Project no. IST-033576

XtreemOS

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

Third Draft Specification of Programming Interfaces
D3.1.5

Due date of deliverable: November 30^{th}, 2008
Actual submission date: January 5^{th}, 2009

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

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

Revision history:

Version	Date	Authors	Institution	Section affected, comments
0.1	04/11/08	Ana Oprescu, Thilo Kielmann	VUA	initial draft
1.0	19/12/08	Thilo Kielmann	VUA	complete draft for internal review
1.0.1	05/01/09	Mathijs den Burger	VUA	processed comments of internal reviewers

Reviewers:
Michael Schöttner (UDUS), Eugenio Cesario (CNR)

Tasks related to this deliverable:

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

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

Executive Summary

This document presents the third draft specification of the API for XtreemOS. In the earlier deliverable D3.1.2 [17], we have proposed an extended SAGA API, called XOSAGA, as the second draft API. In this report, we present additions to XOSAGA, covering the so-far missing XtreemOS-specific functionality. In particular, we provide API's for the following XtreemOS functionalities: transparently distributed services (i.e., Distributed Servers), a scalable, transactional, distributed key-value store (i.e., Scalaris) and a transparent and consistent data sharing service (i.e., Object Sharing Service).

Contents

Executive Summary **1**

1 Introduction **3**

2 XtreemOS application requirements revisited **5**

3 Distributed servers and XOSAGA **5**
3.1 Distributed server sockets to SAGA Streams 5
3.2 Specification 6
3.3 Specification details 7

4 XtreemFS and XOSAGA **8**

5 OSS, Scalaris, and XOSAGA **9**
5.1 Shared buffers 9
5.1.1 Specification 9
5.1.2 Specification details 11
5.2 Shared events 19
5.2.1 Specification 19
5.2.2 Specification details 20
5.3 Shared properties 22
5.3.1 Specification 22
5.3.2 Specification details 23

6 Virtual Nodes and XOSAGA **25**

7 Summary **25**

Bibliography **25**

1 Introduction

The specification of the XtreemOS API is being performed by agreement between representatives of all project partners, representing both groups of interface providers and interface users. This is an iterative process: a first specification, intended as an early draft, has been agreed upon early in the project, such that immediate implementation experience can be gained and strengths and weaknesses can be identified. The first API specification has been documented in deliverable D3.1.1 [13]. The second API specification has been documented in deliverable D3.1.2 [17]. This document describes the third API specification draft, the first one to cover all XtreemOS-specific functionality. Future milestones during the project will cover revisions and improvements of this API, until the final version will meet all requirements of both applications and system implementors.

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

1. Existing Linux applications, using POSIX-standardized interfaces.
2. Existing grid applications, using OGF-standardized interfaces.
3. New applications, using functionality uniquely provided by XtreemOS.

In D3.1.1, we have selected the emerging OGF standard *Simple API for Grid Applications* (SAGA) as the first draft API for XtreemOS. SAGA had been selected because it combines OGF-standardized API's (namely JSDL [2], BES [3], GridFTP [7], GridRPC [9], DRMAA [10]) with POSIX-like interfaces wherever possible (e.g., for files and streams). Also, SAGA has been cited as a possible candidate API for XtreemOS by the project-internal application groups in deliverable D4.2.1 [14].

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

Figure 1 illustrates the interaction between the SAGA-related standardization and the XtreemOS API specification. Initially (in D3.1.1), XtreemOS has adopted the SAGA draft recommendation, documented in OGF's draft GWD-R.90 [4][1], as its first own API specification. Since then, XtreemOS has contributed to finalize the SAGA-CORE specification, with imminent publication as a draft recommendation, in OGF's document GFD.90 [5]. The publication of GFD.90 marks a crossroads at which the SAGA-CORE specification itself becomes fixed, and

[1] The draft [4] unfortunately refers to itself as GWD-R.72, which is wrong.

XtreemOS-related modifications or additions require the definition of separate API documents.

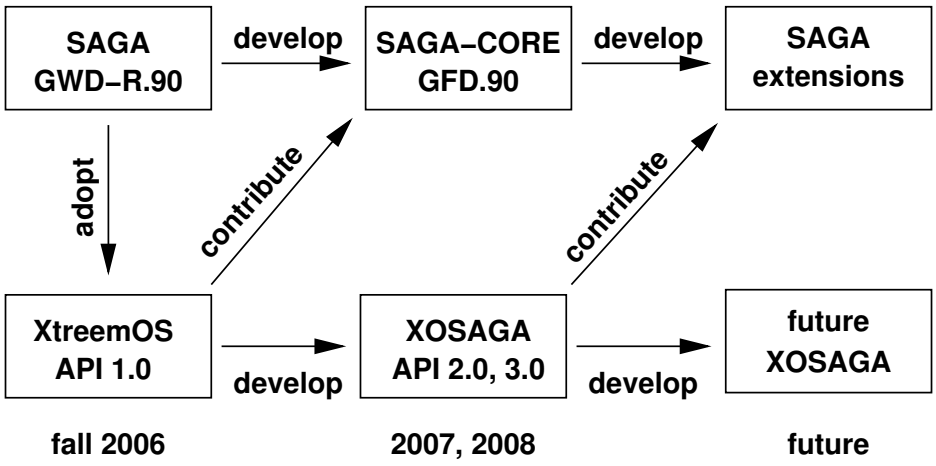

Figure 1: Co-development of XtreemOS API and SAGA standard.

This document covers the third draft API specification for XtreemOS. Its main focus is providing API extensions (w.r.t. to XOSAGA, the second draft API specification) that provide the so-far missing access to XtreemOS-specific functionality. For XtreemOS-specific extentions to SAGA, in D3.1.2 we defined an API name space called XOSAGA (XtreemOS extensions to SAGA) that mirrors the SAGA API name space. XOSAGA contains only those packages, classes, and interfaces that require XtreemOS-specific extensions to SAGA. Together, SAGA and XOSAGA form the XtreemOS API. By this design, applications require only minimal changes for being ported from "standard SAGA" to XOSAGA: applications simply have to create objects from classes from the XOSAGA name space, instead of classes with identical names, but from the SAGA namespace.

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

The bulk of this document consists of an analysis of functionality either specified or implemented so far by the XtreemOS work packages from sub projects SP2 and SP3, as identified in D3.1.2 as to be defined in this report. This report covers only the additions to XOSAGA as it had been defined in D3.1.2. The complete XtreemOS API thus consists of SAGA, the XOSAGA extensions from D3.1.2, and the XOSAGA extensions describe in this document.

The remainder of this document is organized as follows. First, we discuss

the updated application requirements in Section 2. Then, we analyze XtreemOS-specific functionality identified as missing in D3.1.2, namely regarding transparently distributed services (Section 3), the XtreemFS file system (Section 4), transparent data sharing services (Section 5) and transparent service replication (Section 6). Finally, Section 7 summarizes our findings.

2 XtreemOS application requirements revisited

The ulterior requirements to other work packages described by WP4.2 in deliverable D4.2.5 [16]. In that document, there are no further requirements to be addressed by the XtreemOS API, compared to the previous requirements document D4.2.3 [15]. In consequence, this document is focusing on providing the so-far missing, XtreemOS-related functionality. Other modifications have not been required.

3 Distributed servers and XOSAGA

Distributed servers are developed by WP3.2, and described in D3.2.2 (*First Prototype Version of Ad Hoc Distributed Servers*) [20]. A distributed server is an abstraction of a group of server processes, perceived by its clients as a single entity with a single stable IP address. These server processes together provide scalable, high-performance client-to-server communication without any client-awareness of the system.

The key technology employed in distributed servers is *versatile anycast* [12], which uses the Linux implementation [8] of the Mobile IPv6 protocol [6]. Since the system also provides mid-connection handoffs, server TCP sockets rely on TCP connection passing (TCPCP) technology [1].

3.1 Distributed server sockets to SAGA Streams

In XOSAGA, distributed server sockets are accessible via *streams* in SAGA. The `saga.stream` package provides a simple interface for TCP/IP socket connections. Integrating the functionality of distributed servers with XOSAGA requires two additions to SAGA:

1. A new scheme '`tcpcp`' for the endpoint URLs of distributed server sockets
2. An extension of the SAGA API to allow a server socket to perform mid-connection handoffs

The syntax of a URL of a distributed server socket is as follows:

```
tcpcp:// host :  port
```

host is the stable server address of a distributed server.

port is the port number of the distributed server socket.

The extension of server sockets is provided in the `xosaga.stream` package, which inherits all classes of the `saga.stream` package. The new `stream_service` object can be constructed with a `saga::url` that has a '`tcpcp`' scheme. Its `serve()` method will return a new `stream` object that implements two additional `handoff()` methods. These methods can be used by a server application to handoff its connection to another server within the group. A client-side application can just use a `saga::stream` object to communicate with a distributed server application.

3.2 Specification

```
package xosaga.stream {

 enum state
 {
    // as in saga.stream
 }

 enum activity
 {
    // as in saga.stream
 }

 class stream_service : extends saga::stream_service
 {
    serve       (in  float  timeout = -1.0,
                 out stream s);
 }

 class stream : extends saga::stream
 {
    // more management methods
    handoff     (in bool nonblock);

    handoff     (in saga::url url,
                 in bool nonblock);
 }
}
```

3.3 Specification details

Class `stream_service`

As described in the SAGA core document, the `stream_service` object establishes a listening/server object that waits for client connections. It can only be used as a factory for client sockets. It does not do any read/write I/O, but provides access to xosaga::stream objects.

```
- serve
  Purpose:  wait for incoming client connections
  Format:   serve   (in  float  timeout,
                     out stream s);
  Inputs:   timeout:  number of seconds to wait for a client
  InOuts:   -
  Outputs:  s:        new 'Connected' stream object
  PreCond:  -
  PostCond: - the returned client is in 'Open' state.
            - the session of the returned client is that of
              the stream service.
  Perms:    - Exec.
            - Exec for the connecting remote party.
  Throws:   NotImplemented
            IncorrectState
            PermissionDenied
            AuthorizationFailed
            AuthenticationFailed
            NoSuccess
            Timeout
  Notes:    All the notes of saga::stream_service.serve() apply
```

Class `stream`

This is the object that encapsulates all functionalities of client streams. It includes the handoff management methods.

```
- handoff
  Purpose:  handoff this connection to a different server.
  Format:   handoff  (in  bool nonblock);
  Inputs:   nonblock: whether this handoff is blocking until
                      completion.
  InOuts:   -
  Outputs:  -
  PreCond:  stream is 'New' or 'Open' state
```

```
  PostCond: stream is in 'Closed' state
  Perms:    -
  Throws:   IncorrectState
            NoSuccess
  Notes:    -

- handoff
  Purpose:  handoff this connection to a different server.
  Format:   handoff  (in  saga::url url,
                      in  bool      nonblock);
  Inputs:   url:      the acceptor server
            nonblock: whether this handoff is blocking until
                      completion.
  InOuts:   -
  Outputs:  -
  PreCond:  stream is 'New' or 'Open' state
  PostCond: stream is in 'Closed' state
  Perms:    -
  Throws:   IncorrectState
            NoSuccess
  Notes:    -
```

---

4 XtreemFS and XOSAGA

An XOSAGA application can access the XtreemFS file system via the existing `saga.namespace` and `saga.file` packages. Referring to files and directories on a certain XtreemFS volume is done via URLs with the scheme '`xtreemfs`'. The syntax of these URLs is as follows:

```
xtreemfs:// volume @ host [:port] path
```

`volume` is the name of the XtreemFS volume.

`host` is the host name of the MRC that manages the volume.

`port` is the port number the MRC listens to.

`path` is the path of the file or directory on the volume.

All URL parts are mandatory, except for `port`. If no port number is specified, a default port number is used. The default port number can be specified in the configuration file of the XtreemFS adaptor. An example XtreemFS URL is:

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

This URL refers to the file '/dir/file.txt' on an XtreemFS volume named 'vol42', which is managed by an MRC that runs on 'host.example.com' and listens to port 32636.

5 OSS, Scalaris, and XOSAGA

XOSAGA provides a new package `xosaga.sharing`. This package provides three types of objects that can be shared between the processes of a distributed SAGA application: *shared buffers*, *shared properties* and *shared events*.

Shared buffers expose the functionality of the Object Sharing Service (OSS) at the SAGA level. OSS provides a transparent and consistent data sharing service, as described in D3.4.3 (*Design report for advanced XtreemFS and OSS features*) [18]. Currently, it features memory-mapped files and transactional memory for volatile memory objects. In XOSAGA, such memory regions are made available as special SAGA *buffers*.

Shared properties and shared events allow an XOSAGA application to use the Scalaris system [11] developed in WP3.2. Scalaris provides a publish-subscribe ring on top of a scalable, transactional, distributed key-value store. In XOSAGA, the publish-subscribe rings are expressed as *shared events*, while the key-value stores are available as *shared properties*.

5.1 Shared buffers

A *shared buffer* is a special SAGA buffer that can be shared between multiple application processes. Each shared buffer lives in a *domain* with a certain consistency model. All shared buffers in the same domain are synchronized with each other using the consistency model of the domain. Each buffer has a unique identifier by which different application processes can identify the same shared buffer.

5.1.1 Specification

```
package xosaga.sharing {

  class shared_buffer_service
  {
    CONSTRUCTOR         (in  saga::url             bootstrap_info,
                         out shared_buffer_service obj);
    DESTRUCTOR          (in  shared_buffer_service obj);
```

```
  create_transactional_domain
                     (in  string name,
                      out transactional_consistency_domain d);

  create_weak_domain (in  string name,
                      out weak_consistency_domain d);
}

class consistency_domain
{
  get_name             (out string           name);

  create_buffer        (in  int              size,
                        out shared_buffer    buf);

  memory_map           (in  saga::file       file,
                        in  int              offset,
                        in  int              length,
                        out shared_buffer    buf);

  get_buffer           (in  shared_buffer_id id,
                        out shared_buffer    buf);

  free_buffer          (in  shared_buffer_id id);

}

class transactional_consistency_domain :
                                 extends consistency_domain
{
  begin                (out transaction_id   tid);

  commit               (in  transaction_id   tid);

  abort                (in  transaction_id   tid);

  permit_abort         (in  transaction_id   tid);

}

class weak_consistency_domain : extends consistency_domain
{
  sync                 ();
}

class shared_buffer : extends      saga::buffer
                    // from buffer  saga::object
                    // from buffer  saga::error_handler
```

```
{
  DESTRUCTOR     ();

  get_id         (out shared_buffer_id       id);

  set_size        ();

  set_data        ();

  close           ();
}

class shared_buffer_id
{
  CONSTRUCTOR    (in  string                 s,
                  out shared_buffer_id       id);

  DESTRUCTOR     (in  shared_buffer_id       id);

  to_string      (out string                 s);
}

class transaction_id
{
  // no public methods, immutable object
}
```

5.1.2 Specification details

Class `xosaga::shared_buffer_service`

The `xosaga::shared_buffer_service` class offers consistency domain management functionalities for shared buffers. Domains can be created with a specific consistency model to be enforced upon the shared buffers of each domain. At this point, the API includes transactional and weak consistency models.

```
- CONSTRUCTOR
  Purpose:  create an service to manage shared buffers with
            various forms of consistency.
  Format:   CONSTRUCTOR (in  saga::url             bootstrap_info,
                         out shared_buffer_service obj);
  Inputs:   bootstrap_info: the bootstrap information for the
                            service.
                            Example URL: 'oss://host.com:12345',
                            which connects to another OSS node
```

```
                          at host.com, port 12345
  InOuts:   -
  Outputs:  shared_buffer_service:  the newly created service
  PreCond:  -
  PostCond: -
  Perms:    -
  Throws:   IncorrectState
            IncorrectURL
  Notes:    - An implementation may only allow a single instance
              of a shared buffer service. In that case, all
              subsequently created instances MUST throw
              an 'IncorrectState' exception.

- DESTRUCTOR
  Purpose:  destroys the manager of shared buffers
  Format:   DESTRUCTOR           (in  shared_buffer_service obj);
  Inputs:   shared_buffer_service:      the service to destroy
  InOuts:   -
  Outputs:  -
  PreCond:  -
  PostCond: consistency domains and buffers created by
            this service are not affected.
  Perms:    -
  Throws:   -
  Notes:    -

- create_transactional_domain
  Purpose:  create a domain for buffers with transactional
            consistency
  Format:   create_transactional_domain
                        (in  string name,
                         out transactional_consistency_domain d);
  Inputs:   name:   the name of the transactional consistency
                    domain. It uniquely identifies this domain
                    on all nodes that participate in the same
                    shared buffer service.
  InOuts:   -
  Outputs:  d:      the transactional consistency domain
                    with the given name.
  PreCond:  -
  PostCond: -
  Perms:    -
  Throws:   -
  Notes:    -

- create_weak_domain
  Purpose:  create a domain for buffers with weak consistency
  Format:   create_weak_domain (in  string name,
                                out weak_consistency_domain d);
```

```
  Inputs:   name:   the name of the weak consistency
                    domain. It uniquely identifies
                    this domain on all nodes
                    that participate in the same
                    shared buffer service.
  InOuts:   -
  Outputs:  d:      the weak consistency domain
                    with the given name.
  PreCond:  -
  PostCond: -
  Perms:    -
  Throws:   -
  Notes:    -
```

Class `xosaga::consistency_domain`

The `xosaga::consistency_domain` class offers generic management operations on a consistency domain, independent of its consistency model. It also provides the API for obtaining the handle to a shared buffer and releasing it.

```
- get_name
  Purpose:  returns the name of this consistency domain
  Format:   get_name (out string           name);
  Inputs:   -
  InOuts:   -
  Outputs:  name:    the name of this consistency domain
  PreCond:  -
  PostCond: -
  Perms:    -
  Throws:   -
  Notes:    -

- create_buffer
  Purpose:  create a new shared buffer in this
            consistency domain. All buffers
            in the same consistency domain
            (i.e. with the same name) are kept
            consistent with each other.
  Format:   create_buffer (in  int              size,
                           out shared_buffer    buf);
  Inputs:   size:    the size of the new buffer in bytes
  InOuts:   -
  Outputs:  buf:     the created buffer
  PreCond:  -
  PostCond: -
```

```
  Perms:    -
  Throws:   BadParameter
  Notes:    - if size < 0, a 'BadParameter' exception
            MUST be thrown

- memory_map
  Purpose:  map a file into a new shared buffer
            in this consistency domain. All buffers
            in the same consistency domain
            (i.e. with the same name)
            are kept consistent with each other.
  Format: memory_map (in  saga::file       file,
                      in  int              offset,
                      in  int              length,
                      out shared_buffer    buf);

  Inputs:   file:    the file to map into memory
            offset:  the offset in the file where
                     the mapping starts
            length:  the amount of bytes to map,
                     starting from offset
  InOuts:   -
  Outputs:  buf:     a new shared buffer containing
                     the memory-mapped file
  PreCond:  -
  PostCond: -
  Perms:    -
  Throws:   BadParameter
            NoSuccess
  Notes:    - if the given file cannot be read from
            or written to, a 'BadParameter' exception
            MAY be thrown
            - if offset < 0, length < 0, or
            offset + length > file.get_size(),
            a 'BadParameter' exception MUST be thrown

- get_buffer
  Purpose:  get a shared buffer that is already created
            in this consistency domain
            (possibly on another node)
  Format: get_buffer (in  shared_buffer_id id,
                      out shared_buffer    buf);
  Inputs:   id:    the identifier of a shared buffer
  InOuts:   -
  Outputs:  buf:     the existing shared buffer
  PreCond:  -
  PostCond: -
  Perms:    -
  Throws:   DoesNotExist
```

```
  Notes:    - if no buffer with the given identifier
            exists in this consistency domain,
            a 'DoesNotExist' exception MUST be thrown.

- free_buffer          (in  shared_buffer_id id);
  Purpose:  globally remove a buffer from this
            consistency domain, and release its memory.
            If the buffer contains a memory-mapped file,
            its content is synchronized to disk first.
  Format:   free_buffer (in  shared_buffer_id id);
  Inputs:   id:      the identifier of a shared buffer
  InOuts:   -
  Outputs:  -
  PreCond:  -
  PostCond: -
  Perms:    -
  Throws:   DoesNotExist
  Notes:    - if no buffer with the given identifier
            exists in this consistency domain,
            a 'DoesNotExist' exception MUST be thrown.
```

Class `xosaga::transactional_consistency_domain`

The `xosaga::transactional_consistency_domain` class provides specific operations for the transactional consistency model.

```
- begin
  Purpose:  begin a transaction on all shared buffers
             in this domain
  Format:   begin (out transaction_id   tid);
  Inputs:   -
  InOuts:   -
  Outputs:  tid:     the identifier of this transaction
  PreCond:  -
  PostCond: -
  Perms:    -
  Throws:   NoSuccess
  Notes:    -

- commit
  Purpose:  end a transaction
  Format:   commit (in  transaction_id   tid);
  Inputs:   tid:     the identifier of this transaction
  InOuts:   -
  Outputs:  -
  PreCond:  -
```

```
  PostCond: -
  Perms:    -
  Throws:   DoesNotExist
            NoSuccess
  Notes:    - if the given transaction id is not known,
            a 'DoesNotExist' exception MUST be thrown

- abort
  Purpose:  unconditionally abort a transaction
  Format:   abort (in  transaction_id   tid);
  Inputs:   tid:     the identifier of this transaction
  InOuts:   -
  Outputs:  -
  PreCond:  -
  PostCond: -
  Perms:    -
  Throws:   DoesNotExist
            NoSuccess
  Notes:    - if the given transaction_id is not known,
            a 'DoesNotExist' exception MUST be thrown

- permit_abort
  Purpose:  permit aborting a transaction during
            the duration of this method call
  Format:   permit_abort (in  transaction_id   tid);
  Inputs:   tid:     the identifier of this transaction
  InOuts:   -
  Outputs:  -
  PreCond:  -
  PostCond: -
  Perms:    -
  Throws:   DoesNotExist
            NoSuccess
  Notes:    - if the given transaction_id is not known,
            a 'DoesNotExist' exception MUST be thrown
```

Class **xosaga::weak_consistency_domain**

The `xosaga::weak_consistency_domain` class provides specific operations for the weak consistency model.

```
- sync
  Purpose:  synchronize all shared buffers
            in this consistency domain
  Format:   sync ();
  Inputs:   -
```

```
  InOuts:   -
  Outputs:  -
  PreCond:  -
  PostCond: -
  Perms:    -
  Throws:   NoSuccess
  Notes:    -
```

Class `xosaga::shared_buffer`

The `xosaga::shared_buffer` provides access to a shared buffer.

```
- DESTRUCTOR
  Purpose:  destroys this shared buffer
  Format:   DESTRUCTOR      (in  shared_buffer obj);
  Inputs:   shared_buffer:   the shared buffer to destroy
  InOuts:   -
  Outputs:  -
  PreCond:  -
  PostCond: - the local memory used by this shared buffer
              is not released. Releasing the memory can
              only be done globally, using
              consistency_domain.free_buffer().
  Perms:    -
  Throws:   -
  Notes:    -

- get_id
  Purpose:  return the identifier of this shared buffer
  Format:   get_id (out shared_buffer_id       id);
  Inputs:   -
  InOuts:   -
  Outputs:  id:    the identifier of this shared buffer
  PreCond:  -
  PostCond: -
  Perms:    -
  Throws:   -
  Notes:    -

- set_size
  Notes:    - this method MUST always throw
            a 'NotImplementedException'

- set_data
  Notes:    - this method MUST always throw
            a 'NotImplementedException'
```

```
- close
  PostCond: - other nodes can still access
            the contents of this shared buffer
```

Class `xosaga::shared_buffer_id`

The `xosaga::shared_buffer_id` uniquely identifies a shared buffer in a consistency domain.

```
- CONSTRUCTOR
  Purpose:  create a shared buffer id
  Format:   CONSTRUCTOR (in  string              s,
                         out shared_buffer_id    id);
  Inputs:   s:     a string description of the identifier
  InOuts:   -
  Outputs:  id:    the new identifier
  PreCond:  -
  PostCond: -
  Perms:    -
  Throws:   BadParameter
  Notes:    - if the string description does not
            describe a valid shared buffer identifier,
            a 'BadParameter' exception MUST be thrown.

- DESTRUCTOR
  Purpose:  destroys this shared buffer id
  Format:   DESTRUCTOR        (in  shared_buffer_id obj);
  Inputs:   shared_buffer_id:  the shared buffer id to destroy
  InOuts:   -
  Outputs:  -
  PreCond:  -
  PostCond: -
  Perms:    -
  Throws:   -
  Notes:    -

- to_string      (out string                 s);
  Purpose:  return a string description of this identifier
  Format:   to_string (out string            s);
  Inputs:   -
  InOuts:   -
  Outputs:  s:     a string description of this identifier
  PreCond:  -
  PostCond: -
  Perms:    -
```

```
Throws:   -
Notes:    -
```

Class `xosaga::transaction_id`

This class contains no public methods, since it is an immutable object.

5.2 Shared events

The `shared_events` object in the `xosaga.sharing` package provides access to a publish-subscribe system. It was designed to provide access to the publish-subscribe functionality of the Scalaris system, but the interface is generic enough to support other publish-subscribe systems too.

An XOSAGA application process can publish events under a certain topic. Both events and topics are string values. Processes can also subscribe to certain topics, after which they will receive the events that are published under these topics. New events are processed in callback functions that are provided when subscribing to a topic.

5.2.1 Specification

```
package xosaga.sharing {

  interface callback
  {
    cb  (in  shared_events  se,
         in  string         topic,
         in  string         content);
  }

  class shared_events
  {
    CONSTRUCTOR (in  saga::url     bootstrap_info,
                 out shared_events obj);

    DESTRUCTOR  (in  shared_events obj);

    publish     (in  string        topic,
                 in  string        content);

    subscribe   (in  string        topic,
                 in  callback      cb);

    unsubscribe (in  string        topic);
```

```
  }
 }
```

5.2.2 Specification details

Interface `callback`

This interface specifies a method that handles incoming events. This method has to be provided when subscribing to a certain topic.

```
- cb
  Purpose:  provide a callback method to handle events for which
            this callback was registered by a subscribe method.
  Format:   publish (in  shared_events  se,
                     in  string         topic,
                     in  string         content);
  Inputs:   se:          the ring of shared events
            topic:       the updated topic
            content:     the content published under 'topic'.
  InOuts:   -
  Outputs:  -
  PreCond:  -
  PostCond: -
  Perms:    -
  Throws:   -
  Notes:    -
```

Class `shared_events`

This class provides the methods to publish events under certain topics, and subscribe to events.

```
- CONSTRUCTOR
  Purpose:  create a service that manages shared events
            within a publish-subscribe ring.
  Format:   CONSTRUCTOR (in  saga::url     bootstrap_info,
                         out shared_events obj);
  Inputs:   bootstrap_info: the bootstrap information for the
                            service. Example URL:
                            'pubsub://host.com:12345', which
                            connects to a publish-subscribe
                            ring at host.com, port 12345
  InOuts:   -
  Outputs:  shared_events:  the newly created service
```

```
  PreCond:  -
  PostCond: -
  Perms:    -
  Throws:   IncorrectState
            IncorrectURL
            NoSuccess
  Notes:    - An implementation may only allow a single
              instance of a shared events service. In that
              case, all subsequently created instances MUST
              throw an 'IncorrectState' exception.

- DESTRUCTOR
  Purpose:  close an service that manages shared events
            within a publish-subscribe ring.
  Format:   DESTRUCTOR  (in  shared_events obj);
  Inputs:   obj:            the service to close
  InOuts:   -
  Outputs:  -
  PreCond:  -
  PostCond: no more events will be received from this service.
  Perms:    -
  Throws:   -
  Notes:    -

- publish
  Purpose:  publish a topic (update) within a pub-sub ring.
  Format:   publish (in  string    topic,
                     in  string    content);
  Inputs:   topic:       the topic to be updated
            content:     the content to be published under
                         this topic.
  InOuts:   -
  Outputs:  -
  PreCond:  -
  PostCond: -
  Perms:    -
  Throws:   BadParameter
            NoSuccess
  Notes:    -

- subscribe
  Purpose:  subscribe to receive updates about a topic within
            a pub-sub ring.
  Format:   subscribe (in  string        topic,
                       in  callback      cb);
  Inputs:   topic:  the topic of interest
            cb:     the callback to process updates for this
                    topic.
  InOuts:   -
```

```
  Outputs:  -
  PreCond:  -
  PostCond: -
  Perms:    -
  Throws:   BadParameter
            NoSuccess
  Notes:    -

- unsubscribe
  Purpose:  stop receiving updates about a topic within a
            pub-sub ring.
  Format:   unsubscribe (in  string      topic);
  Inputs:   topic:  the topic that is not interesting anymore.
  InOuts:   -
  Outputs:  -
  PreCond:  -
  PostCond: -
  Perms:    -
  Throws:   BadParameter
            NoSuccess
            NotImplemented
  Notes:    -
```

5.3 Shared properties

The `shared_properties` object in the `xosaga.sharing` package provides access to a distributed key-value storage. It was design to provide access to the transactional distributed key-value storage of Scalaris.

A `shared_properties` object is identified by a URL. When multiple XOSAGA application processes use a shared properties object with the same URL, they can see each others modifications.

5.3.1 Specification

```
package xosaga.sharing
{
  class shared_properties
  {
    CONSTRUCTOR (in  saga::url         bootstrap_info,
                 out shared_properties obj);

    DESTRUCTOR  (in  shared_properties obj);

    put         (in  string  key,
                 in  string  value);
```

```
    get          (in  string  key,
                  out string  value);

    remove       (in  string   key);

  }
}
```

5.3.2 Specification details

Class `shared_properties`

This class offers methods for shared management of properties.

```
- CONSTRUCTOR
  Purpose:  create an service to manage shared properties
            within a key-value store.
  Format:   CONSTRUCTOR (in  saga::url         bootstrap_info,
                         out shared_properties obj);
  Inputs:   bootstrap_info: the bootstrap information for the
                            service.
                            Example URL: 'transstore://host.com:12345',
                            which connects to a key-value store
                            at host.com, port 12345
  InOuts:   -
  Outputs:  shared_property:  the newly created service
  PreCond:  -
  PostCond: -
  Perms:    -
  Throws:   IncorrectState
            IncorrectURL
  Notes:    - An implementation may only allow a single instance
              of a shared properties service. In that case, all
              subsequently created instances MUST throw
              an 'IncorrectState' exception.

- DESTRUCTOR
  Purpose:  close an service that manages shared properties
            within a key-value store.
  Format:   DESTRUCTOR  (in  shared_properties obj);
  Inputs:   obj:            the service to close
  InOuts:   -
  Outputs:  -
  PreCond:  -
  PostCond: -
  Perms:    -
```

```
  Throws:   -
  Notes:    -

- put
  Purpose:  store a (new) value for this key.
  Format:   put (in  string  key,
                 in  string  value);
  Inputs:   key:       the key to store the value for
            value:     the value to store.
  InOuts:   -
  Outputs:  -
  PreCond:  -
  PostCond: -
  Perms:    -
  Throws:   BadParameter
            NoSuccess
  Notes:    - An implementation may throw a BadParameter
              exception if the value is a reserved string
              (e.g. THISKEYHASBEENDELETED).

- get
  Purpose:  lookup the value store under this key.
  Format:   get (in  string  key,
                 out string  value);
  Inputs:   key:   the key to lookup
  InOuts:   -
  Outputs:  value: the value store under this key.
  PreCond:  -
  PostCond: -
  Perms:    -
  Throws:   BadParameter
            DoesNotExist
            NoSuccess
  Notes:    - if there is no such key in the store
              a 'DoesNotExist' exception is thrown. An
              implementation may throw a 'DoesNotExist'
              exception if the returned value is a
              reserved string.

- remove
  Purpose:  delete this key and the value stored under it
  Format:   remove (in  string   key);
  Inputs:   key:  the key to delete.
  InOuts:   -
  Outputs:  -
  PreCond:  -
  PostCond: -
  Perms:    -
  Throws:   BadParameter
```

```
            NoSuccess
            NotImplemented
  Notes:    - An implementation may store a special string
              as the value of a deleted key
              (e.g. THISKEYHASBEENDELETED).
```

---

6 Virtual Nodes and XOSAGA

The goal of virtual nodes is to make service replication management transparent to the application, as described in D3.2.5 (*Design and Specification of a Virtual Node System*) [19]. By design, a service that also uses SAGA should therefore not be concerned with its own replication at all. One the other hand, some services may benefit from having some high-level control of their replication (e.g. selecting the consistency model to be maintained). Such controls are offered by the virtual nodes system. How much of this functionality to expose in an XOSAGA API is the subject of future work, to be performed by WP3.2. Based on their findings, a future XOSAGA release may add such functionality. Based on the state of the work at the time of writing, no such functionality has been identified so far.

7 Summary

In this document, we have presented the third draft specification of programming interfaces for XtreemOS, adding the so-far missing access to parts of XtreemOS-specific functionality. In particular, API's for transparently distributed services (from WP3.2), a naming scheme for XtreemFS volumes (from WP3.4), and an integrated API package for transparent data sharing services (from WP3.2 and WP3.4) have been presented. For transparent service replication (WP3.2), no direct API can be considered useful at the time of writing this document.

Upcoming implementations of the API will cover the functionality described in this document. Future API revisions will be based on practical experience with this API draft, and will possibly revise and improve the API as defined here.

References

[1] Werner Almesberger. TCP connection passing. In *Ottawa Linux Symposium*, July 2004.

[2] A. Anjomshoaa, F. Brisard, M. Drescher, D. Fellows, A. Ly, S. McGough, D. Pulsipher, and A. Savva. Job Submission Description Language

(JSDL) Specification v1.0. Grid Forum Document GFD.56, Open Grid Forum (OGF), 2005. http://www.ogf.org/sf/documents/GFD.56.pdf.

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

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

[5] Tom Goodale, Shantenu Jha, Hartmut Kaiser, Thilo Kielmann, Pascal Kleijer, Andre Merzky, John Shalf, and Christopher Smith. A Simple API for Grid Applications (SAGA). Grid Forum Document GFD.90, Open Grid Forum (OGF), 2007. Version 1.0 http://forge.ogf.org/short/saga-core-wg/saga-core-v1.

[6] D. Johnson, C. Perkins, and J. Arkko. Mobility Support in IPv6. RFC 3775, June 2004.

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

[8] MIPL: mobile ipv6 for linux. Available on the WWW, July 2006. http://www.mobile-ipv6.org/.

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

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

[11] T. Schütt, F. Schintke, and A. Reinefeld. Scalaris: Reliable Transactional P2P Key/Value Store - Web 2.0 Hosting with Erlang and Java. In *Proceedings of the 7th ACM SIGPLAN Erlang Workshop*, Victoria, BC, Canada, September 2008.

[12] Willem van Duijn. A versatile anycast framework for distributed servers. Master's thesis, Vrije Universiteit, Amsterdam, The Netherlands, February 2008. http://www.globule.org/publi/VAFDS_master2008.html.

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

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

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

[16] Evaluation Report and Revision of Application Requirements. Deliverable D4.2.3, XtreemOS Consortium, 2007.

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

[18] Design report for advanced XtreemFS and OSS features. Deliverable D3.4.3, XtreemOS Consortium, 2008.

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

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