Project no. IST-033576

XtreemOS

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

Revised System Architecture
D3.1.7

Due date of deliverable: November 30^{th}, 2008
Actual submission date: January 12^{th}, 2009

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

Type: Deliverable
WP number: WP3.1
Task number: T3.1.3

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

Version 1.0.3 / Last edited by Mathijs den Burger / January 12^{th}, 2009

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

Revision history:

Version	Date	Authors	Institution	Section affected, comments
0.1	22/09/08	Mathijs den Burger	VUA	initial draft
0.2	5/11/08	Guillaume Pierre	VUA	WP3.2 contribution
0.3	6/11/08	Luis Pablo Prieto	TID	WP2.3 and WP3.6 contribution
0.4	12/11/08	Christine Morin	INRIA	WP2.2 contribution with the help of other WP2.2 participants
0.5	13/11/08	An Qin	ICT	WP2.1 contribution on local VO managment
0.6	08/11/08	Oscar David Sanchez	INRIA	Update on communication and configuration sections
1.0	12/12/08	Thilo Kielmann	VUA	Complete version, containing all contributions.
1.0.1	05/01/09	Mathijs den Burger	VUA	Processed comments of internal reviewers
1.0.2	09/01/09	John MehnertSpahn & Michael Schoettner	UDUS	Update of WP3.3 checkpointing part
1.0.3	12/01/09	Mathijs den Burger	VUA	Processed comments of internal reviewers

Reviewers:
Yvon Jégou (INRIA), Christine Morin (INRIA), and Arnaud Laprévote (EDGE-IT)

Tasks related to this deliverable:

Task No.	Task description	Partners involved°
T3.1.3	XtreemOS system architecture	INRIA, STFC, BSC, VUA*, XLAB, ZIB, TID

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

Executive Summary

This document describes the overall system architecture for the software packages developed by the XtreemOS project, as they have been developed by the current stage of the project (Month 30). We are presenting the individual packages of XtreemOS, their layering in the overall software system, as well as the individual components of each package. Based here upon, we are describing the interactions of the software packages with each other, for the purpose of jointly providing capabilities to users, applications, or to other XtreemOS software packages. Most of these capabilities have been implemented by now. The remaining ones are planned to be added during the remaining project life time. The Sections 2 and 3 covering these aspects are revised and updated versions, based on deliverable D3.1.4 [1], the *First Version of System Architecture*.

New in this document, compared to D3.1.4, are the sections on the communication layer, Section 4, and configurations, Section 5. We conclude this document by summarizing our findings and outlining aspects for consideration with respect to the overall XtreemOS system architecture.

Glossary

Capabilities Higher-level *functionality* achieved by the combination and interaction of different *services*.

Component Generic name for a piece of software that makes up XtreemOS. Focuses on the software-engineering aspect rather than the *functionality* the software provides.

Functionality Specific actions or activities that can be performed.

Layer Set of *components* that provide *functionality* at approximately the same level of abstraction from the underlying hardware.

Module Part of a component.

Package Set of *components* provided by the same Work Package.

Service Set of *components* providing a certain *functionality*. Focuses on the functionality that is provided rather than the underlying software.

System A combination of *components* and hardware forming a unitary whole. Historically, the term is mostly used to describe either rather low-level software close to the actual hardware, or the high-level concept of the 'whole system' including all software and hardware it consists of.

Contents

1 Introduction

The major research challenge in grids is scalability. Large numbers of machines (e.g., 10.000's) populating virtual organizations are becoming unmanageable, requiring decentralized (P2P) management solutions. The numbers of users is getting equally large, with similar implications on user authentication and authorization management. Another important trend is the increasing diversity of platforms, ranging from high-end clusters, via stand-alone PC's, to less powerful, mobile devices. While integrated (operating system) support for these heterogeneous platforms is highly desirable, their different requirements and capabilities keep asking for tailor-made configurations.

To address these challenges, the XtreemOS project is building a Linux-based operating system to support virtual organizations (VOs) in next-generation grids. Unlike the traditional, middleware-based approaches, it is a prominent goal to provide seamless support for VOs, on all software layers involved, ranging from the operating system of a node, via the VO-global services, up to direct application support. In terms of the Open Grid Service Architecture (OGSA) [2], as shown in Figure 1, XtreemOS is providing support on all layers involved in a virtual organization:

- On the *fabric layer*, XtreemOS provides VO-support by Linux kernel modules.
- On the *connectivity layer*, XtreemOS provides VO membership support for (compute and file) resources, application programs, and users.
- On the *resource layer*, XtreemOS provides application execution management.
- On the *collective layer*, XtreemOS provides the XtreemFS file system, and VO management services.
- On the *Application layer*, finally, XtreemOS provides runtime support via its *XOSAGA* API, integrating the *Simple API for Grid Applications* (SAGA) [3] with native POSIX interfaces and XtreemOS-specific extensions.

Not only does XtreemOS cover the whole spectrum of OGSA layers. XtreemOS also integrates operating systems for the various computer architectures used in VOs:

- For stand-alone PCs (single CPU, or SMP, or multi-core), XtreemOS provides its Linux-XOS flavour with full VO support.

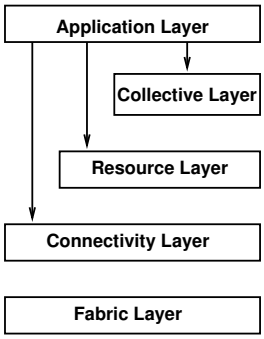

Figure 1: The layered Grid middleware architecture, diagram simplified from [2].

- For clusters of Linux machines, the LinuxSSI flavour combines VO support with a single system image (SSI) functionality.
- For mobile devices, finally, XtreemOS provides the XtreemOS-MD flavour with VO support and specially-tailored, lightweight services for application execution, common data access, and user management.

This document describes the overall system architecture for the software packages developed by the XtreemOS project, as they are planned and are already partially developed at the current phase of the project (Month 30). This document is intended to serve the following purposes:

1. Provide an overview of the XtreemOS software packages and their components,
2. Summarize the functionality of XtreemOS components:
   (a) functionality provided to other components,
   (b) functionality required from other components,
3. Analyze the interactions (and flows of information) between components, for jointly providing capabilities to users, applications, or to other XtreemOS software packages.
4. Describe the communication layer among services and components
5. Describe the different XtreemOS configurations

This document has four main sections. Section 2 presents the individual packages of XtreemOS, their layering in the overall software system, as well as the

individual components of each package. Section 3 describes the interactions of the software packages with each other, for the purpose of jointly providing capabilities to users, applications, or to other XtreemOS software packages. Section 4 describes the communication layer; Section 5 the XtreemOS configurations. We conclude this document with Section 6 by summarizing our findings and outlining aspects for consideration with respect to the XtreemOS system architecture.

2 XtreemOS software packages

The XtreemOS project is producing various software components, ranging from Linux kernel modules to application-support libraries. The overall layering of these components, grouped to *software packages*, is shown in Fig. 2. It shows all layers in the infrastructure on a very high level of abstraction. Each *layer* abstracts further from the underlying physical structure of a Grid, and consists of one or more software packages.

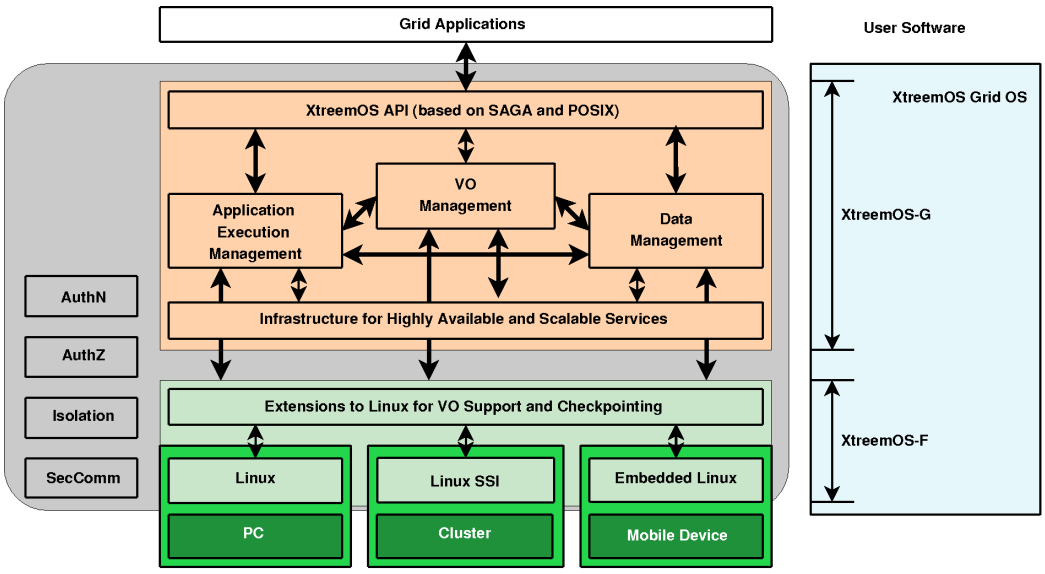

Figure 2: Layering of the XtreemOS software packages.

The development within XtreemOS is organized in work packages; each work package is responsible for one of these software packages. A software package provides one or more *services* of XtreemOS. Each service implements its functionality by interacting with other services in the same layer, and the layer below. Here, services can be either "classical" services within the XtreemOS-G layer, or Linux extensions (kernel modules etc.) within the XtreemOS-F layer.

This document describes the overall architecture for all software packages developed by XtreemOS. As such, it also describes software that is still under development or planned for later stages of the project. In the following, we outline the individual services being produced by the work packages from SP2 and SP3.

2.1 Extensions to Linux for VO Support and checkpointing (WP2.1)

This work package provides two main components of XtreemOS [1]:

Node-level VO support in Linux-XOS: This component provides the mapping from VO user identities to local user identities and the enforcement of VO-level policies on the local node. It performs authorization functions, by checking the validity of XtreemOS certificates (XOS-Cert) on user's login. It provides a dynamic mapping between grid identities and local identities, allowing grid users and processes to be linked to their local counterparts. This component was implemented based on existing Linux mechanisms including NSS, PAM and the kernel key retention service, which implies that applications could process VO-level information via standard Linux APIs (e.g. libc).

Checkpointing in Linux-XOS: Checkpointing in Linux-XOS provides the ability to save the state of a process group for a single node, and to restart it later on. Linux-XOS' implementation of checkpoint/restart mechanisms leverages BLCR (Berkeley Lab Checkpoint/Restart) [6], and adapts its uses to an application running on a grid. BLCR (version 0.6.1 for kernel 2.6.22 and version 0.7.3 for kernel 2.6.25) has been modified towards integrating the executable and the applications libraries into the snapshot. An application restart can be executed on other grid nodes, too without depending on its environment Furthermore, BLCR has been extended to support coordinated checkpointing of a distributed application. The checkpoint sequence has been modified to allow synchronization of multiple job-units before a checkpoint is taken. The restart sequence has been adapted appropriately.

[1] The current state of the work is documented in D2.1.2 [4] *Design and implementation in Linux of basic user and resource management mechanisms spanning multiple administrative domains* with respect to VO support and in D2.1.3 [5] *Design and implementation of basic application unit checkpoint/restart mechanisms in Linux* with respect to checkpointing in the F-layer of XtreemOS.

2.2 LinuxSSI (WP2.2)

This work package provides two main components of XtreemOS [2]:

Single System Image for cluster: LinuxSSI LinuxSSI gives the illusion that a Linux cluster is a single Linux node. Based on Kerrighed Single System Image (SSI) technology [14], LinuxSSI is improved in stability and features such as global customizable scheduler, checkpoint/restart of process trees, reconfiguration mechanisms, and distributed file system.

VO support in LinuxSSI: LinuxSSI-XOS This component provides the XtreemOS-F layer for cluster. LinuxSSI-XOS provides the necessary adaptations to the LinuxSSI operating system for clusters, in order to work with virtual organizations and VO users, by using Linux standard mechanisms like PAM and NSSwitch, in a similar fashion as in Linux-XOS node-level VO support (see Section 2.1). LinuxSSI-XOS provides a complete transparency of the cluster for XtreemOS-G layer so that a LinuxSSI-XOS cluster can be considered as a Linux-XOS powerful node.

LinuxSSI comprises of the following services:

Distributed file system: Most of available network file systems for clusters are built on the historical model compute nodes *vs* storage nodes. Available hard drives on compute nodes are only used for the system and temporary files, wasting both a lot of space and throughput, predominant criteria in the current High Performance Computing context. Keeping that in mind we have designed a new kernel Distributed File system, named kDFS, to efficiently exploit storage resources within a cluster. The first prototype, pluggable under the VFS, has been implemented upon kDDM mechanisms, a kernel DSM-like manager which allows consistent data sharing cluster-wide [10]. Thanks to kDDM sets, kDFS provides a cooperative cache for both data and meta-data.

Customizable scheduler: LinuxSSI scheduler [11] is a component which is in charge of placing processes to different cluster nodes. Besides that, it also serves as an interface for submitting jobs to LinuxSSI-XOS from upper layers (especially the Application Execution Management (AEM) layer). In

[2]The current state of the work is documented in D2.2.2 [7] *Design and implementation of scalable SSI mechanisms in LinuxSSI*, D2.2.3 [8] *Design and implementation of basic checkpoint/restart mechanisms in LinuxSSI*, D2.2.4 [9] *Design and implementation of basic reconfiguration mechanisms in LinuxSSI*, D2.2.5 [10] *Design and implementation of high performance disk input-out operations in a cluster*, D2.2.6 [11] *Design and implementation of a basic customizable scheduler*, D2.2.7 [12] *Prototype of the basic version of LinuxSSI*, and D2.2.8 [13] *Design and implementation of first advanced version of LinuxSSI*

the first implementation phase of the XtreemOS project, we were dealing with load balancing schedulers. These schedulers take care of migrating processes from one cluster node to another and thus transferring load from overloaded nodes to less busy ones. We designed a special framework, which we named "Pluggable Probes and Scheduling Policies Framework" (PlugProPol). By using this framework, the upper layers are able to load user-implemented resource measurement probes and scheduling policies (i.e., implementations of scheduling algorithms) and enable them without having to restart the whole cluster.

Checkpointing: Checkpointing in LinuxSSI provides the ability to transparently save the state of a process group of a SSI-application running in a cluster, and to restart it later on. The customized checkpointing implementation is used to support the SSI-internal process migration facility. The checkpointer runs in kernel mode and uses a coordinated checkpointing approach to save the cluster-wide state of a distributed application [8].

Reconfiguration mechanisms: LinuxSSI is implemented by a set of kernel level services distributed on the cluster nodes. The cluster administrator may want to upgrade hardware of one cluster node without stopping the whole cluster and especially without stopping application execution. Reconfiguration mechanisms are needed in LinuxSSI in order to provide node addition(s) and node removal(s) operations [9]. It is also highly desirable to handle node failure and network disconnection to avoid a crash of the whole cluster. XtreemOS consortium cannot take in charge the full implementation of reconfiguration mechanisms as it requires call-backs to be implemented in all LinuxSSI system services, some of them being designed and implemented by key developers outside XtreemOS consortium but involved in the Kerrighed open source community.

2.3 Embedded Linux (WP2.3)

As of this writing, the envisioned components belonging to this software package are [3]:

Terminal Mobility: This component provides XtreemOS-MD nodes with terminal mobility features, through an implementation of Mobile IPv6, in order to be able to change access points in a transparent way, without interrupting the communications of the mobile node with the Grid.

[3] The current state of the work with respect to the F-layer of XtreemOS-MD is documented in D2.3.4 [15] *Linux-XOS for MD/PDA* and D2.3.5 [16] *Requirements and specifications for advanced VO support in mobile devices*.

VO support in Linux-XOS for Mobile Devices: This component provides adaptations to the Linux operating system for mobile devices, in order for mobile users to login and be authenticated with virtual organizations, enabling them to use XtreemOS services like AEM or XtreemFS. These adaptations share the same features and make use of the same Linux standard mechanisms used in the VO support of the standard flavor (see Section 2.1), plus additional mechanisms for credential storage needed in mobile architectures.

2.4 XtreemOS API (WP3.1)

The (envisioned) components belonging to this software package are [4]:

API engine in C++: This engine acts as a runtime library that is to be linked to a user application. The engine is providing the XtreemOS API, and implementing its functions on top of different XtreemOS flavors. The engine is using dynamically loaded libraries, so-called adaptors, to dispatch functionality to different service providers. One set of adaptors is interfacing to the local capabilities of the node the application is running on (local adaptors), another set of adaptors is interfacing to XtreemOS' services.

API engine in Java: This engine works like its C++ counterpart, except that it is written purely in Java, also with Java adaptors in JAR files.

API for other languages: In a later stage, the existing engines in C++ and Java will be used to provide the XtreemOS API to other programming languages via wrapper interfaces.

2.5 Infrastructure for Highly-available and Scalable Services (WP3.2)

This software package comprises of the following (envisioned) components [5]:

[4] The state of the work with respect to the XtreemOS API is documented in D3.1.5 [17] *Third Draft Specification of Programming Interfaces* and D3.1.6 [18] *Second Prototype of XtreemOS Runtime Engine*.

[5] An introductory description about WP3.2 goals is provided in D3.2.1 [19] (*Design of an Infrastructure for Highly Available and Scalable Grid Services*) while individual services are extensively described in, respectively, D3.2.2 [20] (*First Prototype Version of Ad Hoc Distributed Servers*), D3.2.3 [21] (*Simulation-based evaluation of a scalable publish/subscribe system*), D3.2.4 [22] (*Design and Specification of a Prototype Service/Resource Discovery System*), and D3.2.5 [23] (*Design and Specification of a Virtual Node System*).

Distributed Server: A distributed server is an abstraction that allows to present a collection of server processes to its clients as a single entity. The IPv6 address of a distributed server remains stable, even in the case of nodes joining or leaving the application. This technology is exploited in the project both as a support for highly available services (e.g., the job manager or the VO manager) and by those applications willing to make their internal distribution transparent to their clients.

Virtual Nodes: A server object designed to be invoked remotely can request to be organized as a virtual node. A virtual node is a fault-tolerant group of server object replicas where each member can take over the task of the others in case of failure. Several types of virtual nodes may be provided, based on active replication, or passive replication. This technology is being integrated with distributed servers to provide a single platform to support fault-tolerant, highly available services and applications whose distribution is transparent to the clients.

Publish-Subscribe: A common form of communication between a large number of nodes taking part in a given application is publish-subscribe. We provide a fully decentralized pub/sub communication system that applications can use for their own purpose. The current implementation is based on a topic-based publish-subscribe. Later in the project we will evaluate if hierarchical topics or more content-based approaches are also needed.

Resource Selection Service: The Resource Selection Service (RSS) takes care of performing a preliminary selection of nodes to allocate to an application, according to range queries upon static attributes. It exploits a fully decentralized approach, based on an overlay network which is built and maintained through epidemic protocols. This allows to scale up to hundred thousands, if not millions, of nodes and to be extremely resilient to churn and catastrophic failures.

Application Directory Service: The Application Directory Service (ADS) handles the second level of resource discovery, answering queries expressed as predicates over the dynamic attributes of the resources. ADS creates an application-specific "directory service" using the NodeIDs received by the RSS, related to the resources involved in the application execution. To provide scalability and reliability, DHT techniques and their extensions to dynamic and complex queries are used.

The RSS and ADS together form the Scalable Resource Discovery System (SRDS).

2.6 Application Execution Management (WP3.3)

The (envisioned) components belonging to this software package are [6]:

Job Manager: Global information on the jobs running (or submitted) is kept by this distributed service (most nodes in the grid have an instance, and each instance handles some of the current jobs). Among its main functionality, the job manager is in charge of being the contact point to interact with a job, answer information about a job, schedule jobs, decide when a migration is needed, etc.

CRJobMng: This distributed service extends the Job Manager realising job checkpointing and job restart in order to realise job migration and suspension and to provide job fault tolerance. For the latter the CRJobMng Manager decides when to checkpoint and to restart a job. During checkpointing and restart the CRJobMng uses the Execution Manager and the CRExecMng Manager to coordinate all affected job units.

Execution Manager: This service is responsible of managing the job units running on the node it is located. Each node that runs part of a job also runs this service. A *job unit* represents (internally to AEM) all the processes of a job running in one resource, and a running job running consist of one of more job units. The Execution Manager performs the action requested by the Job Manager, e.g. launching processes within a job, monitoring the job unit (information that will be aggregated by the job manager) etc.

CRExecMng: This service realises the job-unit checkpointer responsible of saving and rebuilding a single job-unit. The grid-inherent hard- and software-heterogeneity requires the CRExecMng service to transparently access different kind of underlying kernel checkpointers. This is achieved by the common kernel checkpointer API. Furthermore, for efficiency reasons, it adapts the checkpointing strategy according to the monitored job-unit behaviour.

Resource Manager: Each resource in the grid has a resource manager service that mainly takes care of two tasks. On the one hand, it is responsible for exporting information about the resource (for instance for monitoring issues). And, on the other hand, to negotiate with the Reservation Manager to

[6] The state of the work with respect to the XtreemOS services regarding Application Execution Management is documented in D3.3.1 [24] *Requirements and specification of XtreemOS services for application execution management*. The internal architecture of these services is documented in D3.3.2 [25] *Design of the architecture for application execution management in XtreemOS*, D3.3.3 [26] *Basic services for application submission, control and checkpointing*, and D3.3.4 [27] *Basic services for resource selection, allocation and monitoring*.

manage reservations, and the Job Manager to negotiate with the scheduler, etc.

Reservation Manager: This service is responsible of managing advanced global reservations. Reservations are created by the Job Manager or directly by users and are bounded to one or more jobs. The goal of resource reservations is to provide a negotiated quality of service to running applications. The Reservation Manager interacts with applications (for instance workflow managers), with the Job Manager (in traditional job submission), and with the Resource Manager (to perform local reservations).

Job Directory: In order to locate a job controller (part of the Job Manager) in the system, we need a distributed service that stores the location of the Job manager containing this object. With this information we can get all the information about the job by directly contacting the right instance of the job controller.

XATI: Interface used to communicate with services in the Application Execution management. We have a Java and C version that can either be used by applications directly, or via XOSAGA (which will in turn use XATI to communicate with the AEM services).

2.7 Data Management (WP3.4)

The following (envisioned) components belong to this software package [7]:

Metadata and Replica Catalog: File system metadata is managed by the Metadata and Replica Catalog (MRC). The MRC provides an interface for file system operations related to metadata, such as creating, renaming or retrieving information about files, on which it also enforces access control. To provide resilience, it supports replication of metadata. The MRC will also offer partitioning of metadata among different servers to increase performance and scalability. A querying interface will allow for an advanced attribute-based retrieval of files.

Object Storage Device: Object Storage Devices (OSDs) are responsible for storing file content. File content is internally handled in the form of objects, where an object represents a certain range of bytes of a file. With the aim of increasing read/write performance, OSDs support striping by spreading

[7] The state of the work with respect to Data Management is documented in D3.4.1 [28] *The XtreemOS File System - Requirements and Reference Architecture* and D3.4.4 [29] *XtreemFS and OSS - Second Prototype*.

multiple objects of a single file across several OSDs. OSDs will also support replication of files with automatic fail-over, for the purpose of improving fault tolerance and availability, as well as reducing access latency. The latter can be achieved by placing file replicas close to their users. Replica placement will later be automated by the Replica Management Service (RMS).

Directory Service: This registry is internal to XtreemFS components. It is used by an MRC to discover OSD's.

Client/Access Layer: The Access Layer provides the interface between user processes and the file system infrastructure. Its main task is to handle access to files and directories on behalf of user processes. A POSIX-compliant interface based on the FUSE framework enables arbitrary applications to use the file system without a prior modification or recompilation of their source code. As the client-side part of the file system, the Access Layer interacts with the aforementioned file system services by translating calls from the POSIX API into corresponding interactions with OSDs and MRCs. In addition to the POSIX interface, the access layer will provide tools for creating and deleting XtreemFS volumes, checking file integrity, querying and changing the file striping policies, and other grid-specific features.

Object Sharing Service: The Object Sharing Service (OSS) provides sharing of volatile memory objects (raw memory regions or programming language objects) and memory-mapped files. One of the major goals is to implement transactional consistency (combining speculative transactions and optimistic synchronization) to simplify distributed programming. But the modular and layered design is open for other consistency models. Fault tolerance is provided by replication of shared data and checkpointing (using the XOS grid checkpointer).

2.8 VO and Security Management (WP3.5)

The (envisioned) components belonging to this software package are [8]:

Credential Distribution Authority: In XtreemOS, grid level credentials take the form of XOS Certificates, as defined in D3.5.3, *First Specification of Security Services*. An XtreemOS user runs a command-line CDA client program to contact the CDA service via a secure and authenticated channel - if the

[8] The state of the work with respect to VO and Security Management is documented in D3.5.3 [30] *First Specification of Security Services*, D3.5.4 [31] *Second Specification of Security Services*, D3.5.5 [32] *Security Services Prototype month 18*, and D3.5.6 [33] *Report on Formal Analysis of Security Properties*.

user is a member of a specified VO, the CDA service generates an XOS Certificate, signs and returns it to the user. With the corresponding private key, the user can then use this certificate to authenticate himself to remote entities in subsequent operations, such as submitting a job via the AEM or accessing files through the XtreemFS. XOS certificate is finally passed down to each resource node and consumed by node-level VO support components (this part of codes is produced by WP2.1)

Resource Certification Authority: The Resource Certification Authority (RCA) issues certificates that authenticate resource nodes in an XtreemOS system.

Accounting Service: The Accounting Service aims to record the information about resource usage and by whom the resources are used within a VO. It is currently being designed in WP3.5. It supports both push (information being pushed to the service) and pull (information being pulled by the service) models. It ensures accountability of actual resource consumption about (groups of) individuals by relying on the (real-time) events provided by the AEM services and the AEM communication infrastructure to realize accounting capability.

VO Policy Service: The VO Policy Service (VOPS) is a stand-alone security service that provides a Policy Administration Point (PAP), Policy Information Point (PIP), and Policy Decision Point (PDP) to other XtreemOS services. VOPS is designed to support coordinated access control to VO resources, including computation and storage resources, by offering a VO level policy decision point. Together with node level policy decision points, it forms a hierarchical access control framework that can be tuned to achieve various degrees of control to resource usage within a VO. It is being used by AEM to facilitate VO policy governed resource selection and job scheduling. It can also be used together with the accounting service to enforce constrains (e.g. quota and usage pattern) to certain types of resource consumption in a real-time manner.

VO Lifecycle Management: The Virtual Organization Lifecycle Management (VOLife) is a web-based tool for accessing various VO-related services in XtreemOS. Currently, VOLife only supports the manipulation of XVOMS database and the generation of XOS Certificates for users. Integration with runtime security services such as VOPS and RCA is still under development. VOLife consists of two parts: the backend and the frontend. The backend is a light Java wrapper around current security libraries. The frontend is a web application to be deployed into Tomcat. The backend provides a command-line utility which has almost the same functionality as the web frontend; its main

purpose is to test the integrity of data. The recommended way to use VOLife is via the web frontend.

2.9 Services for Mobile Devices (WP3.6)

As of this writing, the components belonging to this XtreemOS-MD layer are [9]:

Execution Management Client: This component provides client access to XtreemOS Application Execution Management (AEM), allowing mobile users to launch, manage and monitor jobs running in XtreemOS Grid. It consists mainly of a C implementation of the XATI interface to the AEM, able to run in ARM architectures.

XtreemFS Client: This component enables mobile users to access the XtreemFS filesystem through an implementation of the XtreemFS FUSE client for mobile devices architectures, with additional caching mechanisms for enhanced performance in mobile computing environments. This allows mobile users to mount XtreemFS volumes, and access grid files through a POSIX-compatible interface.

Credential Obtention Framework: This framework allows mobile users to obtain credentials for authentication, in a flexible and modular way. In XtreemOS, this component accesses the Credential Distribution Authority (see Section 2.8), either directly or through a proxy server. By using these certificates and the VO support components for Linux-MD, users are able to access the other XtreemOS services securely.

XtreemOS-MD API Engine: This component provides a subset of the XtreemOS XOSAGA API, that covers the needs of a XtreemOS client configuration (access to grid resources). This enables user applications to access and manage grid files and security contexts. Initially, only the C++ engine and the XtreemOS adaptors are available in mobile devices.

XtreemOS-MD Application Integration Kit: This component allows XtreemOS-unaware applications to use the XtreemOS certificates scheme, and allows for easier integration of end-user (e.g. graphical) applications with the XtreemOS system.

[9] The current state of the work with respect to the G-layer of XtreemOS-MD is documented in D3.6.2 [34] *Design of basic services for mobile devices* and D3.6.3 [35] *XtreemOS-G for MD/PDA*.

3 Capabilities

The different services developed by XtreemOS can interact in different ways to achieve certain higher-level functionality. We call such higher-level functionality a *capability* of XtreemOS: it is something XtreemOS as a whole is able to provide to the outside world. XtreemOS has the following capabilities:

Resource discovery: users can search for XtreemOS resources with certain characteristics.

Reservation management: users can exclusively reserve a set of XtreemOS resources for further use.

Job submission: users can submit a job to XtreemOS, which will then be executed on the required or reserved resources.

Checkpointing: jobs can be checkpointed automatically by XtreemOS according to a policy specified by the user, or manually triggered by the user.

Event management: users can send events to their jobs, similar to POSIX signals.

Monitoring: users can monitor various aspects of their jobs and the resources they run on.

Dynamic resource allocation: users can modify the resources used by their running jobs.

Fault-tolerant execution: vital XtreemOS services and user jobs can be replicated transparently to ensure high availability with minimum additional programming overhead.

Data management: users can have a global view of the distributed file system (XtreemFS).

File replication: XtreemFS implements transparent access to replicated files and co-operates with other services to support pro-active replica creation.

VO lifecycle management: VO creators can manage the lifecycle of VOs.

VO entity management: VO admins can manage the identity and attributes for users in a VO. Together with resource admins, VO admins can also manage VO resources.

VO accounting and audit trail management: VO admins can receive, register, and certify audit and accounting information of resource usage. VO administrators distributes such information to users.

Policy management: VO admins can manage VO policies to control the access to and the usage of VO resources.

We structure this section along different capabilities, and show the interactions between different services for each capability individually. Each capability is described by a diagram showing the interaction of the services involved. The diagram is accompanied by a brief description of the interaction.

Each *box* in these capability diagrams corresponds to a service in one of the work packages, as described in Section 2. The text in a box consists of the work package number, followed by the name of the service. An *arrow* between boxes describes the flow of information between services. Each arrow is annotated with a very short description of the information, which is always a noun. The direction of the arrow indicates the direction in which the information is transferred. When services communicate by request-and-reply, an arrow indicates the direction and contents of the reply.

Boxes can consist of multiple layers stacked on top of each other. These layers describe *software layers*, that are linked together and used as one piece of software. Layering is only be included in a diagram if it is relevant for the capability the diagram describes. For complex or more general software layering, a separate diagram is used.

A capability diagram may include grey areas that visually group services that are in the same *scope*. Each scope indicates the *locality* of services. We have identified four scopes:

User Scope contains all services that are local to a user of XtreemOS. The main example is client APIs.

Admin Scope contains all services that are local to an administrator of a virtual organization (VO) [2].

Core Scope contains those services that are operating independent of their physical location, typically in charge of a whole VO.

Node Scope contains all services that are running on an XtreemOS node (being a single machine running Linux-XOS, or a cluster running LinuxSSI-XOS).

Grouping services into these scopes gives the diagrams a more intuitive layout and improves readability.

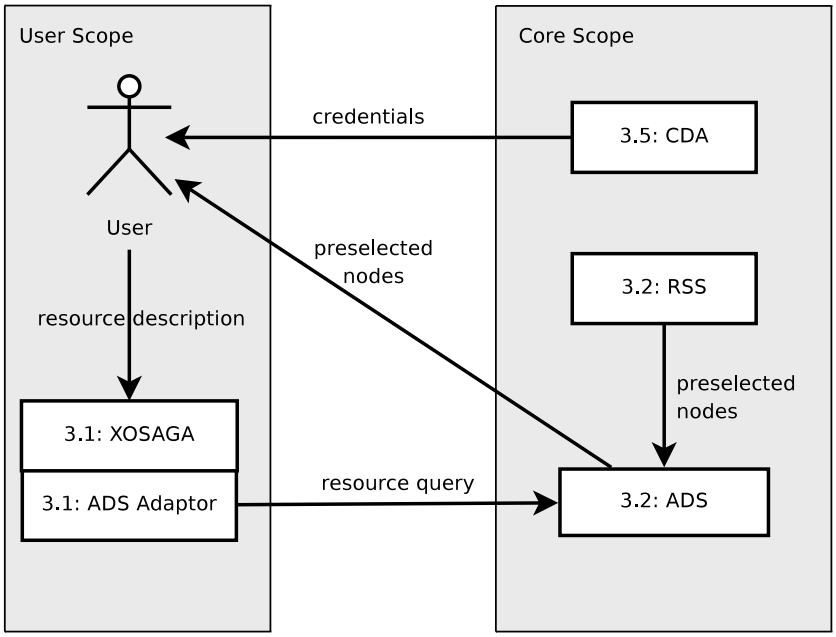

Figure 3: Resource Discovery

3.1 Resource Discovery

Users of XtreemOS can search for XtreemOS nodes with certain characteristics, as depicted in Figure 3. First, the user retrieves its credentials from the Credential Distribution Authority (CDA). Together with its credentials, the user then specifies resource requirements in the XOSAGA API, which are translated by an Application Directory Service (ADS) adaptor to a resource query. The ADS uses the Resource Selection Service (RSS) to perform a preliminary selection of nodes, which is further refined by the ADS. Finally, a description of the resources found is returned to the user.

Note that, although nodes must authenticate themselves to join the Resource Selection Service, this should not imply that nodes found in this service are necessarily trusted. One should therefore check credentials of such nodes before using them to run jobs.

3.2 Reservation Management

XtreemOS users can reserve a set of XtreemOS nodes for future usage. Figure 4 presents the reservation of a set of nodes. Initially, the user knows what resources will be used to make the reservation. These resources can either be already know by the user (i.e. a well known large cluster) or could have been discovered pre-

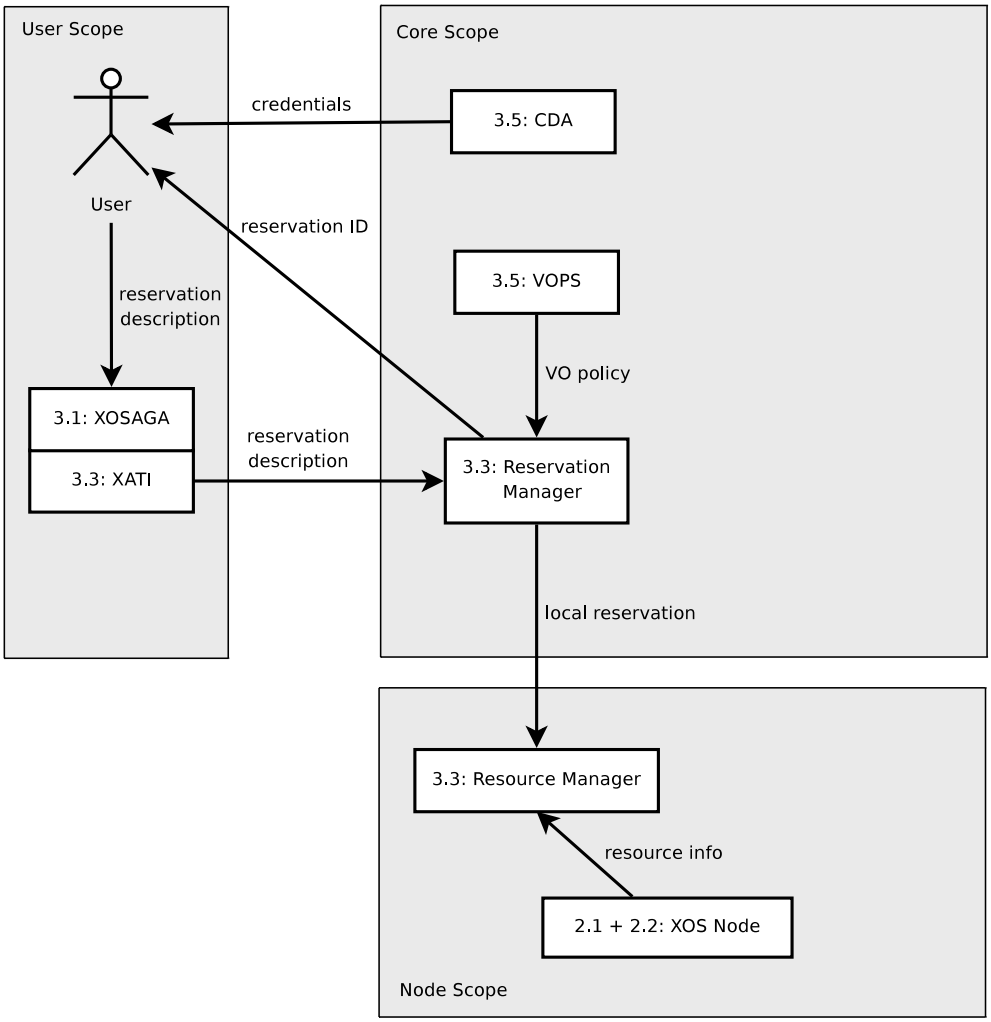

Figure 4: Reservation management

viously as described in Section 3.1. To be able to make a reservation, the user first needs to get credentials from the CDA. With these credentials and the description of the reservation it contacts the Reservation Manager via the XOSAGA and XATI interfaces. The Reservation Manager gets in contact with the Resource Managers of the nodes to reserve, and asks them for a local reservation. Once these reservations are made, information is kept in the Accounting Service and the reservation ID is returned to the user. From this point on, the user can refer to this reservation with this ID.

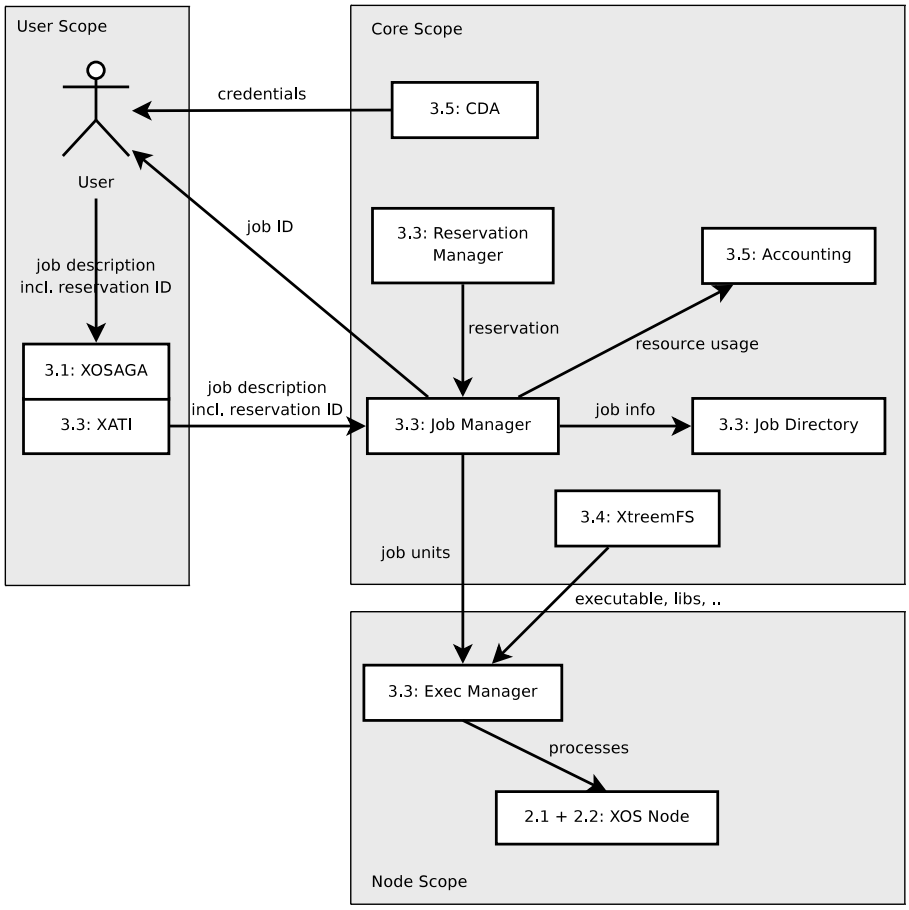

Figure 5: Job submission using an existing reservation

3.3 Job Submission

Users of XtreemOS can submit jobs in two ways. The first way is to first reserve a number of resources, and then submit a job that uses this reservation. The second way is to incorporate the job's resource requirements into the job description, and let XtreemOS handle the reservation and scheduling itself.

Figure 5 shows the submission of a job that uses a previously made reservation. To be able to submit a job, a XtreemOS user should first obtain credentials from the CDA. With these credentials, it can submit a job using XOSAGA, which uses XATI internally to access the Job Manager. The Job Manager obtains the reservation from the Reservation Manager using the reservation ID in the job description. The different job units the job consist of are then submitted by the Job Manager to the Exec Manager of each reserved node, which starts it on all

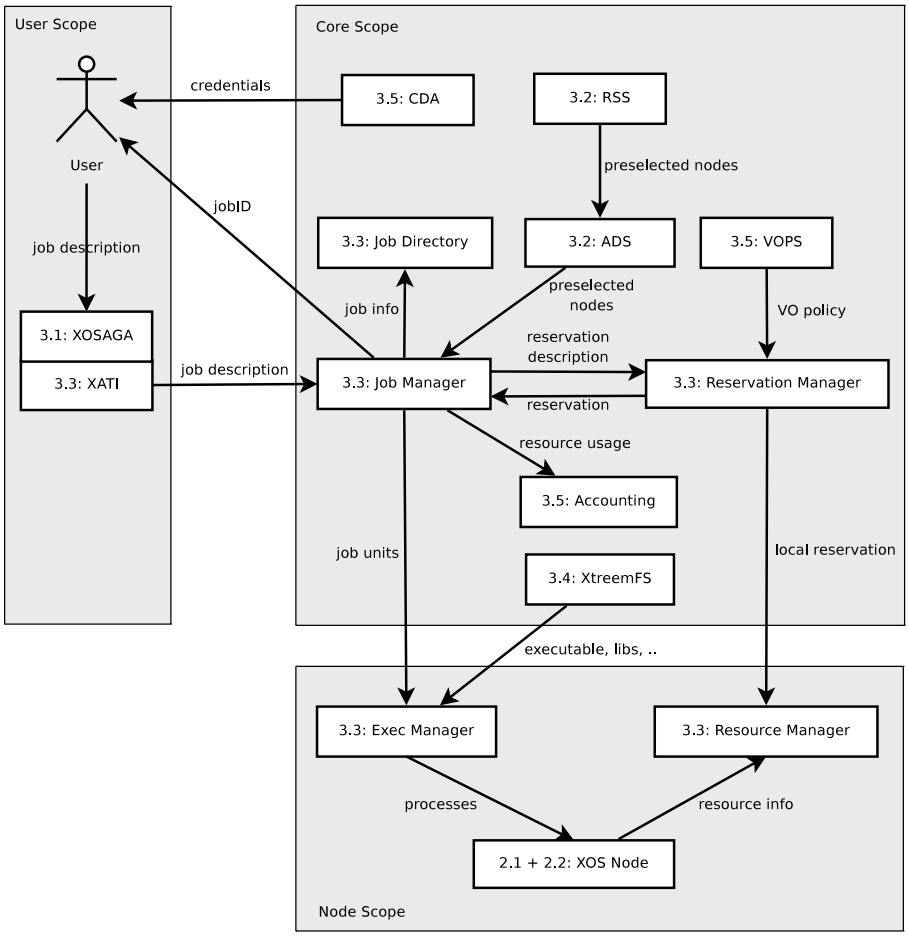

Figure 6: Job submission without an existing reservation

nodes. The executables, libraries and other input files needed by each job unit are obtained from XtreemFS by the Exec Manager of each node. Finally, the information about the job is stored in the Job Directory for future reference. Similarly, the resource usage of the job is stored in the Accounting Service.

Note that the information stored in the Job Directory is minimal, as most of the relevant information is kept in the Job Manager. The Job Directory contains the job ID, the contact point within the Job Manager to get the information of the job, and some security information.

Figure 6 shows the submission of a job for which no resources have been reserved yet. In this case, XtreemOS will first select and reserve the required resources, and then schedule the job on those resources. The resulting diagram is a merger of Figures 4 and 5.

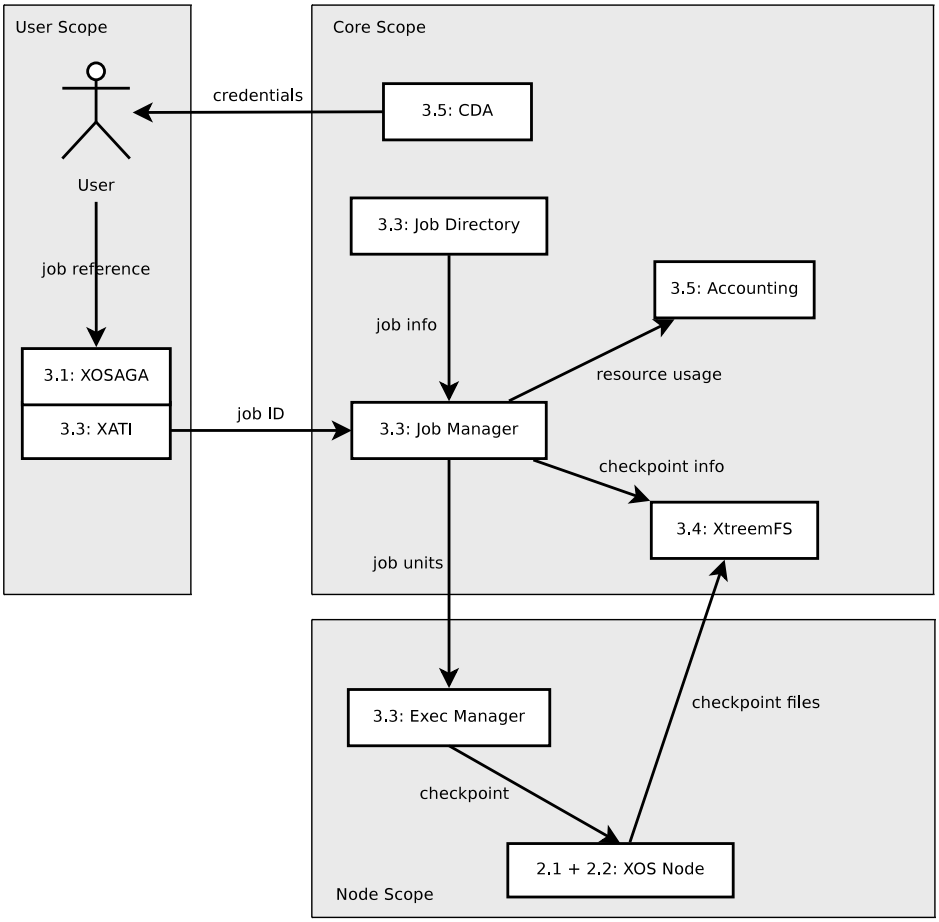

Figure 7: Manually checkpointing a running job

3.4 Checkpointing

Similar to job submission, there are two different scenario for initiating a checkpoint: manually or automatic. Manual checkpointing is initiated by the user, automatic checkpointing by the Job Checkpointer.

To create a manual checkpoint of a job (depicted in Figure 7), the user first needs to get its credentials from the CDA. It then contacts the Job Checkpointer with its credentials and the job ID to request a checkpoint. The Job Checkpointer contacts all nodes where a job unit is running and instructs the local Job Unit Checkpointer to start a checkpoint of this job unit. In turn, each Job Unit Checkpointer will requests a low-level checkpointer, assigned to a job unit, to checkpoint all processes within the job unit. All stored information (checkpoint files and additional checkpoint information) is kept in XtreemFS.

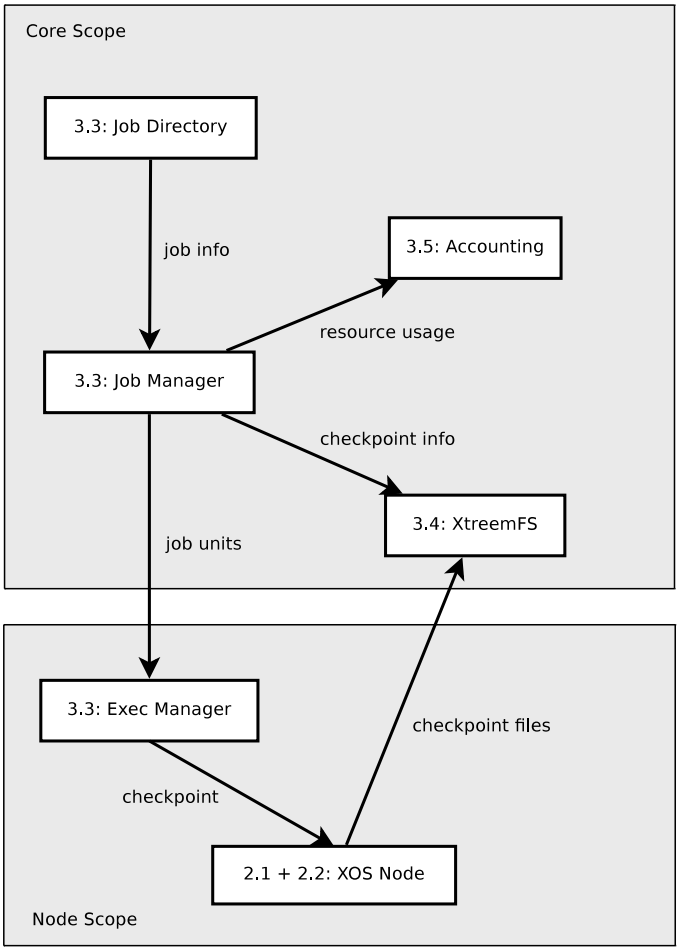

Figure 8: Automatic checkpointing of a running job

In the automatic checkpoint scenario (shown in Figure 8), it is the Job Checkpointer that decides that the job needs to be checkpointed. This can be due to a periodic checkpoint to guarantee some degree of fault tolerance, due to a migration decision etc. Compared to manual checkpointing, the only difference with automatic checkpointing is that the Job Manager triggers the action instead of the user; the resulting interaction of services is the same.

Restarting a job from a checkpoint is handled by the Job Checkpointer. It very much resembles job submission, but includes the checkpoint information and files stored in the checkpointing process.

By combining checkpointing and restarting, a job can be migrated from one set of resources to another. First the job on the old set of resources will be checkpointed, and then restarted on the new set of resources.

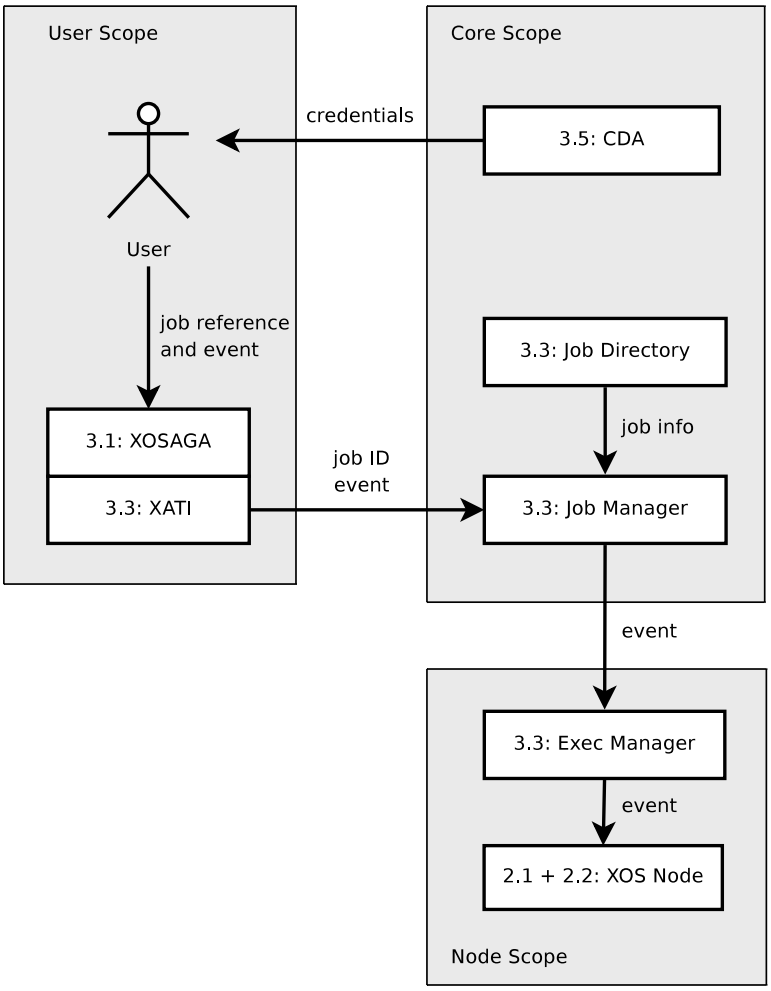

Figure 9: Event management

3.5 Event Management

XtreemOS users can send events to jobs. These events are extended versions of POSIX signals. In the general scenario (see Figure 9), the user gets the needed credentials from the CDA and then requests the Job Manager to send an event to the job identified by a job ID. The Job Manager contacts the Exec Managers in the nodes where the job has running processes and requests them to send an event to these processes. The default case is that all processes in the job receive the event, but mechanisms to decide which ones actually receive them will also be available.

In addition to events sent by the user, events can also be sent by the Job Manager. This resembles the way a Linux kernel can send signals to processes.

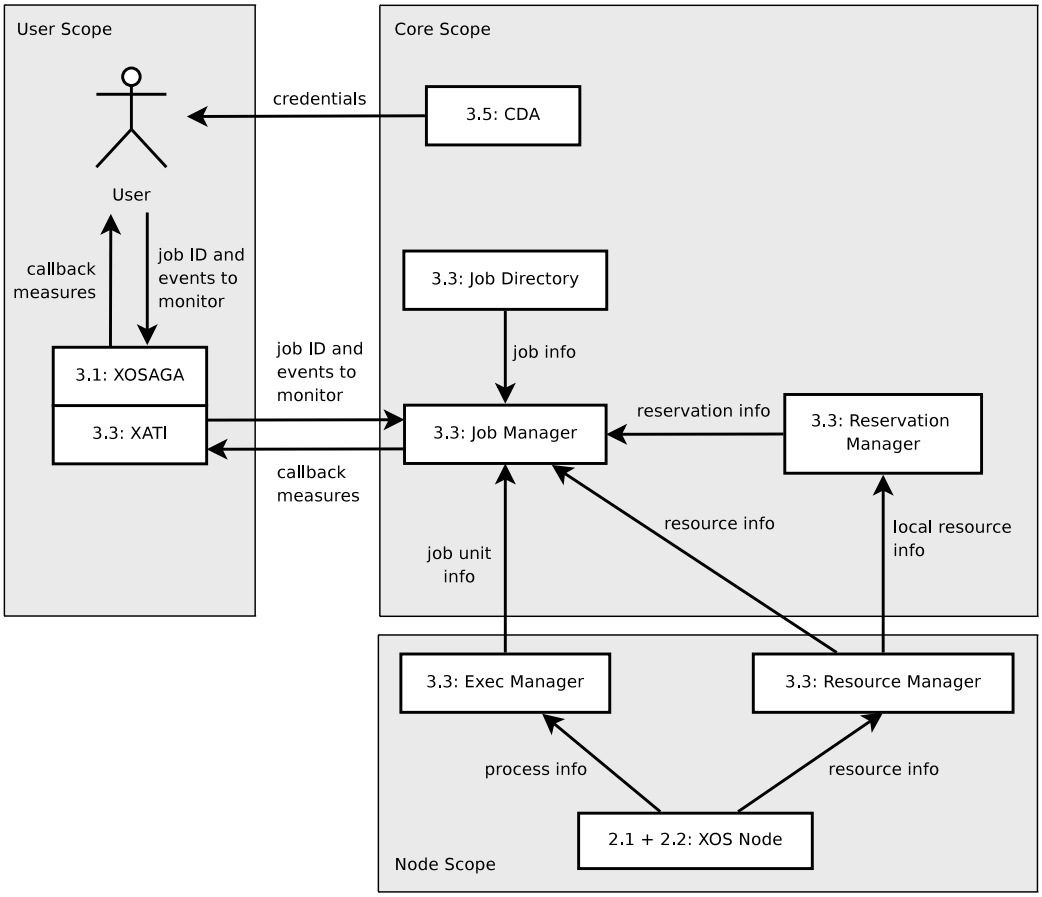

Figure 10: Monitoring

3.6 Monitoring

XtreemOS offers a much more detailed monitoring system than current approaches. Not only jobs, but also reservations and resources can be monitored. Figure 10 presents the components that play a role in monitoring. Once the user has obtained the right credentials it can ask the Job Manager to monitor some events. The Job Manager will request the information from the Reservation Manager, the Resource Manager and the Exec Manager. The monitored information can travel to the user in two ways. The first one is a 'pull' mechanism where the application requests certain information and waits for the reply. The second one is a 'push' mechanism where the Job Manager provokes a callback when a given event is measured or it reaches a certain value.

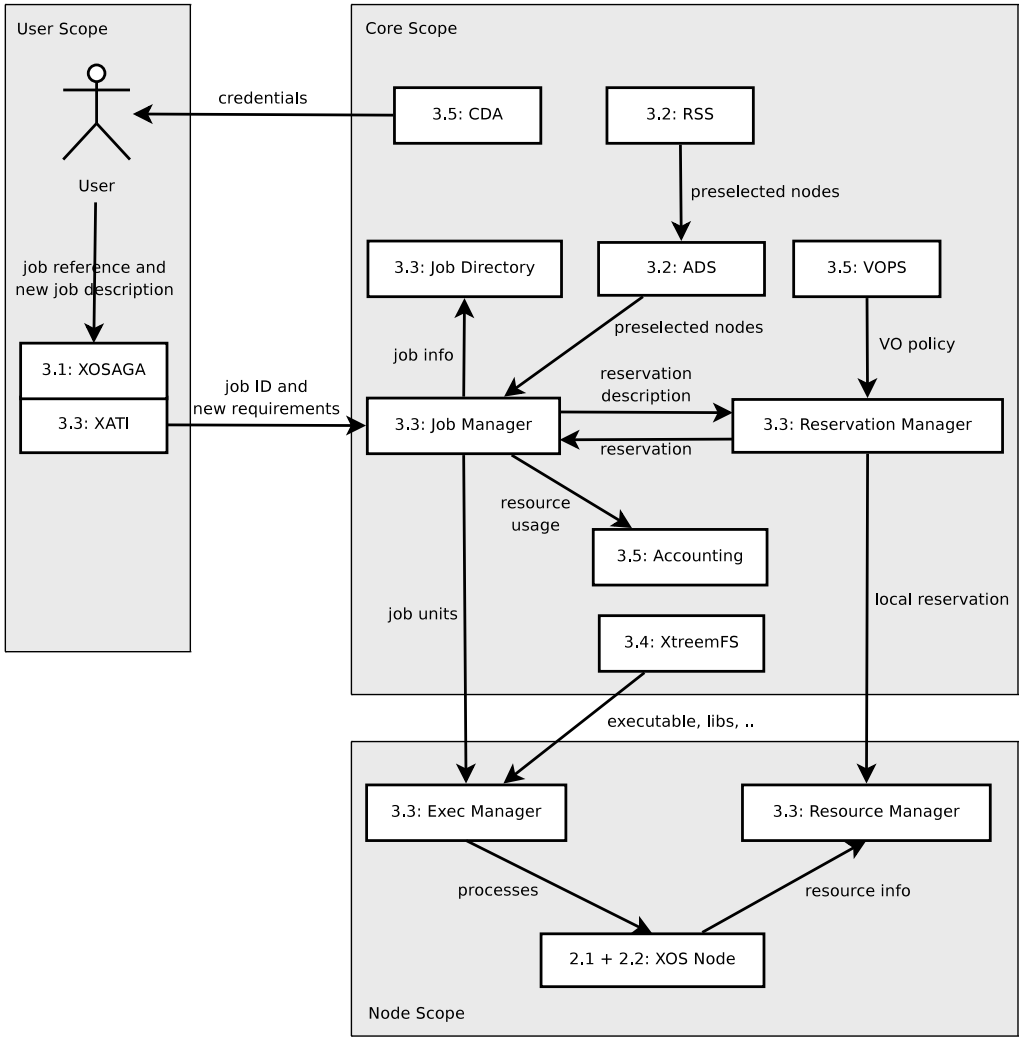

Figure 11: Dynamic resource allocation

3.7 Dynamic Resource Allocation

XtreemOS allows applications to change the number of resources they are using. For this reason, the system allows dynamic resource allocation. Figure 11 presents the services involved when a job requests more resources. The interaction of services is nearly the same as with job submission without having previously reserved resources (Figure 6). The only difference is that the job is already running, which means that instead of creating it, resources have to be requested via ADS, the reservation has to be modified by the Reservation Manager, and processes have to executed by the local Exec Managers.

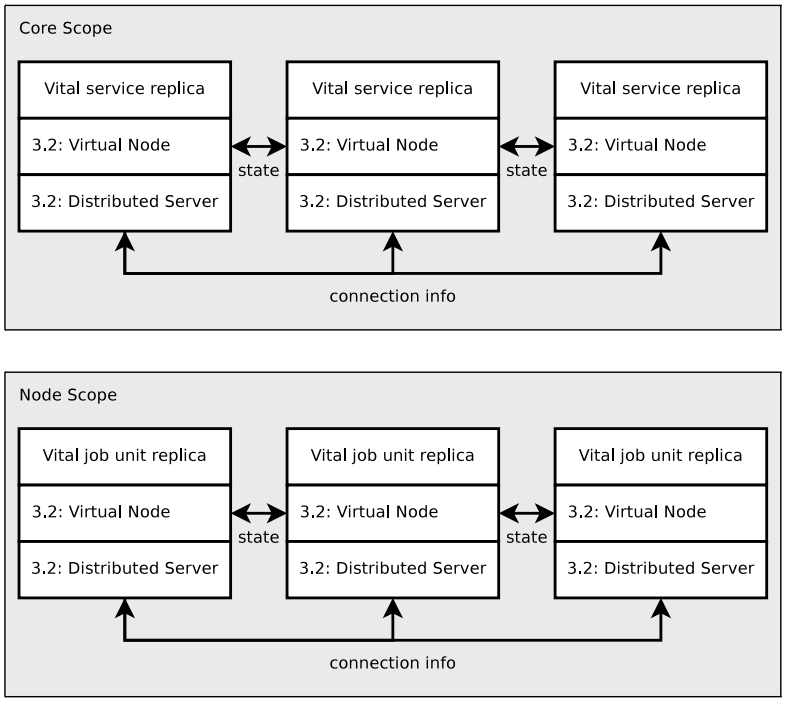

Figure 12: Fault-tolerant execution of vital XtreemOS services and jobs

3.8 Fault-Tolerant Execution

Some XtreemOS services are vital; if they are unavailable, a serious number of capabilities is lost. Examples of such services are the Job Manager and the VO Policy Service. Certain user applications could also desire high-availability. For this reason, XtreemOS provides transparent fault-tolerant execution of services.

As sketched in Section 2.5, fault-tolerant replication of a service or application is achieved by organizing it into a virtual node. Transparent access to a virtual node will be achieved using distributed servers. In case of a server failure, replicas within a virtual node can take over each other's tasks to provide continuous execution. Organizing virtual nodes as distributed servers ensures that they can be reached at a single stable IPv6 address, which makes the fault-tolerance transparent to clients.

Both XtreemOS services and user services can be made fault-tolerant by linking their code to special libraries, as shown in Figure 12. When running multiple copies of the same code on different machines, these libraries take care of all necessary communication between the copies to create a group of virtual nodes and distributed servers.

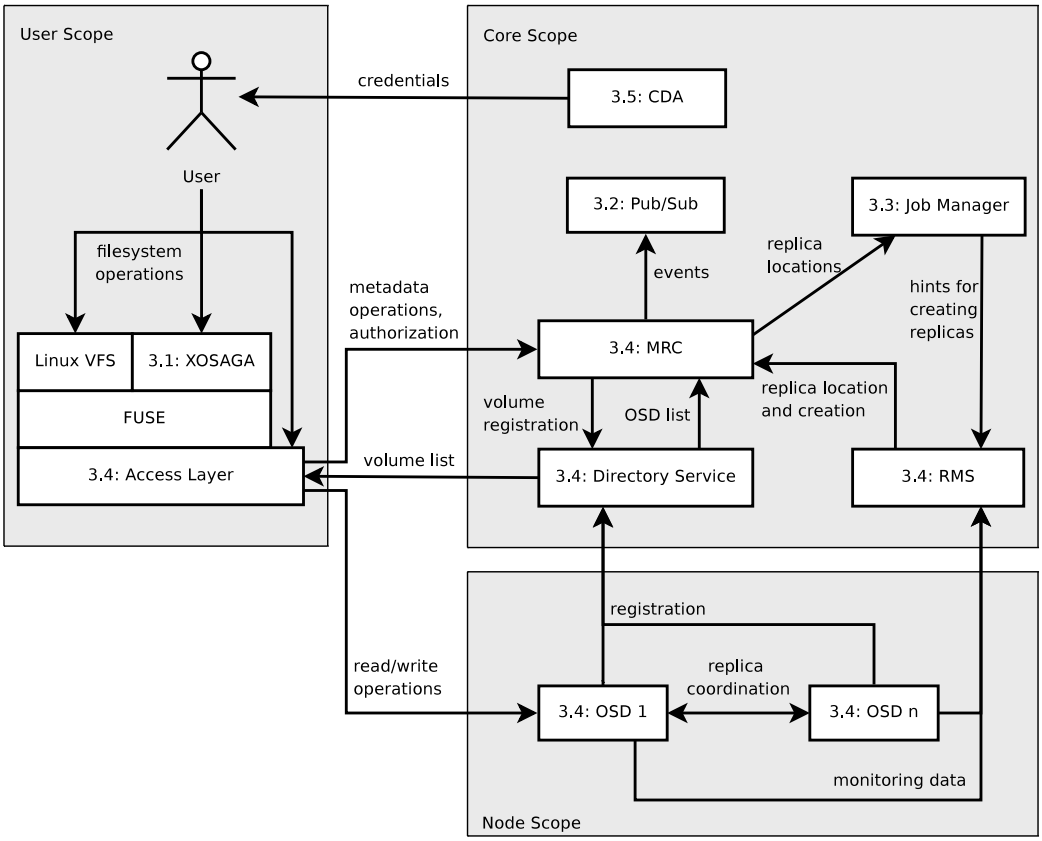

Figure 13: Data management and file replication

3.9 Data Management

Data management in XtreemOS is implemented by XtreemFS which is composed of several components in all scopes (see Figure 13). The file system is composed of the Access Layer, the Metadata and Replica Catalogs (MRC), the Replica Management Service (RMS) and the Object Storage Devices (OSD). The Access Layer implements a POSIX compatible API and translates all file system calls into corresponding invocations of XtreemFS services. Users can manually mount remote volumes via FUSE. In XOSAGA, volumes are referred to via URLs and automatically mounted by the runtime engine. Users can also use the Access Layer directly for more finegrained control and advanced features. The Directory Service is used as a registry for storage servers and volumes. Notification of file changes are disseminated using the Pub/Sub service.

3.10 File Replication

XtreemFS implements transparent file replication while maintaining POSIX compatibility (i.e. same semantics as a local file system). Figure 13 shows the Metadata and Replica catalogs that keep a list of replica locations for every file. Such information can be used by external services like the Job Manager to start jobs close to the storage location of the job's data. The creation and removal of replicas can be done manually by the user. In addition, the Replica Management Service is pro-actively creating and deleting replicas as needed. The Job Manager can send hints on future jobs to the RMS to allow for automatic replica creation before a job is executed. The replica consistency coordination is done transparently among the OSDs.

3.11 VO Lifecycle Management

In XtreemOS, the lifecycle of a virtual organization consists of three stages: creation, evolution, and dissolution. The management of this lifecycle involves a number of actors: a VO creator, VO members, VO administrators, resource administrators, and a VO manager.

The *VO creator* is the person who creates the VO. The *VO members* are users (consuming resources) and resources (providing resources) in the VO. The *VO administrators* perform administrative tasks, including adding VO members to and removing members from the VO, maintaining policies and attributes of the VO, and running services for the VO. The *resource administrator*, one per resource, is responsible for setting up policies for the resource, running services and registering the resource to a VO. The *VO manager* is a person or an organization responsible for the authenticity of the information, such as the identity and attributes of VO members and accounting information and audit trails of users, disseminated from the VO management services, such as CDA and accounting.

These actors are logical groupings by their responsibilities. In practice, one person or software service can take up the responsibilities of one or more actors. For example, a person can simultaneously become a VO creator, a VO member, a VO administrator, and a VO manager. In the very extreme case, a person can have the responsibilities of all these roles. The person (or a service) who creates a VO can become a VO member, a VO administrator, a VO manager, and a resource administrator, given that he also provides resources to the VO.

To set up a VO, a VO creator needs to specify the following information:

1. the public and private key pair of the VO manager (compulsory)
2. VO attributes (compulsory), for example, roles, groups, capabilities and/or other attributes that the VO supports

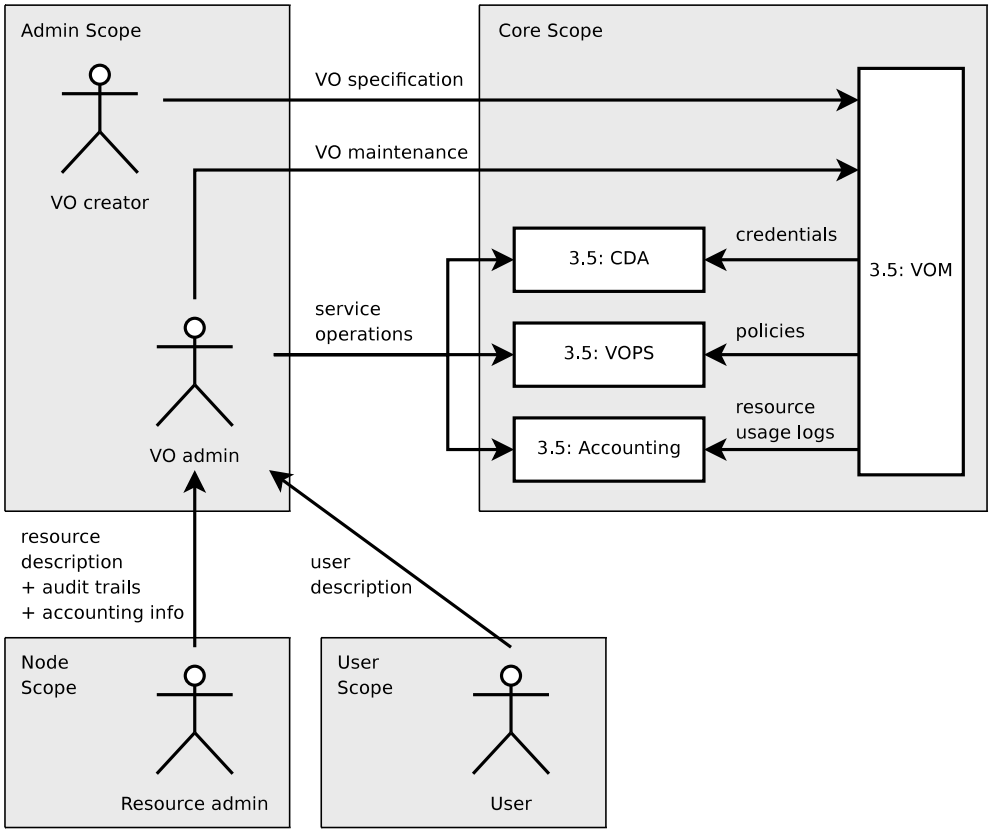

Figure 14: VO lifecycle management

3. a set of VO policies (optional)

4. a set of VO members (optional)

The information of (2 - 4) are maintained in the VO Management (VOM) database(s), whose structure is set up and are administrated by the VO administrators.

During the evolution phase, the VO administrators maintain the sets of VO policies, users, and resources in the corresponding databases. Also, the VO administrators are responsible for running the VO management services.

Upon the dissolution of the VO, all the relevant entries of the VO are deleted from the database(s) and the VO configuration is removed from the resources involved by the resource administrator.

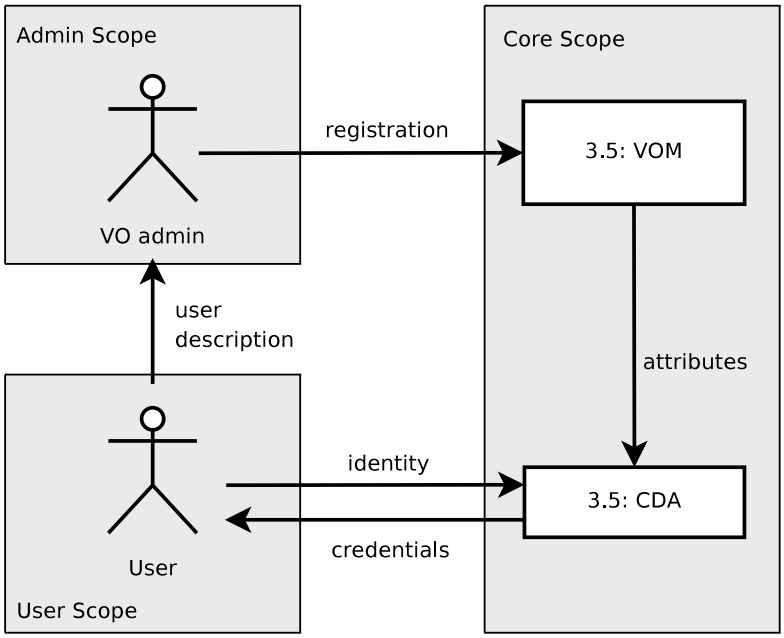

Figure 15: VO user management

3.12 VO Entity Management

There are two types of entities in a VO: users and resources. VO administrators are in charge of managing both of them as illustrated in Figure 15 and 16.

When a user registers with a VO, his identity and attributes (such as role, group, capabilities, and VO membership) are stored in the VO Management (VOM) database by the VO administrator. Based on the information in the database, the CDA service issues short-lived X.509 certificates (signed by the VO manager) to users. A user can associate with multiple attributes (e.g. roles and groups) in a VO. He can also simultaneously register with multiple VOs.

A resource admin can register a resource with multiple VOs. Upon receiving the description from a resource admin, the VO admin creates corresponding entries for the resource in the VOM database. The VO admin sends the configuration of this VO to the resource so that it can be configured as part of the VO.

By performing the VO configuration as instructed, the resource admin is committed to:

1. configure the resource as part of the VO, which implies that it will have the means to check the authenticity of the credentials from this VO by installing the trusted domain certificate.

2. make sure that appropriate local policies have been set up for this VO.

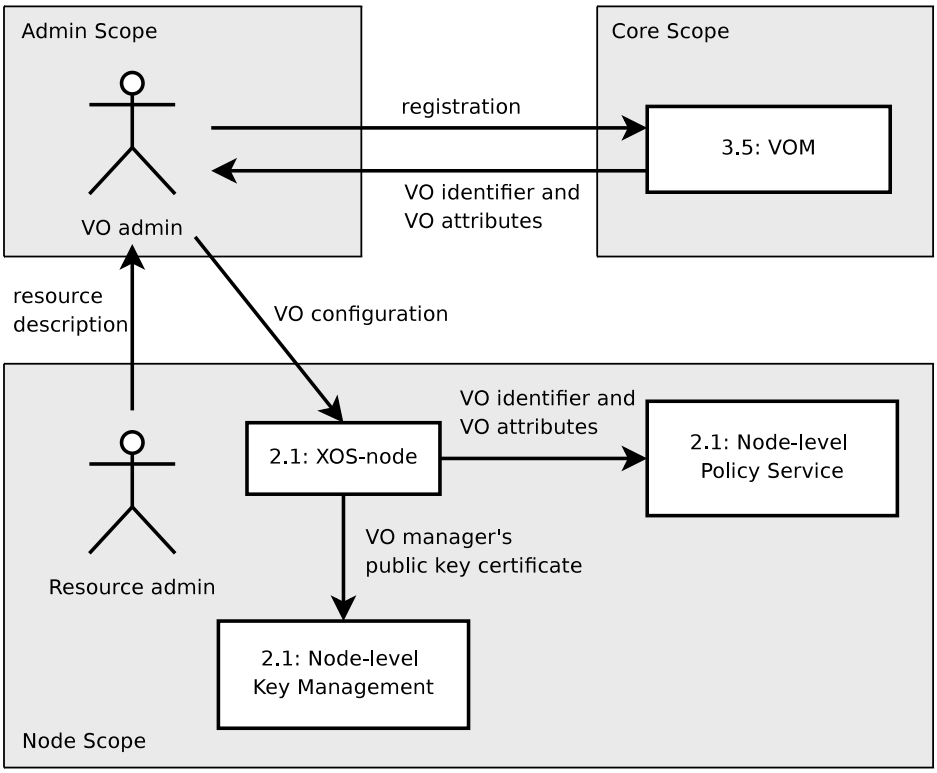

Figure 16: VO resource management

3. provide genuine resource usage data to the VO accounting service.

The removal of a resource from a VO involves a VO admin removing the resource entries from the VOM database and a resource admin removing the VO configuration from the node.

3.13 Policy Management

VO admins are in charge of managing *global* policies of a VO (Figure 17). These policies, used by the VOPS service and stored in a VO policy database, describe a VO-level access and usage control based on the description (characteristics) of users, resources, and/or requests (e.g. job description and resource requirements). VO admins can adapt the policies dynamically to balance the load on VO resources. The VO policy decisions are certified by the VO manager and such decisions can be verified by a resource.

In XtreemOS, nodes can also have node level policy management mechanisms to enforce local policies, which are independent from VO policies. VO policies

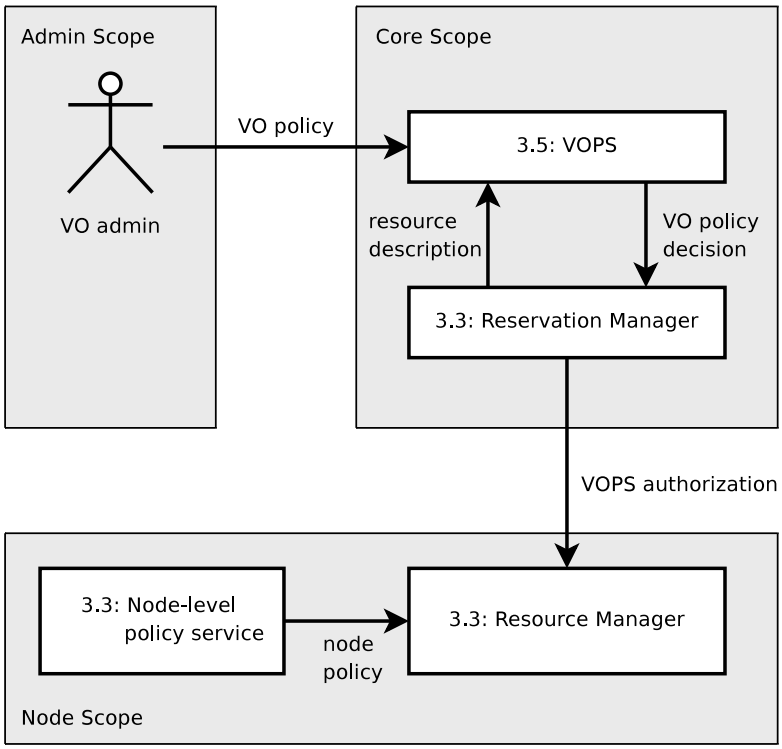

Figure 17: VO policy management

are set and managed by VO admins through the VOPS service whereas node policies are set and managed by resource admins via node level policy mechanisms. Checking whether a job conforms to local policies implies that a job is already compliant to VO policies. Hence, from a policy management point of view, a job running on a node means that it satisfies both VO and node policies.

When a resource is being added to a VO, be it during the setup or evolution of the VO, the resource can set up local policies in accordance to the VO attributes. However, each resource can come with a default set of generic local policies (such as users' file quota) which are agnostic to the VO attributes.

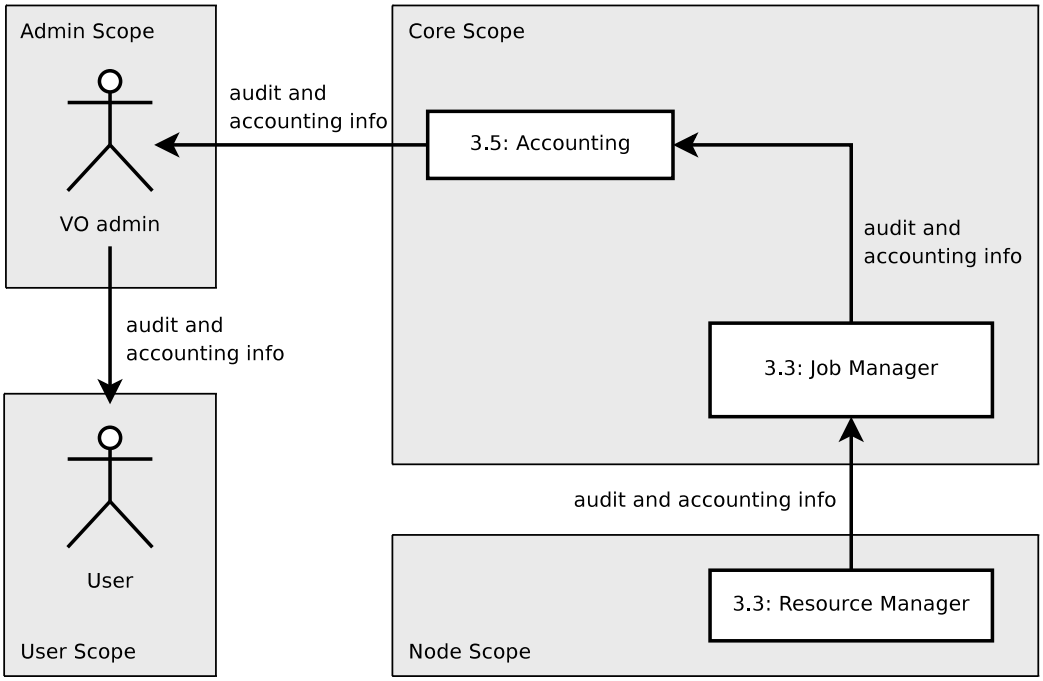

Figure 18: VO audit trail and accounting management

3.14 VO Accounting and Audit Trail Management

The VO accounting service receives audit and accounting information from job and resource management services. Such information is registered in an accounting database and is certified by the VO manager. The database is managed by the VO admin who disseminates the certified information to users (Figure 18).

4 Communication layer

The communication layer used by the XtreemOS components and services is split in two parts. First, the DIXI framework is used for flexible communication among XtreemOS nodes and services. Second, JSON/HTTP is used by XtreemFS to enable communication with clients within and also outside XtreemOS nodes.

4.1 Distributed XtreemOS Infrastructure (DIXI)

XtreemOS features a framework for developing, hosting and exploiting services that compose essential components of the XtreemOS system. We named this framework as DIstributed XtreemOS Infrastructure (DIXI). This framework provides more than just communication for the services, so it is an important part of XtreemOS. We have introduced and described the DIXI framework along with its basic architecture in [27]. In this section we summarize the motivation for the framework's development, summarize its features, and report on the current usage of the framework

4.1.1 Motivation

In the early designing stages of the AEM [24] a need arose for a framework that would

- provide a staging environment for the AEM services,
- simplify the development of a distributed system, composed of services,
- provide a communication layer between services running on the same node,
- provide communication between the nodes.

When designing DIXI, we kept the following guidelines in mind:

- The services should be as light-weight as possible.
- The target system would be highly distributed.
- The system should provide such a level of abstraction that the programmer would not have to implement communication-related logic. The framework would take care of the communication.
- The services would be written in Java.
- Peer services could use ready-made Java interfaces for issuing service calls.

- The client code could be written either in Java or in C.
- The communication layer would be interchangeable with any protocol required by the developers.

We decided for an approach that instantiates a staged event-driven architecture (SEDA) [36]. The outcome is a flexible framework that does not depend on external middleware instances. Instead, it uses ordinary TCP/IP for its communication with an option to use SSL for encryption.

4.1.2 Description

DIXI comprises a daemon for staging and hosting services, the XATI client interface, and some development tools.

Staging and hosting services. The purpose of the framework is to provide the place to run the services and their cooperation and communication. The architecture of the framework consists of the following elements:

Services represent the functionality implemented by the users.

Event Machine is a root object, which is used to load stages, and connects queues. Event Machine is responsible for executing the service calls upon request.

Message Bus Stage is an object that provides communication capabilities between various stages within the same instance of DIXI. It receives service messages from services, and, based on the service header information, passes them to the recipient service's event handler.

Communication Stage passes the service messages to other instances of DIXI on the same node or to those on other nodes using a standard network transport protocol (TCP/IP, SSL).

The services are thus decoupled from the rest of the DIXI layers. The communication between services takes the form of asynchronous Java calls. A service instantiates an interface of the recipient server, obtains the service's access point, and calls the method just like a regular Java class method. These calls, however, cause the framework to form and send a service message, and do not wait for the message's response, blocking the code execution. To obtain the return value of the call, the calling service registers a callback method. The Event Machine ensures the proper calling context is preserved when the callback is invoked, and calls the callback method when it receives the call's response.

XATI client interface. When a developer designs a service, the interfaces of the service become the API exposed to the client programs. To use the distributed services, clients need to import a library that represents a small version of the DIXI daemon. The service calls are conveniently available as static methods which, unlike the intra-service calls, block the execution until a client receives the response from the service.

Development tools. The most important part of the service development involves designing and writing the service implementations. These implementations by themselves cannot take part in a distributed system of services, because we need, for each service, the following items:

Service call entry point which exposes the services' API to other services, implements the translation from a service call into a service message, serialization of the parameters, and representation of the call-back methods.

Event handlers which translate the service messages into the method calls.

XATI client handlers that encapsulate both sending and receiving the service messages from the client programs and back from the services.

The Java implementation of the service already contains all the necessary information needed to automatically generate these items. DIXI thus enables that the developer uses proper `@Decoration` tags, marking up the methods that are to be exposed to other services and clients. The enclosed *code processor* then creates the enumerated items as well as others such as *console command* that let the user call the services directly from the command shell, as well as *client support for other programming languages*.

4.1.3 Usage of DIXI

In the first release of XtreemOS, DIXI hosts the majority of the AEM services: Job Manager, Job Directory, Resource Manager, Resource Monitor, and Execution Manager. We also included a wrapper class for the ADS/RSS services, making them available for the Job Directory and Resource Manager services.

The fact that the services do not have to worry about network communication makes the distinction between a core service and a node service a matter of deployment rather than a matter of the service's implementation. This is due to the fact that the transport layers and utility services will take care of a message originating from a node-level service to be sent to the core-level service, even if they are on different nodes. As a result, a seamless integration was possible with VOPS and RCA (with the RCA client service on each node).

4.2 HTTP/JSON

XtreemFS components communicate using JSON over the HTTP protocol.

4.2.1 Motivation

The main advantages of HTTP and JSON can be summarized as follows:

- Both HTTP and JSON are ASCII-based, which makes them human-readable and therefore easy to debug.
- HTTP is usually not blocked by firewalls; thus, a client can access an XtreemFS installation from any Internet-connected node.
- JSON is standardized and has a fairly simple notation. Compared to XML-based formats like SOAP or XML-RPC, it is easier to read and can be parsed more efficiently.

4.2.2 Description

JSON stands for JavaScript Object Notation and is a lightweight computer data interchange format. It is a text-based, human-readable format for representing simple data structures and associative arrays (called objects). Although it is based on a subset of the JavaScript Programming, JSON is a text format that is completely language independent. These properties make JSON an ideal data-interchange language.

JSON is built on two structures:

- A collection of name/value pairs. In various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array.
- An ordered list of values. In most languages, this is realized as an array, vector, list, or sequence.

These are universal data structures. Virtually all modern programming languages support them in one form or another. It makes sense that a data format that is interchangable with programming languages also be based on these structures.

4.2.3 Usage of HTTP/JSON

HTTP is used as the communication protocol between XtreemFS components, and JSON is used to encode message content. We run HTTP/JSON between the following components: client and MRC, client and OSD, client and Directory Service, OSD and OSD, OSD and Directory Service, and MRC and Directory Service.

5 Configurations

We have identified three main types of node configurations in XtreemOS: *core nodes*, *resource nodes*, and *client nodes*. These three configurations correspond to three scopes in the capability diagrams in Section 3: *core scope*, *node scope*, and *user scope*, respectively.

Core nodes provide the services that allow other nodes to participate in an XtreemOS system. They form the infrastructure.

Resource nodes provide computation and/or storage resources to the XtreemOS system. We distinguish two types of resource nodes: single PCs and clusters of PCs.

Client nodes only access an XtreemOS system, and do not necessarily belong to it. Single PCs and mobile devices can act as clients.

However, this is just a logical (and functional) division, and nothing precludes a certain machine from acting as, e.g., both a client node and a resource node. However, it is not advisable to allow access on core nodes to users other than system administrators.

The different purpose of each configuration also determines which services should be present on it. In the following, we will describe which services belong to which configuration. Each of these services has already been described individually in Section 2.

5.1 Core services

The core services together provide the infrastructure of an XtreemOS system.

Core VOM services handle Virtual Organization management (i.e. certificate issuing and signing authorities for the users and the resources). They also provide the means to edit VO-level policies and making policy decisions. The core VOM services are the XVOMS database, the VOLife management application, the CDA service, the RCA service, and the VOPS service.

Core AEM services oversee the job submission process, select the nodes for the jobs, schedule their execution, and book-keep the jobs submitted so far. The core AEM services are the Job Directory, Job Manager and Resource Manager.

Core XtreemFS services keep control of metadata as well as storage devices committed to the system. Core XtreemFS services are the Metadata Replica Catalog (MRC) and XtreemFS Directory Service.

5.2 Resource services

The resource services run on nodes that provide compute power and/or storage space to an XtreemOS system.

Resource VOM services are the VO-support PAM module that enables mapping of Grid accounts to local accounts and hence provides users with access to Grid nodes. The RCA Client service also extends the functionality to the resource nodes, helping the admins with an easier resource registration, and provides the RCA Server with a way to check the node's machine certificates.

Resource Discovery services support distributed information management. They set up essential overlay networks within the platform and provide highly scalable management and location of resources. These services include ADS and RSS, which together form the Scalable Resource Discovery System (SRDS).

Resource AEM services include the Execution Manager and the Resource Manager, which together handle job execution on a resource.

Resource XtreemFS services are called Object Storage Devices (OSDs), which store arbitrary objects of files.

5.3 Client services

Client service allow PCs and mobile devices to access an XtreemOS system.

Client VOM services are the CDA client (to get user XOS-Certificates) and sshxos (to login to an XtreemOS system with an XOS-certificate and access the home XtreemFS volumes).

Client AEM services are XATI (the API to the AEM services) and xconsole_dixi, a command-line interface for retrieving available resources and submitting jobs.

Client XtreemFS services consist of a FUSE user-level driver that runs as a non-priviledged process. FUSE itself is a kernel/user-level hybrid that connects the user-level driver to Linux' Virtual File System (VFS) layer where file system drivers usually live. A user can thus mount remote XtreemFS volumes locally.

6 Summary

This document describes the overall system architecture of XtreemOS, at the current stage of the project (Month 30). In combination with the overall layering of the XtreemOS software packages, as shown in Figure 2, we have followed a bottom-up approach by first describing the individual software packages (Section 2), followed by descriptions of the capabilities provided by the software packages together (Section 3), outlining the interactions and information exchanged between the components within the packages. This material has been revised and updated since the first system architecture report, D3.1.4. Two sections (4 and 5) have been added since D3.1.4, describing the communication layer among XtreemOS components and services, and the different node configurations, respectively.

As such, this document provides a comprehensive description of all software packages being produced by the XtreemOS project. It reflects the recently performed activities on package integration and software bundling that have lead to the first public release of XtreemOS.

The current status of the overall system architecture for XtreemOS is considered to be stable and complete. It is actually being used for building the current and upcoming releases of the XtreemOS operating system. As some of the described functionality is (according to the work plan) left for the remainder of the project, possibly new insights will only be gained later on, possibly leading to unforeseen dependencies or other issues. In case of such issues, another revised, final version of this system architecture description will be produced towards the completion of the project.

References

[1] XtreemOS Consortium. First Version of System Architecture. Deliverable D3.1.4, November 2007.

[2] Ian Foster, Carl Kesselman, and Steve Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International J. Supercomputer Applications*, 15(3), 2001.

[3] 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, 2007. Open Grid Forum (OGF).

[4] XtreemOS Consortium. Design and implementation in Linux of basic user and resource management mechanisms spanning multiple administrative domains. Deliverable D2.1.2, November 2007.

[5] XtreemOS Consortium. Design and implementation of basic application unit checkpoint/restart mechanisms in Linux. Deliverable D2.1.3, November 2007.

[6] Paul H. Hargrove and Jason C. Duell. Berkeley lab checkpoint/restart (blcr) for linux clusters. In *In Proceedings of SciDAC 2006*, June 2006.

[7] XtreemOS Consortium. Design and implementation of scalable SSI mechanisms in LinuxSSI. Deliverable D2.2.2, November 2007.

[8] XtreemOS Consortium. Design and implementation of basic checkpoint/restart mechanisms in LinuxSSI. Deliverable D2.2.3, November 2007.

[9] XtreemOS Consortium. Design and implementation of basic reconfiguration mechanisms in LinuxSSI. Deliverable D2.2.4, November 2007.

[10] XtreemOS Consortium. Design and implementation of high performance disk input-out operations in a cluster. Deliverable D2.2.5, November 2007.

[11] XtreemOS Consortium. Design and implementation of a basic customizable scheduler. Deliverable D2.2.6, November 2007.

[12] XtreemOS Consortium. Prototype of the basic version of LinuxSSI. Deliverable D2.2.7, November 2007.

[13] XtreemOS Consortium. Design and implementation of first advanced version of LinuxSSI. Deliverable D2.2.8, November 2008.

[14] Christine Morin, Renaud Lottiaux, Geoffroy Vallée, Pascal Gallard, David Margery, Jean-Yves Berthou, and Isaac Scherson. Kerrighed and data parallelism: Cluster computing on single system image operating systems. In *Proc. of Cluster 2004*. IEEE, September 2004.

[15] XtreemOS Consortium. Linux-XOS for MD/PDA. Deliverable D2.3.4, June 2008.

[16] XtreemOS Consortium. Requirements and Specifications for Advanced VO Support in Mobile Devices. Deliverable D2.3.5, December 2008.

[17] XtreemOS Consortium. Third Draft Specification of Programming Interfaces. Deliverable D3.1.5, November 2008.

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

[19] XtreemOS Consortium. Design of an Infrastructure for Highly Available and Scalable Grid Services. Deliverable D3.2.1, December 2006.

[20] XtreemOS Consortium. First Prototype Version of Ad Hoc Distributed Servers. Deliverable D3.2.2, December 2007.

[21] XtreemOS Consortium. Simulation-based evaluation of a scalable publish/subscribe system. Deliverable D3.2.3, December 2007.

[22] XtreemOS Consortium. Design and Specification of a Prototype Service/Resource Discovery System. Deliverable D3.2.4, December 2007.

[23] XtreemOS Consortium. Design and Specification of a Virtual Node System. Deliverable D3.2.5, December 2007.

[24] XtreemOS Consortium. Requirements and specification of XtreemOS services for application execution management. Deliverable D3.3.1, November 2006.

[25] XtreemOS Consortium. Design of the architecture for application execution management in XtreemOS. Deliverable D3.3.2, May 2007.

[26] XtreemOS Consortium. Basic services for application submission, control and checkpointing. Deliverable D3.3.3, November 2007.

[27] XtreemOS Consortium. Basic service for resource selection, allocation and monitoring. Deliverable D3.3.4, November 2007.

[28] XtreemOS Consortium. The XtreemOS File System - Requirements and Reference Architecture. Deliverable D3.4.1, November 2006.

[29] XtreemOS Consortium. XtreemFS and OSS - Second Prototype. Deliverable D3.4.4, November 2008.

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

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

[32] XtreemOS Consortium. Security Services Prototype month 18. Deliverable D3.5.5, December 2007.

[33] XtreemOS Consortium. Report on Formal Analysis of Security Properties. Deliverable D3.5.6, December 2007.

[34] XtreemOS Consortium. Design of Basic Services for Mobile Devices. Deliverable D3.6.2, June 2008.

[35] XtreemOS Consortium. XtreemOS-G for MD/PDA. Deliverable D3.6.3, December 2008.

[36] Matt Welsh. SEDA: an architecture for highly concurrent server applications. http://www.eecs.harvard.edu/m̃dw/proj/seda.