Project no. IST-033576

XtreemOS

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

Third Prototype of XtreemOS Runtime Engine

D3.1.8

Due date of deliverable: May 31^{st}, 2009
Actual submission date: May 27^{th}, 2009

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

Type: Deliverable
WP number: WP3.1
Task number: T3.1.2

Responsible institution: VUA
Editor & and editor's address: Thilo Kielmann
Vrije Universiteit
Dept. of Computer Science
De Boelelaan 1083
1081HV Amsterdam
The Netherlands

Version 0.99 / Last edited by Thilo Kielmann / May 27, 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
0.9	26/05/09	Mathijs den Burger	VUA	complete draft version
0.99	27/05/09	Thilo Kielmann	VUA	complete version, for internal review

Reviewers:
Ramon Nou (UPC), Samuel Kortas (EDF)

Tasks related to this deliverable:

Task No.	Task description	Partners involved°
T3.1.2	A runtime engine for dynamic call dispatching	VUA*

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

Executive Summary

This document presents the third prototype implementation of a runtime engine for the XtreemOS API as specified in deliverable D3.1.5 [8]. (Some restrictions apply due to the limited availability of some XtreemOS features and services.) Compared to the second prototype implementation, as documented in D3.1.6 [7], the following functionality has been added:

- Job submission and resource reservation in the Java programming language.
- A separate implementation in the Python programming language.

We outline the design of the implementations, explain how to install and deploy them, and provide some code examples. In addition, we describe *Xterior*: an application on top of XOSAGA that provides a simple graphical user interface for managing files and jobs.

Contents

1 Introduction

For the successful adoption of the XtreemOS grid operating system, it is extremely important to provide a well-accepted API to its potential application programs. To accomplish this goal, we are following an iterative approach to specifying and implementing this API. In our previous deliverable, D3.1.1 [5], we have presented the *Simple API for Grid Applications (SAGA)* [2] as the first draft API for XtreemOS. Deliverables D3.1.2 [6] and D3.1.5 [8] added XtreemOS-specific extensions to SAGA, which together were named XOSAGA. These specifications are the basis of the implementations presented by this report.

In this document, we provide three prototype implementations of the XOSAGA API, written in the programming languages C++, Java, and Python. These implementations cover those parts of the XtreemOS-specific functionality described in D3.1.2 and D3.1.5 for which stable implementations have been available so far and XOSAGA bindings have been developed. In particular, the current implementations support:

- The XtreemFS file system (in C++, Java, and Python)
- The XtreemOS security context (in C++, Java, and Python)
- Job submission on top of AEM (in Java and Python)
- Resource reservation on top of AEM (in Java)

The XOSAGA 'sharing' package described in D3.1.5 is *not* covered by the currently existing implementations. It will be an essential part of the fourth prototype implementation (D3.1.10, due month 46).

The functionality available in this third prototype implementation is summarized in Table 1. New functionality (compared to the previous prototype implementation) is shown in bold face.

In the remainder of this document, we will outline the general design of all three XOSAGA implementations, describe their integration with the Application Execution Management system in XtreemOS, and provide information for downloading, installing and using the software. For providing a self-contained report, we include some description of general architecture and of the C++ and Java implementations that were also part of D3.1.5. We augment it by the respective information about the new, Python-based implementation, the XtreemOS job adaptor, the resource reservation API, and the GUI application Xterior.

SAGA package	**C++**	**Java**	**Python**
Physical Files	local, XtreemFS, **SSH**, **HDFS**, **KFS**, Globus GT4 GridFTP	local, XtreemFS, SSH, FTP, SFTP, Globus GT2 & GT4.0 WS GridFTP	**local**, **XtreemFS**, **SSH**, **FTP**, **SFTP**, **Globus GT2 & GT 4.0** **WS GridFTP**
Replicated Files	Globus GT4 RLS	via physical files	**via physical files**
Job submission	local, GridSAM, Condor, Globus GT4 GRAM2	local, **AEM**, GridSAM, gLite, SGE, LocalQ, **Unicore**, **Koala**, Globus GT2 & GT4.0 WS GRAM	**local**, **GridSAM**, **gLite**, **SGE**, **LocalQ**, **Unicore**, **Koala**, **Globus GT2 & GT4.0** **WS GRAM**
Streams		TCP sockets	**TCP sockets**
RPC	XMLRPC	XMLRPC	**XMLRPC**

XOSAGA package	**C++**	**Java**	**Python**
Resource reservation		**AEM**	

Table 1: SAGA and XOSAGA functionality provided by the third prototype implementation

2 General Architecture

In its general architecture, our SAGA implementations follow the lessons we have learned with the SAGA predecessor GAT [1]: a small dynamic *engine* provides dynamic call switching of SAGA API calls to middleware bindings (*adaptors*) which are dynamically loaded on demand, and bound at runtime (*late binding*). The relation between these components are illustrated in Figure 1.

Unlike the GAT, SAGA provides an extensible API framework, consisting of a look-and-feel part, and an extensible set of functional packages. The look-and-feel consists of the following parts:

Base object which provides all SAGA objects with a unique identifier, and associates *session* and shallow-copy semantics.

Session object that isolates independent sets of SAGA objects from each other.

Context object that contains security information for Grid middleware. A session can contain multiple contexts. XtreemOS certificates are managed with an XtreemOS context object.

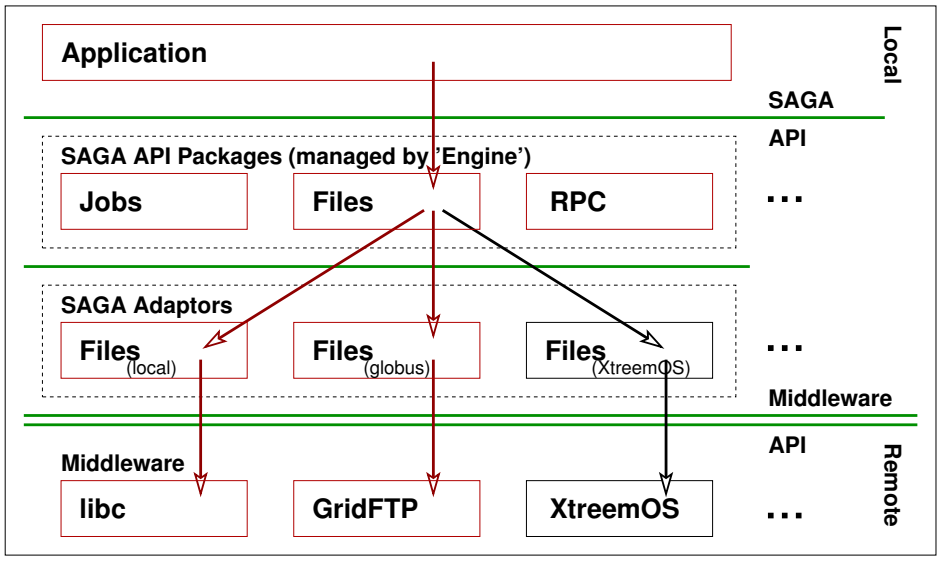

Figure 1: General architecture: a lightweight *engine* dispatches SAGA calls to dynamically loaded middleware *adaptors*.

URL object to uniformly name remote jobs, files, services etc.

I/O buffer providing unified access to data in memory, either managed by the application or by the SAGA engine.

Error handling using exceptions.

Monitoring of certain SAGA objects using callback functions.

Task model which allows both synchronous and asynchrous execution of methods and object creation.

Permission model lets an application allow or deny certain operations on SAGA objects.

Orthogonal to the look-and-feel are the functional packages, providing the actual functionality of the grid. Currently, the set of standardized functional packages consists of:

Job Management to run and control jobs.

Name Spaces to manipulate entries in an abstract hierarchical name space.

File Management to access files.

Replica Management to manage replicated files.

Streams for network communication.

Remote Procedure Calls for inter-process communication.

The complete specification of the language-independent SAGA API can be found in [2].

3 The SAGA C++ Engine

The C++ implementation of SAGA uses the PIMPL mechanism (**p**rivate **impl**ementation) to simplify the internal state management of SAGA objects and to resolve the lifetime dependencies between SAGA objects, SAGA sessions, and adaptors [4]. The SAGA object does not maintain any state itself, but is merely a facade maintaining a private, shared pointer to the implementation of the (stateful) SAGA object, and all method invocations are simply forwarded to that implementation instance. On copies, a new facade instance is created which maintains another shared copy to the same implementation instance, using, by definition, shallow copy semantics, as the stateful implementation is not copied at all. Also, depending objects and task instances (which represent asynchronous operations) maintain additional shared pointers to the implementation instance and are thus extending the lifetime of that instance: only when all shared pointer copies are finally freed (i.e., when all depending objects are deleted and all asynchronous operations are completed) is the stateful implementation deleted.

At the same time, the engine provides the complete SAGA task model, e.g., it implements all SAGA operations asynchronously, even if that is not explicitly supported by the backend services. Both the central call routing and the central management of asynchronous operations, allow for smart runtime optimizations of the remote method invocations [3], which are, for example, exploited for bulk optimizations.

We also use the shared pointer abstraction for the internal lifetime management of the adaptor instances: multiple of those instances can co-exist and provide the implementation (i.e., middleware binding) of the SAGA object implementations.

The SAGA C++ Adaptors

Along with the SAGA C++ engine, which is providing the SAGA API itself, several middleware bindings (i.e. SAGA adaptors) have been implemented. Firstly, local adaptors have been provided which interface to the local operating system

(in the case of XtreemOS: Linux) and provide the SAGA functionality on the local host machine, as well as LinuxSSI clusters. Besides, the local adaptor set is also important for (a) development and debugging purposes, and (b) as reference for other, non-local adaptors.

In addition to the local adaptors, the SAGA C++ implementation includes various adaptors to access remote services. Table 1 provides an overview of the accessible services, which include the XtreemFS file system. XtreemOS user certificates are also automatically recognized and used by the XtreemFS adaptor to securely access remote volumes. Interfaces to other XtreemOS services are currently being designed and implemented.

3.1 Installation and deployment

In the XtreemOS distribution, the C++ implementation of SAGA is available as several rpm packages:

libsaga-devel contains everything to develop SAGA applications.

libsaga contains all libraries to run SAGA applications.

saga contains some example SAGA programs and environment settings.

xosaga contains XtreemOS-specific additions to SAGA.

Installing C++ SAGA can be done using urpmi:

```
$> urpmi xosaga
```

You will be given a choice between the '`libsaga`' and the '`libsaga-devel`' package. Choose the '`libsaga`' package if you only want to *run* XOSAGA applications (e.g. on an XtreemOS node). Choose the '`libsaga-devel`' package if you also want to develop XOSAGA applications on your machine.

The development source tree of the C++ implementation of XOSAGA can be found in the Subversion repository of XtreemOS located at INRIA, France:

```
svn+ssh://scm.gforge.inria.fr
   /svn/xtreemos/grid/xosaga/cpp/trunk
```

XOSAGA includes the latest SAGA 1.2.1 release, which can also be downloaded separately from the SAGA website. For historical reasons, the SAGA website is located at `http://saga.cct.lsu.edu`.

The C++ implementation of SAGA depends on the free Boost C++ libraries, version 1.33.1 or higher. They can usually be found in the package repository of your Linux distribution. Alternatively, they can be downloaded from the Boost website: `http://www.boost.org`.

3.2 API documentation

API documentation of the C++ implementation is available in three different formats. Firstly, the OGF SAGA API standard document [2] is, naturally, a comprehensive documentation source for the SAGA API. Secondly, a number of tutorials are included in the released code package. And finally, a detailed API documentation is generated by doxygen. It is available from:

`http://saga.cct.lsu.edu/cpp/apidoc/`

4 The SAGA Java Engine

The SAGA Java engine implements release 1.0 of the Java SAGA language binding. Like its C++ counterpart, the engine takes care of dynamically selecting and loading SAGA adaptors, contains base classes for adaptors, and default implementations for SAGA's attributes, tasks, monitorable, buffer, session, and context.

4.1 Java language binding of SAGA

The Java language binding defines the precise syntax and semantics of the SAGA functionality in the Java language. The language binding can be seen as a contract between applications and SAGA implementors: both parties can safely assume that exactly the classes and interfaces described in the language binding will be either provided or requested for. SAGA's language binding for Java is provided in the form of directly usable files that contain a set of *interfaces*. A SAGA implementation has to provide classes that implement these interfaces. For allowing applications to create SAGA objects, the interfaces are accompanied by factory classes. The factory objects for each SAGA package are created by a `SagaFactory` meta class. This setup requires a bootstrap meachanism to locate the implementation of the `SagaFactory` class. A user is therefore obliged to set the system property `saga.factory` to the class name of an implementation-specific `SagaFactory` object. In our SAGA Java engine, this property must be set to:

```
saga.factory=org.ogf.saga.impl.bootstrap.MetaFactory
```

The Java language binding of SAGA can downloaded from Sourceforge:

```
http://sourceforge.net/projects/saga/
```

The language binding is also available online in the form of Javadoc:

```
http://saga.cct.lsu.edu/java/apidoc/
```

4.2 XOSAGA additions

The XtreemOS-specific functionality added to SAGA is bootstrapped using a `XosagaFactory` meta class, which extends `SagaFactory`. XOSAGA users should therefore use an XOSAGA-specific factory instead of the default class `org.ogf.saga.impl.bootstrap.MetaFactory`, namely:

```
saga.factory=eu.xtreemos.xosaga.bootstrap.MetaFactory
```

This class extends `org.ogf.saga.impl.bootstrap.MetaFactory`, so all SAGA functionality is also available via the XOSAGA factory. Currently, XOSAGA adds support for resource reservation in XtreemOS. Other XtreemOS-specific functionality is currently being developed.

4.3 Configuration

The scripts that are included in the SAGA Java engine use the environment variable `JAVA_SAGA_LOCATION`, which should point to the root directory of the SAGA Java installation. The engine recognizes a number of system properties, which are either provided to the engine by means of a `saga.properties` file, or by means of the `-D` option of Java. The `saga.properties` file is searched for in the classpath and in the current directory. If both are present, values specified in the file in the current directory override values specified in the file in the classpath. Values specified on the command line override both.

The property `saga.adaptor.path` tells the engine where to find the adaptors. Its default value is `JAVA_SAGA_LOCATION/lib/adaptors`. This property is interpreted as a path, which may either be specified in the "unix" way (with '/' and ':'), or in the system-dependent way.

All properties with names ending in `.path` are subjected to the following replacements: all occurrences of the string `SAGA_LOCATION` are replaced with the value of the `JAVA_SAGA_LOCATION` environment variable, all occurrences of

'/' are replaced with the system-dependent separator character, and all occurences of ':' are replaced with the system-dependent path separator character. This allows for a system-independent way of specifying paths in a `saga.properties` file.

All XtreemOS-specific adaptors (i.e. to access XtreemFS, XtreemOS certificates, and AEM) are configured via an `xosaga.properties` file. Properties are load from the following locations, in this order:

1. The system-wide property file `/etc/xos/config/xosaga.properties`
2. The per-user property file `/.xos/xosaga.properties`
3. The file `xosaga.properties` the current working directory
4. Properties specified at the command line

Properties loaded later replace properties loaded earlier.

4.4 The SAGA Java Adaptors

During startup, the engine examines which adaptors are available, and loads these. When a SAGA object is created, a corresponding set of adaptors is instantiated. An invocation of a method on a SAGA object is dynamically dispatched to one or more of adaptors, until one succeeds or all adaptors fail.

The adaptors implement one or more specific Service Provider Interfaces (SPI), which correspond to particular interfaces of the SAGA language binding for Java. Some adaptors only implement one SPI (e.g. the Gridsam adaptor, which only implements the SPI for SAGA's job package). Other adaptors implement multiple SPIs, as they are able to provide more functionality. An important adaptor of the latter catagory is the one built on top of the JavaGAT. This adaptor implements almost all SPIs and acts like a "swiss army knife". Table 1 lists all available functionality provided by all adaptors.

All adaptors that ship with the SAGA Java engine are located in the subdirectory `lib/adaptors`. Each adaptor has its own subdirectory, named `<adaptorname>Adaptor`, in which a jar-file `<adaptorname>Adaptor.jar` exists, and which also contains all supporting jar-files. The manifest of `<adaptorname>Adaptor.jar` specifies which adaptors actually are implemented by this jar-file. For instance, the manifest of `XtreemFsAdaptor.jar` specifies:

```
FileSpi-class:  org.ogf.saga.adaptors.xtreemfs.FileAdaptor
```

which indicates that it contains a class 'org.ogf.saga.adaptors.xtreemfs.FileAdaptor' that implements the `FileService` Service Provicer Interface. By default, the

SAGA engine tries all adaptors that it can find on the list specified by the property `saga.adaptor.path`. It is, however, possible to select a specific adaptor, or to not select a specific adaptor by specifying certain properties in a file named `saga.properties` or on the command line. Some examples are:

`StreamService.adaptor.name=socket,javagat` will load both the socket and the JavaGAT adaptor for the StreamService SPI, but no others. Also, the adaptors will be tried in the specified order.

`StreamService.adaptor.name=!socket` will load all StreamService adaptors, except for the socket adaptor.

4.5 Installation and documentation

The development source tree of the XOSAGA Java implementation can be found in the Subversion repository of XtreemOS located at INRIA, France:

```
svn+ssh://scm.gforge.inria.fr
   /svn/xtreemos/grid/xosaga/java/trunk
```

This tree includes the latest SAGA Java release, which can also be downloaded separately from Sourceforge:

```
http://sourceforge.net/projects/saga/
```

The release contains a short user guide that describes the steps required to compile and run a Java SAGA application. The complete specification of the Java-to-SAGA language binding is the subject of ongoing work.

5 The SAGA Python engine

The SAGA Python engine implements a preliminary version of the Python SAGA language binding. This language binding is under development, and will be a candidate for future standardization.

Unlike its C++ and Java counterparts, the Python SAGA engine does not use Python-specific SAGA adaptors to implement the functionality of the various SAGA packages. Instead, it acts as a wrapper on top of a Java SAGA implementation. All SAGA functionality is therefore available via Python-specific constructs. Internally, the Python SAGA engine uses the Java SAGA engine to implement all functionality.

The Python SAGA engine relies on Jython[1], a Python interpreter written in Java. Hence, we nicknamed this engine *JySaga*. The current stable release of Jython is 2.2.1, which implements most of the features of CPython 2.2. Jython allows a Python application to use Java objects and methods, which makes it relatively easy to implement a Python SAGA engine on top of the Java SAGA engine.

5.1 Python language binding of SAGA

The Python language binding defines the precise syntax and semantics of the SAGA functionality in the Python language. The language binding can be seen as a contract between applications and SAGA implementors: both parties can safely assume that exactly the classes and interfaces described in the language binding will be either provided or requested for. SAGA's language binding for Python is provided in the form of a set of files that contain a Python module for each of the SAGA packages. Each module defines a number of *skeleton objects* that define the method signatures of all SAGA objects, but do not implement any functionality. A Python SAGA engine programmer can use these skeleton objects as a basis for implementing a functional SAGA engine in Python. To ensure that the engine implements all SAGA functionlity correctly, the language binding is accompanied by a set of *unit tests*. These unit tests provide a run-time check of a Python SAGA implementation, and check the precise syntax and semantics of all SAGA objects. The Python language binding of SAGA and the engine on top of a Java SAGA engine (JySaga) can be retrieved from Subversion:

```
svn+ssh://scm.gforge.inria.fr
     /svn/xtreemos/grid/xosaga/python/trunk/
```

The language binding is documented in the 'doc' directory. The JySaga code is located in the 'saga' directory. The unit tests can be found in the 'test' directory. The 'bin' directory contains scripts to easily start JySaga and run the unit tests. Finally, the 'examples' directory contains a few example programs, which are described in more detail in Section 5.4).

5.2 Installation

The Python SAGA engine JySaga requires:

- Jython >= 2.2.1
- Java >= 1.6

[1] http://www.jython.org

- The Java SAGA engine

The `jysaga` script starts the Jython interpreter with the right classpath and Java properties set to load the Python XOSAGA classes and modules. The script assumes that Jython is started via `/usr/bin/jython`.

The Java SAGA implementation is assumed to be located in the directory `/usr/share/java/saga`. This directory must contain a subdirectory 'lib' that corresponds to the 'lib' directory created by the `build.xml` script of the Java SAGA distribution. Another location can be specified by setting the environment variable `JAVA_SAGA_LOCATION`.

JySaga's `.py` files are assumed to live in a subdirectory 'saga' in the default Jython path (usually `/usr/share/jython/Lib`). Another location can be specified by setting the environment variable `JYSAGA_LOCATION`.

5.3 Unit tests

The 'test' directory contains a set of generic unit tests for a Python SAGA implementation. For JySaga, the whole test suite can be easily run using the shell script `bin/jysaga-test`. The script also accepts a single unittest file name as parameter, in which case only that test is run.

5.4 Code Examples

The subdirectory 'examples' contains several example Python applications that use SAGA. We will describe these examples in the following sections.

5.4.1 Listing a directory

The program `examples/listdir.py` shows the contents of a (possibly remote) directory. Figure 2 shows its source code. The program accepts a URL as parameter that indicates the directory to show. If the URL scheme is 'ftp', we add a SAGA context for anonymous FTP access to the default SAGA session. We then create an `NSDirectory` object with the given URL, and print the names of all files in that directory.

An example run of `listdir.py` could be:

```
$ bin/jysaga examples/listdir.py ftp://ftp.xs4all.nl/
dev
pub
shlib
welcome.msg
```

```
import sys

from saga import context, error, namespace, session, url

if __name__ == "__main__":

  if len(sys.argv) < 2:
    print "usage: jysaga " + sys.argv[0] + " <url>"
    sys.exit(1)

  try:
    dir_url = url.URL(sys.argv[1]);

    # check if we need an anonymous FTP context
    if dir_url.scheme == "ftp":
      default_session = session.Session()
      ftp_context = context.Context("ftp")
      default_session.add_context(ftp_context)

    d = namespace.NSDirectory(dir_url)

    for file_url in d.list():
      print file_url.string

  except error.SagaException, e:
    print e.message
```

Figure 2: Source code of `examples/listdir.py`

Possible others URLs are:

- Local files: `file://localhost/dir/`
- XtreemFS: `xtreemfs://volume@host.example.com/dir/`
- FTP files: `ftp://host.example.com/dir/`
- Secure Shell: `ssh://host.example.com/dir/`
- GridFTP: `gridftp://host.example.com/dir/`
- Try all adaptors: `any://host.example.com/dir/`

5.4.2 Showing file contents

The program `example/cat.py` shows the contents of a (possibly remote) file. Its source code is show in Figure 3.

```
import sys

from saga import buffer, context, error, file, session, url

if __name__ == "__main__":

  if len(sys.argv) < 2:
    print "usage: jysaga " + sys.argv[0] + " <url>"
    sys.exit(1)

  try:
    u = url.URL(sys.argv[1]);

    # check if we need an anonymous FTP context
    if u.scheme == "ftp":
      default_session = session.Session()
      ftp_context = context.Context("ftp")
      default_session.add_context(ftp_context)

    f = file.File(u)

    bufsize = 2048
    buf = buffer.Buffer(bufsize);

    done = False
    while not done:
      len = f.read(bufsize, buf);
      if len > 0:
        sys.stdout.write(buf.data[:len].tostring())
      else:
        done = True

    f.close()

  except error.SagaException, e:
    print e.message
```

Figure 3: Source code of `examples/cat.py`

The first part of the program is similar to `listdir.py`. It accepts a URL as parameter that indicates the files whose context will be printed. After adding an anonymous FTP context (if needed), we create a SAGA `File` object with the

given URL. The file is then read in chunks of 2048 bytes that are interpreted as ASCI and printed to stdout.

An example run of `cat.py` could be:

```
$ bin/jysaga examples/cat.py ftp://ftp.kernel.org/welcome.msg
                Welcome to the

            LINUX KERNEL ARCHIVES
                ftp.kernel.org

(remaining output omitted)
```

5.4.3 Running a job

The program `examples/runjob.py` executes a (possibly remote) job. Figure 4 shows the source code of this program. It needs at least three parameters: the URL of the job service to contact, the URL of the file in which the output of the command is stored, and the command to execute. Any additional parameters are provided to the command.

The program first creates a SAGA `JobService` object with the given URL. It then creates a simple job description using the other parameters. The job description is given to the job service to create a SAGA `Job` object. Two callback functions are then registered with this job object. These callbacks are triggered when the job's status and detailed status changes. The callbacks simply print the current job status. After the job is run, the contents of the output file is read and printed.

An example run of `runjob.py` (assuming SSH access to 'host.example.com') could be:

```
$ bin/jysaga examples/runjob.py ssh://host.example.com \
  file://localhost/tmp/out hostname -f
job.state: RUNNING
job.state_detail: JavaGAT.PRE_STAGING
job.state_detail: JavaGAT.RUNNING
job.state_detail: JavaGAT.POST_STAGING
job.state_detail: JavaGAT.STOPPED
job.state: DONE
Output:
host.example.com
```

Other possible job service URLs are:

- Local jobs: `local://localhost`

```
import sys

from saga import context, error, file, job
from saga import monitoring, session, url

class JobCallback(monitoring.Callback):

  def cb(self, monitorable, metric, context):
    print metric.Name + ": " + metric.Value
    return True

if __name__ == "__main__":
  if len(sys.argv) < 3:
    print "usage: jysaga " + sys.argv[0] + " <jobservice_url> ",
    print "<output_file> <command> [arguments...]"
    sys.exit(1)

  try:
    job_service_url = url.URL(sys.argv[1]);
    output = sys.argv[2]
    cmd = sys.argv[3]
    args = sys.argv[4:]

    js = job.JobService(job_service_url)

    jd = job.JobDescription()
    jd.Executable = cmd
    jd.Arguments = args
    jd.Output = output

    j = js.create_job(jd)

    c = JobCallback()
    j.add_callback(job.Job.STATE, c)
    j.add_callback(job.Job.STATEDETAIL, c)

    j.run()
    j.wait();

    print "Output:"
    output_url = url.URL(output)
    output_file = file.File(output_url);
    print output_file.read(),
    output_file.close()

  except error.SagaException, e:
    print e.message
```

Figure 4: Source code of `examples/runjob.py`

- XtreemOS jobs: `xos://host.example.com`
- Gridsam jobs: `gridsam://host.example.com`
- gLite jobs: `glite://host.example.com`
- Globus jobs: `globus://host.example.com`

- Try all adaptors: `any://host.example.com`

6 XtreemOS Job Adaptor

The XtreemOS job adaptor lets SAGA applications submit jobs on an XtreemOS cluster by using the familiar SAGA job API. The adaptor uses AEM through the XATI API package.

URLs recognized by the job adaptor must have a scheme 'xos' or 'any'. A URL without a hostname (i.e. `xos://`) will signal the adaptor to use the default XATI settings in `~/.xos/XATIConfig.conf` to connect to an XOSD. When the URL includes a hostname and port (e.g. `xos://host.example.com:12345`), the job adaptor will reconfigure XATI to connect to the XOSD running at the given hostname and port. When no port is specified, the default port number 60000 is used.

The user certificate used to talk to XATI is configured through the SAGA security context mechanism. If a context of type 'xtreemos' exists which contains a Context.USERCERT attribute, the XtreemOS job adaptor tries to load and use the certificate at the location given by the attribute value. If the certificate is encrypted, the Context.USERPASS attribute should also contain the password for decryption. If no Context.USERCERT attribute was found, the default XATI security settings found in `~/.xos/XATIConfig.conf` are used.

Currently, the XtreemOS job adaptor is only available in Java and Python. Support for XtreemOS job submission in C++ is currently being developed.

7 Resource Reservation

XOSAGA add an extra 'resource' package to allow resource discovery and reservation management. It also allows more job submission use cases by using the extended XOSAGA job API. A language-independent specification of the resource package and extended job API can be found in deliverable D3.1.2 [6].

Current, resource reservation is only available in the SAGA Java engine. The XOSAGA API classes for resource reservation can be found in the Java packages `eu.xtreemos.xosaga.resource` and `eu.xtreemos.xosaga.job`.

Users can access the extra XOSAGA functionality by using the SagaFactory `eu.xtreemos.xosaga.bootstrap.MetaFactory`. This class extends the Java SAGA class `org.ogf.saga.impl.bootstrap.MetaFactory`, and adds the method `createResourceFactory()` to obtain a `ResourceFactory` object. Figure 5 shows the methods that the resource factory contains.

```
/**
 * Creates a resource service object which connects to a
 * resource management system.
 *
 * @param session
 *      SAGA session to associate with the service.
 * @param rm
 *      Contact url for the resource manager. If blank or null,  the implementa-
 *      tion must either perform resource discovery, fall back to a fixed value,
 *      or find a rm contact in any other way.  If that is not possible, a
 *      BadParameter exception MUST be thrown, and MUST indicate that an rm
 *      contact string is needed.
 * @return
 *      the created ResourceService object.
 * @throws NotImplementedException
 * @throws IncorrectURLException
 * @throws PermissionDeniedException
 * @throws AuthorizationFailedException
 * @throws AuthenticationFailedException
 * @throws TimeoutException
 * @throws NoSuccessException
 */
public static ResourceService createResourceService(Session s, URL rm)
throws NotImplementedException, IncorrectURLException, PermissionDeniedException,
    AuthorizationFailedException, AuthenticationFailedException,
    TimeoutException, NoSuccessException;

/**
 * Creates a new ResourceDescription to be used with the resource reservation
 * service.
 *
 * @return the resource description.
 */
public static ResourceDescription createResourceDescription();
```

Figure 5: Methods in the `ResourceFactory` class

The `ResourceService` object represents a resource management system. It allows for resource discovery and reservation. The `ResourceDescription` object adds a number of SAGA attributes to describe resources.

8 Xterior

Xterior is an GUI application for easily managing files and jobs in a virtual organization. Xterior is implemented in Java, and uses the Java SAGA engine underneath to access XtreemOS functionality. The first release of Xterior (0.2.0) only supports file management.

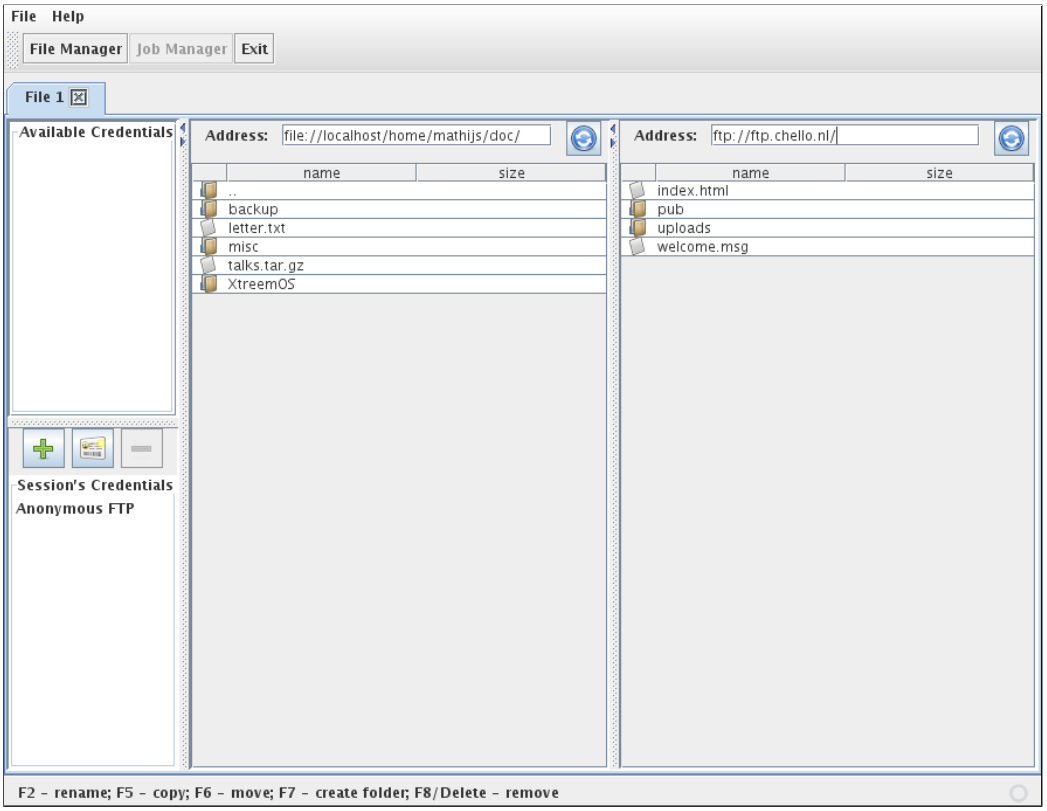

Figure 6: Example of remote directory access via FTP

8.1 File management interface

Figure 6 shows the file management interface of Xterior. Clicking the 'File Manager' button in the top left corner creates a new file management tab. Each tab has its own SAGA session object, with its own set of associated SAGA security contexts. The contexts used in the tab's session can be edited in the side pane on the left. The lower box in this side pane contains all contexts used in the tab's session, and the upper box contains the remaining contexts. In Figure 6, the tab's session only contains one 'Anonymous FTP' context. This context defines a username 'anonymous' and an empty password to access anonymous FTP servers. Contexts can be dragged between these two boxes to add and remove them to and from the tab's session. Alternative, the 'plus' and 'minus buttons can be used.

Each file management tab shows the contents of two directories, which is retrieved via a SAGA `NSDirectory.list()` call. The URL of each directory is shown above each directory contents pane. In Figure 6, the left one is the local directory `/home/mathijs/doc/`. The right directory shows the contents of the root directory of the FTP server `ftp.chello.nl`. This FTP server allows anony-

mous FTP access. Hidden from the user, the FTP adaptor in the Java SAGA engine uses the 'Anonymous FTP' context to login to this server and retrieve contents of the root directory.

The icon in front of each directory entry indicates whether it is a file or a directory. Double-clicking a file opens it with the default associated program of the operating system. Double-clicking a directory changes the current working directory.

8.2 Credential management

Clicking the middle button in the credentials side pane pops up the credential management interface of Xterior. Figure 7 shows a screenshot of this interface.

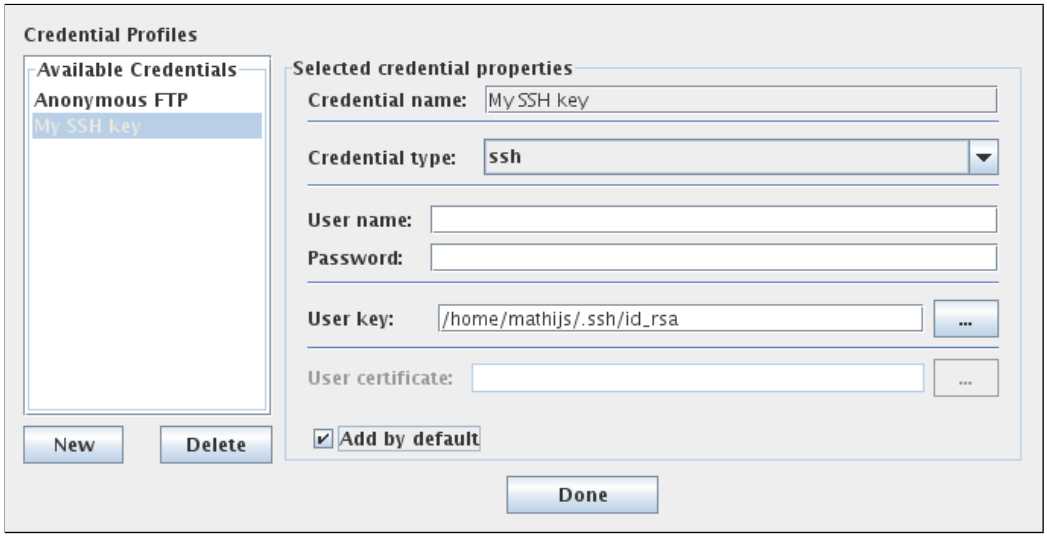

Figure 7: Creating a new 'SSH' context in Xterior

In this interface, users can manage the available credentials in Xterior. Each credential translates to a SAGA security context, and consists of a user-specified name, a type, and a set of attributes. Various credential types are available, including including 'FTP', 'SSH', 'Globus', and 'XtreemOS'. Depending on the type, certain attributes can be entered in the interface.

In Figure 7, a new context of type 'SSH' is created, and named 'My SSH key'. The attribute 'User key' is set to the filename of the local private SSH key `/home/mathijs/.ssh/id_rsa`. The option 'Add by default' is marked, which indicates that when a new tab is created, this context should be added automatically to the tab's session.

After clicking the 'Done' button, all credential information is automatically saved to a local file `~/.xterior/saved_credentials.data`. The file is encrypted with a master password.

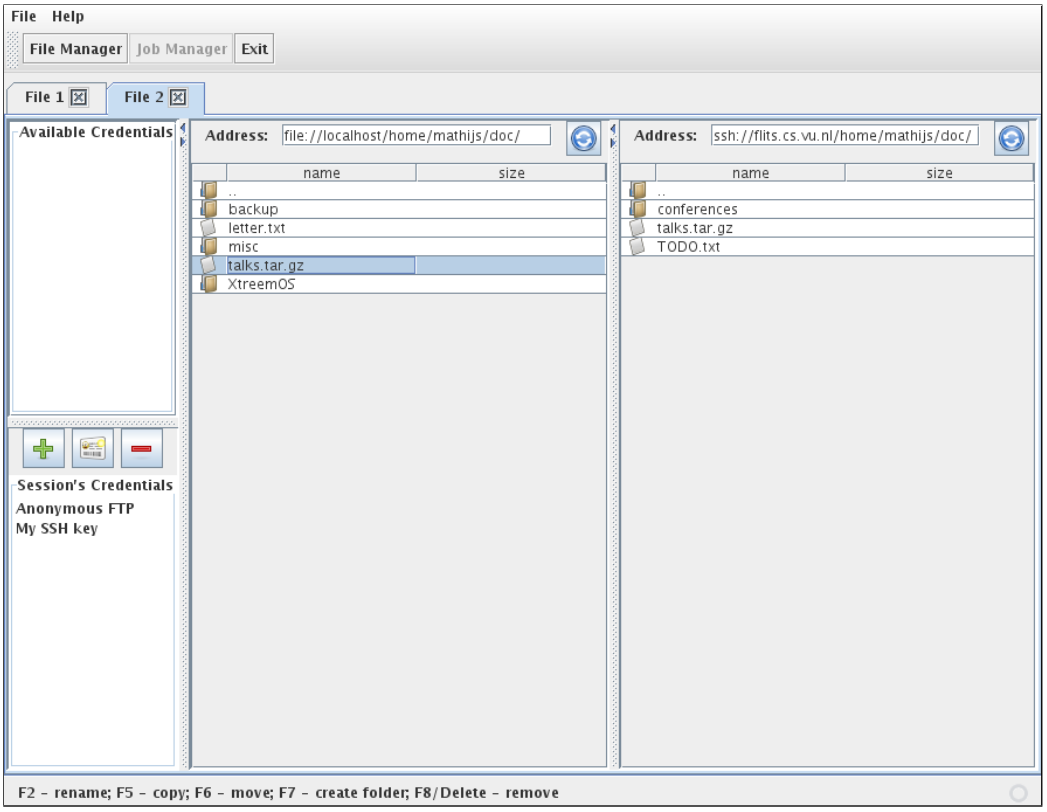

Figure 8: Example of remote directory access via SSH. The file `talks.tar.gz` is copied from the remote directory to the local one.

In Figure 8, a second tab is created by clicking the 'File Manager' button again. The new 'My SSH key' context is now automatically added to this tab's session. To demonstrate that SAGA can now access remote SSH servers, the URL of the right directory is changed to `ssh://flits.cs.vu.nl/home/mathijs/doc/`. The private key referred to in the 'My SSH context' is then used to retrieve the contents of the remote directory.

8.3 File management operations

Xterior users can copy and move files between the two directories shown in the file management interface. Copying files and directories is done by selecting one or more of them in one of the directories, and pressing 'F5'. The files are then copied to the other directory. Moving files and directories is done similarly by pressing

'F6'. Alternatively, users can *drag* entries from one directory to the other. When both directories refer to the same filesystem, dragged files will be moved. When the directories refer to different filesystems, dragged files will be copied. Figure 8 shows how the file `talks.tar.gz` was copied from the remote SSH server to the local directory `/home/mathijs/doc/`.

Other file management operation are also possible. Renaming a file or directory is done by selecting it and pressing 'F2'. Deleting files is done by selecting them and pressing 'F8' or 'Delete'. New directories can be created by pressing 'F7'.

9 Summary and Future Work

In this report, we have presented the third prototype of the XtreemOS runtime engine. The prototype implements the XOSAGA API, according to our previous deliverables D3.1.2 and D3.1.5. We have outlined the underlying design principles of our implementations in C++, Java, and Python, and have provided information for download, installation, and use. All implementations support the XtreemFS file system and XtreemOS certificates. In addition, the Java and Python engine support XtreemOS job submission. The Java engine also supports resource reservation.

Deliverable D3.1.5 defines several other XtreemOS-specific extensions to the SAGA API that are not covered in the current prototype engine. These XOSAGA interfaces provide access to publish-subscribe systems, shared properties, shared buffers, and distributed servers. Implementing these interfaces in the runtime engines for C++, Java, and Python is the subject of ongoing work.

References

[1] Gabrielle Allen, Kelly Davis, Tom Goodale, Andrei Hutanu, Hartmut Kaiser, Thilo Kielmann, Andre Merzky, Rob van Nieuwpoort, Alexander Reinefeld, Florian Schintke, Thorsten Schütt, Ed Seidel, and Brygg Ullmer. The Grid Application Toolkit: Towards Generic and Easy Application Programming Interfaces for the Grid. *Proceedings of the IEEE*, 93(3):534–550, 2005.

[2] Tom Goodale, Shantenu Jha, Hartmut Kaiser, Thilo Kielmann, Pascal Kleijer, Andre Merzky, John Shalf, and Christopher Smith. A Simple API for Grid Applications (SAGA). Grid Forum Document GFD.90, January 2008. Open Grid Forum (OGF).

[3] Stephan Hirmer, Hartmut Kaiser, Andre Merzky, Andrei Hutanu, and Gabrielle Allen. Generic Support for Bulk Operations in Grid Applications. In *MCG '06: Proceedings of the 4th International Workshop on Middleware for Grid Computing*, page 9, New York, NY, USA, November 2006. ACM Press.

[4] Hartmut Kaiser, Andre Merzky, Stephan Hirmer, and Gabrielle Allen. The SAGA C++ Reference Implementation – Lessons Learnt from Juggling with Seemingly Contradictory Goals. In *Workshop on Library-Centric Software Design LCSD'06, at Object-Oriented Programming, Systems, Languages and Applications conference (OOPSLA'06)*, Portland, Oregon, USA, October 2006.

[5] XtreemOS Consortium. First Draft Specification of Programming Interfaces. Deliverable D3.1.1, 2006.

[6] XtreemOS Consortium. Second Draft Specification of Programming Interfaces. Deliverable D3.1.2, November 2007.

[7] XtreemOS Consortium. Second Prototype of XtreemOS Runtime Engine. Deliverable D3.1.6, November 2008.

[8] XtreemOS Consortium. Third draft specification of programming interfaces. Deliverable D3.1.5, November 2008.