8x0 devices is the Internet Connectivity daemon (ICD2)[9]. To get network information for the Context Awareness API, `libCONTEXT` uses the internal D-Bus API `com.nokia .icd2.addrinfo_req` to connect to ICD2 and get the type of network connection used in the device.

4.4.3 Current GPS information

The current GPS position of the device is returned by getGPSPosition() function. The implementation of this function in `libCONTEXT` gets the GPS position using the utility `GPSpipe`, which belongs to `gps-utils` [4] package of the OpenMoko repository. GPS Information is stored in a binary file at `/etc` directory of the device, `GPSpipe` reads the date and position information provided and shows it via standard output (in this case the device display). The module that implements the Context Awareness API invokes the `GPSpipe` utility collecting the response with the date and position information. This information is formatted following the Context Awareness API rules and it is returned as a string of characters.

To avoid problems with the device, the first step for this process is to check the status (off/on) of the GPS device. If the GPS module is off-line, an error code will be returned by the function and the process finalizes immediately, aborting the GPS consult. The process is shown in the figure 4.3

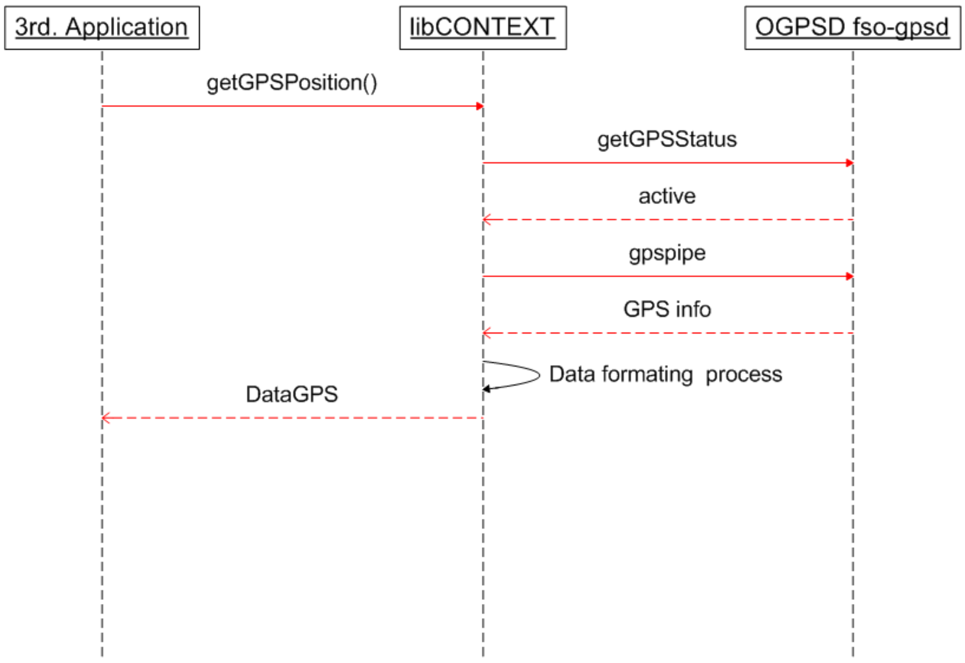


Figure 4.3: getGPSposition operation

Tracing GPS information

The Context Awareness API permits 3rd applications to keep the GPS date and position during a period of time and return this information to the 3rd application. When the `startGPSTrace` function is invoked, `libCONTEXT` implementation launches a program, in an independent thread, whose mission is to invoke the `GPSpipe` to get the GPS information and to keep the information in a regular file. The operation is repeated each period (by default 2 minutes). The time period is defined as a parameter of the function. While the API is keeping the GPS information, it can attend any other request given that the process is running in an independent thread.

When the `stopGPSTrace()` function is invoked the thread, which is keeping the GPS information, is stopped. And when the `getGPSTrace()` is invoked, `libCONTEXT` reads the file containing the GPS traces (stored with the correct format for this API) and returns them as a string of characters. The complete process of getting GPS traces is shown in the figure 4.4

Profiles information

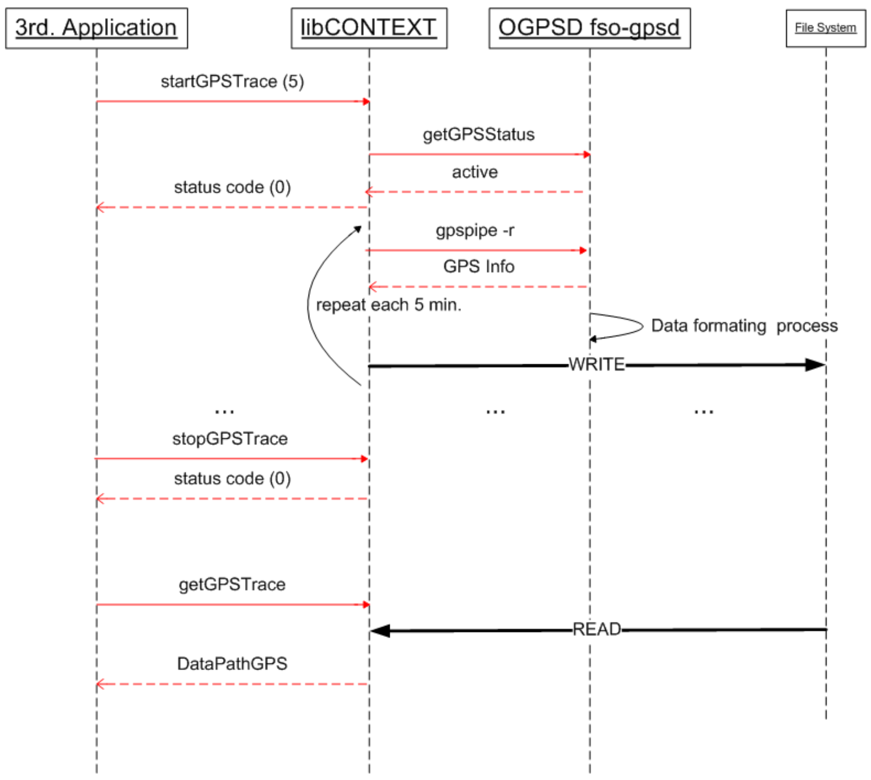


Figure 4.4: getGPSTraces operation

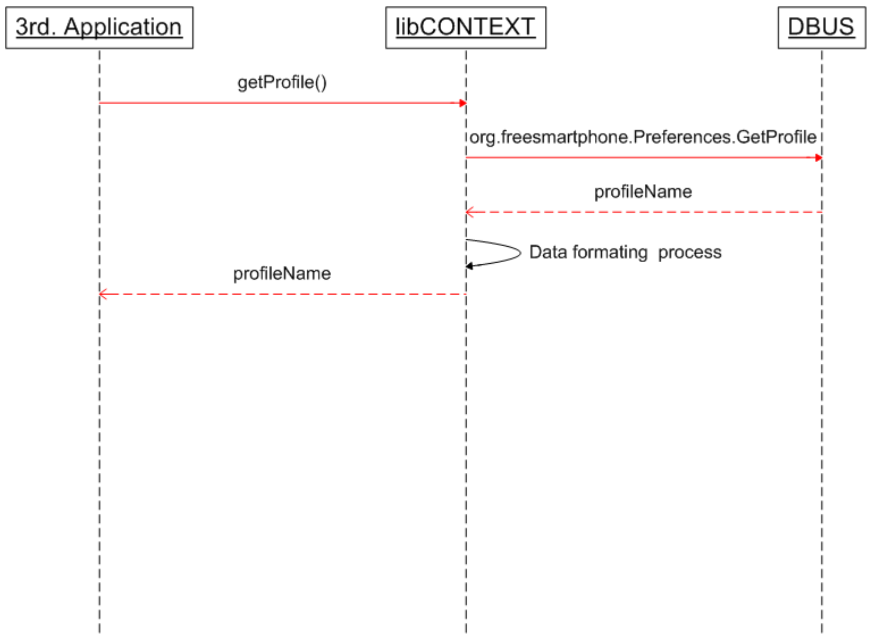


Figure 4.5: getProfile operation

The active profile is selected by the user device and the D-BUS interface of OpenMoko allows consulting it when the method `org.freesmartphone.device.PowerControl.getPower` is invoked. So, the API implementation invokes it and gets the name of the active profile as a string that is returned by the `getProfile()` function. The complete process of getting the profile name is shown in the figure 4.5

4.4.4 Mobile Status information

The method `getStatus`, in order to generate a response (*online* or *offline*), consults other Context Awareness API methods getting information of battery, network connection and profile (if available). With this information, it generates a response taking into consideration the rules defined in a configuration file with the following format:

```
#Context Awareness configuration File
batteryLevelMin 30

#network
wifi TRUE
usb FALSE
3G FALSE
bluetooth TRUE
```

Just the network connections marked as TRUE in the configuration file will be enabled for the Grid services, but only when the remaining battery percentage is higher than the `batteryLevelMin` parameter indicated. For example, in the example shown, the *online* mode will be assumed while the remaining battery is higher than 30% and there are WiFi or Bluetooth connectivity, but the MD will work in *offline* mode in the rest of cases.

4.5 Installation and Usage

4.5.1 Installation

The implementation of the Context Awareness API is created and delivered in the form of an static library (`libCONTEXT.a`) which will be offered with the XtreemOS-MD version as a `deb` or `ipk` package for Maemo/Openmoko respectively, although it is possible to install the API implementation manually, executing the following commands:

- Openmoko devices:

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

- Openmoko devices:

  ```
  #opkg install libcontext-1.0.ipk
  ```

This command installs and configures the library in the system.

4.5.2 Usage example

The following source code exemplifies the use of this Context Awareness API from a 3RD mobile device application:

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

int main(){

//battery level
char *statusBattery = NULL;
statusBattery = getLevelBattery();

//network info
char *statusNetwork = NULL;
statusNetwork = getNetworkAccess();
```

```
//GPS

//current GPS position
char *CurrentGPS = NULL;
CurrentGPS = getGPSPosition();

//Starting GPS traces service
startGPSTrace();

//to get several GPS traces
sleep(60);

//Stopping GPS traces service
stopGPSTrace();

//Getting GPS traces service
char *pathTraces = NULL;
pathTraces=getTraceGPS();

return 0;

}
```

Chapter 5

Installation, configuration and additional features

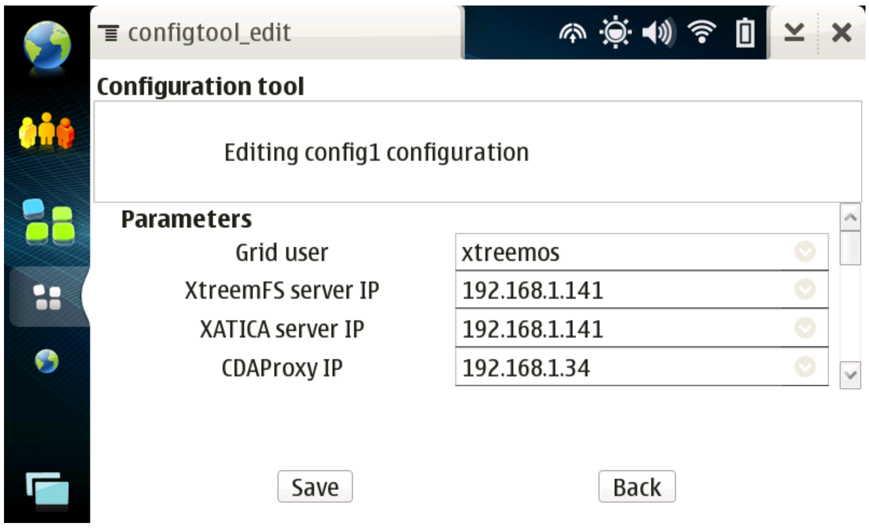
5.1 Installation enhancements

XtreemOS-MD basic version was already very easy to install and almost plug-and-play: *username* was the only parameter required during the installation. For this advanced version we have included some additional features, with the main objective of reaching a full unattended installation whenever possible:

- XtreemOS-MD advanced version allows a better granularity in installation, so that it's possible to install separately the AEM and XtreemFS modules if only one of them is needed.
- XtreemOS-MD could be installed as a dependency of an application. This way, when installing the JobMA application for example, XtreemOS-MD will be automatically installed in the system. As usual, the Grid's username will be requested during the installations, as it's needed by the XtreemOS-MD installer.

5.2 Configuration enhancements

XtreemOS-basic version was not only easy to install, but also it was fully configured automatically during the installation process. The administrators or service providers were in charge of setting the needed configuration in a single file included in the installation. XtreemOS-MD advanced version offers additionally the possibility of editing the configuration without gaining root privileges in the system. A very simple GUI is provided, allowing the users to modify, in a very simple and intuitive way, the default settings like Grid user, IP addresses of the XATICA and XtreemFS servers, etc, as shown in figure 5.1. Anyway, the GUI for configuration could also be limited (or even not provided within a concrete XtreemOS-MD distribution) in order to keep the classical behavior of the XtreemOS-MD basic version, which could be interesting to implement parental control, or let the service provider keep a full control over the configuration, or simply to avoid the possibility that malicious applications could the configuration.



Configuration tool

Editing config1 configuration

Parameters

Grid user	xtreemos
XtreemFS server IP	192.168.1.141
XATICA server IP	192.168.1.141
CDAProxy IP	192.168.1.34

Figure 5.1: Configuration tool: editing window

This graphical interface also allows the support of multiple configurations, so that the users can manage several different configuration and select the active one each time, as shown in figure 5.2. The implementation is done just by creating the default one as a virtual link of the configuration selected each time.

On the other hand, XtreemOS-MD manages automatically the credential acquisition process, so that users don't need to worry about getting manually the credential from the CDA. This behavior provided by the basic version is very convenient, but there are special circumstances under which some additional actions could be needed (like expired credentials, or not desired cached credential for example). This advanced version offers the following features to cope with this situations:

- Automatic renewal of expired credentials: `startxtreemos` sets a timeout in the *credstore* to disable the expired credentials. This implies a process of credential automatic renewal, taking into account that `startxtreemos` is invoked when the *credstore* is empty, obtaining a new credential from the CDA server.

- A new function `xos_purgecred` in `libxos_getcred` library that will force the removal of cached credentials and consequently the request of a new one. This could be useful when an application detects an invalid cached credential, like for example when the credential has been revoked. This new function implies in turn a new one (`xos_credagent_purgecred`)in

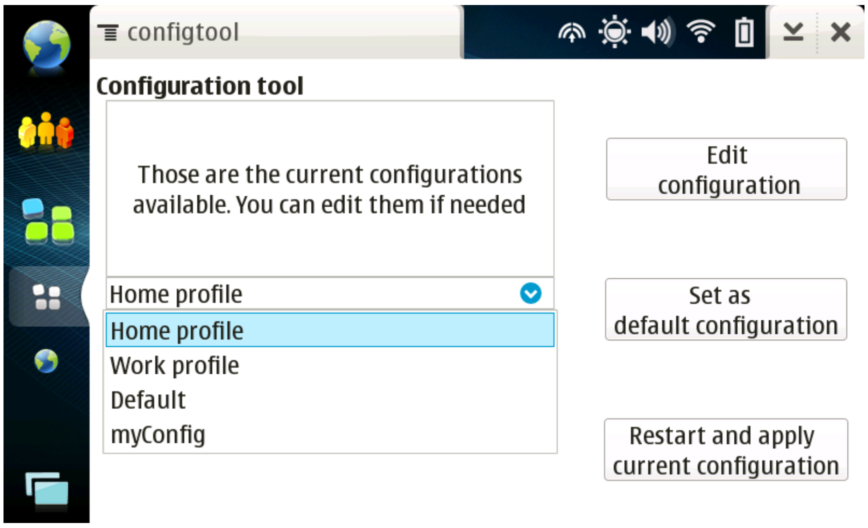

configtool
Configuration tool
Those are the current configurations available. You can edit them if needed
Home profile
Home profile
Work profile
Default
myConfig
Edit configuration
Set as default configuration
Restart and apply current configuration

Figure 5.2: Configuration tool: selecting configuration window

`libxos_credagent` library that will search the module associated to the credential, requesting its removal if cached. Also the `credagent` modules where it make sense, will implement a `credagent_purgecred` method to remove the cached credentials.

The API offered by `libxos_getcred` is:

```
void xos_purgecred(char *credencial);
```

but as *libxos_getcred* is setgid "credagent", like *startxtreemos* utility, only *libxos_credagent* has access to it, and then the API is offered by the latter:

```
void xos_credagent_purgecred(char *configuration);
```

Finally, while the basic version supported the mounting of just one volume at the same time, XtreemOS-MD advanced version support as well the configuration of multiple volumes to be mounted when starting the connection to the Grid. This has been implemented through a simple modification of `starxtreemos`, which is able to mount several volumes configured instead of just one.

5.3 On-demand starting enhancements

The `startxtreemos` console application was already provided by the XtreemOS-MD basic version, but in this advanced version, as commented in the previous section, it has also been included a graphical interface to select the configuration to apply. With this GUI it's also possible to launch `startxtreemos` graphically just by selecting one of the available configurations offered (see 5.2).

`Startxtreemos` was as well automatically launched when using applications linked with `libxos_getcred` library or in systems where the open calls were overwritten using the LD_PRELOAD mechanism. During the design phase of the advanced version, two new alternatives were identified:

- a first one based on a FUSE pseudo filesystem.
- a second alternative based on a wrapper to libxos_getcred through a D-BUS service, so that libxos_getcred would also be available for programs written in non-C languages as Python or Java.

It has been decided to implement the first one, and then in this advanced version we are providing four virtual files to read the credential, certificate, key and configuration names using a FUSE-based pseudo file-system that makes use of libxos_getcred.

5.4 Service resuming

Mobile devices environment might be less stable than the desktop one. For example, network connection may be lost for a time and it is possible that after recovery the device will get a different IP address. This is problematic with software that establishes a persistent connection, or when the other side of the communication does not expect an IP address change. Another problem with mobile devices is that, sometimes, the processes are killed if the device is getting out of memory.

Manual restarting of applications or services is not a good option for mobile device users. For this reason, some critical processes are launched with a wrapper, which guarantees that the process will be restarted if it ends unexpectedly.

The implemented wrapper is xos_launcher. This software does a fork to exec the program, while the main process will wait until the child ends. Then, xos_launcher will evaluate the exit status of the process: if it detects an abnormal termination, the launcher will relaunch the process and will wait again, but it the software ended with a successful state, the launcher will invoke exit(0).

Of course, xos_launcher does not relaunch repeatedly the process when it fails. The following algorithm is applied:

- if the process ends at first run immediately (the precise meaning of "immediately" is a configurable number of seconds), it is considered a configuration problem and the launcher ends with a error state
- if the process ends two times in the last minute, process relaunched is delayed a configurable time which is specified as a parameter in xos_launcher invocation.

5.5 Transparent access to Grid resources

XtreemOS provides transparent access to file resources through XtreemFS FUSE module, but applications like xsub or JobMA are needed to launch jobs in the

Grid. In order to gain transparency and improve the execution of jobs in the Grid, XtreemOS-MD advanced version offers the possibility of executing directly JSDL files to launch the jobs described inside those files, without the need of using explicitly any other application to open the JSDL files. This way, if a JSDL file containing the definition of a job is available, the user will just execute this file (for example with a classical double click or any other mechanism used by the terminal's file manager to execute a program) and the job will be launched to the Grid.

In order to implement this feature we have made use of the `binfmt_misc` mechanism provided by the Linux kernel (used for example to run directly the Java class files), so that a specific interpreter of the code is executed to "translate" it to something executable by the operating system. In this case, the interpreter consist of a program that reads the JSDL file and pass it as an argument to `xsub.sh` command, that is then executed to run the corresponding job in the Grid.

5.6 XtreemOS-MD ported to netbooks

The basic version of XtreemOS-MD was only available for Maemo devices and Angstrom [1] over QEMU. Even if the advanced version supports as well ARM architectures (quite common in mobile phones), it was already suggested to migrate XtreemeOS-MD to netbooks based on x86 architecture. During the design phase, and after a careful analysis (see [11] for the details), it was decided to select Ubuntu as the GNU/Linux distribution for netbooks. Given that porting only the F-layer now does not provide Grid services for end-users, we have decided to wait for the final implementation of the G-layer and make a full porting of XtreemOS-MD software then.

Chapter 6

Conclusions and Future work

All the features, processes and information contained in this document apply to the implementation of the advanced XtreemOS F-layer carried out for mobile devices, including VO support, context awareness API and resource sharing features, apart from the installation and configuration enhancements. This advanced version of XtreemOS-MD F-layer is now available for PDAs based on Maemo platform and also for smartphones based on OpenMoko platform.

It should be noted that there is an ongoing design and implementation of the Grid services for mobile devices (G-layer), which will make use of the context awareness and resource sharing APIs offered by layer F, and which will be ready before the end of the project. This means that the final packaging of the XtreemOS-MD advanced F-layer version, that includes all the features detailed in this deliverable, will be available along with the packaging of the advanced G-layer just after the corresponding integration with the future last release of XtreemOS. During this intermediate phase before the final release, and after the integration of F and G layers, a testing phase including bug fixing will be carried out.

Finally, as explained in section 5.6, the porting to Ubuntu distribution for netbooks will be delayed until the final release of the whole XtreemOS-MD software is available.

References

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

[2] DBUS Homepage.
http://www.freedesktop.org/wiki/Software/dbus/.

[3] P. Saint-Andre et al. Extensible messaging and presence protocol (xmpp): Core. Technical Report, RFC 3920, Internet Task Force, October 2004.

[4] Neo FreeRunner Homepage.
http://wiki.openmoko.org/wiki/Neo_FreeRunner_GPS/.

[5] Hardware Abstraction Layer Homepage.
http://www.freedesktop.org/wiki/Software/hal/.

[6] Maemo Homepage.
http://maemo.org/.

[7] OpenMoko Homepage.
http://wiki.openmoko.org/wiki/Main_Page/.

[8] UUID Wikipedia page.
http://en.wikipedia.org/wiki/UUID#Random_UUID_probability_of_duplicates/.

[9] Internet Connectivity daemon version 2 Homepage.
http://maemo.org/api_refs/5.0/beta/icd2/index.html/.

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

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

[12] XtreemOS Consortium. Fourth Specification, Design and Architecture of the Security and VO Management Services D3.5.13. Integrated Project, 2009.