Project no. IST-033576

XtreemOS

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

Second Prototype of XtreemOS Runtime Engine

D3.1.6

Due date of deliverable: November 30^{th}, 2008
Actual submission date: December 8^{th}, 2008

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 1.0.3 / Last edited by Mathijs den Burger / January 5^{th}, 2009

Project co-funded by the European Commission within the Sixth Framework Programme		
Dissemination Level		
PU	Public	$\surd$
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	28/10/08	Mathijs den Burger	VUA	initial draft
1.0	01/12/08	Thilo Kielmann	VUA	complete version
1.0.1	05/12/08	Mathijs den Burger	VUA	incorporated internal review comments
1.0.2	08/12/08	Thilo Kielmann	VUA	final version
1.0.3	05/01/09	Mathijs den Burger	VUA	updated URL to the software after SVN repository re-organization

Reviewers:
Björn Kolbeck (ZIB), Enric Tejedor (BSC)

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 second prototype implementations of a runtime engine for the XtreemOS API as specified in deliverable D3.1.2 [8]. The first prototype implementation, D3.1.3 [6], implemented the SAGA API, according to D3.1.1 [5] in the C++ programming language. The second implementation described here is adding the following functionality:

- a separate implementation in the Java programming language
- support for the XtreemFS file system and XtreemOS certificates, for both the C++ and Java implementations

In this report, we outline the general design of both implementations, describe their installation and configuration process, explain their integration with the XtreemFS filesystem and XtreemOS certificates and give a code example.

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. Deliverable D3.1.2 [8] added XtreemOS-specific extension to SAGA, named XOSAGA. This specification is the basis for the implementations presented by this report.

In this deliverable, we provide two prototype implementations of the XOSAGA API, written in the C++ and Java programming languages. Both implementations cover those parts of the XtreemOS-specific functionality described in D3.1.2 for which stable implementations have been available so far. In particular, the current implementations both support:

- the XtreemFS file system
- XtreemOS certificates

The XOSAGA package for Application Execution Management is **not** covered by the currently existing implementations. It will be an essential part of the third prototype implementation (D3.1.9, due month 42). This future implementation will also cover the third draft API for XtreemOS, focusing on SAGA extensions to accomodate specific XtreemOS features missing in D3.1.2. These currently missing extensions are described in a companion deliverable, D3.1.5 [9]. The functionality available with this second prototype implementation is summarized in Table 1.

In this document, we outline the general design of both XOSAGA implementations, describe their integration with the XtreemFS filesystem and XtreemOS certificates, and provide information for downloading, installing and using the software. For providing a self-contained report, we include some description of the C++ implementation that was also part of D3.1.3. We augment it by the respective information about the new, Java-based 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*)

SAGA package	C++	Java
Physical Files	local	local
	XtreemFS	XtreemFS
	Globus GT4 GridFTP	Globus GT2 & GT4.0 WS GridFTP
		SSH, FTP, SFTP
Replicated Files	Globus GT4 RLS	generic (via physical files)
Job submission	local	local
	Globus GT4 GRAM2	Globus GT2 & GT4.0WS GRAM
	GridSAM	GridSAM
	Condor	gLite, Sun Grid Engine
		LocalQ, Zorilla
Streams		TCP sockets
RPC	XMLRPC	XMLRPC

Table 1: SAGA functionality provided by the current prototype implementation

which are dynamically loaded on demand, and bound at runtime (*late binding*). The relation between these components are illustrated in Figure 1.

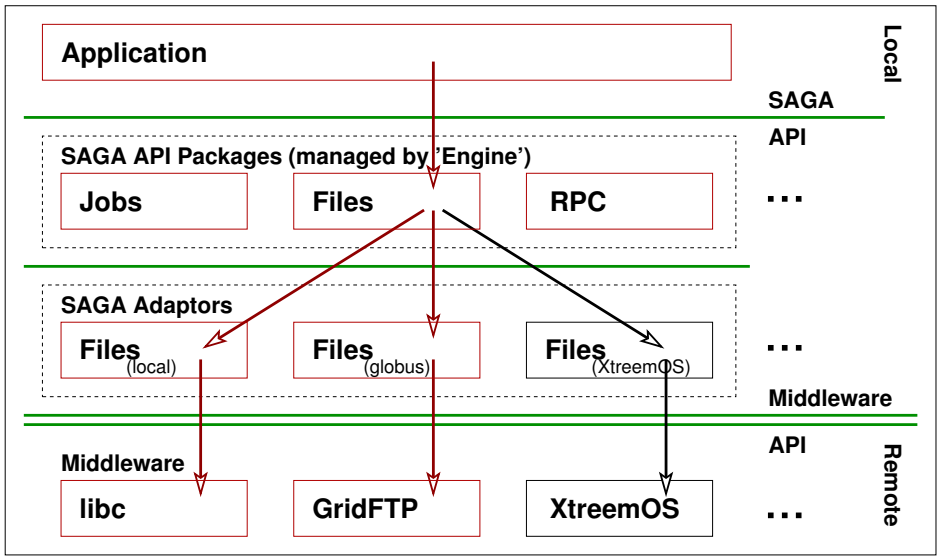

Figure 1: General architecture: a lightweight *engine* dispatches SAGA calls to dynamically loaded middleware *adaptors*.

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.

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].

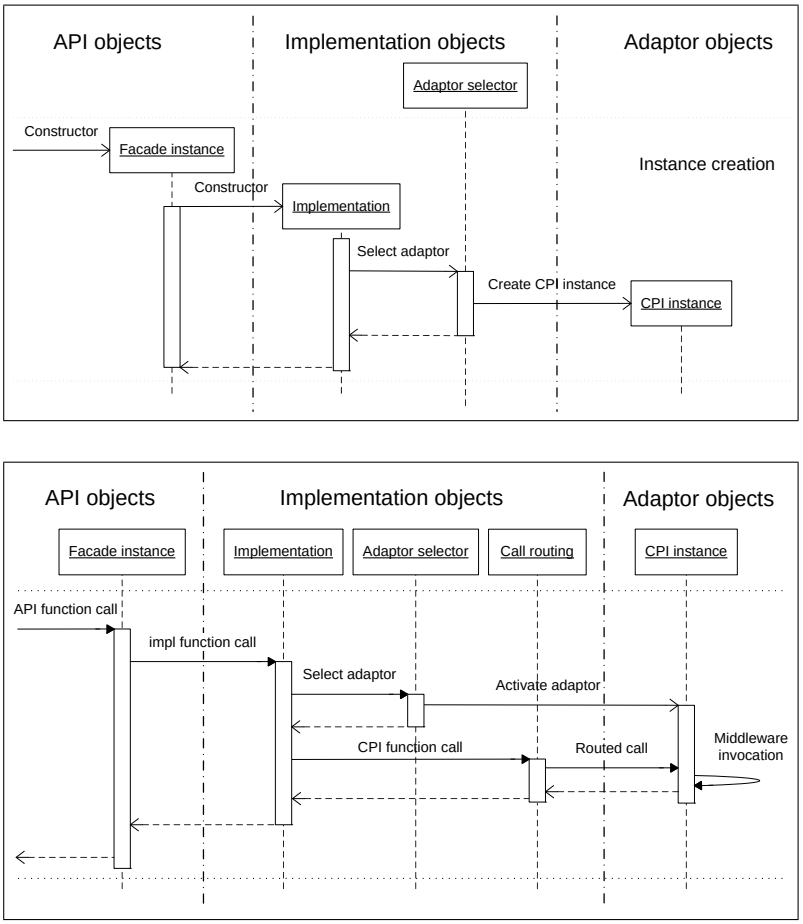

Figure 2: The PIMPL mechanism hides the implementation from the end user. Shown are object creation (top) and invocation of a SAGA function call (bottom).

3 The SAGA C++ Engine

One of the technically challenging requirements of the SAGA Core API specification is that SAGA object copy operations are shallow copies by default, so copies do not perform a deep copy of object state. These semantics are performant in remote environments as they avoid remote operations (state query and duplication) in most cases. A second challenge is that the lifetime of a SAGA object is *not* only defined by its scope in the program, but depends (a) on the lifetime of objects depending on that instance, (b) pending asynchronous operations for that instance, and (c) shallow copies of that instance.

To address these challenges, our SAGA implementation uses a technique called the PIMPL mechanism (**p**rivate **impl**ementation), shown in Figure 2. Using this

technique, we were able to simplify the internal state management of SAGA objects and to resolve the lifetime dependencies between SAGA objects, SAGA sessions, and adaptors [4]. At the same time, the engine provides the complete SAGA task model, e.g. 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.

Using the PIMPL mechanism, 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.

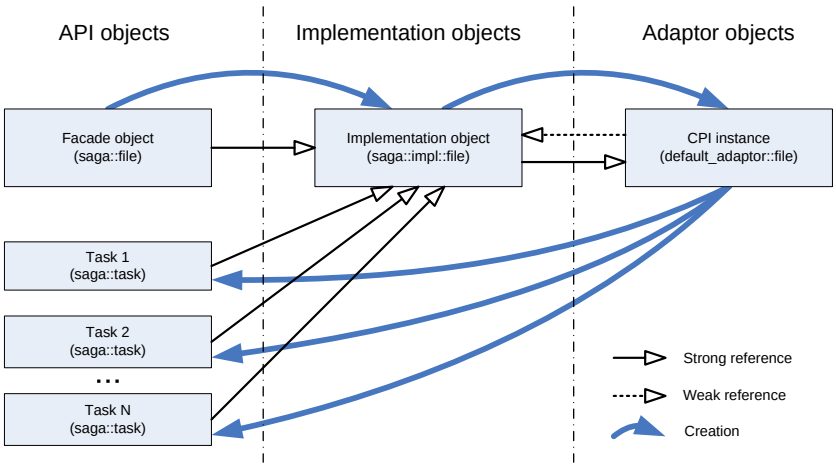

Figure 3: Shared pointers to the implementation object instance define the lifetime of the SAGA objects.

As can be seen in Figure 3, 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 adaptors to access the following services:

- XtreemFS for file access and file management
- Globus (GT4) GRAM2 service for job submission and management
- Globus (GT4) GridFTP service for file access and file management
- Globus (GT4) RLS service for replica management
- GridSAM for job submission and management
- Condor for job submission and management
- XMLRPC for remote procedure calls

Interfaces to other XtreemOS services are currently being designed and implemented.

3.1 Installation and deployment

In the XtreemOS distribution, 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 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 in INRIA, France:

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

XOSAGA includes the latest SAGA 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 define 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 meta `SagaFactory` 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 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.

4.3 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 the 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 as a swiss army knife.

Currently, all adaptors in the SAGA Java implementation together provide the following functionality:

- File access and file management using XtreemFS, SSH, FTP, SFTP, GridFTP, Global GT4.0, and local files.
- Job submission and control using SSH, Globus (GT2 and GT4.0 WS), GridSAM, gLite, Sun Grid Engine, LocalQ, Zorilla, and local jobs.
- Generic replica management of logical files.
- Streams on top of sockets.
- Remote procedure calls using XMLRPC.

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 `saga.properties` file 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.

4.4 Installation and documentation

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

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

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 XtreemFS adaptor

Both the C++ and Java implementation of SAGA contain a file adaptor to access the XtreemFS file system. XtreemFS provides access to remote data on *volumes*, where physical location, internal distribution, and replication are hidden from a user. A volume can be mounted to the local filesystem via FUSE. As a result, a user has to to execute a reasonably complicated mount command for every volume

it wants to access. The mount command includes the address and port of the Metadata and Replica Catalog (MRC) that manages the volume, some special FUSE or XtreemFS-specific options and a local mount point. Users also have to take care of unmounting mounted volumes when they are not used anymore.

The XtreemFS file adaptor saves a SAGA application from all this explicit mounting and unmounting by acting as an *automounter*: whenever a user accesses a file or directory on a certain XtreemFS volume, the adaptor automatically ensures that the volume is mounted at some local directory. The adaptor remembers which volume is mounted where, and ensures that a volume is only mounted once. All volumes that are accessed in a certain SAGA session are automatically unmounted when that session is destroyed. As a result, accessing files on XtreemFS volumes in a SAGA application is very easy.

A SAGA application can refer to a file on an XtreemFS volume using a special URL. An example of such a URL is:

```
xtreemfs://vol42@host.example.com:32636/dir/file.txt
```

This example URL points to a file '/dir/file.txt' on an XtreemFS volume 'vol42' that is managed by an MRC located at 'host.example.com' and listens to port 32636. When no explicit port number is specified, a default port number is used.

The XtreemFS adaptor also understands local files, using relative and absolute URLs. A relative URL looks like `dir/file.txt`, and points a file relative to the base directory of the object the URL is given to (e.g. a directory object). An absolute URL looks like `file://localhost/tmp/file.txt`, which points to a file '/tmp/file.txt' on the local filesystem. The XtreemFS adaptor can therefore directly copy and move files between the local file system and remote XtreemFS volumes.

The XtreemFS file adaptor can be configured by editing key-value pairs in the `xtreemfs_file_adaptor.ini` file (in the C++ implementation) or editing some properties in the `xosaga.properties` file (in the Java implementation). A user can specify, among others, the directory to create mount points in, the default port number to use in XtreemFS URLs and the commands for mounting and unmounting volumes.

5.1 Code Examples

In this section we will develop a simple XOSAGA application that mimics the 'cp' command. We can then use this program to easily copy files to and from XtreemFS volumes. The program reads the source and destination file from the command line arguments, creates a SAGA file object and uses the copy() function to perform the copying. We will first show the C++ versioon of the program, and then the Java version.

5.1.1 C++ code example

The source code of the C++ example program looks like this:

```
#include <saga.hpp>
#include <iostream>

using namespace std;

int main(int argc, char** argv) {
    if (argc != 3) {
        cout << "usage: " << argv[0] << " src dst" << endl;
        return 1;
    }

    saga::url src(argv[1]);
    saga::url dst(argv[2]);

    try {
        saga::filesystem::file f(src, saga::filesystem::Read);

        cout << "Copying " << src << " to " << dst << endl;
        f.copy(dst);
    } catch (saga::exception const & e) {
        cout << "SAGA exception: " << e.get_message() << endl;
    }
}
```

We save the program in a file called 'copyfile.cpp', and then create the following Makefile:

```
SAGA_SRC        = $(wildcard *.cpp)
SAGA_OBJ        = $(SAGA_SRC:%.cpp=%.o)
SAGA_BIN        = $(SAGA_SRC:%.cpp=%)

include /usr/share/saga/make/saga.application.mk
```

Finally, we compile our program:

```
$> make
      compiling  copyfile.o
      binlinking copyfile
$>
```

We can now use our `copyfile` program to copy files around. Assume we have a simple XtreemFS installation, with a directory service, MRC, and OSD process all running locally. We will create a volume 'vol42', and copy a file to it using our example program. First, we create the volume:

```
$> mkvol http://localhost/vol42
```

We then create a file, and copy it to the volume:

```
$> echo 'hello world!' > hello.txt
$> ./copyfile hello.txt xtreemfs://vol42@localhost/hello.txt
```

We can check whether the file was copied succesfully by mounting the XtreemFS volume manually:

```
$> mkdir /tmp/vol42
$> xtfs_mount -o volume_url=http://localhost:32636/vol42 \
-o direct_io -o logfile=/dev/null /tmp/vol42
$> ls /tmp/vol42
hello.txt*
$> cat /tmp/vol42/hello.txt
hello world!
$> xtfs_umount /tmp/vol42
$> rmdir /tmp/vol42
```

5.1.2 Java Code Example

The Java code of the example program looks like this:

```
import org.ogf.saga.error.*;
import org.ogf.saga.file.*;
import org.ogf.saga.namespace.*;
import org.ogf.saga.url.*;

public class CopyFile {

    public static void main(String argv[]) {
        if (argv.length != 2) {
            System.out.println("usage: java FileCopy src dst");
                        System.exit(1);
        }

        try {
            URL src = URLFactory.createURL(argv[0]);
            URL dst = URLFactory.createURL(argv[1]);

            File f = FileFactory.createFile(src);
            f.copy(dst);
        } catch (SagaException e) {
            System.out.println("SAGA exception: " + e);
        }
    }

}
```

We save the program in a file called 'CopyFile.cpp'. Assume that the environment variable `JAVA_SAGA_LOCATION` already points to the installation directory of the SAGA Java engine. We can then compile our program like this:

```
$> javac -cp $JAVA_SAGA_LOCATION/lib/saga-api-1.1rc1.jar \
   CopyFile.java
$>
```

We can now use our `CopyFile` program similarly to its C++ counterpart. For example, we can copy the file `hello.txt` to the XtreemFS volume 'vol42' again (which were both already created in the C++ example in the previous section).

```
$> $JAVA_SAGA_LOCATION/bin/run-saga-app CopyFile \
   hello.txt xtreemfs://vol42@localhost/hello.txt
```

6 XtreemOS certificates

The grid level credentials of an XtreemOS user consist of XOS certificates [7]. An XtreemOS user obtains such a certificate by contacting the Credential Distribution Authority. The XOS certificate is then made accessible in SAGA via a context object with the type '`xtreemos`'. The `UserCert` property of this context object should point to the file containing the XOS certificate the user.

Every time a SAGA application accesses a file on an XtreemFS volume, the XtreemFS adaptor first checks whether the current session contains a context with the type '`xtreemos`'. If such a context is found, the file pointed to by the `UserCert` property is then automatically added to the command that automatically mounts a remote XtreemFS volume locally. Furthermore, the scheme of the URL to contact the MRC of that volume at will then change from '`http`' to '`https`' to enable SSL.

7 Summary and Future Work

In this report, we have presented the second prototype of the XtreemOS runtime engine, implementing the XOSAGA API, according to our previous deliverable D3.1.2 (except for the Application Execution Management package). We have outlined the underlying design principles of our implementations in C++ and Java, and have provided information for download, installation, and use. Both implementations support the XtreemFS file system as well as XtreemOS certificates.

In a companion deliverable, D3.1.5, we are presenting the third draft API for XtreemOS, focusing on SAGA extensions for exposing XtreemOS-specific functionality that had not yet been covered by D3.1.3. This functionality will be implemented in the next implementation release, along with the Application Execution Management package.

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. First Prototype of XtreemOS Runtime Engine. Deliverable D3.1.3, November 2007.

[7] XtreemOS Consortium. First Specification of Security Services. Deliverable D3.5.3, May 2007.

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

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