Project no. IST-033576

XtreemOS

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

First Draft Specification of Programming Interfaces
D3.1.1

Due date of deliverable: November 30^{th}, 2006
Actual submission date: December 21^{st}, 2006

Start date of project: June 1^{st} 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 1.1 / Last edited by Thilo Kielmann / December 21^{st}, 2006

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	06/10/27	Andre Merzky	VUA	initial draft
0.9	06/11/06	Thilo Kielmann	VUA	first complete draft
0.99	23/11/06	Thilo Kielmann	VUA	after internal review
0.991	27/11/06	Andre Merzky	VUA	almost final
1.0	28/11/06	Thilo Kielmann	VUA	final version
1.1	18/12/06	Thilo Kielmann	VUA	covering comments from deliverable harmonization
1.1	21/12/06	Guillaume Pierre	VUA	very final warp-up

Executive Summary

This document presents the first draft specification of programming interfaces for the XtreemOS operating system and services. It first outlines the design space for grid API's into which the XtreemOS programming interfaces can be positioned. Then, the requirements on XtreemOS API's, as stated by the other work packages of XtreemOS, are listed and discussed. From this discussion we conclude to declare the upcoming OGF standard *SAGA* (the "Simple API for Grid Applications") to be the first draft of the XtreemOS API. XtreemOS-specific extensions and "native" API's will be added as soon as possible, namely in the next API version, due in month 18 of the project.

1 Introduction

The objective of work package 3.1 is to provide the API of XtreemOS, and a corresponding runtime system for grid applications. The goal is to allow applications to run on any XtreemOS platform, be it a handheld device, a PC, a federation of machines, or a virtual organization (VO) comprising systems of all these categories. As far as possible, existing Linux applications shall be able to run on XtreemOS with little or no modifications.

To achieve this objective, various interfaces need to be coordinated. A distinction has to be made between application interfaces and management interfaces; the latter being those necessary for XtreemOS-specific services and Linux extensions.

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, is being agreed upon early in the project, such that immediate implementation experience can be gained and strengths and weaknesses identified. 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. The above-mentioned first specification is described in this document.

This document is structured as follows. Section 2 will first outline the design space for grid API's into which the XtreemOS programming interfaces can be positioned. Section 3 lists and discusses the requirements on XtreemOS API's, as stated by the other work packages of XtreemOS. Section 4 presents the first draft specification of XtreemOS programming interfaces. Section 5 concludes and outlines the following steps of work package 3.1.

2 Grid Programming Models and API's

In previous work [5], we have investigated programming models for grid applications and their possible incarnation in API's. In this section, we briefly outline the grid API design space into which the XtreemOS programming interfaces can be positioned.

2.1 Properties of Grid API's

To summarize out findings from [5], grid API's have to provide several properties. These can be divided in two categories, functional and non-functional properties. The functional properties describe the functionality required by applications to

run in grid environments, while the non-functional properties determine the constraints on grid API functionality. As such, issues like performance, security, and fault-tolerance influence the suitability of certain programming abstractions.

Functional properties

- Job submission, spawning, and scheduling:
  Users as well as running jobs need to submit new application jobs to the grid, usually via some job submission service.

- Access to file and data resources:
  Most real-world application have to process some form of input data, be it files, data bases, or streams. Similarly, generated output data has to be stored on behalf of the users.

- Inter process communication:
  Often, the processes of a parallel or distributed application need to communicate with each other. Several programming models for grid applications have been developed, among which are MPI [4], shared objects [7], or remote procedure calls [10].

- Application monitoring and steering:
  In case of long running applications, users need to track their progress in order to avoid costly repetition of unsuccessful jobs. For this purpose, users need to inspect and possibly modify the status of their application while it is running on some nodes in a grid.

Non-functional properties

- Performance:
  As high-performance computing is one of the driving forces behind grids, performance is the most prominent, non-functional property of grid operations.

- Fault tolerance:
  Frequently, grid applications fail due to faulty configurations, middleware failures, application failures, and also due to hardware failures and network outages. Consequently, error handling becomes an integral part of grid runtime environments and grid APIs.

- Security and trust:
  A grid API needs to support mutual authentication of users and resources, as well as access control to resources (authorization).

- Platform independence:
  It is an important property for programming environments to keep the application code independent from details of the grid platform, like machine names or file system layouts for application executables and data files.

2.2 Scoping of Grid API's

Taking into account both the diversity of the above-identified properties, and the diversity of grid platforms (HPC machines, clusters, individual PC's) and middleware services, it becomes obvious that there can not be a single, unifying grid API. Instead, a palette of grid programming abstractions is needed, each suitable for its respective problem domain.

Virtualization is the predominant purpose of all (grid) middleware. Likewise, operating systems virtualize the resources of a single computer; and cluster operating systems like, e.g., Kerrighed [9] virtualize the resources of a cluster computer in order to provide a single, powerful system instead of a collection of individual nodes. XtreemOS will virtualize a wide spectrum of different systems, ranging from mobile devices, via individual PC's and clusters, to virtual organization composed from these. Figure 1 outlines the XtreemOS software stack, ranging from Linux-XOS for PC's, clusters, and mobile devices, to the service layer spanning whole virtual organizations. The task of the XtreemOS API will be to provide unified and coordinated interfaces to all of these.

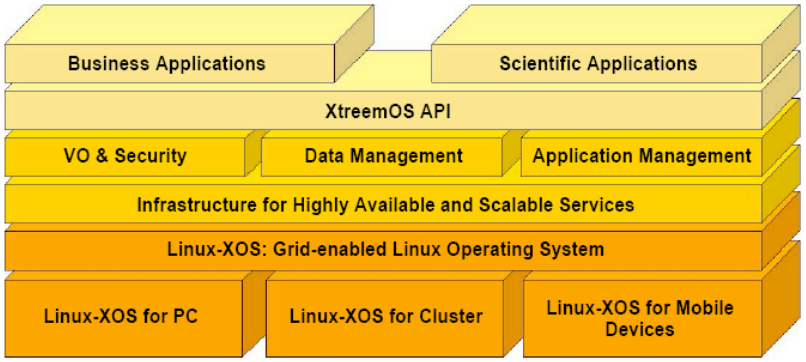

Figure 1: The XtreemOS software stack.

Each level of virtualization provides a uniform interface to a variety of heterogeneous, individual entities. With each virtualization layer, the use of the resources becomes simpler, however at the price of loosing part of the control over

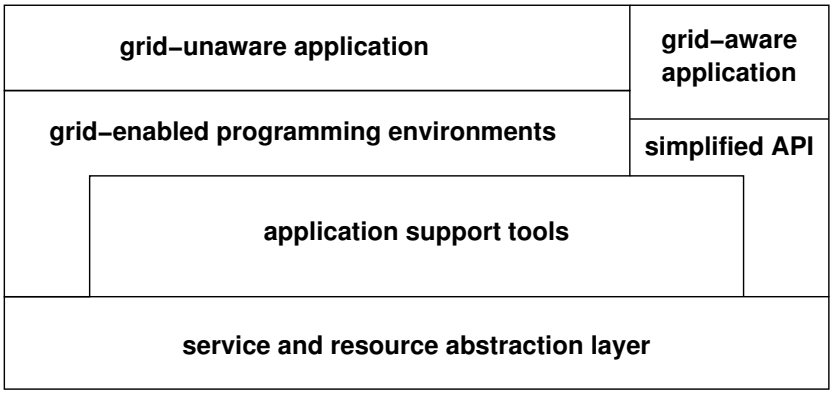

Figure 2: Hierarchy of grid programming abstractions.

the resources which may cause problems with non-functional aspects, like system performance.

Figure 2 is providing a more detailed view on a stack of virtualization abstractions provided to grid applications. The bottom of the stack forms the service and resource abstraction layer. This layer provides an API that virtualizes the different grid middleware or XtreemOS services. The most prominent example of such a layer is the Grid Application Toolkit (GAT) [1].

On top of this layer may sit a set of (optional) application support tools. Such tools provide additional functionality, adding value to plain middleware services, likely tailored to application needs. Examples for these tools can be found in CoreGRID's proposed mediator component toolkit [6]. These components are supposed, for example, to provide application-level metadata or to dynamically tune and steer running applications. Such components add another layer of virtualization by providing more comprehensive functionality, on top of the plain service and resource virtualization.

An important, original design goal of the Grid Application Toolkit [1] (GAT) was to provide a simplified API for programmers of grid-aware applications (see figure 2). Meanwhile, the GAT has been recognized as an attractive programming platform because of its service and resource abstraction functionality. Many developers are currently also trying to build application support tools using the GAT, requesting and leading to additional functionality and hence less simplicity in the GAT interface.

Independent of the GAT, within the Open Grid Forum, we are working on a Simple API for Grid Applications (SAGA) that will provide a standardized, simplified grid API [3]. With SAGA, simplicity stems from both uniformity across different middleware platforms, and also from the reduction to functionality that is

needed to run application code, purposefully excluding features needed for management or monitoring of the grid services and resources themselves.

Both GAT and SAGA interfaces aim at grid-aware (or "grid-enabled") applications. These are applications that are explicitly using grid resources like submitting additional jobs to other machines or accessing remote files or data bases. A different class of applications is grid-unaware. Such applications treat a grid as a completely virtualized execution environment. Examples of such programming environments are special MPI versions [4] and remote object-based systems like Ibis [13] or ProActive [12]. Part of the XtreemOS work will lead to further grid-enabled programming abstractions that hide the platform diversity behind functionality that slightly extends existing Linux (POSIX) API's with grid-enabling features. One prominent example is the support for virtual organizations in XtreemOS.

For the design of the XtreemOS API there is a strong tension between the application requirements (for the diverse set of applications that are more or less grid-aware) on one side, and the operating system and service-layer on the other side, where different services provide their shares to the overall XtreemOS functionality. Coordinating the proposed service functionality and application needs is the most important goal of the API work package.

To conclude, XtreemOS will need three kinds of API's:

1. Application-level API's for grid-aware applications
2. Application-level API's for grid-unaware but XtreemOS-aware applications
3. Management API's for the XtreemOS systems and services

In the next section, we will put these kinds of anticipated API's in the context of the other XtreemOS work packages.

3 Requirements on the XtreemOS API

As outlined before, the XtreemOS API is supposed to meet the requirements of both the application users and the service and extension providers of XtreemOS. We summarize both sets of requirements in turn.

3.1 Application requirements

Work package 4.2 (Applications, Experiments, Evaluation) has specified a number of application-level requirements [14]. The requirements on the XtreemOS API are summarized in Table 1.

API Requirements	
R43	*Other API Standards as basis for XtreemOS API* XtreemOS API must consider the following standards as a basis: SAGA (especially the subsets DRMAA, GAT). Furthermore, any other standard allowing application to access user and/or job information is welcome. Additionally, one application requires the following Globus/Globus-related components: GridFTP, Apache Axis, and the GSI public key infrastructure. Here, an equivalent XtreemOS functionality is needed.
R44	*Demand for POSIX like extension* Mandatory access control (ACL) as defined in e.g. in POSIX.1e, IEEE 1003.1e/2c (which was withdrawn) is required by one application. Note, all 28 calls must be provided. Furthermore, It would prove useful if functions for management of processes on remote machines would be part of the XtreemOS API.
R45	*XtreemOS API language support* XtreemOS must support C, C++, Java, and Fortran 77. Furthermore, Ada, Python and Perl should be supported. Fortran 77 should be supported via C bindings.
R46	*Degree of Interoperability* It should be possible to use XtreemOS as a backend for GT4 WS-GRAM.

Table 1: API requirements as specified by WP 4.2

From this group, R43–R45 are shaping the overall API structure and will be addressed by the XtreemOS API. Requirement R46, however, while striving at integration of XtreemOS and existing grid middleware (here: Globus), can technically not be addressed at the API level; it has to be supported on the infrastructure (service) level. During discussions at the XtreemOS Düsseldorf workshop it was agreed that WP 3.3 will take responsibility of this requirement.

Besides those requirements that are specifically targeted at the XtreemOS API, WP 4.2 has also stated several *General Requirements*. Table 2 summarizes those general requirements that need to be taken care of on the API level.

The requirements from this group will be taken into account for the API. Several of these requirements are marked as *Optional* by WP 4.2. We are anticipating, however, to support the complete set of requirements.

Besides API specifications for these requirements, the service-providing work packages (from sub projects 2 and 3) will have the foremost task of addressing

General Requirements	
R1	XtreemOS supports data- intensive and compute-intensive applications
R9	XtreemOS needs to provide software licensing mechanisms
R10	XtreemOS has to provide for fast and reliable communication
R11	XtreemOS must support IPv6
R14	XtreemOS shall support multicast
R15	XtreemOS needs to provide access to various grid services
R17	XtreemOS must support the execution of interactive and batch jobs

Table 2: General requirements as specified by WP 4.2

these. API definition can only be performed at a later phase, when proper services will be available, or at least well-enough specified to allow to design APIs.

3.2 Service-driven requirements

Besides the application-driven requirements, the technical discussions among representatives of the technical work packages (sub projects 2 and 3) of XtreemOS have lead to a set of additional requirements on the API, listed in Table 3. These requirements are more related to the upcoming XtreemOS services and Linux extensions.

For this group of requirements, API definition can only be performed at a later phase, when proper services will be available, or at least specified well-enough to allow API design.

At the current stage of the project, however, the services and extensions to be provided by XtreemOS are not yet sufficiently advanced to start defining API's for them. This perception is one of the results of the technical project meeting, held in October 2006 in Düsseldorf. Consequently, the first version of the XtreemOS API will be confined to meeting the requirements of the application users within the project.

3.3 Summary

API definition requires maturity about concepts, requirements, and provided service functionality. At the current status of the XtreemOS project, we have to confine ourselves to specify the application-level API's for grid-aware applications. Both application-level and management-level API's for XtreemOS-specific services can not be sensibly specified yet; such specifications would be doomed to be incomplete or erratic or otherwise of little use to potential applications, due to the early status (albeit according to the project schedule) of the XtreemOS service

Additional Requirements	
RA1	shared memory API
RA2	resource discovery/exploration
RA3	POSIX-ACL on meta data items (it was agreed to provide that as a separate library/package)
RA4	notification on file size and meta data changes
RA5	file versioning
RA6	Ada support is not necessary immediately; it needs further discussion in later stages of the project
RA7	job description: allow to specify QoS requirements, such as bandwidth, no other application runs on a resource, location (country), latency/proximity.
RA8	support for job dependencies
RA9	application-level signalling support

Table 3: Additional requirements on the API

development. These kinds of API's will thus be added in the next API specification, as defined in deliverable D3.1.2, due at month 18. In the following, we will present the application-level API's for grid-aware applications that will define the XtreemOS API version 1.

4 API Specification

Based on the above discussion, we define the first draft XtreemOS API. We focus on application-level API's for grid-aware applications, addressing the application requirements, R1, R9–R11, R14, R15, R17, and R43–R45.

4.1 Rationale for choosing the SAGA interface

The currently most promising candidate for an Application level Grid Programming Interface is the SAGA API [3] as specified by the Open Grid Forum (OGF [11]). Also, the SAGA API has been explicitly cited as possible candidate for the XtreemOS Grid API by several application groups (R39), together with DRMAA [2] and GAT [1], which are both superceeded by SAGA. For this reason, the first draft specification of an application-level XtreemOS API will be a SAGA compliant API implementation with bindings to the XtreemOS services.

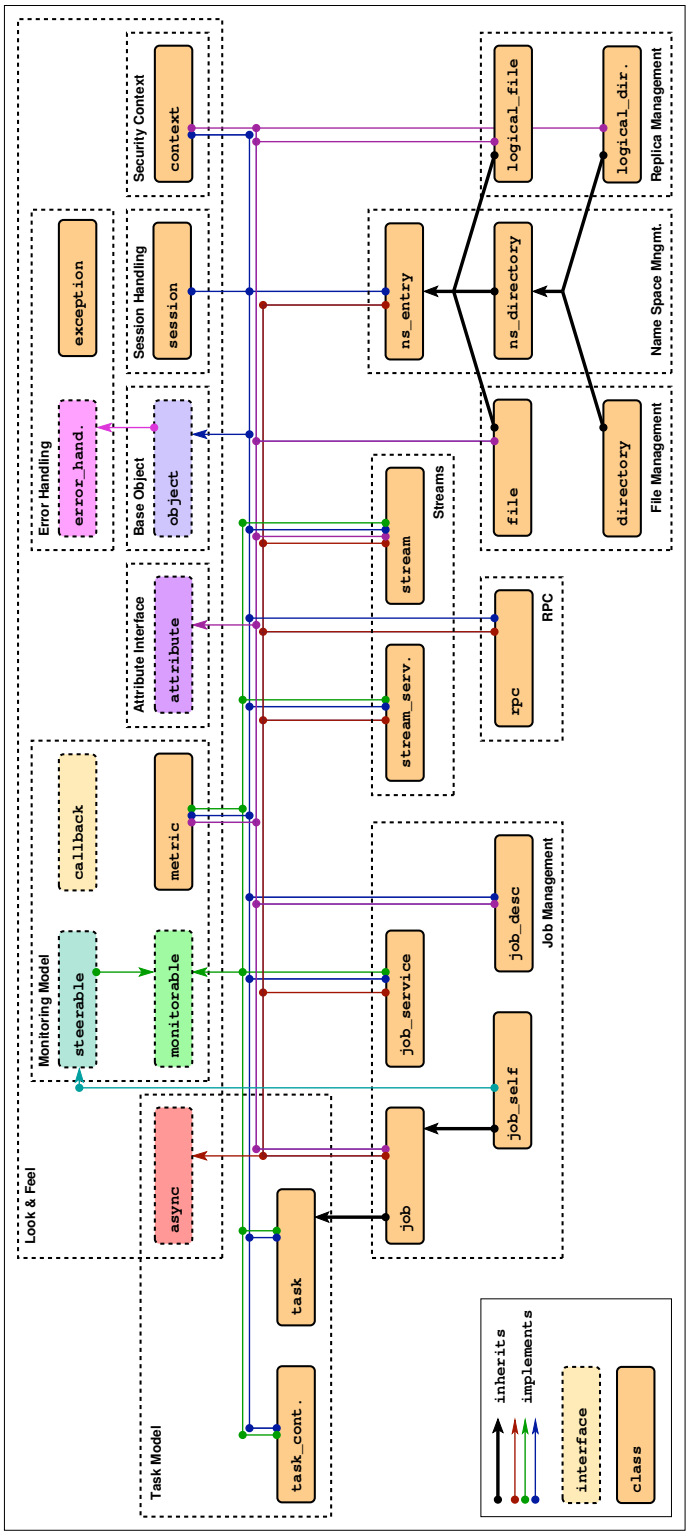

Figure 3: The SAGA class and interface hierarchy

We consider the following document [3] to be a part of this deliverable. Due to its size (250 pages), we refrain from directly including the SAGA specification in this document.

> T. Goodale, S. Jha, T. Kielmann, A. Merzky, J. Shalf, C. Smith. **A Simple API for Grid Applications (SAGA).** Grid Forum Working Draft GWD-R, Open Grid Forum (OGF), 2006.

4.2 SAGA scope

According to Figure 2, the scope of the SAGA API can be identified. SAGA thus is aiming at grid-aware applications to which a simplified API is provided. Aiming at *grid-aware* applications means exposing the fact that there are several, distributed resources, while a *simplified* API is driven by application needs rather than service capabilities.

In contrast, SAGA does *not* provide abstractions of grid-enabled programming environments which aim at *grid-unaware* applications. For example, interfaces likes the ones provided by MPI [8], Ibis [13] or ProActive [12] are *outside* the scope of SAGA.

Likewise, SAGA does *not* (directly) provide interfaces to particular grid services. Only the implementations of SAGA will use existing services for providing the simplified SAGA API. Also, service and resource management interfaces are *outside* the scope of SAGA.

4.3 SAGA overview

The SAGA API consists of a number of interface and class specifications. The relation between these is shown in Figure 3. This figure also marks which interfaces are dominating the SAGA look-and-feel (the non-functional part of the API), and which classes are combined to functional API packages.

4.3.1 SAGA Look-and-Feel

Error handling Each SAGA API call has an associated list of exceptions it may throw. These exceptions all extend the `saga::exception` class.

All objects in SAGA implement the `error_handler`, which allows a user of the API to query for the latest error associated with a saga object. In languages with exception facilities, such as Java, C++ and Perl, the language binding may allow exceptions to be thrown *instead*. Bindings for languages without exception handling capabilities MUST stick to

the `error_handler` interface described here, but MAY define additional language native means for error reporting.

Object The SAGA object interface provides methods which are essential for all SAGA objects. It provides a unique ID which helps maintain a list of SAGA objects at the application level as well as allowing for inspection of objects type and its associated session.

Session The session object provides the functionality of a session handle, which isolates independent sets of SAGA objects from each other. Sessions also support the management of security information (see `saga::context`).

Context The `saga::context` class provides the functionality of a security information container. A context is created, and attached to a session handle. As such it is available to all objects instantiated in that session. Multiple contexts can co-exist in one session.

Attributes There are various places in the SAGA API where attributes need to be associated with objects, for instance for job descriptions and metrics. The 'Attribute' interface provides a common interface for storing and retrieving attributes.

Monitoring The ability to query Grid entities about state is requested in several SAGA use cases. Also, the SAGA task model introduces numerous new use cases for state monitoring.

This package definition approaches the problem space of monitoring to unify the various usage patterns, and to transparently incorporate SAGA task monitoring. The paradigm is realised by introducing monitorable SAGA objects, which expose metrics to the application, which represent values to be monitored.

A closely related topic is Computational Steering, which is (for our purposes) not seen independently from Monitoring: in the SAGA approach, the steering mechanisms extend the monitoring mechanisms by the ability to push values back to the monitored entity, i.e. to introduce writable metrics.

Tasks Operations performed in highly heterogenous distributed environments may take a long time to complete, and it is thus desirable to have the ability to perform operations in an asynchronous manner. The SAGA task model provides this ability to all other SAGA classes.

4.3.2 SAGA Functionality

Jobs This packages has been designed for submitting jobs to a grid resource, either in batch mode, or in an interactive mode. It also describes how to control these submitted jobs (e.g. to `cancel()`, `suspend()`, or `signal()` a running job), and how to retrieve status information for both running and completed jobs.

This API is also intended to incorporate the design of the DRMAA API [2] from where much of the SAGA jobs specification was taken, with many of the differences arising from an attempt to make the job API consistent with the overall SAGA look and feel.

Name Spaces Several SAGA packages share the notion of name spaces and operations on these namespaces. In order to increase consistency in the API, those packages share the same API paradigms. The SAGA name spaces allow to operate on arbitrary hierarchical name spaces, such as used in physical, virtual and logical file systems, and in information systems.

The API is inspired by the POSIX standard, which defines tools and calls to handle the name space of physical files (directories). The methods listed for the interfaces have POSIX like syntax and semantics.

Files The ability to access the contents of files regardless of their location is central to many of the SAGA use cases. It is useful to note that interactions with files as opaque entities (i.e., as entries in file name spaces) are covered by the name space package. The classes of the Files package supplement the namespace package with operations for the reading and writing of the *contents* of files. For all methods, the descriptions and notes of the equivalent methods in the name space package apply if available, unless noted here otherwise.

The described classes are syntacically and semantically POSIX oriented. Large numbers of simple POSIX-like remote data access operations are, however, prone to latency related performance problems. To allow for efficient implementations, the presented API borrows ideas from GridFTP and other specifications which are widely used for remote data access. These extentions should be seen as just that: optimizations. Implementations of this package MUST implement the POSIX-like `read()`, `write()` and `seek()` methods, and MAY implement the additional optimized methods.

Logical Files This section of the SAGA API describes the interaction with replica systems. Numerous SAGA use cases required replica management functionality in the API – however, only a small number of operations have

been requested. The methods in these interfaces are hence limited to the creation and maintainance of logical files, replicas, and to search on logical file meta data.

Streams A number of use cases involved launching of remotely located components in order to create distributed applications. These use cases require simple, remote socket connections to be established between these components and their control interfaces.

The target of the stream API is to establish the simplest possible authenticated socket connection with hooks to support authorization and encryption schemes. The stream API is not performance-oriented, focused on TCP socket connections, and does not attempt to create a new IPC programming paradigm.

GridRPC GridRPC is one of the few high level APIs that have been specified by the GGF [10]. Thus including the GridRPC specification in the SAGA API benefits both SAGA and the GridRPC effort: SAGA becomes more complete and provides a better coverage of its use cases with a single look-and-feel, whilst GridRPC gets embedded into a set of other tools of similar scope, which opens it to a potentially wider user community, and ensures its further development.

4.4 Summary

We have chosen the upcoming OGF standard *SAGA* as the first draft API for XtreemOS, as it is the best match to the requirements from the other work packages in the project. Those requirements which are not covered by the current SAGA specification will be provided by additional API packages in the same framework (SAGA is extensible), and will, where sensible and possible, pushed into the OGF SAGA specification work. XtreemOS-specific extensions will, where needed, be added in the next version of the XtreemOS API.

5 Conclusion

In this document, we have presented the first draft specification of programming interfaces for the XtreemOS operating system and services. We have outlined the design space for grid API's into which the XtreemOS programming interfaces can be positioned. We have found that, for the design of the XtreemOS API, there is a strong tension between the application requirements (for the diverse set of applications that are more or less grid-aware) on one side, and the operating

system and service-layer on the other side, where different services provide their shares to the overall XtreemOS functionality. Coordinating the proposed service functionality and the application needs is the most important goal of the API work package.

We have identified three kinds of API's needed by XtreemOS: (1) application-level API's for grid-aware applications, (2) application-level API's for grid-unaware but XtreemOS-aware applications, and (3) management API's for the XtreemOS systems and services.

API definition requires maturity about concepts, requirements, and provided service functionality. At the current status of the XtreemOS project, we have to confine ourselves to specify the group of application-level API's for grid-aware applications. Both application-level and management-level API's for XtreemOS-specific services can not be sensibly specified yet; such specifications would be doomed to be incomplete or erratic or otherwise of little use to potential applications, due to the early status (albeit according to the project schedule) of the XtreemOS service development. These kinds of API's will thus be added in the next API specification, as defined in deliverable D3.1.2, due at month 18.

According to the requirements on XtreemOS API's, as stated by the other work packages of XtreemOS, we have concluded to declare the upcoming OGF standard *SAGA* (the "Simple API for Grid Applications") to be the first draft of the XtreemOS API. XtreemOS-specific extensions and "native" API's will be added as soon as possible, namely in the next API version, due in month 18 of the project.

In the period until month 18, the anticipated functionality extensions of the XtreemOS operating system and its services will have matured to a status at which API's can be defined with good confidence. In the same period, the first implementation of the SAGA API to XtreemOS will be available.

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, 2004.

[2] R. Brobst, Waiman Chan, F. Ferstl, J. Gardiner, J. P. Robarts, A. Haas, B. Nitzberg, H. Rajic, and J. Tollefsrud. Distributed Resource Management Application API Specification 1.0. Grid Forum Document GFD.22, Global Grid Forum, September 2002.

[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. `http://forge.ggf.org/sf/projects/saga-core-wg`.

[4] N. Karonis, B. Toonen, and I. Foster. MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface. *Journal of Parallel and Distributed Computing*, 2003.

[5] Thilo Kielmann. Programming Models for Grid Applications and Systems: Requirements and Approaches. In *2006 IEEE John Vincent Atanasoff International Symposium on Modern Computing (JVA 2006)*, pages 27–32, Sofia, Bulgaria, 2006.

[6] Thilo Kielmann, Gosia Wrzesinska, Natalia Currle-Linde, and Michael Resch. Redesigning the SEGL Problem Solving Environment: A Case Study of Using Mediator Components. In *Integrated Research in Grid Computing*. Springer Verlag, 2006.

[7] Jason Maassen, Thilo Kielmann, and Henri E. Bal. Parallel Application Experience with Replicated Method Invocation. *Concurrency and Computation: Practice and Experience*, 13(8–9):681–712, 2001.

[8] Message Passing Interface Forum. MPI: A Message Passing Interface Standard. *International Journal of Supercomputing Applications*, 8(3/4), 1994.

[9] Christine Morin, Pascal Gallard, Renaud Lottiaux, and Geoffroy Vallée. Towards an Efficient Single System Image Cluster Operating System. *Future Generation Computer Systems*, 20(2), 2004.

[10] Hidemoto Nakada, Satoshi Matsuoka, Keith Seymour, Jack Dongarra, Craig Lee, and Henri Casanova. A GridRPC Model and API for End-User Applications. Grid Forum Document, GFD.52, 2005. Global Grid Forum.

[11] Open Grid Forum (OGF). `http://www.ogf.org/`.

[12] ProActive. http://www.inria.fr/oasis/ProActive.

[13] Rob V. van Nieuwpoort, Jason Maassen, Gosia Wrzesinska, Rutger Hofman, Ceriel Jacobs, Thilo Kielmann, and Henri E. Bal. Ibis: a Flexible and Efficient Java-based Grid Programming Environment. *Concurrency and Computation: Practice and Experience*, 17(7–8):1079–1107, 2005.

[14] XtreemOS project consortium. Requirements capture and use case scenarios. Deliverable D.4.2.1, 2006.