Overview of XOSAGA Programming Interfaces

D3.1.10

Due date of deliverable: April 1st 2010
Actual submission date: May 10th 2010

Start date of project: June 1st 2006

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

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

Version 1.0 / Last edited by Thilo Kielmann / May 10th 2010

<table>
<thead>
<tr>
<th>Dissemination Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>PU Public</td>
</tr>
<tr>
<td>PP Restricted to other programme participants (including the Commission Services)</td>
</tr>
<tr>
<td>RE Restricted to a group specified by the consortium (including the Commission Services)</td>
</tr>
<tr>
<td>CO Confidential, only for members of the consortium (including the Commission Services)</td>
</tr>
</tbody>
</table>

Project co-funded by the European Commission within the Sixth Framework Programme
Revision history:

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Authors</th>
<th>Institution</th>
<th>Section affected, comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>16/03/10</td>
<td>Mathijs den Burger</td>
<td>VUA</td>
<td>initial draft</td>
</tr>
<tr>
<td>0.99</td>
<td>09/04/10</td>
<td>Thilo Kielmann</td>
<td>VUA</td>
<td>version for internal review</td>
</tr>
<tr>
<td>1.0</td>
<td>10/05/10</td>
<td>Thilo Kielmann</td>
<td>VUA</td>
<td>final version</td>
</tr>
</tbody>
</table>

Reviewers:
Michael Schöttner (UDUS), André Lage (INRIA)

Tasks related to this deliverable:

<table>
<thead>
<tr>
<th>Task No.</th>
<th>Task description</th>
<th>Partners involved†</th>
</tr>
</thead>
<tbody>
<tr>
<td>T3.1.1</td>
<td>Specification of XtreemOS API extensions to the set of POSIX specifications</td>
<td>VUA*, all partners except CDC</td>
</tr>
</tbody>
</table>

*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 an overview of XOSAGA, the programming interfaces to XtreemOS, its components and services. XOSAGA forms a coherent set of modular API packages, based on OGF’s Simple API for Grid Applications (SAGA) [2]. XOSAGA extends SAGA by packages for handling XtreemOS user certificates (XtreemOS contexts), job submission and resource management, the XtreemFS file system, the Scalaris publish-subscribe system, the Object Sharing System (OSS), and the Distributed Servers.

This document contains the programming-language independent specifications of the XOSAGA package API’s. XOSAGA has been implemented in C++, in Java, and in Python. The programming-language bindings (the concrete syntax and semantics) for these programming languages are described separately, along with the respective implementations.
Contents

Executive Summary

1 Introduction

2 XtreemOS context
   2.1 Example

3 Job Submission and Resource Management
   3.1 Specification
   3.2 Specification Details
      3.2.1 Class resource_description
      3.2.2 Class resource
      3.2.3 Class reservation
      3.2.4 Class resource_service
      3.2.5 Class application_description
      3.2.6 Class job_service
   3.3 Example

4 XtreemFS

5 Scalaris and OSS
   5.1 Shared buffers
      5.1.1 Specification
      5.1.2 Specification details
      5.1.3 Example
   5.2 Shared events
      5.2.1 Specification
      5.2.2 Specification details
      5.2.3 Example
   5.3 Shared properties
      5.3.1 Specification
      5.3.2 Specification details
      5.3.3 Example

6 Distributed Servers
   6.1 Specification
   6.2 Specification Details
   6.3 Example

7 Summary
1 Introduction

The API for the XtreemOS operating system has to meet multiple and conflicting requirements. First of all, it has to be congruent with POSIX API’s and their look-and-feel, in order to serve traditional Linux applications, as XtreemOS is based on Linux. Second, the XtreemOS API should also serve existing grid applications, thus following grid-related standards. And finally, XtreemOS-specific functionality needs to be exposed to new applications that wish to exploit XtreemOS to the fullest extent.

The resolution of these conflicting requirements lies in defining an API for XtreemOS that is based on the Simple API for Grid Applications (SAGA), a standard defined by the Open Grid Forum (OGF) [2]. SAGA has been accepted as a middleware and service-independent API for grid infrastructures, thus allows XtreemOS to serve grid applications that had been developed for other, middleware-based systems. Also, SAGA has been designed following the look-and-feel of POSIX API’s, also making Linux applications feel “at home” on XtreemOS. Finally, SAGA has a modular and extensible design, allowing to add new packages that give access to XtreemOS-specific functionality and services.

SAGA thus forms the core of the XtreemOS API. Together with the XtreemOS-specific extension packages we call the API XOSAGA. These packages provide interfaces for handling XtreemOS user certificates (XtreemOS contexts), job submission and resource management, the XtremFS file system, the Scalaris publish-subscribe system, the Object Sharing System (OSS), and the Distributed Servers.

This document contains the programming-language independent specifications of the XOSAGA package API’s. XOSAGA has been implemented in C++, in Java, and in Python. The programming-language bindings (the concrete syntax and semantics) for these programming languages are described separately, along with the respective implementations.

An integral part of the XtreemOS API is defined by the OGF recommendation document GFD.90 [2], which has been shaped and contributed by the XtreemOS team throughout the development of the XtreemOS software. We refrain from including its 324 pages in this document. Instead, we briefly summarize the SAGA core packages, and only present the XOSAGA extensions in full detail in the following sections. (The API documentation that is part of the three implementations also covers the core SAGA API, rendered in the respective programming languages, C++, Java, and Python.)

Figure 1 shows the classes and interfaces of the SAGA core API. At the top, the so-called “look and feel” packages are shown. These packages deal with all non-functional aspects and mandate a uniform look-and-feel for all the functional packages (shown in lower part), as well as for the XOSAGA extension packages as defined in the following sections. From the look-and-feel packages, it is im-
important to mention the context class that gets extended for handling XtreemOS user certificates. From the functional packages we highlight the job management package, that is extended by XOSAGA for providing access to the Application Execution Management system (AEM). Likewise, the name space and file management packages are extended to provide access to the XtreemFS file system. The following sections provide the detailed specifications of all XOSAGA extension packages. Structure, organization, and layout of these sections follow the SAGA standard document [2].

Figure 1: The SAGA classes and interfaces, according to [2].

2 XtreemOS context

In XtreemOS, VO management is based on XtreemOS-specific certificates. These certificates are issued and administered by VO management services, and used and interpreted both by other XtreemOS-specific services as well as the different flavours of the XtreemOS operating system. The latter is done via kernel modules that authenticate and authorize users via these XtreemOS certificates [7].

In the SAGA API [2], the saga::context class provides the functionality of a security information container. A saga::context object can be attached to a saga::session handle, and as such be available to all SAGA objects instantiated in that session. Multiple contexts can co-exist in one session, and it is up to the implementation to choose the correct context for a specific method call.

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

The XtreemOS API therefore uses saga::context objects to encapsulate XtreemOS certificates. The Type attribute of such an XtreemOS context has the value 'xtreemos'.

If a user has installed an XtreemOS certificate in his home directory, an XOSAGA implementation provides default values for the following attributes:

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

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

Applications can also set these attributes themselves to use another user certificate than the default one.

In addition, the implementation provides the following (read only) attributes for XtreemOS contexts, providing the relevant information from XtreemOS certificates [7]:

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

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

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

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

Figure 2 shows a Java program that creates a SAGA session and adds an XtreemOS context to it. When a local XtreemOS certificate is installed, various attributes of the certificate will be printed.

```
import org.ogf.saga.error.SagaException;
import org.ogf.saga.session.Session;
import org.ogf.saga.session.SessionFactory;
import org.ogf.saga.context.Context;
import org.ogf.saga.context.ContextFactory;
import eu.xtreemos.xosaga.context.XosContext;

public class XtreemOSContextExample {
    public static void main(String... args) {
        try {
            // add a deep copy of the context to the default session
            Session defaultSession = SessionFactory.createSession();
            Context c = ContextFactory.createContext("xtreemos");
            defaultSession.addContext(c);

            // get the initialized copy of the context from the session
            Context[] contexts = defaultSession.listContexts();
            c = contexts[0];

            // print some attributes of the context
            System.out.println("XtreemOS key file: "+
                                 c.getAttribute(Context.USERKEY));
            System.out.println("XtreemOS certificate: "+
                                 c.getAttribute(Context.USERCERT));
            System.out.println("- Global primary VO name: "+
                                 c.getAttribute(XosContext.GLOBAL_PRIMARY_VO_NAME));
            System.out.println("- Global primary role name: "+
                                 c.getAttribute(XosContext.GLOBAL_PRIMARY_ROLE_NAME));
            System.out.println("- Global primary group name: "+
                                 c.getAttribute(XosContext.GLOBAL_PRIMARY_GROUP_NAME));

            System.out.println("- Global secondary group names:");
            String attr = XosContext.GLOBAL_SECONDARY_GROUP_NAMES;
            for (String name : c.getVectorAttribute(attr)) {
                System.out.println(" - "+name);
            }
        }
        catch (SagaException e) {
            System.err.println("Exception: "+e.getMessage());
        }
    }
}
```

Figure 2: Example Java program that prints attributes of an XtreemOS context
3 Job Submission and Resource Management

XOSAGA applications can submit and monitor XtreemOS jobs via the existing SAGA package `saga.job`. However, XtreemOS also provides resource management and the feature to restart jobs. This requires an extension of the existing SAGA API, which is provided by the XOSAGA resource management extension package. It consists of eight classes, partially extending existing SAGA classes, partially implementing existing SAGA interfaces. The relationships between the new XOSAGA classes and the 'old' SAGA classes and interfaces is shown in Figure 3. We specify the XOSAGA classes in the following.

![Diagram of XOSAGA resource management](image_url)

Figure 3: The relations between the XOSAGA resource management package and the existing SAGA classes and interfaces.

3.1 Specification

```java
package xosaga.resource {

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

    CONSTRUCTOR (out resource_description obj)
    DESTRUCTOR (in resource_description obj)

    // Attributes:
    //
    // name: TotalCPUCount
    // desc: total number of cpus to be provided
    // mode: ReadWrite, optional
```
// type: Int
// value: '1'
// notes: - semantics as defined in JSDL
//         - available in JSDL, DRMAA

// name: TotalPhysicalMemory
// desc: Estimated amount of memory to be provided
// mode: ReadWrite, optional
// type: Float
// value: -
// notes: - unit is in MegaByte
//         - memory usage of the job is aggregated
//         across all processes of the job
//         - semantics as defined by JSDL
//         - available in JSDL

// name: CPUArchitecture
// desc: compatible processor for job submission
// mode: ReadWrite, optional
// type: Vector String
// value: -
// notes: - allowed values as specified in JSDL
//         - semantics as defined by JSDL
//         - available in JSDL

// name: OperatingSystemType
// desc: compatible operating system for job submission
// mode: ReadWrite, optional
// type: Vector String
// value: -
// notes: - allowed values as specified in JSDL
//         - semantics as defined by JSDL
//         - available in JSDL

// name: CandidateHosts
// desc: list of host names which are to be considered
//       by the resource manager as candidate targets
// mode: ReadWrite, optional
// type: Vector String
// value: -
// notes: - semantics as defined by JSDL
//         - available in JSDL

// name: Queue
// desc: name of a queue to place the job into
// mode: ReadWrite, optional
// type: String
// value: -
// notes: - While SAGA itself does not define the
// semantics of a "queue", many backend systems // can make use of this attribute. // - not supported by JSDL

class xosaga::resource : implements saga::object
    implements saga::async
    implements saga::attributes
    implements saga::permissions
    implements saga::monitorable
{
    // no CONSTRUCTOR
    DESTRUCTOR (in xosaga::resource obj);

    get_resource_description (out xosaga::resource_description rd);
}

enum state
{
    New       = 1,
    Running   = 2,
    Done      = 3,
    Canceled  = 4,
}

class reservation : implements saga::object
{
    // no CONSTRUCTOR
    DESTRUCTOR (in xosaga::reservation obj);

    get_state (out state state);

    get_resources (out array<resource> reserved);

    // Attributes:
    // 
    // name: ReservationID
    // desc: reservation identifier as returned by the
    // resource service
    // mode: Read, optional
    // type: String
    // value: -
    // notes: -
    // 
    // name: CreationTime
    // desc: time stamp of the reservation creation in
    // the resource manager
    // mode: Read, optional
    // type: Int
// value: -
// notes: - format: number of seconds since epoch
//
// name: Starttime
// desc: time stamp indicating when
// the reservation starts
// mode: Read
// type: Int
// value: -
// notes: - format: number of seconds since epoch
//
// name: ExpirationTime
// desc: time stamp indicating when
// the reservation ends
// mode: Read
// type: Int
// value: -
// notes: - format: number of seconds since epoch

class resource_service : implements saga::object
    implements saga::async
{
    CONSTRUCTOR (in session s,
        in saga::url rm = "",
        out resource_service obj);
    DESTRUCTOR (in resource_service obj);
    discover (in resource_description rd,
        out array<string> resource_ids);
    reserve (in resource_description rd,
        in int start_time,
        in int expiration_time,
        out reservation reserved);
    reserve (in array<string> resource_ids,
        in int start_time,
        in int expiration_time,
        out reservation reserved);
    cancel (in reservation res,
        in float timeout);
    list (out array<string> reservation_ids);
    getReservation (in string reservation_id,
        out reservation res);
get_resource (in string resource_id,
         out resource res);
}

class application_description : implements saga::object
    implements saga::attributes
{
    CONSTRUCTOR (out application_description obj);
    DESTRUCTOR (in application_description obj);

    // Attributes:
    //
    // name: Executable
    // desc: command to execute.
    // type: String
    // mode: ReadWrite
    // value: '
    // notes: - this is the only required attribute.
    //         - can be a full pathname, or a pathname relative
    //           to the 'WorkingDirectory' as evaluated on the
    //           execution host.
    //         - semantics as defined in JSDL
    //         - available in JSDL, DRMAA
    //
    // name: Arguments
    // desc: positional parameters for the command.
    // mode: ReadWrite, optional
    // type: Vector <string
    // value: -
    // notes: - semantics as specified by JSDL
    //         - available in JSDL, DRMAA
    //
    // name: SPMDVariation
    // desc: SPMD job type and startup mechanism
    // mode: ReadWrite, optional
    // type: String
    // value: -
    // notes: - as defined in the SPMD extension of JSDL
    // notes: - semantics as defined in JSDL
    //         - available in JSDL, SPMD extension
    //         - the SPMD JSDL extension defines the value to be
    //           an URI. For simplicity, SAGA allows the
    //           following strings, which map into the respective
    //           URIs: MPI, GridMPI, IntelMPI, LAM-MPI, MPICH1,
    //                  MPICH2, MPICH-GM, MPICH-MX, MVAPICH, MVAPICH2,
    //                  OpenMP, POE, PVM, None.
- the value ‘Empty’ (default) indicates that the application is not a SPMD application.
- as JSDL, SAGA allows other arbitrary values.
  The implementation must clearly document which values are supported.

name: NumberOfProcesses
desc: total number of processes to be started
mode: ReadWrite, optional
type: Int
value: ‘1’
notes: - semantics as defined in JSDL
    - available in JSDL, SPMD extension

name: ProcessesPerHost
desc: number of processes to be started per host
mode: ReadWrite, optional
type: Int
value: ‘1’
notes: - semantics as defined in JSDL
    - available in JSDL, SPMD extension

name: ThreadsPerProcess
desc: number of threads to start per process
mode: ReadWrite, optional
type: Int
value: ‘1’
notes: - semantics as defined in JSDL
    - available in JSDL, SPMD extension

name: Environment
desc: set of environment variables for the job
mode: ReadWrite, optional
type: Vector String
value: -
notes: - exported into the job environment
    - format: ‘key=value’
    - semantics as specified by JSDL
    - available in JSDL, DRMAA

name: WorkingDirectory
desc: working directory for the job
mode: ReadWrite, optional
type: String
value: ‘.’
notes: - semantics as specified by JSDL
    - available in JSDL, DRMAA

name: Interactive
// desc: run the job in interactive mode
// mode: ReadWrite, optional
// type: Bool
// value: 'False'
// notes: - this implies that stdio streams will stay
//         connected to the submitter after job submission,
//         and during job execution.
//         - if an implementation cannot handle interactive
//         jobs, and this attribute is present and 'True',
//         job creation MUST throw an 'IncorrectParameter'
//         error with a descriptive error message.
//         - not supported by JSDL, DRMAA

// name: Input
// desc: pathname of the standard input file
// mode: ReadWrite, optional
// type: String
// value: -
// notes: - semantics as specified by JSDL
//         - available in JSDL, DRMAA
//         - will not be used if 'Interactive' is 'True'

// name: Output
// desc: pathname of the standard output file
// mode: ReadWrite, optional
// type: String
// value: -
// notes: - semantics as specified by JSDL
//         - available in JSDL, DRMAA
//         - will not be used if 'Interactive' is 'True'

// name: Error
// desc: pathname of the standard error file
// mode: ReadWrite, optional
// type: String
// value: -
// notes: - semantics as specified by JSDL
//         - available in JSDL, DRMAA
//         - will not be used if 'Interactive' is 'True'

// name: FileTransfer
// desc: a list of file transfer directives
// mode: ReadWrite, optional
// type: Vector String
// value: -
// notes: - translates into jsdl:DataStaging
//         - used to specify pre- and post-staging
//         - semantics as specified in JSDL
//         - staging is part of the 'Running' state
- syntax similar to LSF (see earlier notes)
- available in JSDL, DRMAA

name: Cleanup
desc: defines if output files get removed after the job finishes
mode: ReadWrite, optional
type: String
value: 'Default'
notes: - can have the Values 'True', 'False', and 'Default'
- On 'False', output files MUST be kept after the job finishes
- On 'True', output files MUST be deleted after the job finishes
- On 'Default', the behaviour is defined by the implementation or the backend.
- translates into 'DeleteOnTermination' elements in JSDL

name: JobStartTime
desc: time at which a job should be scheduled
mode: ReadWrite, optional
type: Int
value: -
notes: - Could be viewed as a desired job start time, but that is up to the resource manager.
- format: number of seconds since epoch
- available in DRMAA
- not supported by JSDL

name: TotalCPUTime
desc: estimate total number of CPU seconds which the job will require
mode: ReadWrite, optional
type: Int
value: -
notes: - intended to provide hints to the scheduler.
- available in JSDL, DRMAA
- semantics as defined in JSDL

name: JobContact
desc: set of endpoints describing where to report job state transitions.
mode: ReadWrite, optional
type: Vector String
value: -
notes: - format: URI (e.g. fax:+123456789, sms:+123456789, mailto:joe@doe.net).
- available in DRMAA
- not supported by JSDL

name: CheckpointPeriodicity
desc: how frequently should the job be checkpointed, in seconds
type: Int
mode: ReadWrite, optional
notes: - a value of 0 means no periodic checkpointing
- default value is implementation dependant
  - proposed by D2.1.1

name: NumberOfKeptCheckpoints
desc: how many checkpoints should be kept for this job
type: Int
mode: ReadWrite, optional
value: '1'
notes: - proposed by D2.1.1

name: FinalStorage
desc: set of pathnames to use to store the checkpoint
type: Vector string
mode: ReadWrite, optional
value: -
notes: - if no path is given, a default path will be selected by the System Checkpointer, presumably on the local node
  - proposed by D2.1.1

name: CheckpointPolicy
desc: how the checkpoint is produced
type: Vector string
mode: ReadWrite, optional
value: -
notes: - if no policy is given, a default policy will be chosen
  - If more than one policy is given, the first policy available for the checkpoint service will be used
  - possible CheckpointPolicies include:
    Safe: the checkpoint file is completly written before the checkpoint call returns
    LocalFirst: the checkpoint file is written locally before the end of the system checkpoint and moved to its final destination later
    MemoryFirst: the checkpoint is saved in memory at the end of the system checkpoint and moved to its final destination later
class job_service : extends saga::job_service
{
    CONSTRUCTOR (in session s,
                 in url rm = ",",
                 out job_service obj)

    DESTRUCTOR (in job_service obj)

    create_job (in application_description ad,
                in resource_description rd,
                out job job);

    create_job (in application_description ad,
                in array<string> resource_ids,
                out job job);

    create_job (in application_description ad,
                in string reservation_id,
                out job job);
}

3.2 Specification Details
3.2.1 Class resource_description

The resource_description class is collecting those attributes from SAGA’s job_description class that are related to selecting suitable resources.
### 3.2.2 Class resource

The `resource` class is a container for the information identifying a compute resource. It has a single method for retrieving its resource description.

- **DESTRUCTOR**
  
  **Purpose:** destroy the object
  
  **Format:** `DESTRUCTOR (in resource_description obj);`
  
  **Inputs:** `obj:` the object to destroy
  
  **Outputs:** `-`
  
  **PreCond:** `-`
  
  **Postcond:** `-`
  
  **Perms:** `-`
  
  **Throws:** `-`

- **get_resource_description**
  
  **Purpose:** Retrieve the description of the discovered resource.
  
  **Format:** `get_resource_description (out resource_description rd);`
  
  **Inputs:** `-`
  
  **InOuts:** `-`
  
  **Outputs:** `rd:` a description of the resource
  
  **PreCond:** `-`
  
  **PostCond:** the returned resource description is a deep copy
  
  (no state is shared after method invocation)
  
  **Perms:** `Query`
  
  **Throws:** `NotImplemented`
  
  `DoesNotExist`
  
  `PermissionDenied`
  
  `AuthorizationFailed`
  
  `AuthenticationFailed`
  
  `Timeout`
  
  `NoSuccess`

  **Notes:** There may be cases when the resource description is not available, e.g. when the resource is one of many discovered resources and/or a description of the individual resource can not be constructed.
  
  In this case, a ‘DoesNotExist’ exception is thrown, with a descriptive error message.
3.2.3 Class reservation

The reservation class is a container for the information identifying a reservation. Like jobs, reservations have different states, shown in Figure 4. A newly constructed reservation can either be in the state New or Running. New denotes that the start time of the reservation has not yet been reached. Running denotes that the resource(s) reserved by the reservation are currently accessible, i.e., the time at the resource(s) lies between start time and expiration time. Once the time at the resource has reached the expiration time, the reservation’s state changes to Done. The state Canceled can only be reached from the state Running. A reservation can be canceled by invoking the cancel() method on the reservation object, or by some external party like the remote resource itself or a resource broker service.

![Diagram of reservation state model]

Figure 4: The XOSAGA reservation state model.

- DESTRUCTOR
  Format: DESTRUCTOR (in reservation obj);
  Purpose: destroy the object
  Format: DESTRUCTOR (in resource obj);
  Inputs: obj: the object to destroy
Outputs: -
PreCond: -
Postcond: -
Perms: -
Throws: -

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

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

3.2.4 Class resource_service

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

- CONSTRUCTOR
  Purpose: create the object
Format: CONSTRUCTOR (in session s,
in saga:url rm = ",
out resource_service obj);

Inputs:  s: session to associate with the object
         rm: contact point of the resource manager
In Outs: -
Out puts: obj: the newly created object
 PreCond: -
 PostCond: -
 Perms: -
 Throws: NotImplemented
         IncorrectURL
         PermissionDenied
         AuthorizationFailed
         AuthenticationFailed
         Timeout
         NoSuccess

Notes:  - ‘rm’ defaults to an empty URL – in that case, the
         implementation must perform a resource discovery,
         or fall back to a fixed value, or locate a valid
         resource manager in any other way. If that is not
         possible, a ‘NoSuccess’ exception MUST be
         thrown, and MUST indicate that a resource manager
         URL is needed. The expected behaviour MUST be
         documented (i.e. if a default is available).
         - if the resource manager identified by the rm URL
           cannot be contacted (e.g. does not exist), a
           ‘NoSuccess’ exception is thrown.

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

- discover
   Purpose: discover resources matching the resource description
   Format: discover (in resource_description rd,
             out array<string> resource_ids);
   Inputs: rd: description of resource to be discovered
   In Outs: -
Outputs: resource_ids: the identifiers of the discovered resources

PreCond: -
PostCond: - rd is deep copied (no state is shared after method invocation)

Perms: -

Throws: NotImplemented
BadParameter
PermissionDenied
AuthorizationFailed
AuthenticationFailed
Timeout
NoSuccess

reserve

Purpose: reserve the resources that match a resource description
Format: reserve (in resource_description rd,
            in int start_time,
            in int expiration_time,
            out reservation reserved);

Inputs: rd: description of the resource(s) to reserve
start_time: requested start time of the reservation, in number of seconds since the epoch
expiration_time: requested expiration time of the reservation, in number of seconds since the epoch

InOuts: -

Outputs: reservation: a reservation object representing the successful reservation

PreCond: -
PostCond: - rd is deep copied (no state is shared after method invocation)

Perms: -

Throws: NotImplemented
BadParameter
PermissionDenied
AuthorizationFailed
AuthenticationFailed
Timeout
NoSuccess

Notes: - if the resource description contains values that are outside of the allowed range, or cannot be parsed, or are otherwise invalid and not usable for creating a resource instance, a ‘BadParameter’ exception is thrown, which MUST indicate which attribute(s) caused this exception, and why.
- if the reservation fails because no matching resources are available in the requested time interval, a 'NoSuccess' exception MUST be thrown, which MUST indicate the failure.
- An implementation MAY use default values for start time and expiration time (like ‘`as soon as possible,’’ and ‘`15 minutes duration’’) and MAY deviate from the requested time interval. An implementation MUST document such behavior.

```
reserve
Purpose: reserve the resources identified by their resource ids
Format: reserve
        (in array<string> resource_ids,
        in int start_time,
        in int expiration_time,
        out reservation reserved);
Inputs: resource_ids: array of resource ids
        start_time: requested start of reservation,
                    in number of seconds since the epoch
        expiration_time: requested expiration of reservation,
                        in number of seconds since the epoch
InOuts: -
Outputs: reservation: a reservation object representing the successful reservation
PreCond: -
PostCond: - rd is deep copied (no state is shared after method invocation)
Perms: -
Throws: NotImplemented
        BadParameter
        PermissionDenied
        AuthorizationFailed
        AuthenticationFailed
        Timeout
        NoSuccess
Notes: - if any of the resource ids is invalid, a 'BadParameter' exception is thrown, which MUST indicate which id(s) caused this exception.
- if the reservation fails because some identified resources are unavailable in the requested time interval, a 'NoSuccess' exception MUST be thrown, which MUST indicate the failure. In this case, no resource will be reserved at all.
- An implementation MAY use default values for start time and expiration time (like ‘`as soon as possible,’’ and ‘`15 minutes duration’’) and MAY deviate from the requested time interval. An implementation MUST document such behavior.
```
- cancel
Purpose: cancel a reservation
Format: cancel (in reservation res,
in float timeout);
Inputs: res: the reservation to cancel
timeout: time to free resources
InOuts: -
Outputs: -
PreCond: - the reservation is in the state ‘New’ or ‘Running’.
PostCond: - the reservation is in ‘Canceled’ state.
Perms: -
Throws: NotImplemented
IncorrectState
Timeout
NoSuccess
Notes: - for resource deallocation semantics, see Section 2 of the SAGA specification.
- if cancel() fails to cancel the reservation immediately, and tries to continue to cancel the reservation in the background, the reservation state remains ‘Running’ until the cancel operation succeeded. The state then changes to ‘Canceled’.
- if the reservation is in the ‘Done’ state, the call has no effect, and, in particular, does NOT change the state to ‘Canceled’. This is to avoid race conditions.
- a ‘NoSuccess’ exception indicates that the backend was not able to initiate the cancelation of the reservation.
- for timeout semantics, see Section 2 of the SAGA specification.

- list
Purpose: Get a list of reservations that are currently known by the resource manager.
Format: list (out array<string> reservation_ids);
Inputs: -
InOuts: -
Outputs: reservation_ids: an array of reservation identifiers
PreCond: -
PostCond: -
Perms: Query on reservations identified by the returned ids
Throws: NotImplemented
PermissionDenied
AuthorizationFailed
AuthenticationFailed
Timeout
NoSuccess
Notes: - which reservations are viewable by the calling user
context, and how long a resource manager keeps reservation information, are both implementation dependent.

- a returned reservation id may translate into a reservation (via get_reservation()), which is not controllable by the requesting application (e.g. it could cause an ‘AuthorizationFailed’ exception).

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

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

3.2.5 Class application_description

The application_description class is collecting those attributes from SAGA's job_description class that are related to the application itself, augmented by the attributes for checkpointing from D2.1.1 [6].

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

- DESTRUCTOR
3.2.6 Class job_service

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

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

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

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

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

Inputs:  ad: description of application to submit
resource_ids: identification for a reservation
holding resources provided to the job

InOuts: -

Outputs: job: a job object representing the
submitted job instance

PreCond: - ad has an 'Executable' attribute.
PostCond: - job is in 'New' state
- ad is deep copied (no state is shared after method
invocation)
- 'Owner' of the job is the id of the context used
  for creating the job.

Perms: -

Throws: NotImplemented
BadParameter
PermissionDenied
AuthorizationFailed
AuthenticationFailed
Timeout
NoSuccess

Notes: - calling run() on the job will submit it to the
resource, and advance its state.
- if the application description does not have a
  valid 'Executable' attribute, a 'BadParameter'
  exception is thrown.
- if the application description contains values that
  are outside of the allowed range, or cannot be
  parsed, or are otherwise invalid and not usable for
  creating a job instance, a 'BadParameter' exception
  is thrown, which MUST indicate which attribute(s)
  caused this exception, and why.
- if the reservation_id is invalid, a 'BadParameter'
  exception is thrown.
3.3 Example

Figure 5 shows an example Java program that uses the SAGA job package to execute `/bin/hostname` on a single node. The URL `xos://` of the job service will use the local AEM configuration. The stdout and stderr output is put in the files `hostname.out` and `hostname.err`, respectively. The program registers two callback methods: one for the job’s state, and one for AEM-specific detailed state. The program waits until the job has been executed, after which the output files can be found in the XtreemFS home volume of the user.

```java
import org.ggf.saga.context.Context;
import org.ggf.saga.error.SagaException;
import org.ggf.saga.job.*;
import org.ggf.saga.monitoring.*;
import org.ggf.saga.url.*;

public class JobExample implements Callback {
    public static void main(String[] args) {
        try {
            URL serverURL = URLFactory.createURL("xos://");
            JobService js = JobFactory.createJobService(serverURL);
            JobDescription jd = JobFactory.createJobDescription();
            jd.setAttribute(JobDescription.EXECUTABLE, "/bin/hostname");
            jd.setAttribute(JobDescription.TOTALCPUCOUNT, "1");
            jd.setAttribute(JobDescription.OUTPUT, "hostname.out");
            jd.setAttribute(JobDescription.ERROR, "hostname.err");
            Job job = js.createJob(jd);
            job.addTarget(JOB_STATE, new JobExample());
            job.addTarget(JOB_STATEDETAIL, new JobExample());
            job.run();
            job.waitFor();
        } catch (SagaException e) {
            System.err.println("Exception: "+e.getMessage());
        }
    }

    public boolean cb(Monitorable m, Metric metric, Context c) {
        try {
            String value = metric.getAttribute(Metric.VALUE);
            String name = metric.getAttribute(Metric.NAME);
            System.out.println("Callback called for metric "+name+
                                ", value = "+value);
        } catch (SagaException e) {
            System.err.println("Error: "+e.getMessage());
        }
        return true; // keep the callback
    }
}
```

Figure 5: Java SAGA program that executes `/bin/hostname` on a single node
4 XtreemFS

XtreemFS provides access to remote files via a local proxy file system using FUSE and Linux VFS. XtreemFS file systems are organized in named volumes that are registered in an XtreemFS Directory Service.

An XtreemFS volume can be mounted into the client machine’s local file system via the XtreemFS client application. After mounting has succeeded, files can be accessed via the POSIX file API to local files.

An XOSAGA application can access the XtreemFS file system via the existing SAGA packages saga.namespace and saga.file; no further API extensions are needed. However, the XtreemFS access layer exposes the local file system mounting to the application, which introduces a small but additional management overhead. XOSAGA relieves users from this overhead by mounting required XtreemFS volumes automatically.

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 XtreemFS Directory Service at which the volume is registered.
- **port** is the port number the XtreemFS Directory Service listens to.
- **path** is the path of the file or directory in the volume.

All URL parts are mandatory, except for the port number. Without a port number, the default port 32638 is used. An example XtreemFS URL is:

```
xtreemfs://vol42@host.example.com:12345/dir/file.txt
```

This URL refers to the file ’/dir/file.txt’ on an XtreemFS volume named ’vol42’. This volume is registered at the XtreemFS Directory Service at ’host.example.com’ that listens to port 12345.

5 Scalaris and OSS

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) [8]. 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 [4] 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 name specified by the user. Different application processes can identify the same shared buffer using its name.

5.1.1 Specification

```java
package xosaga.sharing {

    class shared_buffer_service {
        CONSTRUCTOR (in saga::url bootstrap, 
                     in saga::url local = "", 
                     out shared_buffer_service obj);
        DESTRUCTOR (in shared_buffer_service obj);

        create_strict_domain (in string name, 
                              out strict_domain d);

        create_transactional_domain (in string name, 
                                     out transactional_domain d);
    }

    class consistency_domain {
        get_name (out string name);
    }

```
create_buffer (in string name,
in int size,
out shared_buffer buf);

memory_map (in string path,
in int offset,
in int length,
out shared_buffer buf);

get_buffer (in string name,
in float timeout = -1.0,
out shared_buffer buf);

}  

class strict_domain extends consistency_domain
{
  // no additional methods
}

class transactional_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 shared_buffer extends saga::buffer
  // from buffer saga::object
  // from buffer saga::error_handler
{
  DESTRUCTOR ()
  get_name (out string name);
}

class transaction_id
{
  // no public methods, immutable object
}
5.1.2 Specification details

Class 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 types of consistency.
  Format: CONSTRUCTOR (in saga::url bootstrap, in saga::url local, out shared_buffer_service obj);
  Inputs: bootstrap: the bootstrap information for the service, e.g., an address of a peer or server to contact. Example URL: ‘oss://host.com:12345’, which connects to another OSS at host.com, port 12345
           local: the local address to bind to. If empty, the default local address is used.
  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_strict_domain

35
Purpose: create a domain for buffers with strict consistency
Format: create_strict_domain (in string name,
          out strict_domain d);
Inputs: name: the name of the strict consistency domain. It
        uniquely identifies this domain on all nodes
        that participate in the same shared buffer
        service.
InOuts: -
Outputs: d: the strict consistency domain
         with the given name.
PreCond: -
PostCond: -
Perms: -
Throws: -
Notes: -

- create_transactional_domain
Purpose: create a domain for buffers with transactional
        consistency
Format: create_transactional_domain
        (in string name,
         out transactional_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: -

Class consistency_domain

The xosaga::consistency_domain class offers generic management oper-
ations 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
          NoSuccess
  Notes: - if size < 0, a ‘BadParameter’ exception
          MUST be thrown

- memory_map
  Purpose: map a local 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 string path,
             in int offset,
             in int length,
             out shared_buffer buf);
  Inputs: path: the path of the local 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). This
method blocks until the buffer is available or a
timeout occurs.
Format: get_buffer (in string buf_name,
in float timeout,
out shared_buffer buf);
Inputs: name: the name of a shared buffer
timeout: the amount of seconds to wait until
the buffer is available.
InOuts: -
Outputs: buf: the existing shared buffer
PreCond: -
PostCond: -
Perms: -
Throws: Timeout
NoSuccess
Notes:  - if no buffer with the given name exists in
this consistency domain after <timeout> seconds,
a 'Timeout' exception MUST be thrown.

Class strict_domain

The xosaga::strict_domain class creates shared buffers with a strict consistency model. It provides no additional methods.

Class transactional_domain

The xosaga::transactional_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

39
NoSuccess

Notes: - if the given transaction_id is not known, a 'DoesNotExist' exception MUST be thrown

---

**Class shared buffer**

This class 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 released automatically when the buffer was created on this node
  - **Perms:** -
  - **Throws:** -
  - **Notes:** -

- **get_name**
  - **Purpose:** return the name of this shared buffer
  - **Format:** get_name (out string name);
  - **Inputs:** -
  - **InOuts:** -
  - **Outputs:** name: the name of this shared buffer
  - **PreCond:** -
  - **PostCond:** -
  - **Perms:** -
  - **Throws:** -
  - **Notes:** -

- **set_size**
  - **Notes:** - overrides set_size() in saga::buffer. This method MUST always throw a 'NotImplementedException'

- **set_data**
  - **Notes:** - overrides set_data() in saga::buffer. This method MUST always throw a 'NotImplementedException'

- **close**
  - **PostCond:** - other nodes can still access the contents of this shared buffer
**Class transaction_id**

This class is an immutable object without any public methods.

### 5.1.3 Example

The example C++ program in Figure [6] demonstrates the use of a shared buffer with transactional consistency. Without arguments, the program acts as a server that creates a shared buffer called 'example_buffer' in the transactional domain 'example_domain'. The server writes 'hello' into the buffer and then waits until it contains 'world'. With arguments, the program runs as a client. The client first looks up the buffer 'example_buffer' using a timeout of 5 seconds. When the buffer is found, the client waits until it contains 'hello'. It then prints the size of the buffer and writes 'world' into it. When the server notices the new value it terminates.
#include <iostream>
#include <string.h>
#include <xosaga/xosaga.hpp>

using namespace xosaga::sharing;
using namespace std;

int main(int argc, const char* argv[]) {
    string bootstrap_url("*");
    string local_url("oss://127.0.0.1");
    if (argc > 1) { // I'm the client
        bootstrap_url = local_url;
        local_url = "oss://127.0.0.2";
    }

    shared_buffer_service sbs(bootstrap_url, local_url);
    transactional_domain dom = sbs.create_transactional_domain("example_domain");
    string buf_name("example_buffer");
    saga::size_t buf_size = 20;
    if (!bootstrap_url.empty()) { // I'm the server
        shared_buffer buf = dom.create_buffer(buf_name, buf_size);
        char* data = (char*)buf.get_data();
        strcpy(data, "hello");
        while (data[0] != 'w') { // wait until shared buffer contains 'world'
            dom.commit(transaction_id tid = dom.begin());
            usleep(100000);
            tid = dom.begin();
        }
        cout << "Content of " << buf_name << ": " << data << endl;
        dom.commit(tid);
    } else { // I'm the client
        shared_buffer buf = dom.get_buffer(buf_name, 5);
        char* data = (char*)buf.get_data();
        transaction_id tid = dom.begin();
        while (data[0] != 'h') { // wait until shared buffer contains 'hello'
            dom.commit(tid);
            usleep(100000);
            tid = dom.begin();
        }
        cout << "Size of " << buf_name << ": " << buf.get_size() << endl;
        strcpy(data, "world");
        dom.commit(tid);
    }
}

usleep(200000); // give OSS time to sync
return 0;

Figure 6: Example C++ XOSAGA program using shared buffers
5.2 Shared events

The `shared_events` object in the `xosaga.sharing` package provides access to a publish-subscribe system. It is 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

```cpp
package xosaga.sharing {

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);
}

interface callback {

    cb (in shared_events se, in string topic, in string content);
}
}
```
5.2.2 Specification details

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: -

**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: -
5.2.3 Example

The Java program shown in Figure 7 demonstrates the use of the shared events package. The program connects to a local publish/subscribe daemon and subscribes to the topic "example topic". It then publishes the value "Hello world!" in this topic and waits for the updated value to arrive in the callback method. Finally, it unsubscribes from "example topic" again.

```java
import org.ofog.saga.error.SagaException;
import org.ofog.saga.url.+
import eu.xtreemos.xosaga.sharing.+

public class SharedEventsExample implements Callback {
    public static void main(String[] args) {
        new SharedEventsExample().run();
    }

    public void run() {
        String topic = "example topic";
        try {
            URL sagaURL = URLFactory.createURL("boot$localhost");
            SharedEvents s = SharingFactory.createSharedEvents(sagaURL);
            System.out.println("Subscribing to " + topic);
            s.subscribe(topic, this);
            System.out.println("Publishing value in " + topic);
            s.publish(topic, "Hello world!");
            System.out.println("Waiting for update...");
            synchronized(this) {
                try {
                    wait();
                } catch (InterruptedException ignored) {
                }
            }
            System.out.println("Unsubscribing from " + topic);
            s.unsubscribe(topic);
        } catch (SagaException e) {
            System.err.println("Exception: " + e.getMessage());
        }
    }

    public void cb(SharedEvents se, String topic, String content) {
        System.out.println("Received update of " + topic + ": " + content);
        synchronized (this) {
            notifyAll();
        }
    }
}
```

Figure 7: Example Java XOSAGA program using shared events
5.3 Shared properties

The shared_properties object in the xosaga.sharing package provides access to a distributed key-value storage. It is designed to provide access to the transactional distributed key-value storage of Scalaris, but can be applied to any key-value storage.

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

```java
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: obj: the newly created service
PreCond: -
PostCond: -
Perms: -
Throws: IncorrectState
IncorrectURL
NoSuccess
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 stored under this key.
Format: get (in string key,
out string value);

Inputs: key: the key to look up

Outputs: value: the value stored under this key.

PreCond: -
PostCond: -

Perms: -

Throws: BadParameter
        DoesNotExist
        NoSuccess

Notes: - if there is no such key in the store, a 'DoesNotExist' exception MUST be 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
        NoImplemented

Notes: - An implementation may store a special string as the value of a deleted key (for example, 'THISKEYHASBEENDELETED').
5.3.3 Example

The use of shared properties is demonstrated in the Java program shown in Figure 8. The program connects to a local publish/subscribe daemon, creates the shared key-value pair (‘example key’, ‘hello world!’) and removes it again.

```java
import org.ofgsa.error.SagaException;
import org.ofgsa.url.*;
import eu.xtreemos.xosaga.sharing.*;

public class SharedPropertiesExample {
    public static void main(String[] args) {
        try {
            URL u = URLFactory.createURL("boot@localhost");
            SharedProperties sp = SharingFactory.createSharedProperties(u);
            String key = "example key";
            System.out.println("Creating shared key-value pair");
            sp.put(key, "hello world!");
            System.out.println("Shared value of " + key + ": " + sp.get(key));
            System.out.println("Removing key-value pair");
            sp.remove(key);
        } catch (SagaException e) {
            System.err.println("Exception: " + e.getMessage());
        }
    }
}
```

Figure 8: Example Java XOSAGA program using shared properties
6 Distributed Servers

The distributed servers, as implemented by WP3.2, provide a TCP stream interface to their clients. They achieve high availability and fault tolerance through forming a redundant group of server machines that can hand-over client connections to each other, without the clients noticing.

Distributed Servers provide location transparent networked services [5]. Clients connect to a single distributed server address for a service and may be moved transparently among multiple locations. Mobile IPv6 (MIPv6) route optimization [3] does the heavy lifting: all IPv6 connections from a client are atomically changed directly to each location, avoiding triangular routing. The distributed server address is simply an IPv6 [1] address. In the terminology of Distributed servers, a client first connects to a contact node. A client may then be transparently handed off to different servers for load-balancing or for client-specific processing. The server endpoint of all of the client’s connections are transferred along with the handoff operation. Distributed servers are described in Deliverables D3.2.2, D3.2.6 and D3.2.11 [9] [10] [11].

6.1 Specification

```java
package xosaga.ds {

class ds_service {

    CONSTRUCTOR (in saga::session s,
                  in string name,
                  in handoff_policy policy = NULL,
                  out ds_service obj);

    DESTRUCTOR (in ds_service obj);

    serve (in float timeout = -1.0,
           out saga::stream stream);

    get_client (in saga::stream stream,
                out ds_client client);

    get_all_clients (out array<ds_client> clients);

    get_all_streams (out array<saga::stream> streams);

    get_all_targets (out array<saga::url> targets);

    handoff (in ds_client client,
             in float timeout = -1.0,
```
6.2 Specification Details

Class ds_service

The ds_service accepts new incoming connections as SAGA streams, allows to hand off all streams connected with the same client to another Distributed Servers
node, and accepts such a handoff operation.

- CONSTRUCTOR

  Purpose: create a service to access a running Distributed Servers daemon

  Format: CONSTRUCTOR (in saga::session s,
in string name,
in handoff_policy policy,
out ds_service obj);

  Inputs: s: session to be used for object creation
  name: a name recognized by the local Distributed Server daemon that maps to a local address
  policy: the handoff policy to use

  InOuts: -
  Outputs: obj: the newly created service
  PreCond: -
  PostCond: obj can now serve incoming client connections.
  Perms: -
  Throws: IncorrectState
          IncorrectURL
          NotImplemented
          BadParameter
          NoSuccess

  Notes: - if there is no local Distributed Servers daemon running, a ‘NoSuccess’ exception MUST be thrown.
  - if the local daemon does not know the given name, a ‘DoesNotExist’ exception MUST be thrown.
  - the local daemon MAY only allow one instance of a ds_service per name per machine. In that case, all subsequently created instances with a name that has already been used MUST throw an ‘AlreadyExists’ exception.
  - the handoff policy can be NULL; in that case, the handoff() method without a target URL parameter will always throw an ‘IncorrectState’ exception.

- DESTRUCTOR

  Purpose: destructor of the ds_service object

  Format: DESTRUCTOR (in ds_service obj)

  Inputs: obj: the ds_service object to destroy

  InOuts: -
  Outputs: -
  PreCond: -
  PostCond: - the service is closed
  Perms: -
  Throws: -
- serve

Purpose: Wait for an incoming client connection

Format: `serve (in float timeout,
         out saga::stream stream);`

Inputs: timeout: number of seconds to wait

InOuts: -

Outputs: stream: new connected stream object

PreCond: -

PostCond: - all postconditions of saga::stream_service.serve() apply.

- the session of the returned stream is that of the
  ds_service object.

- the associated ds_client object also contains the
  new stream.

Perms: - all permissions of saga::stream_service.serve() apply.

Threws: NotImplemented
         BadParameter
         PermissionDenied
         AuthorizationFailed
         AuthenticationFailed
         IncorrectState
         Timeout
         NoSuccess

Notes: - all notes from saga::stream_service.serve() apply.

- get_client

Purpose: returns the client associated with a stream

Format: `get_client (in saga::stream stream,
                 out ds_client client);`

Inputs: stream: a connected stream

InOuts: -

Outputs: client: the ds_client object associated with
         the given stream

PreCond: -

PostCond: -

Perms: -

Threws: DoesNotExist
         BadParameterException
         IncorrectState
         NotImplemented

Notes: - if no client is associated with the given stream, a
       'DoesNotExist' exception MUST be thrown.

- if the given stream is not connected, a
  'BadParameter' exception MUST be thrown.
- get_all_clients
  Purpose: returns all clients currently handled by this ds_service object.
  Format: get_all_clients (out array<ds_client> clients);
  Inputs: -
  InOuts: -
  Outputs: clients: all clients currently handled by this ds_service object.
  PreCond: -
  PostCond: -
  Perms: -
  Throws: IncorrectState
          NotImplemented
  Notes: -

- get_all_streams
  Purpose: returns all streams of all clients.
  Format: get_all_streams (out array<saga::stream> streams);
  Inputs: -
  InOuts: -
  Outputs: streams: all streams of all clients
  PreCond: -
  PostCond: -
  Perms: -
  Throws: NotImplemented
  Notes: - the array is a shallow copy; streams served later are not reflected in the array.

- get_all_targets
  Purpose: get the URLs of all the handoff targets.
  Format: get_all_targets (out array<saga::url> targets);
  Inputs: -
  InOuts: -
  Outputs: targets: all possible handoff targets.
  PreCond: -
  PostCond: -
  Perms: -
  Throws: NotImplemented
          PermissionDenied
          AuthorizationFailed
          AuthenticationFailed
          IncorrectState
          NoSuccess
  Notes: - the number of possible handoff targets CAN be zero.

- handoff
  Purpose: hand off all streams of a client to another node determined by the handoff policy of this ds_service.
Format:    handoff (in ds_client client,
in float  timeout,
out saga:url target);

Inputs:    client:  the client to hand off
timeout:   number of seconds to wait

InOuts:    -

Outputs:   target:  the URL of the node selected for the handoff

PreCond:   - if an application-specific 'message' object has been set via ds_client.set_message(), it will be passed to the other server along with the handoff.

PostCond:  - the client object does not contain any streams anymore.

Perms:     -

Throws:    NotImplemented
           PermissionDenied
           AuthorizationFailed
           AuthenticationFailed
           BadParameter
           DoesNotExist
           Timeout
           IncorrectState
           NoSuccess

Notes:     - the handoff target is determined by calling the get_target() method of the handoff policy of this ds_service
- any exception thrown by the handoff policy MUST be forwarded
- if the given client is not handled by this ds_service, a 'DoesNotExist' exception MUST be thrown.
- if no handoff policy was given in the CONSTRUCTOR, an 'IncorrectState' exception MUST be thrown.
- after a successful handoff, all subsequent method calls on the client MUST throw an 'IncorrectState' exception (except for the DESTRUCTOR and close()).
- which streams are contained in the given client object is irrelevant; ALL streams from this client known by the local daemon are handed off.

- handoff

Purpose:   hand off all streams of a client to a specific node

Format:    handoff (in saga:url target,
in ds_client client,
in float  timeout);

Inputs:    target:  the target server to which the client has to be handed off
client:    the client that is to be handed off
timeout:   number of seconds to wait
InOuts: -
Outputs: -
PreCond: - if an application-specific ‘message’ object has been set via ds_client.set_message(), it will be passed to the target server along with the handoff.
PostCond: - the client object does not contain any streams anymore.
Perms: -
Throws: NotImplemented PermissionDenied AuthorizationFailed AuthenticationFailed BadParameter DoesNotExist Timeout IncorrectStateException NoSuccess
Notes: - if the target does not exist, a ‘DoesNotExist’ exception MUST be thrown
- if the given client is not handled anymore by this ds_service, a ‘DoesNotExist’ exception MUST be thrown.
- after a successful handoff, all subsequent method calls on the client MUST throw an ‘IncorrectStateException’ exception (except for the DESTRUCTOR and close()).
- which streams are contained in the given client object is irrelevant; ALL streams from this client known by the local daemon are handed off.

- receive_handoff
  Purpose: receive all streams of a client that are handed off by another node.
  Format: receive_handoff (in float timeout, out ds_client client);
  Inputs: timeout: number of second to wait
  InOuts: -
  Outputs: client: the client object that has been handed off
  PreCond: -
  PostCond: -
  Perms: -
  Throws: NotImplemented IncorrectStateException PermissionDenied AuthorizationFailed AuthenticationFailed Timeout NoSuccess
  Notes: - if the contact node could not be contacted,
an ‘IncorrectState’ exception MUST be thrown.
- the returned client MUST contain the
  application-specific ‘message’ object that was
  set by the node that handed off the client.

- close
  Purpose: Closes all streams of all clients handled by the
  local daemon and cleans up the daemons state.
  Format: close (in bool binding_reset);
  Inputs: binding_reset: whether to clear all bindings of all
           clients handled by the current node
           or not
  InOuts: -
  Outputs: -
  PreCond: -
  PostCond: -
  Perms: -
  Throws: NotImplementedException
          PermissionDenied
          AuthorizationFailed
          AuthenticationFailed
          Timeout
  Notes: - all subsequent method calls on this object MUST
         throw an ’IncorrectState’ exception, except for the
         CONSTRUCTOR, DESTRUCTOR and close().
         - if binding_reset is true, all clients will connect
         to the contact node when they start a new connec-
         tion.

Class ds_client

A ds_client object contains all streams that are connected with a particular
client. It can also contain an application-specific message (a SAGA buffer) that
is passed to the target of a handoff operation, or has been received from another
node during a handoff operation.

  There are two ways to retrieve a ds_client object:

  1. via ds_service.get_client(), using one of its associated streams re-
     turned by ds_service.serve()

  2. via ds_service.receive_handoff()

  In the second case, the ds_client may contain a message that was set by the
  node that handed off the client.
A ds_client object is a shallow copy of a part of the state of the local Distributed Servers daemon. In particular, the object will not contain any new streams that arrived after it was constructed. Hence, a ds_client object simply acts as the identifier of a remote node. Each call to ds_service.get_client() may return a new object with independent state. The message contained in a ds_client is also local to that particular instance.

- DESTRUCTOR
  Purpose: destroys the object
  Format: DESTRUCTOR (in ds_client obj);
  Inputs: obj: the object to destroy
  InOuts: -
  Outputs: -
  PreCond: -
  PostCond: - the client is closed
  Perms: -
  Throws: -
  Notes: - if the client was not closed before, the destructor performs a close() on the instance, and all notes to close() apply.

- get_url
  Purpose: returns the url that identifies the client
  Format: get_url (out saga::url obj_url);
  Inputs: -
  InOuts: -
  Outputs: obj_url: url of the client
  PreCond: -
  PostCond: -
  Perms: -
  Throws: NotImplemented
           IncorrectState
           NoSuccess
  Notes: - the URL MUST consist of the scheme 'ipv6://'
         followed by the IPv6 address of the client.

- get_streams
  Purpose: returns all streams connected with this client
  Format: get_streams(out array<saga::stream> streams);
  Inputs: -
  InOuts: -
  Outputs: streams: list of streams connected with this client
  PreCond: -
  PostCond: -
  Perms: -
  Throws: NotImplemented
           IncorrectState
NoSuccess
PermissionDenied

Notes: - the returned array is a shallow copy; any subsequent streams connected with this client that are served later will not be included in the array.

- set_message

Purpose: sets application specific data that will be sent along with a handoff

Format: set_message (in saga::buffer msg);

Inputs: msg: buffer containing application-specific data, or NULL.

InOuts: -

Outputs: -

PreCond: -

PostCond: -

Perms: -

Throws: NotImplemented
IncorrectState

Notes: - any message set previously will be overwritten
- using NULL as a message effectively removes it

- get_message

Purpose: returns the application-specific data of this client

Format: get_message (out saga::buffer msg);

Inputs: -

InOuts: -

Outputs: msg: the application-data associated with this client

PreCond: -

PostCond: -

Perms: -

Throws: NotImplemented
IncorrectState

Notes: - the data can have been set by another node that handed off this client.
- if no data has been set, NULL MUST be returned.

---

**Interface handoff_policy**

A handoff_policy chooses one target node from a set of possible targets. An implementation of such a policy must be provided to a ds_service object, which will use it for all handoff operations. Handoff policies can be very application-specific.
- get_target
  Purpose: returns the URL of the node a client should be handed off to.
  Format: get_target (in ds_client client,
               in array-url options,
               out saga::url target)

  Inputs: client: the client object to select a target for
          options: the possible targets to select
  InOuts: -
  Outputs: target: the selected target
  PreCond: -
  PostCond: -
  Perms: -
  Throws: DoesNotExist
          NoSuccess
  Notes: - if there are no possible targets, a 'DoesNotExist' exception MUST be thrown.

6.3 Example

Figure 9 shows an example Java SAGA program that uses Distributed Servers. Without arguments, it runs a ‘frontend’ process that accepts incoming streams (i.e. TCP sockets). Each stream is handed off to the first available other node (determined by the ‘PickFirstPolicy’ object given to the constructor of the DsService object). The frontend counts the incoming connections, and sends a message along with the handoff containing the latest count. With arguments, the program runs a ‘backend’ process that accepts clients that are handed off. The backend then closes all streams of each client.
```java
import java.util.List;
import org.ofg.saga.buffer.*/;
import org.ofg.saga.error.*/;
import org.ofg.saga.stream.Stream;
import org.ofg.saga.url.URL;
import eu.xtreemos.xosaga.d*s.*/;

class Example {
    public static void main(String args[]) {
        try {
            HandoffPolicy p = new PickFirstPolicy();
            DsService service = DsFactory.createDsService("/tmp/dsdaemon", p);
            if (args.length == 0) { // run frontend
                for (long i = 1; i++ ) {
                    Stream stream = service.serve();
                    DsClient client = service.getClient(stream);
                    String data = "client " + i;
                    Buffer dataBuf = BufferFactory.createBuffer(data.getBytes());
                    client.sendMessage(dataBuf);
                    URL dst = service.handoff(client);
                    System.out.println("Handed off " + data + " to " + dst);
                }
            } else { // run backend
                while (true) {
                    DsClient client = service.receiveHandoff();
                    Buffer b = client.getMessage();
                    System.out.println("Accepted " + new String(b.getData()));
                    for (Stream s: client.getStreams()) s.close();
                }
            }
        } catch (SagaException e) {
            System.err.println("Fatal error: " + e.getMessage());
        }
    }
}

private static class PickFirstPolicy implements HandoffPolicy {
    public URL getTarget(DsClient client, List<URL> options) throws DoesNotExistException, NoSuccessException {
        if (options.isEmpty()) {
            throw new DoesNotExistException("No options to choose from");
        } else {
            return options.get(0);
        }
    }
}
```

Figure 9: Example Java XOSAGA program using Distributed Servers
7 Summary

This document has presented an overview of XOSAGA, the programming interfaces to XtreemOS, its components and services. The API for the XtreemOS operating system has to meet multiple and conflicting requirements. First of all, it has to be congruent with POSIX API’s in order to serve traditional Linux applications. Second, the XtreemOS API should also serve existing grid applications, and finally, XtreemOS-specific functionality needs to be exposed to applications.

XOSAGA addresses these conflicting requirements by being based on OGF’s Simp-le API for Grid Applications (SAGA). Together with the XtreemOS-specific extension packages we call the API XOSAGA. These packages provide interfaces for handling XtreemOS user certificates (XtreemOS contexts), job submission and resource management, the XtreemFS file system, the Scalaris publish-subscribe system, the Object Sharing System (OSS), and the Distributed Servers.

This document contains the programming-language independent specifications of the XOSAGA package API’s. XOSAGA has been implemented in C++, in Java, and in Python. The programming-language bindings (the concrete syntax and semantics) for these programming languages are described separately, along with the respective implementations. An integral part of the XtreemOS API is defined by the OGF recommendation document GFD.90 [2]. In here, we only present the XOSAGA extension packages themselves. (The API documentation that is part of the three implementations also covers the core SAGA API, rendered in the respective programming languages, C++, Java, and Python.)

To summarize, this document provides a comprehensive description of the XOSAGA extension packages that provide access to XtreemOS-specific modules and services. Together with the SAGA core specification, and most prominently with the language-specific documentation of the three XOSAGA implementations, it allows application programmers to access the XtreemOS operating system.

References


