Project no. IST-033576

XtreemOS

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

Second Draft Specification of Programming Interfaces D3.1.2

Due date of deliverable: November 30th, 2007
Actual submission date: January 10th, 2008

Start date of project: June 1st 2006

Type: Deliverable
WP number: WP3.1
Task number: T3.1.1

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 / Last edited by Thilo Kielmann / January 10th, 2008

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

Revision history:

Version	Date	Authors	Institution	Section affected, comments
0.1	22/10/07	Ana Oprescu, Thilo Kielmann	VUA	initial draft
0.2	30/10/07	Guillaume Pierre	VUA	minor update vs. WP3.2
0.99	03/01/08	Ana Oprescu, Thilo Kielmann	VUA	complete version
1.0	10/01/08	Thilo Kielmann	VUA	final version, after WP leader review

Reviewers:
All work package leaders from SP2 and SP3

Tasks related to this deliverable:

Task No.	Task description	Partners involved[°]
T3.1.1	Specification of XtreemOS API extensions to the set of POSIX specifications	VUA[*], all partners except CDC

[°]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 draft specification of the API for XtreemOS. In the earlier deliverable D3.1.1 [9], we have proposed the upcoming, OGF-standardized SAGA API [3] as the first draft API. In this report, we present refinements to SAGA, especially for facilitating XtreemOS-specific functionality. Besides improvements made on the SAGA specification itself (thus XtreemOS contributing to the upcoming OGF standard), we present how the following XtreemOS functionalility can be accessed via the API: application management (including checkpoint/recovery), the XtreemFS file system, the XtreemOS VO management, as well as access to the LinuxSSI cluster operating system.

Contents

1 Introduction

The specification of the XtreemOS API is being performed by agreement between representatives of all project partners, representing both groups of interface providers and interface users. This is an iterative process: a first specification, intended as an early draft, has been agreed upon early in the project, such that immediate implementation experience can be gained and strengths and weaknesses can be identified. The first API specification has been documented in deliverable D3.1.1 [9]. Several milestones during the project will cover revisions and improvements of the API, until the final version will meet all requirements of both applications and system implementors.

In general, the XtreemOS API has to serve three classes of applications:

1. Existing Linux applications, using POSIX-standardized interfaces.

2. Existing grid applications, using OGF-standardized interfaces.

3. New applications, using functionality uniquely provided by XtreemOS.

In D3.1.1, we have selected the emerging OGF standard *Simple API for Grid Applications* (SAGA) as the first draft API for XtreemOS. SAGA had been selected because it combines OGF-standardized API's (namely JSDL [1], BES [2], GridFTP [5], GridRPC [7], DRMAA [8]) with POSIX-like interfaces wherever possible (e.g., for files and streams). Also, SAGA has been cited as a possible candidate API for XtreemOS by the project-internal application groups in deliverable D4.2.1 [11].

Part of the XtreemOS API design process is to actively contribute to the standardization efforts, most importantly within OGF for the SAGA specification. These active contributions not only give XtreemOS a better visibility, but also (and most importantly) make sure that XtreemOS stays in sync with ongoing standardizations, and can contribute its own technical findings to the ongoing standardization process.

Figure 1 illustrates the interaction between the SAGA-related standardization and the XtreemOS API specification. Initially (in D3.1.1), XtreemOS has adopted the SAGA draft recommendation, documented in OGF's draft GWD-R.90 [3][1], as its first own API specification. Since then, XtreemOS has contributed to finalizing the SAGA-CORE specification, with imminent publication as a draft recommendation, in OGF's document GFD.90 [4]. The publication of GFD.90 marks a crossroads at which the SAGA-CORE specification itself becomes fixed, and XtreemOS-related modifications or additions require the definition of separate API documents.

[1]The draft [3] unfortunately refers to itself as GWD-R.72, which is wrong.

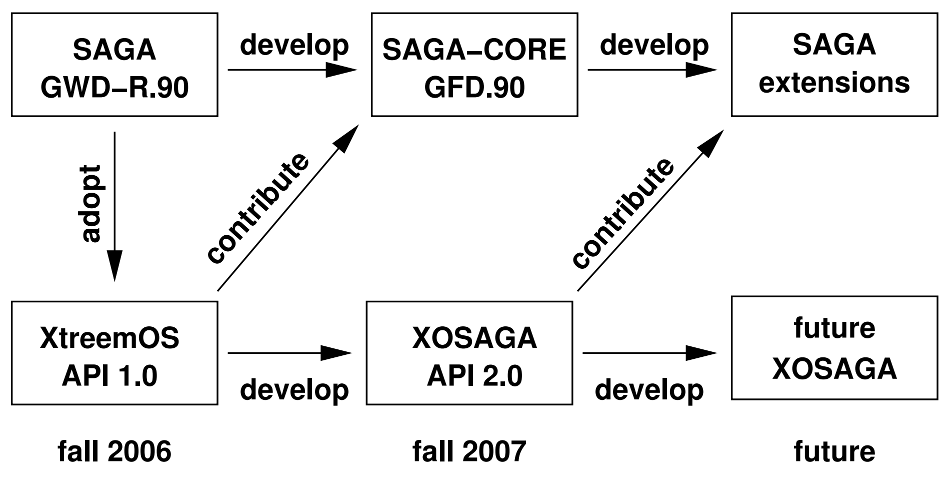

Figure 1: Co-development of XtreemOS API and SAGA standard.

This document covers the second draft API specification for XtreemOS. Its main focus is providing API extensions (w.r.t. to SAGA, the first draft API specification) that provide access to XtreemOS-specific functionality, as far as it is available at this stage of the project. For this purpose, we define an API name space called XOSAGA (XtreemOS extensions to SAGA) that mirrors the SAGA API name space. XOSAGA contains only those packages, classes, and interfaces that require XtreemOS-specific extensions to SAGA. Together, SAGA and XOSAGA form the XtreemOS API, as of this second draft version. By this design, applications require only minimal changes for being ported from "standard SAGA" to XOSAGA: applications simply have to create objects from classes from the XOSAGA name space, instead of classes with identical names, but from the SAGA namespace.

Within OGF, the SAGA-related standardization efforts are focusing on the development of extension packages to the SAGA-CORE. Obviously, this is the mechanism for XtreemOS to contribute by proposing XOSAGA extensions for such packages.

The bulk of this document consists of an analysis of functionality either specified or implemented so far by the XtreemOS work packages from sub projects SP2 and SP3. Goal of this analysis is to find out to which extent the existing SAGA API is sufficient to provide application access to XtreemOS-specific functionality, and where API extensions (part of XOSAGA) become necessary.

Our analysis is based on the assumption of having an engine-based SAGA implementation, as also provided by the companion deliverable D3.1.3 [15]. Figure 2 shows the architecture of such an implementation. The SAGA (and XOSAGA) API consists of several packages (like Jobs, Files, RPC). The engine implements these API packages by using so-called adaptors, dynamically-loaded interfaces to middleware backends. The engine itself forms a rather thin layer, mostly in charge of dispatching incoming API calls to the right adaptor, for a given middleware intallation. The main result of our functionality analysis is to decide which XtreemOS-specific functionality can be made available by implementing XtreemOS-specific adaptors for the SAGA engine, and which functionality requires actual API extensions, within the XOSAGA name space.

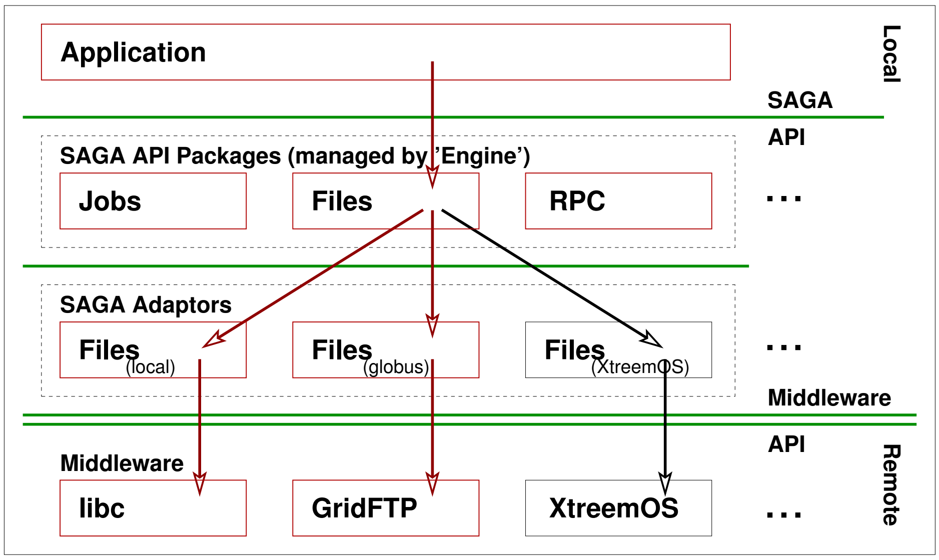

Figure 2: Architecture of an engine-based SAGA implementation.

The remainder of this document is organized as follows. First, we discuss the updated application requirements in Section 2. Section 3 discusses the extensions that XtreemOS has already contributed to the SAGA-CORE specification. Then, we analyze XtreemOS-specific functionality regarding VO management (Section 4), application execution management (Section 5), and the XtreemFS file system (Section 6). We complete our analysis by detailing which XtreemOS functionality is not yet ready for designing a specific API (Section 7). Finally, Section 8 summarizes our findings.

2 XtreemOS application requirements revisited

In their deliverable D4.2.1 [11], work package 4.2 had already described their initial requirements to other work packages, among which is the XtreemOS API. In deliverable D3.1.1 [9], we have already outlined how, and to which extent, these requirements can be addressed by the XtreemOS API.

2.1 Updated requirements on XtreemOS interfaces

Deliverable D4.2.3 [13] presents a revised set of requirements. We discuss the applicability of the API-related requirements from D4.2.3 here. In particular, the following API-related requirements are expressed:

R45 Other API Standards as basis for XtreemOS API (revised, was R43)
"XtreemOS API must consider the following standard as a basis: SAGA (especially the subsets DRMAA, GAT). Furthermore, any other standard allowing applications to access user and/or job information is welcome. WP3.1 must ensure that applications can adapt to XtreemOS in a way that complex start and stop procedures can be specified that are used to start/stop the various parts of the overall application. To this end, WP3.1 must ensure that the interfaces are sufficient enough and compatible with WP3.3.

Additionally, one application requires the following functionalities that are actually provided by Globus/Globus-related components: GridFTP, Apache Axis, and the GSI public key infrastructure. Here, an equivalent XtreemOS functionality is needed."

R46 Demand for POSIX compliance (revised, was R44)
"XtreemOS must provide access to the distributed resources in the Grid from any node using the standard Posix interface. Mandatory access control (ACL) as defined e.g. in POSIX .1e, IEEE 1003.1e/2c (which was withdrawn). It is yet to be clarified which calls have to be provided. Furthermore, it would prove useful if functions for management of processes on remote machines would be part of the XtreemOS API."

R47 XtreemOS API language support (revised, was R45)
"XtreemOS must support several different programming languages. The mandatory languages and their priorities are C (high priority), C++ (high priority), Java (high priority) and Fortran 77 (low priority). Fortran 77 can be supported via C bindings. The optional languages are Python (medium priority), Perl (medium priority) and Ada (low priority)."

R48 Degree of Interoperability (was R46)
"It should be possible to use XtreemOS as a backend for GT4 WS-GRAM."

2.2 Implementation of these requirements

R45 SAGA indeed is the basis of the XtreemOS API. Providing application-level interfaces to the functionality from WP3.3 is part of this document (see Section 5).

Providing special interfaces as mentioned in the second text paragraph of R45 is debatable. GridFTP is a way to access files, and as such is its functionality covered by SAGA. Apache Axis is an implementation of Web services, containers for providing access to functionality in general. Applications should not rely on technologies by which needed functionality is provided. The XtreemOS API is providing the functionality itself, according to OGF-based standards, and independent of the underlying technology providing this functionality. Besides, providing a GridFTP interface to XtreemFS, or a Web service interface to XtreemOS' application execution management, would require support from the work packages realizing the underlying functionality, and could not be properly addressed on the API level alone.

Public key infrastructures are an authentication mechanism, something to be dealt with by WP3.5. Providing interfaces to the respective VO management functionality is part of this document, addressed in Section 4.

R46 Being based on SAGA, the XtreemOS API is providing POSIX syntax and semantics wherever possible. Access Control Lists (ACL) functionality based on the withdrawn proposal IEEE 1003.1e/2c is not practically useful (without the respective functionality being available).

However, providing access control functionality is a prominent extension made to the original SAGA draft [3]. In the new, final, SAGA specification [4] we have added the *SAGA Permission Model*, covering access control based on a POSIX-inspired permission model.

Application execution management, rather than process management, is subject to this document (see Section 5).

R47 In the companion deliverable D3.1.3 [15] we have released an implementation of the first draft version XtreemOS API (according to deliverable D3.1.1 [9]). This implementation is written in C++, thus providing the API for the C++ and C languages. Implementing the XtreemOS API in Java is subject to ongoing work. Currently, there are no immediate plans to provide the API in other languages. This can, however, be achieved in later project stages via wrapper interfaces to the C++ implementation.

R48 As already outlined in D3.1.1 [9], this requirement can technically not be addressed on the API level. It has to be supported by the infrastucture level, namely by WP3.3.

3 Extensions included in the SAGA-CORE specification

We consider the following document [4] to be part of this deliverable. Due to its size (about 300 pages), we refrain from directly including the SAGA specification in this document.

> 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, Open Grid Forum (OGF), 2007, available from:
> `http://forge.ogf.org/short/saga-core-wg/saga-core-v1`

The changes from the previous SAGA specification draft, as used for the first XtreemOS API [9], can be summarized as follows:

- Introduce a buffer class for uniform management of I/O buffers.
- Introduce a URL class for uniform parsing and handling of URLs and names.
- Radical overhaul of the permission model: ACLs are gone, and POSIX-like permissions are introduced on all first class SAGA objects.
- Added preconditions and postconditions for all method calls.
- Added permissions for all method calls.
- Several SAGA interfaces now implement the task interface if they are used by classes which implement the task interface themselves.
- Several SAGA interfaces now implement the permission interface if they are used by classes which implement permission task interface themselves.
- Copy semantics for SAGA objects (shallow versus deep copy) has been clarified.
- The lifetime of the default SAGA session has been clarified.
- The SAGA class diagram has been updated, see Figure 3.

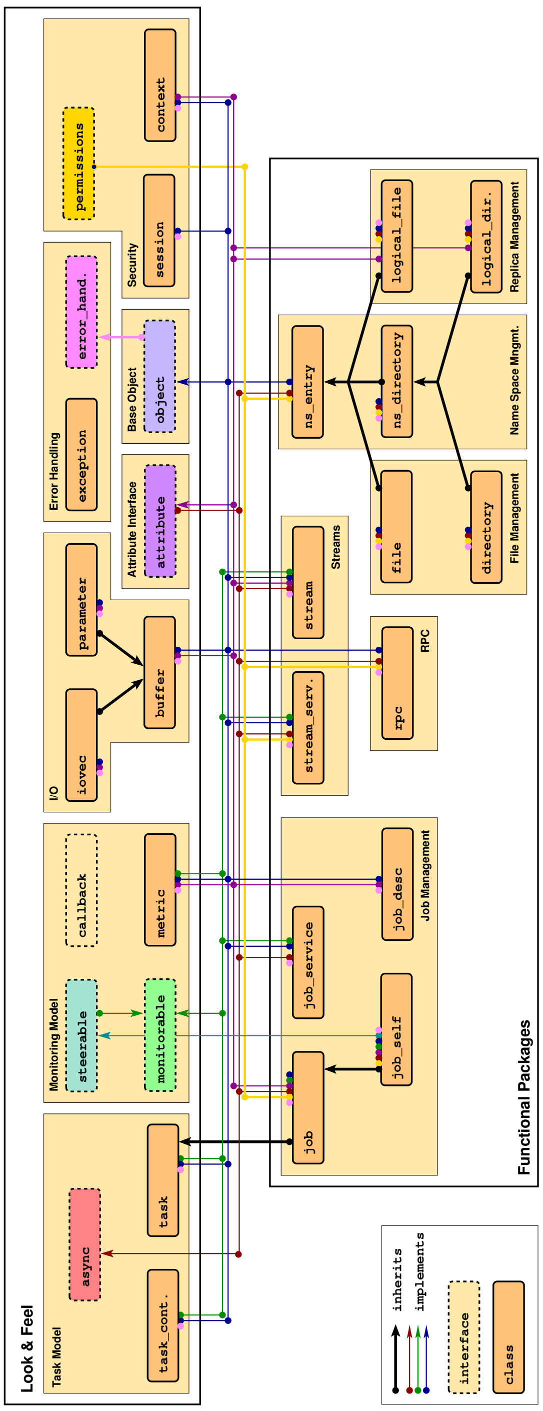

Figure 3: The revised SAGA class and interface hierarchy, according to [4].

- The usage of URLs has been clarified.
- Byte ordering issues have been clarified.
- Relation of SAGA exceptions and middleware exceptions have been clarified.
- Fixed order/precedence of SAGA exceptions.
- Allow for nested exceptions in bindings (for implementations with late binding).
- Clarify usage of object types.
- Change saga::context to be more extensible and backend specific, while simplifying its attributes and initialization (set_defaults()).
- Clarify the formatting of user IDs for saga::context instances.
- Attributes cannot implicitly converted between vector and scalar attributes anymore – that was confusing.
- Simplified the signature of find().
- Changed the task interface: return values for async ops are essentially created at the point where they are used.
- Updated/fixed all state diagrams according to the new task model.
- get_object() has been re-introduced on the task.
- Clarified that async object construction is a matter for the language bindings.
- Added verbose description for all enum types.
- Allow bindings to inherit the task container from native containers.
- Added support for the JSDL SPMD extension to the job description.
- Removed some unusable job description attributes.
- Clarified consistency semantics of saga::logical_file instances.
- Clarify why no raw JSDL is supported.
- Clarify semantics of find() calls.

- Removed (unused) attribute interface from file.
- Unified semantics over the various packages:
  - open on object construction
  - exception semantics
  - close() semantics
  - state management
  - bitwise ORing of flags
- Several examples have been clarified.
- Countless spelling, grammar fixes and formatting fixes.

4 VO Management

In XtreemOS, VO management is based on XtreemOS-specific certificates, that are issued and administered by VO management services, and are used and interpreted both by other XtreemOS-specific services as well as the different flavours of the XtreemOS operating system, the latter via kernel modules that authenticate and authorize users via these XtreemOS certificates [17].

In the SAGA API [4], the `saga::context` class provides the functionality of a security information container. A `saga::context` object can be attached to a `saga::session` handle, and as such be available to all SAGA objects instatiated in that session. Multiple contexts can co-exist in one session, and it is up to the implementation to choose the correct context for a specific method call.

A context has a set of attributes which can be set/get via the SAGA attributes interface (that is implemented by the `saga::context` class). Exactly which attributes a context actually evaluates, depends on its type. A SAGA implementation can implement multiple types of contexts. The implementation must document which context types it supports, and which values to the `Type` attribute are used to identify these context types. Also, the implementation must document which default values it supports for the various context types, and which attributes need to be or can be set by the application.

We therefore specify that any implementation of the XtreemOS API, according to this document, must support[2] `saga::context` objects, that are representing XtreemOS certificates, and are identified by the value `xtreemos` for the `Type` attribute.

[2]possibly among other types

For such `xtreemos` contexts, the implementation must provide useful default values for the following attributes. The application can get and set these attributes, too:

```
// name: CertRepository
// desc: location of the trusted certificates
// mode: ReadWrite
// type: string

// name: UserCert
// desc: location of a user certificate to use
// mode: ReadWrite
// type: string

// name: UserKey
// desc: location of the private key for a user
// mode: ReadWrite
// type: string
```

In addition, the implementation must provide the following (read only) attributes for `xtreemos` contexts, providing the relevant information from XtreemOS certificates [17]:

```
// name: GlobalPrimaryVOName
// desc: the primary VO that a user is associated with
// mode: ReadOnly
// type: string

// name: GlobalPrimaryRoleName
// desc: the primary role that a user is associated with
// mode: ReadOnly
// type: string

// name: GlobalPrimaryGroupName
// desc: the primary group that a user is associated with
// mode: ReadOnly
// type: string

// name: GlobalSecondaryGroupNames
// desc: the list of secondary groups that a user
//       is associated with
// mode: ReadOnly
// type: array<string>
```

5 Application Execution Management

Application execution management (AEM) is a set of functionalities that are central to XtreemOS. The precise functionality for application execution management is still evolving [14, 18], so a final API can not be defined as of the current state of the project (month 18). Because we consider AEM as central and important to XtreemOS, we nevertheless provide an API to this set of functionalities, being aware that some details may only be filled in at later stages, and that future API versions might deviate from what is described in this document.

The following subsets of the XtreemOS basic AEM functionality are identified in deliverable D3.3.2 [14] (Table 1). In the following, we analyze how these subsets can be mapped to (accessed using) the XtreemOS API, version 1 (SAGA), or which extensions to SAGA become necessary.

Job Submission: mapped to **saga::job**; detailed analysis in Section 5.2.

Job Management: mapped to **saga::job**, **saga::metric**, and **saga::session**; detailed analysis in Section 5.3.

Job Checkpointing and Migration: mapped to **saga::job** and an extension in an XOSAGA class; detailed analysis in Section 5.4.

Job Monitoring: mapped to **saga::job**, and to SAGA's monitoring model; detailed analysis in Section 5.5.

Resource Management: not available in SAGA. We propose a new package in XOSAGA, detailed design in Section 5.6.

5.1 General Terminology and Scoping

Within different work packages of XtreemOS, terminology differs. SP2 tends to describe applications from a system-level view while SP3 tends to describe processes and jobs from an application-level view. SAGA (and the XtreemOS API) naturally has an application-centric view. Here, we put the respective terminology next to each other. More specific terms are defined per functionality subset, as needed.

XtreemOS Basic API: client-side Application Execution Management (AEM) API, according to deliverable D3.3.2 [14] and the ongoing work in [18].

application: composed of application units running on different grid nodes (see D2.1.1 [16] and D2.2.1 [12]). This is a system-level definition.

application unit: a collection of processes under the control of one operating system instance, i.e. a grid node (e.g. Linux-SSI or Linux-XOS) (see D2.1.1 [16] and D2.2.1 [12]). This is a system-level definition.

process: part of a job consuming resources (see D3.3.1 [10]). This is an application-level definition.

job: one or more Linux processes which collaborate to achieve a certain goal or objective (see D3.3.1). This is an application-level definition.

job context: defined via grid user and virtual organization.

job unit: same as process.

The XtreemOS API (XOSAGA) is, much like SAGA, intended to serve application programmers. As such, it is the foremost goal to provide a simple interface that abstracts from the underlying infrastructure while exposing the application-relevant aspects. As such, our API must deal with application execution management in a simple and application-centric way, avoiding the use of redundant terminology. While it is evident to use system-level and application-level terminology where appropriate for the different XtreemOS operating-system flavours and services, the application-level API has to confine itself to what is relevant to the applications themselves. As such, we omit the explicit handling of *processes* in XOSAGA, as processes are merely implementing running *applications*, referred to as *jobs*.

In the following, we analyse the functionality subsets for AEM in XtreemOS. For each subset, we identify which functionality can be mapped to existing concepts (classes and interfaces) within SAGA. For missing functionality, we propose extensions to SAGA. We collect all these extensions in a *Resource Management* package for XOSAGA, presented in section 5.7.

5.2 Job Submission

The job submission functionality offered by the XtreemOS Basic API (see D3.3.2 [14]) can be expressed by the SAGA API as shown in Table 1. Therefore, only implementation efforts (i.e., coding XtreemOS adaptors for SAGA) are required.

5.2.1 Terminology and Data Structures

jobDefinition, used by the XtreemOS Basic API (see D3.3.1 [10]) as input parameter for **Create_job**, including:

- job description,

- resource requirements,
- resource hints,
- scheduling hints,
- job dependencies.

job_description is the respective class in the job management package from the SAGA API, including:

- executable,
- arguments,
- environment variables,
- standard input/output/error.

5.2.2 The API Mapping

The XtreemOS Basic API (according to [10] and [14]) is providing the following calls for job submission. We describe how they are mapped to SAGA methods, summarized in Table 1:

Create_job: creates a job according to user credentials; mapped to **saga::job_service.create_job**.

Run_job: starts an existing job; mapped to **saga::job.run**

createProcess: creates a process for a job in a resource, similarly to a Linux *fork*. We omit mapping the **createProcess** system call to the SAGA/XOSAGA API as it cannot be called by a job itself. It is called by a process that is part of the job, hence XtreemOS job execution management services will register the newly created process as part of the job. In the SAGA API context, process creation is specific to the **saga::job_description** and is taken care of by the implementation (e.g., an adaptor for XtreemOS services).

XtreemOS	SAGA
Create_job	saga::job_service.create_job
Run_job	saga::job.run (inherited from saga::task)
createProcess	not applicable

Table 1: Job Submission

The **job dependencies** field of jobDefinition (see Section 5.2.1) raises the open issue of workflows. As stated in D3.3.1 [10] (Section 3.1.1), XtreemOS will not provide a workflow management system, though it will sustain managing job dependencies. Currently, SAGA does not support any workflow definition language, but provides methods for jobs spawning other jobs, for testing and waiting for completion of other jobs, as well as for monitoring state changes of jobs. The XtreemOS feature for providing job dependencies might be mimicked by the SAGA API using the **saga::task_container** class. If later decided to add a workflow definition and management mechanism to XtreemOS, the SAGA API can be extended to express this functionality, likely as another XOSAGA package.

Aside: Assuming a job has been created outside the SAGA API (i.e., using the XtreemOS command line API), it can still be handled by the SAGA API using the **get_job** method from the **saga::job_service** class.

5.3 Job Management

The job management functionality offered by the XtreemOS Basic API (see D3.3.2 [14]) can be expressed by the SAGA API as shown in Table 2. Therefore, only implementation efforts (i.e., coding XtreemOS adaptors for SAGA) are required.

5.3.1 Terminology and Data Structures

Events, as defined by XtreemOS AEM (see D3.3.1 [10]) can be exchanged among processes. Within the application-level API, thus on job level, events can be handled using SAGA's monitoring model [4], namely the **saga::steerable** interface.

5.3.2 The API Mapping

The XtreemOS Basic API (according to [10] and [14]) is providing the following calls for job management. We describe how they are mapped to SAGA methods, summarized in Table 2:

jobControl(jobID,controlOperation,...): perform scheduling control operations on a previously created job; the control operations are:

- CHG_UID: changes user credentials for the job identified by **jobID**. In the SAGA API, credentials are expressed as *contexts* (compare also Section 4). A change of user credentials can be expressed by removing a context and adding another one (**saga::session.remove_context** and **saga::session.add_context**).

- CANCELJOB: cancels the execution of the job identified by **jobID**; mapped to **saga::job.cancel**.
- ENDJOB: ends the execution of the job identified by **jobID**; has the same mapping as **jobControl(CANCELJOB)**. According to the current AEM API from WP3.3, this call will be substituted by **exitJob**.
- SUSPENDJOB: suspends the execution of job identified by **jobID**; mapped to **saga::job.suspend**.
- RESUMEJOB: resume the execution of job identified by **jobID**; mapped to **saga::job.resume**.
- WAITFORJOB: blocks the calling job until the job identified by **jobID** has completed; mapped to **saga::job.wait**.

updateJobRequirements(jobID, reqOperation, requirementList, ...): modifies job requirements for the job identified by **jobID**; it has the following possible **reqOperation** values:

- REQMORERESOURCES: request more resources, specified by **requirementList**.
- RELEASERESOURCES: release resources specified by **requirementList**.
- MIGRATIONHINTS: provide hints about resource characteristics that could improve the job execution if available.

Ongoing work from WP3.3 [18] suggests splitting this method in two parts: one to deal with managing resources for the job identified by **jobID**, the other to provide hints for its migration/checkpointing. Final results are to appear in the advanced AEM release. They will be covered in the following XtreemOS API release.

sendEvent(jobID, ...): sends a specified event to the job specified by **jobID**; mapped to **saga::metric.fire**.

waitForEvent: this call blocks the calling process until it receives the specified event. Explicit waiting for an event can not be directly expressed in SAGA as all events are handled via callbacks that are associated with a metric. If necessary, an application will have to synchronize with its own callback function. We will reconsider adding an explicit method to wait for an event to the XOSAGA API in future API versions, if this will turn out to be necessary, based on application experience feedback.

addEventCallback(event, processID, callbackFunction, ...): adds a callback to the process associated to a specified event; mapped to **saga::metric.add_callback**.

exitJob: this call will finalize the execution of all processes belonging to the job that owns the calling process. It will be used inside the (adaptor) implementation only, as this is a process-level call. The application itself might either terminate (for successful termination) or call **saga::job.cancel** on **saga::job_self**, in case of a detected, fatal error.

attachProcess: this system call converts a UNIX process to a XtreemOS standalone job or job unit (see D3.3.1 [10], Section 3.1.4). This functionality is only needed for implementation purposes and can not be meaningfully provided on the application-level API.

XtreemOS		SAGA
jobControl		
	CHG_UID	saga::session.remove_context
		saga::session.add_context
	CANCELJOB	saga::job.cancel
	ENDJOB	saga::job.cancel
	SUSPENDJOB	saga::job.suspend
	RESUMEJOB	saga::job.resume
	WAITFORJOB	saga::job.wait
updateJobRequirements		(subject to change; will be
	REQMORERESOURCES	covered in the next
	RELEASERESOURCES	XtreemOS API release)
	MIGRATIONHINTS	
sendEvent		saga::metric.fire
waitForEvent		(implicit within callbacks)
addEventCallback		saga::metric.add_callback
exitJob		not applicable
attachProcess		not applicable

Table 2: Job Management

5.4 Job Checkpointing and Migration

Both checkpointing and migration can be expressed in the SAGA API. The XtreemOS AEM method **restartJob**, however, has no counterpart in SAGA. We therefore propose an extended job class in XOSAGA.

5.4.1 The API Mapping

The XtreemOS Basic API (according to [10] and [14]) is providing the following calls for job checkpointing and migration. We describe how they are mapped to SAGA and XOSAGA methods, summarized in Table 3:

migrateJob: migrates an existing job from a set of resources to a new one specified by a set of resource requirements; mapped to **saga::job.migrate**.

checkpointJob: checkpoint a job; mapped to **saga::job.checkpoint**. Deliverable D2.1.1 [16] makes a proposal to extend the **saga::job_description** class with checkpointing information to be used by the **saga::job_service**. We merge that proposal with our present XOSAGA proposal as presented in Section 5.7.

restartJob(jobID, ...): restarts a job context by creating a new job which will maintain its old **jobID** and re-scheduling it; though the **saga::job.resume** method would seem a straightforward mapping for **restartJob**, there is a subtle difference between their semantics. The **restartJob** method is supposed to re-create the job and re-schedule it, while maintaining its "old" context, i.e. **jobID** and user credentials. In contrast, the **saga::job.resume** method simply resumes the job's execution from the point where it was **saga::job.suspend**-ed. Therefore, we propose a XtreemOS-customized extension of the **saga::job** class, namely **xosaga::job**, adding a **restart** method.

XtreemOS	SAGA
migrateJob	saga::job.migrate
checkpointJob	saga::job.checkpoint
restartJob	xosaga::job.restart

Table 3: Job Checkpointing and Migration

5.5 Job Monitoring and Steering

Both functionalities can be expressed in the SAGA API, therefore only implementation efforts (i.e. coding XtreemOS adaptors for SAGA) are required.

5.5.1 Terminology and Data Structures

Attributes: though not clearly defined, this term is used in the sense of both static and dynamic information, related to a job (see D3.3.1 [10]). This term will be mapped to **metrics** in SAGA terminology [4]. However, both terms, **attributes** and **metrics**, are mentioned in subsection 3.2.7 of D3.3.1. The exact semantics of these terms in XtreemOS terminology needs further clarification. Modifications to the job monitoring and steering API might become necessary in future API releases, accordingly.

5.5.2 The API Mapping

The XtreemOS Basic API (according to [10] and [14]) is providing the following calls for job monitoring and steering. We describe how they are mapped to SAGA methods, summarized in Table 4:

getJobIDs(jobFilter, ...): returns the jobIDs matching the **jobFilter** parameter, which can have one the following values:

- SELF - jobID of the calling process,
- USER - jobIDs of all the jobs of the user,
- VO - jobIDs of all the jobs running in the VO,
- INSTATE - all the jobIDs in a certain state.

This method is mapped to **saga::job_service.list**.

getJobInfo(jobIDList, flags, infoLevel): returns the available information for each job in the provided list. For static information on jobs identified by **jobIDList**, this call can be mapped to **saga::job.get_job_description**. For dynamic information, we make use of the **saga::metric**'s of the jobs identified by **jobIDList**.

addAtribute(jobID, attributeName, ...): mentioned in [14], changed to **addJobAttribute** in the current AEM API (see WP3.3 Wiki [18]); adds the attribute identified by **attributeName** to the job information of the job referred to by **jobID**. Mapped to **saga::steerable.add_metric**. See also 5.5.1 for a discussion on the semantics of **attribute**'s.

addCallback(jobID, callbackFunction, metric, value, callbackEvent): mentioned in [14], changed to **addJobCallback** in the current AEM API (see WP3.3 Wiki [18]); defines a callback function **callbackFunction** associated to a metric **metric** already existent in the **jobID** job information. **callbackEvent** specifies when the job should be notified:

- CHANGE,
- EQUALTO,
- GREATERTHAN,
- LESSTHAN

with respect to a specified **value**. Mapped to **saga::monitorable.add_callback**. See also 5.5.1 for a discussion on the semantics of **metric**.

monitoringControl: mentioned in [14], changed to **jobMonitoringControl** in current AEM API (see WP3.3 Wiki [18]); starts, stops or changes level of monitoring. Mapped to **saga::job.add/remove_callback**. According to the current status of the AEM API (see WP3.3 Wiki [18]), the **monitoringControl** method needs clear defined semantics for `CHANGELEVEL` functionality. The application-level API definition might change in future releases, accordingly.

XtreemOS		SAGA
getJobIds		saga::job_service.list
getJobInfo		saga::job.get_job_description
		saga::metric.get_value
addAttribute		saga::steerable.add_metric
addCallback		saga::monitorable.add_callback
		saga::metric.add_callback
monitoringControl		
	START	saga::job.add_callback
	STOP	saga::job.remove_callback
	CHANGELEVEL	(subject to change; will be covered in future XtreemOS API releases)

Table 4: Job Monitoring and Steering

5.6 Resource Management

XtreemOS proposes resource management functionalities in D3.3.2 [14]. Specifically, finding/matching compute resources according to a resource description, and reserving such resources is provided, independent of the execution of a given application.

No such functionality is available in SAGA. On purpose, SAGA restricts itself to job submission and execution, abstracting from resource management, for the sake of application simplicity. In SAGA, resource requirements are expressed inside **saga::job_description** objects, along with the details of the application, like the program binary, etc. Resource management and brokering issues are dealt with inside the implementation of a **saga::job_service** object.

Because resource management is an important functionality of XtreemOS, and because there are important use cases that separate resource matching and reservation from job submission, we are providing a resource management package within the XOSAGA API. Its main features are:

- Introduce classes for resources and reservations.
- Split SAGA's **job_description** in a **resource_description** and an **application_description**.
- Add a **resource_service** for matching and reserving resources.
- Extend the **job_service** by methods to submit jobs either to a set of resources or to a given reservation.

Details will be presented in Section 5.7. The XOSAGA resource management package may, in future releases, also be extended to provide resource monitoring functionalities, as the client-side AEM API is currently looking into this as well (see XtreemOS wiki:WP3.3 [18]). Whether and to which extent resource monitoring should be included in an application-level API, however, has to be decided later in the project.

5.6.1 The API Mapping

The XtreemOS Basic API (according to [10] and [14]) is providing the following calls for resource management. We describe how they are mapped to XOSAGA methods, summarized in Table 5:

XtreemOS	XOSAGA
resMatching	xosaga::resource_service.discover
setReservation	xosaga::resource_service.reserve
rmReservation	xosaga::resource_service.cancel

Table 5: Resource Management

5.7 The XOSAGA Resource Management Package

Both resource management (Section 5.6) and the feature for restarting a job (Section 5.4) require extensions to the existing SAGA API, for which we are providing the XOSAGA resource management extension package to SAGA. It consists of eight classes, partially extending existing SAGA classes, partially implementing existing SAGA interfaces. The relationships between the new XOSAGA classes and the "old" SAGA classes and interfaces is shown in Figure 4. We specify the XOSAGA classes in the following.

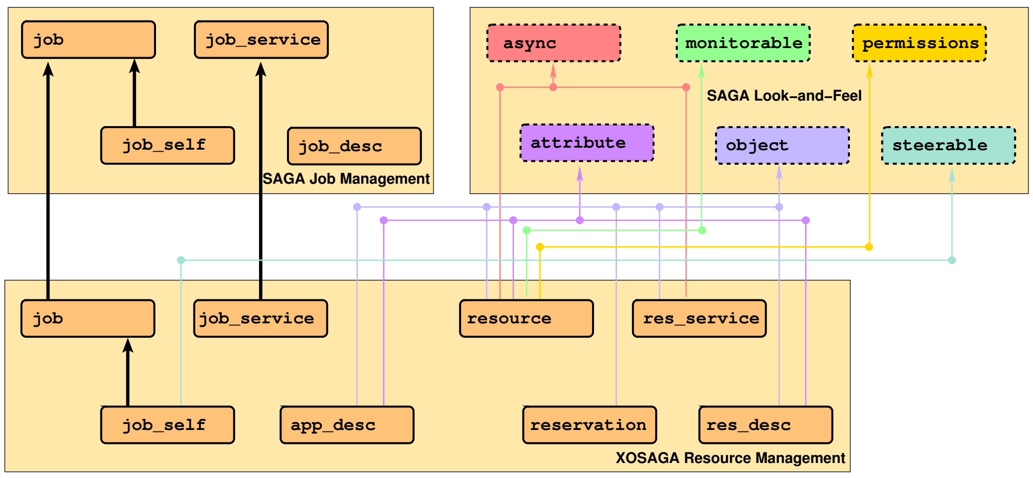

Figure 4: The relations between the XOSAGA resource management package and the existing SAGA classes and interfaces.

5.7.1 Class `xosaga::job`

The `job` class inherits from `saga::job` and adds a method for restart.

```
class xosaga::job :  extends saga::job
{
    // no CONSTRUCTOR

    DESTRUCTOR           (in  xosaga::job obj);
    //Purpose:   destroy the object

    restart(void);
    //Purpose:   Restart the job from the beginning; possibly
    //           on a differen resource, however, maintaining
    //           its job id and user credentials.
    //Inputs:   -
    //InOuts:   -
    //Outputs:  -
    //PreCond:  -
    //PostCond: the job is in 'Running' state.
    //Perms:    Exec (job can be controlled).
    //Throws:   NotImplemented
    //          PermissionDenied
    //          AuthorizationFailed
    //          Timeout
    //          NoSuccess
}
```

5.7.2 Class `xosaga::job_self`

The `job_self` class inherits from `xosaga::job`. This replicates the setup in the SAGA job management package. This way, a job can also restart itself.

```
class xosaga::job_self : extends     xosaga::job
                         implements  saga::steerable
                      // from job    saga::async
                      // from job    saga::attributes
                      // from job    saga::task
                      // from job    saga::object
                      // from job    saga::monitorable
                      // from job    saga::permissions
                      // from job    saga::error_handler
{
   // no CONSTRUCTOR

   DESTRUCTOR           (in  xosaga::job_self      obj);
   //Purpose:   destroy the object
}
```

5.7.3 Class `xosaga::resource_description`

The `resource_description` class is collecting those attributes from SAGA's `job_description` class that are related to selecting suitable resources.

```
class xosaga::resource_description
                            : implements saga::object
                              implements saga::attributes
{
```

```
CONSTRUCTOR                (out resource_description obj);
//Purpose:   create the object
//Inputs:    -
//Outputs:   obj: the newly created object
//PreCond:   -
//Postcond:  -
//Perms:     -
//Throws:    NotImplemented
//           NoSuccess

DESTRUCTOR                 (in resource_description obj);
//Purpose:   destroy the object

//Attributes:
//   name:  TotalCPUCount
//   desc:  total number of cpus to be provided
//   mode:  ReadWrite, optional
//   type:  Int
//   value: '1'
//   notes: - semantics as defined in JSDL
//          - available in JSDL, DRMAA
//
//   name:  TotalPhysicalMemory
//   desc:  Estimated amount of memory to be provided
//   mode:  ReadWrite, optional
//   type:  Float
//   value: -
//   notes: - unit is in MegaByte
//          - memory usage of the job is aggregated
//            across all processes of the job
//          - semantics as defined by JSDL
//          - available in JSDL
//
//   name:  CPUArchitecture
//   desc:  compatible processor for job submission
//   mode:  ReadWrite, optional
//   type:  Vector String
//   value: -
//   notes: - allowed values as specified in JSDL
//          - semantics as defined by JSDL
//          - available in JSDL
//
//   name:  OperatingSystemType
```

```
   //   desc:  compatible operating system for job submission
   //   mode:  ReadWrite, optional
   //   type:  Vector String
   //   value: -
   //   notes: - allowed values as specified in JSDL
   //          - semantics as defined by JSDL
   //          - available in JSDL
   //
   //   name:  CandidateHosts
   //   desc:  list of host names which are to be considered
   //          by the resource manager as candidate targets
   //   mode:  ReadWrite, optional
   //   type:  Vector String
   //   value: -
   //   notes: - semantics as defined by JSDL
   //          - available in JSDL
   //
   //   name:  Queue
   //   desc:  name of a queue to place the job into
   //   mode:  ReadWrite, optional
   //   type:  String
   //   value: -
   //   notes: - While SAGA itself does not define the
   //            semantics of a "queue", many backend systems
   //            can make use of this attribute.
   //          - not supported by JSDL
}
```

5.7.4 Class **`xosaga::resource`**

The `resource` class is a container for the information identifying a compute resource. Currently, it has only a single method for retrieving its resource description. Future XOSAGA versions may add more methods to this class.

```
class xosaga::resource
                : implements saga::object
                  implements saga::async
                  implements saga::attributes
                  implements saga::permissions
                  implements saga::monitorable
{
   //no CONSTRUCTOR
```

```
    DESTRUCTOR                (in xosaga::resource obj);
    //Purpose:   destroy the object

    get_resource_description (out xosaga::resource_description rd);
    // Purpose:  Retrieve the resource_description which was used to
    //           discover this resource instance.
    // Inputs:   -
    // InOuts:   -
    // Outputs:  rd:              a resource_description object
    // PreCond:  -
    // PostCond: - rd is deep copied (no state is shared
    //             after method invocation)
    // Perms:    Query
    // Throws:   NotImplemented
    //           DoesNotExist
    //           PermissionDenied
    //           AuthorizationFailed
    //           AuthenticationFailed
    //           Timeout
    //           NoSuccess
    // Notes:    - There are cases when the resource_description
    //             is not available.  This may include cases when
    //             the resource is one of many identified via a
    //             resource description, and a description for
    //             the individual resource can not be contructed.
    //             In this case, a 'DoesNotExist' exception is
    //             thrown, with a descriptive error message.
}
```

5.7.5 Class `xosaga::reservation`

The `reservation` class is a container for the information identifying a reservation. Like jobs, reservations have different states, shown in Figure 5. When constructed (by a resource service), a reservation can either be in state *New*, or in state *Running*. *New* denotes that the start time of the reservation has not yet been reached. *Running* denotes that the resource(s) reserved by the reservation are currently accessible, this means time at the resource(s) is within the interval *start time* and *expiration time*. Once time has reached the expiration time, the reservation's state changes to *Done*. The state *Canceled* can be reached from the state *Running* either by invoking the `cancel` method on the reservation object, or by external cancelation, for example by the remote resource itself, or by a resource broker service.

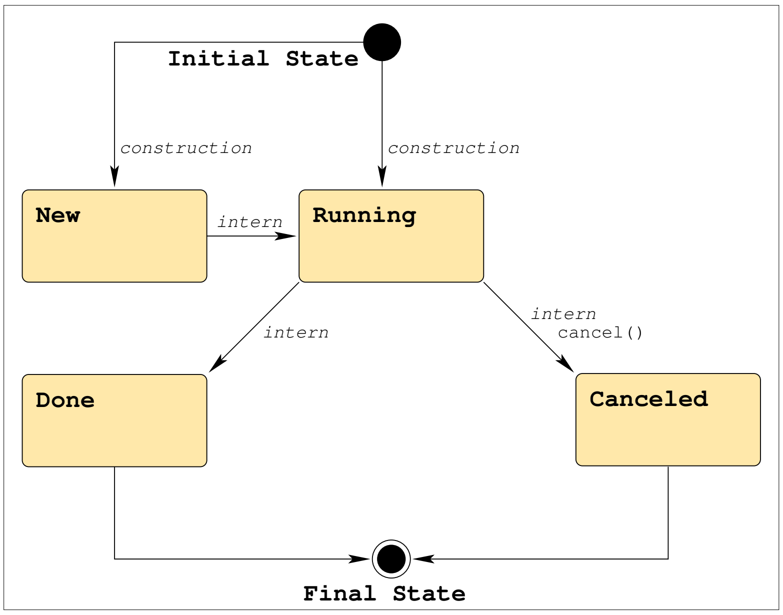

Figure 5: The XOSAGA reservation state model.

```
enum state
{
   New         =  1,  // same as in saga::task::state
   Running     =  2,  // same as in saga::task::state
   Done        =  3,  // same as in saga::task::state
   Canceled    =  4,  // same as in saga::task::state
}

class xosaga::reservation : implements saga::object
{
   // no CONSTRUCTOR

   DESTRUCTOR                 (in xosaga::reservation obj);
   //Purpose:   destroy the object

   //Attributes
   //    name:  CreationTime
   //    desc:  time stamp of the reservation creation in
   //           the resource manager
   //    mode:  Read, optional
   //    type:  Int
   //    value: -
```

```
  //   notes: - format: number of seconds since epoch
  //
  //   name:  Starttime
  //   desc:  time stamp indicating when
  //          the reservation starts
  //   mode:  Read
  //   type:  Int
  //   value: -
  //   notes: - format: number of seconds since epoch
  //
  //   name:  ExpirationTime
  //   desc:  time stamp indicating when
  //          the reservation ends
  //   mode:  Read
  //   type:  Int
  //   value: -
  //   notes: - format: number of seconds since epoch

  get_state   (out state state);
  //Purpose:  Get the state of the task.
  //Inputs:   -
  //InOuts:   -
  //Outputs:  state:                state of the reservation.
  //PreCond:  -
  //PostCond: -
  //Perms:    -
  //Throws:   NotImplemented
  //          Timeout
  //          NoSuccess
  //Notes:    - a 'Timeout' or 'NoSuccess' exception indicates
  //            that the backend was not able to retrieve the
  //            reservation state.

  get_resources              (out array<resource> reserved);
  // Purpose:  Get the reserved resources.
  // Inputs:   -
  // InOuts:   -
  // Outputs:  reserved:      an array of resources
  // PreCond:  -
  // PostCond: -
  // Perms:    -
  // Throws:   NotImplemented
}
```

5.7.6 Class `xosaga::resource_service`

The class `resource_service` is modeled after SAGA's job service. Its constructor has parameters describing a possible back-end resource broker. Further, it has methods for discovering resources according to a resource description, for reserving resources, either from resource id's, or directly from a resource description. Reservations can explicitly be canceled. The `list` method lists all active reservations of the resource service. For completeness, `get_reservation` and `get_resource` map id's to their respective container objects.

```
class resource_service : implements saga::object
                         implements saga::async
{
  CONSTRUCTOR                (in  session          e,
                              in  string           rm = "",
                              out resource_service obj);
  // Purpose:  create the object
  // Inputs:   s:                 session to associate with
  //                              the object
  //           rm:                contact url for resource
  //                              manager
  // InOuts:   -
  // Outputs:  obj:               the newly created object
  // PreCond:  -
  // PostCond: -
  // Perms:    -
  // Throws:   NotImplemented
  //           IncorrectURL
  //           PermissionDenied
  //           AuthorizationFailed
  //           AuthenticationFailed
  //           Timeout
  //           NoSuccess
  // Notes:    - 'rm' defaults to an empty string - in that
  //             case, the implementation must perform a
  //             resource discovery, or fall back to a fixed
  //             value, or find a valid rm contact in any
  //             other way.  If that is not possible, a
  //             'BadParameter' exception MUST be thrown, and
  //             MUST indicate that a rm contact string is
  //             needed.  The expected behaviour MUST be
  //             documented (i.e. if a default is available).
  //           - if the rm identified by the rm URL cannot be
  //             contacted (i.e. does not exist), a
  //             'BadParameter' exception is thrown.

  DESTRUCTOR                 (in  resource_service obj);
  // Purpose:  destroy the object
  // Inputs:   obj:               the object to destroy
  // InOuts:   -
  // Outputs:  -
  // PreCond:  -
  // PostCond: - reservations created by this resource_service
  //             instance are not affected by the destruction,
  //             and are in particular not canceled.
  // Perms:    -
  // Throws:   -

  discover                   (in  resource_description rd,
                              out array<string> resource_ids);
  // Purpose:  discover resources matching the resource
  //           description
  // Inputs:   rd:                description of resource to be
  //                              discovered
  // InOuts:   -
  // Outputs:  resource_ids:      an array of resource identifiers
  // PreCond:  -
  // PostCond: - rd is deep copied (no state is shared
  //             after method invocation)
  // Perms:    -
  // Throws:   NotImplemented
  //           BadParameter
  //           PermissionDenied
  //           AuthorizationFailed
  //           AuthenticationFailed
  //           Timeout
  //           NoSuccess

  reserve                    (in  resource_description rd,
                              in  int start_time,
                              in  int expiration_time,
                              out reservation reserved);
  // Purpose:  reserve the resources matching the resource
  //           description
  // Inputs:   rd:                description of resource to be
  //                              reserved
  //           start_time:        requested start of reservation,
  //                             in number of seconds since epoch
  //            expiration_time:   requested expiration of reservation,
  //                             in number of seconds since epoch
  // InOuts:    -
  // Outputs:   reserved:       a resource object representing
  //                            the reserved resource
  // PreCond:   -
  // PostCond:  - rd is deep copied (no state is shared
  //              after method invocation)
  // Perms:     -
  // Throws:    NotImplemented
  //            BadParameter
  //            PermissionDenied
  //            AuthorizationFailed
  //            AuthenticationFailed
  //            Timeout
  //            NoSuccess
  // Notes:     - if the resource description contains values
  //              which are outside of the allowed range, or
  //              cannot be parsed, or are otherwise invalid
  //              and not usable for creating a resource instance,
  //              a 'BadParameter' exception is thrown, which MUST
  //              indicate which attribute(s) caused this
  //              exception, and why.
  //            - if the reservation fails because no matching
  //              resources are available in the requested time
  //              interval, a 'NoSuccess' exception MUST be thrown,
  //              which MUST indicate the failure.
  //            - An implementation MAY use default values for start
  //              time and expiration time (like ``as soon as
  //              possible,'' and ``15 minutes duration'') and MAY
  //              deviate from the requested time interval. An
  //              implementation MUST document such behavior.

  reserve                    (in  array<string> resource_ids,
                              in  int start_time,
                              in  int expiration_time,
                              out reservation  reserved);
  // Purpose:   reserve the resources identified by their resource
  //            ids
  // Inputs:    resource_ids:    array of resource ids
  //            start_time:      requested start of reservation,
  //                             in number of seconds since epoch
  //           expiration_time: requested expiration of reservation,
  //                            in number of seconds since epoch
  // InOuts:   -
  // Outputs:  reserved:      a resource object representing
  //                          the reserved resource
  // PreCond:  -
  // PostCond: - rd is deep copied (no state is shared
  //             after method invocation)
  // Perms:    -
  // Throws:   NotImplemented
  //           BadParameter
  //           PermissionDenied
  //           AuthorizationFailed
  //           AuthenticationFailed
  //           Timeout
  //           NoSuccess
  // Notes:    - if any of the resource ids are invalid,
  //             a 'BadParameter' exception is thrown, which MUST
  //             indicate which id(s) caused this exception.
  //           - if the reservation fails because some identified
  //             resources are unavailable in the requested time
  //             interval, a 'NoSuccess' exception MUST be thrown,
  //             which MUST indicate the failure. In this case,
  //             no resource will be reserved at all.
  //           - An implementation MAY use default values for start
  //             time and expiration time (like ``as soon as
  //             possible,'' and ``15 minutes duration'') and MAY
  //             deviate from the requested time interval. An
  //             implementation MUST document such behavior.

  cancel                     (in reservation res,
                              in  float  timeout);
  // Purpose:  cancel the reservation
  // Inputs:   res:          the reservation
  //           timeout:      time for freeing resources
  // InOuts:   -
  // Outputs:  -
  // PreCond:  - reservation is in state 'New' or 'Running'.
  // PostCond: - reservation is in 'Canceled' state.
  // Perms:    -
  // Throws:   NotImplemented
  //           IncorrectState
  //           Timeout
  //           NoSuccess
  // Notes:    - for resource deallocation semantics, see
  //             Section 2 of the SAGA specification.
  //           - if cancel() fails to cancel the reservation
  //             immediately, and tries to continue to cancel
  //             the reservation in the background, the reservation
  //             state remains 'Running' until the cancel operation
  //             succeeded.  The state then changes to
  //             'Canceled'.
  //           - if the reservation is in the 'Done' state, the call
  //             has no effect, and, in particular, does NOT change
  //             the state to 'Canceled'.
  //             This is to avoid race conditions.
  //           - a 'NoSuccess' exception indicates
  //             that the backend was not able to initiate the
  //             cancelation for the reservation.
  //           - for timeout semantics, see Section 2 of the SAGA
  //             specification.

  list         (out array<string> reservation_ids);
  // Purpose:  Get a list of reservations that are currently known
  //           by the resource manager.
  // Inputs:   -
  // InOuts:   -
  // Outputs:  reservation_ids:       an array of
  //                                  reservation identifiers
  // PreCond:  -
  // PostCond: -
  // Perms:    Query on reservations identified by the returned ids
  // Throws:   NotImplemented
  //           PermissionDenied
  //           AuthorizationFailed
  //           AuthenticationFailed
  //           Timeout
  //           NoSuccess
  // Notes:    - which reservations are viewable by the calling user
  //             context, and how long a resource manager keeps
  //             reservation information, are both implementation
  //             dependent.
  //           - a returned reservation_id may translate into a
  //             reservation (via get_reservation()), which is not
  //             controllable by the requesting application (e.g.
  //             it could cause an 'AuthorizationFailed' exception).
```

```
get_reservation            (in  string   reservation_id,
                            out reservation res);
// Purpose:  Given a reservation identifier, this method returns
//           a reservation object representing this reservation.
// Inputs:   reservation_id: reservation identifier as returned
//                           by the resource manager
// InOuts:   -
// Outputs:  reservation:    a reservation object representing
//                           the reservation identified by
//                           reservation_id
// PreCond:  - reservation identified by reservation_id is
//             managed by the resource_service.
// PostCond: -
// Perms:    Query on the reservation.
// Throws:   NotImplemented
//           BadParameter
//           DoesNotExist
//           PermissionDenied
//           AuthorizationFailed
//           AuthenticationFailed
//           Timeout
//           NoSuccess
// Notes:    - in general, only a resource_service representing
//             the resource manager which made the reservation
//             may be able to handle the reservation_id, and to
//             identify the reservation -- however, other
//             resource_services may succeed as well.
//           - if the resource manager can handle the
//             reservation_id, but the referenced reservation
//             is not alive, a 'DoesNotExist' exception is thrown.
//           - if the resource manager cannot parse the
//             reservation_id at all, a 'BadParameter' exception
//             is thrown.

get_resource               (in  string   resource_id,
                            out resource res);
// Purpose:  Given a resource identifier, this method returns
//           a resource object representing this resource.
// Inputs:   resource_id: resource identifier as returned
//                        by the resource manager
// InOuts:   -
// Outputs:  resource:    a resource object representing
```

```
   //                        the resource identified by
   //                        resource_id
   // PreCond:  - resource identified by resource_id is
   //             managed by the resource_service.
   // PostCond: -
   // Perms:    Query on the resource.
   // Throws:   NotImplemented
   //           BadParameter
   //           DoesNotExist
   //           PermissionDenied
   //           AuthorizationFailed
   //           AuthenticationFailed
   //           Timeout
   //           NoSuccess
   // Notes:    - in general, only a resource_service representing
   //             the resource manager which discovered the resource
   //             may be able to handle the resource_id, and to
   //             identify the resource -- however, other
   //             resource_services may succeed as well.
   //           - if the resource manager can handle the
   //             resource_id, but the referenced resource
   //             is not alive, a 'DoesNotExist' exception is thrown.
   //           - if the resource manager cannot parse the
   //             resource_id at all, a 'BadParameter' exception
   //             is thrown.
 }
```

5.7.7 Class **xosaga::application_description**

The application_description class is collecting those attributes from SAGA's job_description class that are related to the application itself, augmented by the attributes for checkpointing from D2.1.1 [16].

```
class xosaga::application_description
                              : implements saga::object
                                implements saga::attributes
{
   CONSTRUCTOR            (out application_description obj);
   //Purpose:   create the object
   //Inputs:    -
   //Outputs:   obj: the newly created object
   //PreCond:   -
   //Postcond:  -
```

```
//Perms:      -
//Throws:     NotImplemented
//            NoSuccess

DESTRUCTOR              (in application_description obj);
//Purpose:    destroy the object

//Attributes:
//    name:  Executable
//    desc:  command to execute.
//    type:  String
//    mode:  ReadWrite
//    value: ''
//    notes: - this is the only required attribute.
//           - can be a full pathname, or a pathname
//             relative to the 'WorkingDirectory' as
//             evaluated on the execution host.
//           - semantics as defined in JSDL
//           - available in JSDL, DRMAA
//
//    name:  Arguments
//    desc:  positional parameters for the command.
//    mode:  ReadWrite, optional
//    type:  Vector String
//    value: -
//    notes: - semantics as specified by JSDL
//           - available in JSDL, DRMAA
//
//    name:  SPMDVariation
//    desc:  SPMD job type and startup mechanism
//    mode:  ReadWrite, optional
//    type:  String
//    value: -
//    notes: - as defined in the SPMD extension of JSDL
//    notes: - semantics as defined in JSDL
//           - available in JSDL, SPMD extension
//           - the SPMD JSDL extension defines the value
//             to be an URI.  For simplicity, SAGA allows
//             the following strings, which map into the
//             respective URIs: MPI, GridMPI, IntelMPI,
//             LAM-MPI, MPICH1, MPICH2, MPICH-GM, MPICH-MX,
//             MVAPICH, MVAPICH2, OpenMP, POE, PVM, None
//           - the value 'Empy' (default) indicates that
//            the application is not a SPMD application.
//          - as JSDL, SAGA allows other arbitrary values.
//            The implementation must clearly document
//            which values are supported.
//
//   name:  NumberOfProcesses
//   desc:  total number of processes to be started
//   mode:  ReadWrite, optional
//   type:  Int
//   value: '1'
//   notes: - semantics as defined in JSDL
//          - available in JSDL, SPMD extension
//
//   name:  ProcessesPerHost
//   desc:  number of processes to be started per host
//   mode:  ReadWrite, optional
//   type:  Int
//   value: '1'
//   notes: - semantics as defined in JSDL
//          - available in JSDL, SPMD extension
//
//   name:  ThreadsPerProcess
//   desc:  number of threads to start per process
//   mode:  ReadWrite, optional
//   type:  Int
//   value: '1'
//   notes: - semantics as defined in JSDL
//          - available in JSDL, SPMD extension
//
//   name:  Environment
//   desc:  set of environment variables for the job
//   mode:  ReadWrite, optional
//   type:  Vector String
//   value: -
//   notes: - exported into the job environment
//          - format: 'key=value'
//          - semantics as specified by JSDL
//          - available in JSDL, DRMAA
//
//   name:  WorkingDirectory
//   desc:  working directory for the job
//   mode:  ReadWrite, optional
//   type:  String
//   value: '.'
//   notes: - semantics as specified by JSDL
//          - available in JSDL, DRMAA
//
//   name:  Interactive
//   desc:  run the job in interactive mode
//   mode:  ReadWrite, optional
//   type:  Bool
//   value: 'False'
//   notes: - this implies that stdio streams will stay
//            connected to the submitter after job
//            submission, and during job execution.
//          - if an implementation cannot handle
//            interactive jobs, and this attribute is
//            present, and 'True', the job creation MUST
//            throw an 'IncorrectParameter' error with a
//            descriptive error message.
//          - not supported by JSDL, DRMAA
//
//   name:  Input
//   desc:  pathname of the standard input file
//   mode:  ReadWrite, optional
//   type:  String
//   value: -
//   notes: - semantics as specified by JSDL
//          - available in JSDL, DRMAA
//          - will not be used if 'Interactive' is 'True'
//
//   name:  Output
//   desc:  pathname of the standard output file
//   mode:  ReadWrite, optional
//   type:  String
//   value: -
//   notes: - semantics as specified by JSDL
//          - available in JSDL, DRMAA
//          - will not be used if 'Interactive' is 'True'
//
//   name:  Error
//   desc:  pathname of the standard error file
//   mode:  ReadWrite, optional
//   type:  String
//   value: -
//   notes: - semantics as specified by JSDL
//          - available in JSDL, DRMAA
//          - will not be used if 'Interactive' is 'True'
//
//   name:  FileTransfer
//   desc:  a list of file transfer directives
//   mode:  ReadWrite, optional
//   type:  Vector String
//   value: -
//   notes: - translates into jsdl:DataStaging
//          - used to specify pre- and post-staging
//          - semantics as specified in JSDL
//          - staging is part of the 'Running' state
//          - syntax similar to LSF (see earlier notes)
//          - available in JSDL, DRMAA
//
//   name:  Cleanup
//   desc:  defines if output files get removed after the
//          job finishes
//   mode:  ReadWrite, optional
//   type:  String
//   value: 'Default'
//   notes: - can have the Values 'True', 'False', and
//            'Default'
//          - On 'False', output files MUST be kept
//            after job the finishes
//          - On 'True', output files MUST be deleted
//            after job the finishes
//          - On 'Default', the behaviour is defined by
//            the implementation or the backend.
//          - translates into 'DeleteOnTermination' elements
//            in JSDL
//
//   name:  JobStartTime
//   desc:  time at which a job should be scheduled
//   mode:  ReadWrite, optional
//   type:  Int
//   value: -
//   notes: - Could be viewed as a desired job start
//            time, but that is up to the resource
//            manager.
//          - format: number of seconds since epoch
//          - available in DRMAA
//          - not supported by JSDL
```

```
//
//   name:  TotalCPUTime
//   desc:  estimate total number of CPU seconds which
//          the job will require
//   mode:  ReadWrite, optional
//   type:  Int
//   value: -
//   notes: - intended to provide hints to the scheduler.
//          - available in JSDL, DRMAA
//          - semantics as defined in JSDL
//
//   name:  JobContact
//   desc:  set of endpoints describing where to report
//          job state transitions.
//   mode:  ReadWrite, optional
//   type:  Vector String
//   value: -
//   notes: - format: URI (e.g. fax:+123456789,
//            sms:+123456789, mailto:joe@doe.net).
//          - available in DRMAA
//          - not supported by JSDL
//
// name: CheckpointPeriodicity
// desc: how frequently should the job be checkpointed,
//       in seconds
// type: Int
// mode: ReadWrite, optional
// notes: - a value of 0 means no periodic checkpointing
//        - default value is implementation dependant
//        - proposed by D2.1.1
//
// name: NumberOfKeptCheckpoints
// desc: how many checkpoints should be kept for this job
// type: Int
// mode: ReadWrite, optional
// value: '1'
// notes: - proposed by D2.1.1
//
// name: FinalStorage
// desc: set of pathnames to use to store the checkpoint
// type: Vector string
// mode: ReadWrite, optional
// value: -
```

```
   // notes: - if no path if given, a default path will be
   //        selected by the System Checkpointer,
   //        presumably on the local node
   //        - proposed by D2.1.1
   //
   // name: CheckpointPolicy
   // desc: how the checkpoint is produced
   // type: Vector string
   // mode: ReadWrite, optionnal
   // value: -
   // notes: - if no policy is given, a default policy
   //          will be chosen
   //        - If more than one policy is given, the
   //          first policy available for the checkpoint
   //          service will be used
   //        - possible CheckpointPolicies include
   //           Safe: the checkpoint file is completly
   //             written before the checkpoint call
   //             returns
   //           LocalFirst: the checkpoint file is written
   //             locally before end of system checkpoint
   //             and moved to its final destination later
   //           MemoryFirst: the checkpoint is saved in
   //             memory at the end of the system
   //             checkpoint and moved to its final
   //             destination later
   //        - proposed by D2.1.1
}
```

5.7.8 Class `xosaga::job_service`

The class `job_service` is extending SAGA's job service. It adds three methods for creating jobs using an `application_description`, in combination with a `resource_description`, or with a `reservation_id`, or with an array of `resource_id`'s.

```
class job_service : extends saga::job_service
{
   CONSTRUCTOR       (in  session       s,
                      in  url           rm = "",
                      out job_service   obj)
   // Purpose:  create the object
   // Inputs:   s:                  session to associate with
   //                              the object
   //            rm:               contact url for resource
   //                              manager
   // InOuts:    -
   // Outputs:   obj:              the newly created object
   // PreCond:   -
   // PostCond:  -
   // Perms:     -
   // Throws:    NotImplemented
   //            IncorrectURL
   //            PermissionDenied
   //            AuthorizationFailed
   //            AuthenticationFailed
   //            Timeout
   //            NoSuccess
   // Notes:     - 'rm' defaults to an empty string - in that
   //              case, the implementation must perform a
   //              resource discovery, or fall back to a fixed
   //              value, or find a valid  rm contact in any
   //              other way.  If that is not possible, a
   //              'BadParameter' exception MUST be thrown, and
   //              MUST indicate that a rm contact string is
   //              needed.  The expected behaviour MUST be
   //              documented (i.e. if a default is available).
   //            - if the rm identified by the rm URL cannot be
   //              contacted (i.e. does not exist), a
   //              'BadParameter' exception is thrown.

   DESTRUCTOR    (in  job_service   obj)
   // Purpose:   destroy the object
   // Inputs:    obj:              the object to destroy
   // InOuts:    -
   // Outputs:   -
   // PreCond:   -
   // PostCond:  - jobs created by this job_service instance
   //              are not affected by the destruction, and are
   //              in particular not canceled.
   // Perms:     -
   // Throws:    -
   // Notes:     -

   create_job                  (in  application_description ad,
                               in  resource_description rd,
                               out job job);
   // Purpose:  create a job instance
   // Inputs:   ad:                description of application
   //                              to be submitted
   //           rd:                description of resource
   //                              required for the job
   // InOuts:   -
   // Outputs:  job:               a job object representing
   //                              the submitted job instance
   // PreCond:  - ad has an 'Executable' attribute.
   // PostCond: - job is in 'New' state
   //           - ad and rd are deep copied (no state is shared
   //             after method invocation)
   //           - 'Owner' of the job is the id of the context
   //             used for creating the job.
   // Perms:    -
   // Throws:   NotImplemented
   //           BadParameter
   //           PermissionDenied
   //           AuthorizationFailed
   //           AuthenticationFailed
   //           Timeout
   //           NoSuccess
   // Notes:    - calling run() on the job will submit it to
   //             the resource, and advance its state.
   //           - if the application description does not have a
   //             valid 'Executable' attribute, a 'BadParameter'
   //             exception is thrown.
   //           - if the application or resource descriptions contain
   //             values that are outside of the allowed range, or
   //             cannot be parsed, or are otherwise invalid and not
   //             usable for creating a job instance, a
   //             'BadParameter' exception is thrown, which MUST
   //             indicate which attribute(s) caused this
   //             exception, and why.

   create_job                  (in  application_description ad,
                               in  array<string> resource_ids,
                               out job job);
   // Purpose:  create a job instance
   // Inputs:   ad:                description of application
   //                              to be submitted
   //           resource_ids:      identifications for the resources
   //                             provided to the job
   // InOuts:   -
   // Outputs:  job:              a job object representing
   //                             the submitted job instance
   // PreCond:  - ad has an 'Executable' attribute.
   // PostCond: - job is in 'New' state
   //           - ad is deep copied (no state is shared
   //             after method invocation)
   //           - 'Owner' of the job is the id of the context
   //             used for creating the job.
   // Perms:    -
   // Throws:   NotImplemented
   //           BadParameter
   //           PermissionDenied
   //           AuthorizationFailed
   //           AuthenticationFailed
   //           Timeout
   //           NoSuccess
   // Notes:    - calling run() on the job will submit it to
   //             the resource, and advance its state.
   //           - if the application description does not have a
   //             valid 'Executable' attribute, a 'BadParameter'
   //             exception is thrown.
   //           - if the application description contains
   //             values that are outside of the allowed range, or
   //             cannot be parsed, or are otherwise invalid and not
   //             usable for creating a job instance, a
   //             'BadParameter' exception is thrown, which MUST
   //             indicate which attribute(s) caused this
   //             exception, and why.
   //           - if one or more resource_ids are invalid, a
   //             'BadParameter' exception is thrown, which MUST
   //             indicate which resource_id(s) caused this
   //             exception, and why.

   create_job                 (in  application_description ad,
                               in  string reservation_id,
                               out job job);
   // Purpose:  create a job instance
   // Inputs:   ad:               description of application
   //                             to be submitted
   //           resource_ids:     identification for a reservation
   //                             holding resources provided to
```

```
  //                             the job
  // InOuts:    -
  // Outputs:   job:             a job object representing
  //                             the submitted job instance
  // PreCond:   - ad has an 'Executable' attribute.
  // PostCond:  - job is in 'New' state
  //            - ad is deep copied (no state is shared
  //              after method invocation)
  //            - 'Owner' of the job is the id of the context
  //              used for creating the job.
  // Perms:     -
  // Throws:    NotImplemented
  //            BadParameter
  //            PermissionDenied
  //            AuthorizationFailed
  //            AuthenticationFailed
  //            Timeout
  //            NoSuccess
  // Notes:     - calling run() on the job will submit it to
  //              the resource, and advance its state.
  //            - if the application description does not have a
  //              valid 'Executable' attribute, a 'BadParameter'
  //              exception is thrown.
  //            - if the application description contains
  //              values that are outside of the allowed range, or
  //              cannot be parsed, or are otherwise invalid and not
  //              usable for creating a job instance, a
  //              'BadParameter' exception is thrown, which MUST
  //              indicate which attribute(s) caused this
  //              exception, and why.
  //            - if the reservation_id is invalid, a
  //              'BadParameter' exception is thrown.
}
```

6 XtreemFS

From the application's point of view, the XtreemFS file system is providing two kinds of functionality:

1. Provide access to remote files via a local, proxy file system (using FUSE and Linux VFS). XtreemFS file systems are organized in *volumes* that are identified based on URLs.

Using the (client local) XtreemFS access layer, a XtreemFS volume can be mounted into the client machine's local file system. After mounting has succeeded, files can be accessed via the POSIX file API to local files.

This setup is suited very well for integration in the SAGA-based API used by XtreemOS. What actually needs to be exposed to the application is access to files identified by their URL, as already covered by SAGA's *File Management* package; no API extension is needed. The XtreemFS access layer, however, exposes the local file system mounting to the application, requiring a small but additional management overhead that should be avoided. We therefore suggest to provide access to XtreemFS volumes via a special XtreemFS adaptor for the SAGA engine that performs automounting functionality, thus hiding the management of local volume mount points.

2. Provide access to replicated volumes, via the client-local access layer to the XtreemFS Metadata and Replica Catalog (MRC), the Directory Service, and the Replica Management Service (RMS).

   As of the current state of the project (month 18), volume replication is not yet available in the current prototype implementation. As such, a detailed analysis on providing application-level access can not yet be performed. However, SAGA's *Replica Management* package provides an interface to replicated files and directories. We thus expect that API extensions for volume replication will not become necessary. Details will be provided likely with the third XtreemOS API specification.

7 XtreemOS functionality not covered by this API specification

The main purpose of this (second draft) API specification is to provide access to XtreemOS-specific functionality for applications. This concerns all work packages from SP2 and SP3, as far as these work packages provide relevant functionality and as far as their work progress allows to provide an API at this project stage. In this section, we list all work packages from SP2 and SP3 and their provided functionality. Doing so, we explain which functionality can not be covered by an API right now.

WP2.1 provides node-level VO support in Linux-XOS. Covered.

WP2.2 provides single system image for clusters (LinuxSSI-XOS), most importantly checkpoint/restart functionality. Covered.

WP2.3
- Mobile IPv6 to XtreemOS-MD. This does not provide any *functionality* that could be covered by an API.
- VO support in Linux-XOS for mobile devices. Covered (along with WP2.1).

WP3.1 provides the API to XtreemOS (this work).

WP3.2
- Distributed Server, Virtual Nodes, and Application Bootstrapping:
  These components will be used for setting up application processes on XtreemOS nodes in a way that will ensure both failure resilience and certain communication topologies (like tori or rings). At the current state of the project (month 18), these components exist only as a first prototype version. A sensible API can thus only be designed in a later phase of the project, likely for the third API specification.
- Resource Selection and Application Directory Service:
  These components are used via the job submission mechanism from WP3.3. (Covered.)
- Publish-Subscribe:
  As of month 18, the publish-subscribe system exists only in simulations of its fundamental mechanisms. As such, it is too early to provide an API for this functionality. In the third API specification, we will provide a publish-subscribe API, likely via the Message-Bus extension package to the SAGA API [6].

WP3.3 Provides services for application execution management. Covered.

WP3.4
- XtreemFS file system. Covered.
- Object Sharing Service (OSS), this will provide support for object sharing services, exposing shared-memory access to the objects. While the basic functionality is becoming available as a first prototype, the functionality regarding object consistency (transactions, locking, etc.) will only become available in a later project phase. It is thus not yet possible (or at least sensible) to devise an API for the OSS. For the third API specification, an API will be devised, likely as an extension package to the XOSAGA API framework.

WP3.5 provides VO support. Covered.

WP3.6 provides mobile-client access to XtreemOS services (application execution management, XtreemFS, VO-centric user management), as well as a lightweight SAGA implementation. This functionality is actually providing/implementing the XtreemOS API rather than separate functionality that would require additional API coverage.

8 Summary

In this document, we have presented the second draft specification of programming interfaces for XtreemOS, adding access to XtreemOS-specific functionalities w.r.t. the first draft programming interface (SAGA).

We have analyzed the updated application requirements and the functionality provided (or planned for) by the work packages from sub projects SP2 and SP3. The conclusions from our analysis are that most XtreemOS functionality can be accessed via the SAGA interface, requiring "only" implementation efforts for respective adaptors. One exception is VO management which requires the definition of an XtreemOS-specific type of `saga::context` objects, specified in this document.

The application execution management (AEM) of XtreemOS can not be accessed via the existing SAGA interface, as it exposes resources and resource reservation to the applications, concepts that are hidden on purpose by SAGA, for the goal of API simplicity. To provide access to AEM functionality, we have presented the XOSAGA API name space that mirrors SAGA and tightly integrates with it, providing suitable extensions. In the case of AEM, we have proposed a resource management package for XOSAGA that integrates SAGA's job management package with XtreemOS AEM. It is anticipated that XOSAGA extension packages that have proven useful will be contributed as SAGA extension packages to the standardization process within the Open Grid Forum (OGF).

The programming interface presented in this document is still considered a draft of the XtreemOS API. This is due to the state of the technical developments at the current phase of the project. Some technical details of the covered functionality is still subject to change. Other functionalities are not yet ready for designing an API. Furthermore, practical experience with the produced prototypes of the XtreemOS operating system, its services, and the API may indicate the need for updates and changes in the coming API releases.

References

[1] A. Anjomshoaa, F. Brisard, M. Drescher, D. Fellows, A. Ly, S. McGough, D. Pulsipher, and A. Savva. Job Submission Description Language (JSDL) Specification v1.0. Grid Forum Document GFD.56, Open Grid

Forum (OGF), 2005. http://www.ogf.org/sf/documents/GFD.56.pdf.

[2] I. Foster, A. Grimshaw, P. Lane, W. Lee, M. Morgan, S. Newhouse, S. Pickles, D. Pulsipher, C. Smith, and M. Theimer. OGSA Basic Execution Service Version 1.0. Grid Forum Document GFD.108, Open Grid Forum (OGF), 2007. http://www.ogf.org/sf/documents/GFD.108.pdf.

[3] T. Goodale, S. Jha, T. Kielmann, A. Merzky, J. Shalf, and C. Smith. A Simple API for Grid Applications (SAGA). Grid Forum Working Draft, Open Grid Forum, 2006. Version 1.0 RC.1 http://forge.ogf.org/sf/projects/saga-core-wg.

[4] 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, Open Grid Forum (OGF), 2007. Version 1.0 http://forge.ogf.org/short/saga-core-wg/saga-core-v1.

[5] I. Mandrichenko, W. Allcock, and T.Perelmutov. GridFTP v2 Protocol Description. Grid Forum Document GFD.47, Open Grid Forum (OGF), 2005. http://www.ogf.org/sf/documents/GFD.47.pdf.

[6] Andre Merzky. SAGA API Extension: Message Bus API. Open Grid Forum (OGF), SAGA CORE working group, 2008. Working draft at http://forge.ogf.org/sf/projects/saga-core-wg.

[7] H. Nakada, S. Matsuoka, K. Seymour, J. Dongarra, C. Lee, and H. Casanova. A GridRPC Model and API for End-User Applications. Grid Forum Document GFD.52, Open Grid Forum (OGF), 2005. http://www.ogf.org/sf/documents/GFD.52.pdf.

[8] H. Rajic, R. Brobst, W. Chan, F. Ferstl, J. Gardner, A. Haas, B. Nitzberg, D. Templeton, J. Tollefsrud, and P. Tröger. Distributed Resource Management Application API Specification 1.0. Grid Forum Document GFD-R.022, Open Grid Forum (OGF), 2007. http://www.ogf.org/sf/documents/GFD.22.pdf.

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

[10] Requirements and specification of xtreemos services for job execution management. Deliverable D3.3.1, XtreemOS Consortium, 2006.

[11] Requirements Capture and Use Case Scenarios. Deliverable D4.2.1, XtreemOS Consortium, 2006.

[12] Specification of Federation Resource Management Mechanisms. Deliverable D2.2.1, XtreemOS Consortium, 2006.

[13] Application References, Requirements, Use Cases and Experiments. Deliverable D4.2.3, XtreemOS Consortium, 2007.

[14] Design of the Architecture for Application Execution Management in XtreemOS. Deliverable D3.3.2, XtreemOS Consortium, 2007.

[15] First Prototype of XtreemOS Runtime Engine. Deliverable D3.1.3, XtreemOS Consortium, 2007.

[16] Linux XOS Specification. Deliverable D2.1.1, XtreemOS Consortium, 2007.

[17] Second Specification of Security Services. Deliverable D3.5.4, XtreemOS Consortium, 2007.

[18] WP 3.3 Internal Documents. http://xtreemos.wiki.irisa.fr/tiki-index.php?page=WP3.3, 2007.