

CONTRACT NUMBER 508830

**DEISA**  
**DISTRIBUTED EUROPEAN INFRASTRUCTURE FOR  
SUPERCOMPUTING APPLICATIONS**

**European Community Sixth Framework Programme**  
**RESEARCH INFRASTRUCTURES**  
Integrated Infrastructure Initiative

Final Design for DESHL v2.0

Deliverable ID: DEISA-JRA7-3.7  
Due date: October 31st 2005  
Actual delivery date: November 25, 2005  
Lead contractor for this deliverable: EPCC, UK

Project start date: May 1<sup>st</sup>, 2004  
Duration: 4 years

<b>Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)</b>		
<b>Dissemination Level</b>		
<b>PU</b>	Public	X
<b>PP</b>	Restricted to other programme participants (including the Commission Services)	
<b>RE</b>	Restricted to a group specified by the consortium (including the Commission Services)	
<b>CO</b>	Confidential, only for members of the consortium (including the Commission Services)	

## Table of Content

Table of Content.....	1
1. Introduction.....	2
1.1 Executive Summary.....	2
1.2 References and Applicable Documents.....	2
1.3 Document Amendment Procedure.....	3
1.4 List of Acronyms and Abbreviations.....	3
2. Background.....	4
3. Design Constraints and Assumptions.....	4
4. DESHL v2.0 System Overview.....	5
5. Command Line Tool.....	7
5.1 Component Interfaces.....	9
5.1.1 desh1 Shell Script.....	9
5.1.2 File copy.....	9
5.1.3 Exists.....	10
5.1.4 isDir.....	10
5.1.5 isFile.....	10
5.1.6 list.....	10
5.1.7 makeDir.....	10
5.1.8 move.....	11
5.1.9 remove.....	11
5.1.10 submit.....	11
5.1.11 status.....	14
5.1.12 terminate.....	14
5.1.13 sites.....	14
5.1.14 fetch.....	14
5.1.15 jobs.....	14
5.2 Component packages and modules.....	15
5.3 Component Design.....	15
6. Client Library.....	15
6.1 Data Staging API.....	15
6.1.1 DESHLNSDirFactory.....	16
6.1.2 DESHLNSDirImpl.....	16
6.1.3 DESHLNSDir.....	16
6.1.4 Component packages and modules.....	16
6.2 Job Management API.....	17
6.2.1 DESHLClientFactory class.....	17
6.2.2 DESHLJobDefinitionImpl class.....	18
6.2.3 DESHLJobServiceImpl class.....	19
6.2.4 DESHLJobImpl class.....	19
6.2.5 Component packages and modules.....	20
7. Grid Access Library.....	20
8. Requirements Satisfaction.....	23
8.1 Data Staging.....	23
8.2 Job Management.....	24

## 1. Introduction

### 1.1 Executive Summary

The DESHL (**DEISA Services for the Heterogeneous management Layer**) has been developed by the DEISA Joint Research Activity JRA7. It will provide standards-based access for users and their applications to manage jobs and transfer files in the DEISA heterogeneous supercomputing infrastructure.

This document, 'Final Design for DESHL v2.0', is deliverable DEISA-JRA7-3.7 from Tasks T3.5 'Design of DESHL layer release 2.0' and T3.6 'Implementation of DESHL release 2.0' in Work Package 3 of the DEISA JRA7 activity [1]. This document describes the final design of release 2 of the DESHL. It is a publicly available document but its principal audience is the JRA7 development team.

In this document, section 2 indicates where background information on the functional scope for DESHL v2.0 can be found, with section 3 outlining the design constraints and assumptions. Section 4 gives an overview of DESHL v2.0 with sections 5, 6 and 7 describing respectively the Command Line Tool, Client Library and the Grid Access Library. These are the components that make up DESHL v2.0.

### 1.2 References and Applicable Documents

Where indicated, diagrams use UML notation [10].

- [1] DEISA Annex I – “Description of Work”.
- [2] “JRA7 Quality Plan v2.0”, DEISA JRA7 Report, Deliverable ID DEISA-JRA7-1.2, May 3<sup>rd</sup> 2005.
- [3] “Functional scope for DESHL v2.0”, DEISA JRA7 Report, Deliverable ID DEISA-JRA7-3.6, June 28<sup>th</sup> 2005.
- [4] “Final design for HSM v1.0”, DEISA JRA7 Report, Deliverable ID DEISA-JRA7-3.4, v1.2, July 5<sup>th</sup> 2005.
- [5] UNICORE, <http://unicore.sourceforge.net>
- [6] ARCON Client Library,  
[http://sourceforge.net/project/showfiles.php?group\\_id=102081&package\\_id=127938](http://sourceforge.net/project/showfiles.php?group_id=102081&package_id=127938)
- [7] UniGrids, <http://www.unigrids.org/>
- [8] GridFTP, [http://www.globus.org/grid\\_software/data/gridftp.php](http://www.globus.org/grid_software/data/gridftp.php)
- [9] Open Group specification for Batch Environment Services,  
[http://www.opengroup.org/onlinepubs/009695399/utilities/xcu\\_chap03.html#tag\\_03](http://www.opengroup.org/onlinepubs/009695399/utilities/xcu_chap03.html#tag_03)
- [10] GGF SAGA Research Group, <https://forge.gridforum.org/projects/saga-rg/>
- [11] GGF JSDL Working Group, <https://forge.gridforum.org/projects/jsdl-wg/>
- [12] “Functional scope for HSM v1.0”, DEISA JRA7 Report, Deliverable ID DEISA-JRA7-3.1 v1.0, October 31<sup>st</sup> 2004

- [13] "Integration of GridFTP in Unicore",  
[http://www.summit.unicore.org/2005/Simone\\_Lanzarini.pdf](http://www.summit.unicore.org/2005/Simone_Lanzarini.pdf)

### **1.3 Document Amendment Procedure**

The document procedure is covered in the Quality Plan [2] (Section 6.6).

### **1.4 List of Acronyms and Abbreviations**

<b>API</b>	Application Programming Interface
<b>CLI</b>	Command Line Interface
<b>CLT</b>	Command Line Tool
<b>DEISA</b>	Distributed European Infrastructure for Supercomputing Applications
<b>DESHL</b>	DEISA Services for the Heterogeneous management Layer
<b>DRM</b>	Distributed Resource Manager
<b>ftp or FTP</b>	File Transfer Protocol
<b>GB</b>	Gigabyte
<b>GGF</b>	Global Grid Forum
<b>GUI</b>	Graphical User Interface
<b>HPC</b>	High Performance Computing
<b>HSM</b>	Heterogeneous Service Management layer (superseded by the term DESHL)
<b>JRA</b>	DEISA Joint Research Activity
<b>JRA7</b>	Seventh Joint Research Activity
<b>JSDL</b>	Job Submission Description Language
<b>OGSA</b>	Open Grid Services Architecture
<b>RM</b>	Resource Manager
<b>SAGA</b>	Simple API for Grid Applications
<b>SOAP</b>	Originally Simple Object Access Protocol, now XML-based protocol for exchanging information
<b>UI</b>	User Interface
<b>UNICORE</b>	Uniform Interface to Computing Resources
<b>UniGrids</b>	Uniform Interface to Grid Services
<b>URI</b>	Uniform Resource Identifier
<b>WS</b>	Web Service: any discoverable, self-describing service within a network which communicates at least via XML, independently from any operating system.
<b>WSRF</b>	Web Services Resource Framework

## 2. Background

The functional scope for DESHL v2.0 can be found in the 'Functional scope for DESHL v2.0' document [3].

## 3. Design Constraints and Assumptions

DESHL v1 [4] successfully implements a solution for job submission and management, consisting of a custom client communicating with Web services based around the WSRF standard; these Web services in turn interact with UNICORE [5] via the UNICORE ARCON client library [6] on the server. The UniGrids project [7] is currently developing web services based job management for UNICORE, therefore rather than duplicate effort, the focus for DESHL 2.0 has changed to providing a SAGA-based interfaces in the DESHL.

SAGA is an acronym for Simple API for Grid Applications. The SAGA Research Group in the Global Grid Forum believe that scientific application developers want to use grids (such as that provided by the DEISA heterogeneous infrastructure) but do not have the time to investigate all the available Grid technologies and APIs. The Research Group are therefore developing a simple API for the fairly simple common operations such developers need to perform. The provision of a SAGA-based interface in the DESHL will help further insulate users and their applications from changes to the underlying DEISA middleware.

As explained in [3], the key objective for DESHL v2 is data staging, that is, the movement of files to and from the DEISA global file system. Movement of potentially very large files (> 1GB) over SOAP transport is likely to be slow and inefficient, and more efficient mechanisms such as GridFTP [7] are currently widely used for this instead. Although UNICORE file transfer via UPL is also likely to be slow and inefficient, UNICORE can be configured to use GridFTP [13] to perform file transfers for improved performance. In addition, UniGrids is developing a high performance UNICORE file transfer service.

In DEISA JRA7 there is no wish (nor sufficient resource available) to build yet another high performance file transfer tool. Instead the strategy adopted is to produce a deliverable compatible with the current DEISA infrastructure, but that provides the easiest route towards integration with either a GridFTP based UNICORE configuration or a UniGrids developed file transfer service. The design has therefore taken a layered approach, such that the WS-based functionality provided by UniGrids may be integrated when available at a later stage with minimum disruption to users and applications.

For these reasons the architecture of DESHL v2 is different to that previously employed in DESHL v1. As explained in section 4. of this document, DESHL v2 consists of a layered client that will initially use the ARCON client library to achieve data transfer using UNICORE. If the DEISA UNICORE configuration is updated to work with GridFTP, or a UniGrids file transfer service becomes available in DEISA, then the DESHL will be able to exploit these to achieve better performance for file transfers. With a UniGrids file transfer service this will require a reworking of the lower libraries of the DESHL whilst with a GridFTP configured UNICORE it is expected that no modifications to the DESHL will be necessary.

In DESHL v2, since the data transfer is not WSRF based it made sense to port the job management functionality originally provided in DESHL v1 over to this new architecture. This means the overall DESHL architecture is simplified and provides coherency between job management and data transfer. In addition it lowers the overhead for the installation of the DESHL. With regard to the original OGSA-based objectives of JRA7, now that the direction of UniGrids has become clearer with regard to OGSA, combined with the expectation that the OGSA-compliant version of UNICORE, UNICORE/GS, being developed by UniGrids, will be deployed within DEISA then this shift in architecture allows JRA7 and the DESHL to exploit the efforts of this much larger project in this area.

This means that DESHL v2 and DESHL v3 will not be service-based. With the likely deployment of UNICORE/GS in DEISA, DESHL v4 will be able to take advantage of this much more readily with a reduced impact on users and their applications.

On the other hand, if UNICORE/GS is not deployed then the existing use of OGSA standards such as JSDL means that the most appropriate technology for DESHL may be identified and deployed with minimum disruption to existing DESHL users.

#### 4. DESHL v2.0 System Overview

DESHL v2.0 provides standards-based access for users and their applications to manage jobs and transfer files in the DEISA heterogeneous supercomputing infrastructure. Figure 1 indicates how a user or their application employs the DESHL v2.0 to access the DEISA heterogeneous supercomputing infrastructure.

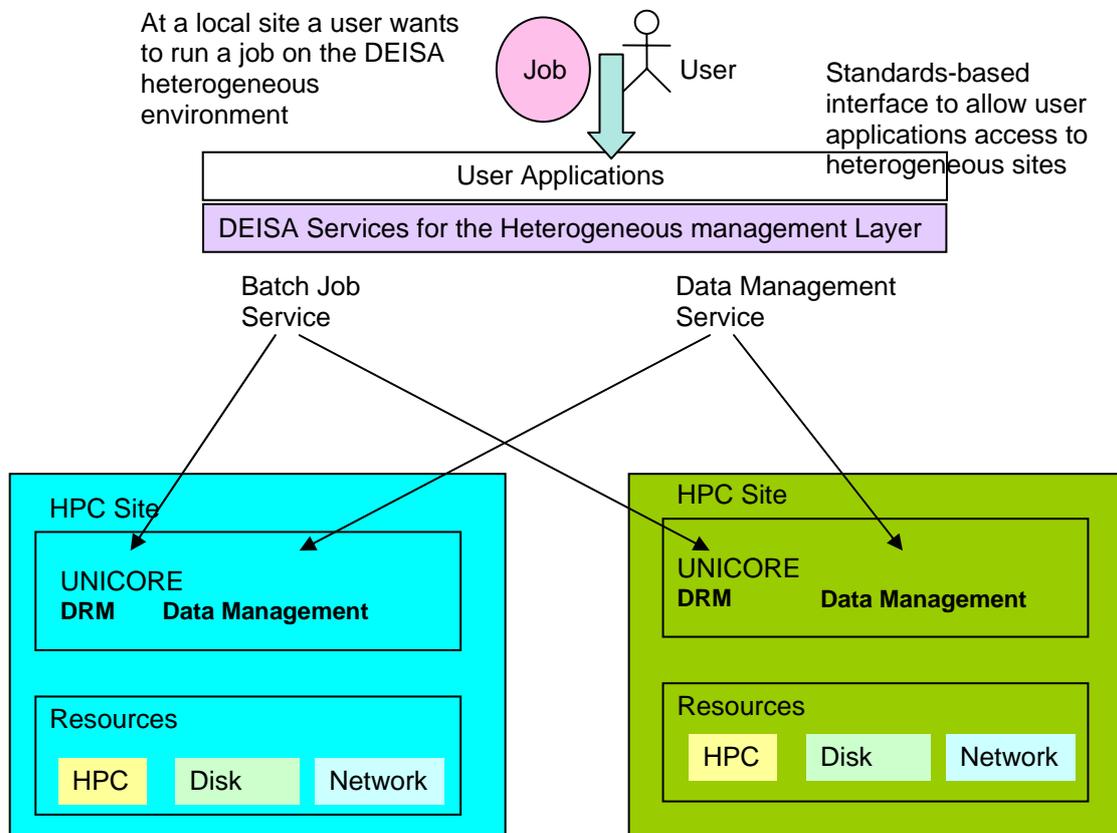


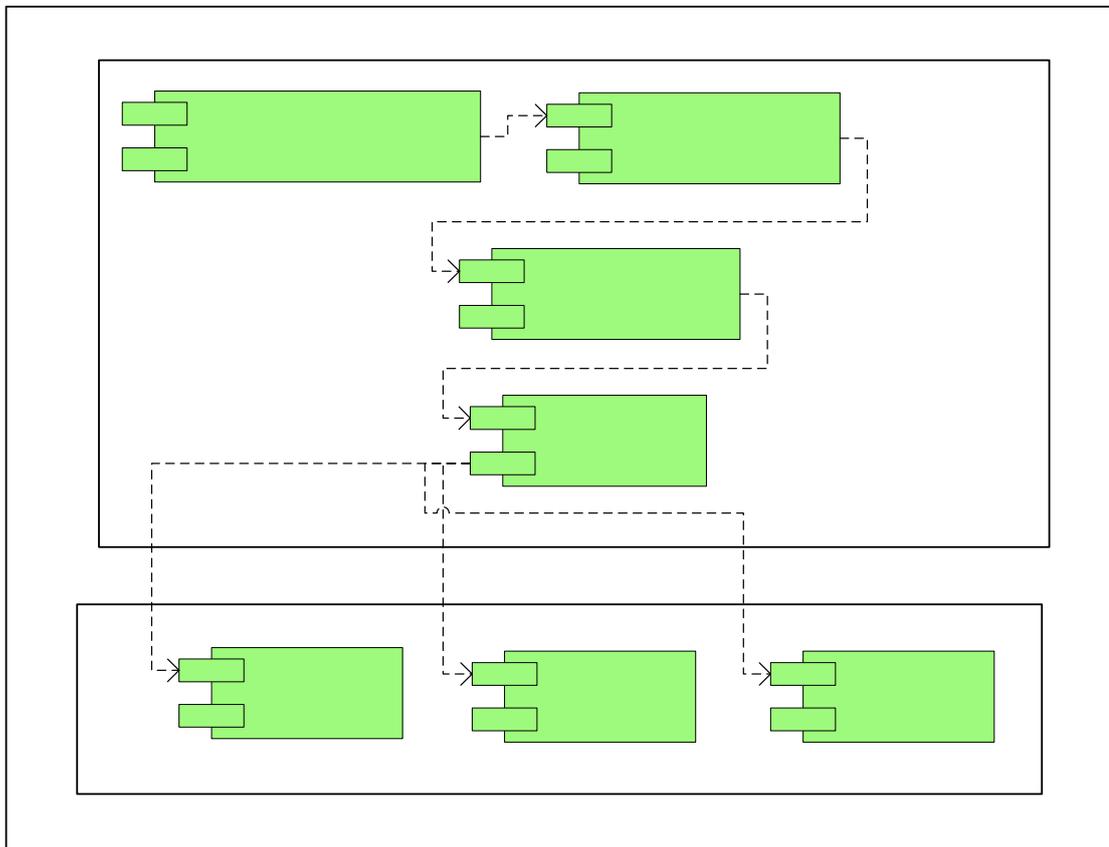
Figure 1: Using the DESHL to access the DEISA heterogeneous supercomputing infrastructure

DESHL v2.0 comprises:

- command line tool (CLT) based on the Open Group Batch Environment Services specification [9];
- client library exposing an API (Application Programming Interface) based on the SAGA [10] standard currently being developed by a GGF (Global Grid Forum) Research Group and
- Grid Access Library for interacting with a UNICORE Grid.

The DEISA heterogeneous supercomputing infrastructure is organised as a UNICORE Grid. UNICORE deals with the heterogeneity of the underlying supercomputing distributed resource managers and operating systems.

As shown in Figure 2, the DESHL Command Line Tool interacts directly with the DESHL client library (the SAGA-based API), which in turn utilises the Grid Access library to access a UNICORE Grid. The GGF JSDL (Job Submission Description Language) [11] standard is used to pass batch job specifications between the client library and the Grid Access Library and then on to the underlying UNICORE Grid.



**Figure 2: A UML diagram showing the DESHL components.**

The DESHL v2 command line tool will provide the following simple job management and file transfer capabilities.

- determine the DEISA sites to which a user can submit a batch job to
- submit a batch job to a DEISA site
- terminate a batch job at a DEISA site
- view the status of a batch job on a DEISA site
- upload a file to a DEISA site
- download a file from a DEISA site

- delete a file from a DEISA site
- determine if a file exists on a DEISA site
- list the contents of a directory on a DEISA site
- rename a file on a DEISA site
- copy/move a file between DEISA sites

The DESHL Client Library contains those SAGA operations necessary to support the file transfer and job management capabilities provided in the DESHL command line tool.

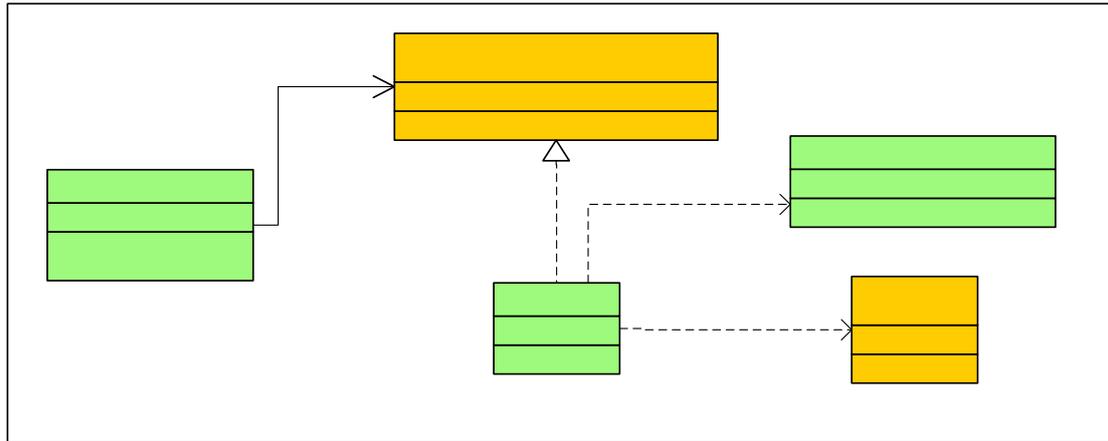
The Grid Access library (also known as Roctopus) presents a generalised object-oriented model for interacting with a UNICORE Grid, wrapping the low-level UNICORE ARCON client library targeted for UNICORE 4.x. The Grid Access library provides a general interface that can have multiple implementations - one for the current version of UNICORE deployed in DEISA and another for the OGSA version of UNICORE that is being developed by the UniGrids project. Future implementations of Roctopus will use web service client tooling or the UniGrids equivalent of the ARCON library.

The DESHL client is installed on a user's workstation or PC, and consists of the Command Line Tool, the Client Library, the Grid Access Library and the ARCON client. (The ARCON client library is included as part of the DESHL release bundle.) In v2, the DESHL client can only interact with a UNICORE grid that uses the UNICORE NJS v4.6.

The DESHL uses x.509 certificates for authentication and authorisation. Provided the user has appropriate certification, the DESHL allows a user to access the DEISA infrastructure regardless of whether or not they have a direct connection to a DEISA supercomputer and the DEISA global file systems. Subject to appropriate certification, access to multiple sites within the DEISA infrastructure may be configured such that access to the different sites is seamless from within the DESHL client.

## 5. Command Line Tool

The CLT (Command Line Tool) component offers a command line based user interface to the DESHL, and provides functionality for data staging and job management. The make up of this component is shown in more detail in Figure 3. This uses the copy command's implementation class `DESHLCopy` as an example of the architecture for the CLT. There is an implementation of `DESHLCommand` for all the CLT commands. There is a further example for the Job Management command 'submit' shown in Figure 4 later in section 5.1.10.



**Figure 3: UML Class diagram of the CLT Component.**

A wrapper script exists around the `DESHLCommandLine` class called `deshl`. This wrapper script provides the bridge from the user's command line shell to the implementation of these classes. This script sets up the environment for the `DESHLCommandLine` class and passes on to it the arguments `deshl` was called with.

The `DESHLCommandLine` main method takes commands of the form:

```
command <options>
```

The `command` maps to implementations of the `DESHLCommand` interface. For example, a supported command is `DESHLCopy`. This can process `copy` commands:

```
copy source target <arguments>
```

A configuration mechanism is provided to map command names to implementation classes with inbuilt defaults, so in the above example `DESHLCopy` is configured to process `copy` commands. The `DESHLCopy` class uses the Client Library interfaces `DESHLNSDirFactory` and `DESHLNSDir` as part of the data staging operations.

Note that `DESHLNSDir` is an implementation of the `NSDir` interface. As such, file attributes such as `read/write/executable` are defined as not being available through this interface. SAGA defines a specific `File` interface to allow access to individual file properties, but this has not been implemented in this release.

The full list of supported commands and the corresponding implementation classes are shown in the following table:

Command	Implementation Class
Copy	<code>org.deisa.jra7.deshl.ui.clt.command.DESHLCopy</code>
exists	<code>org.deisa.jra7.deshl.ui.clt.command.DESHLExists</code>
isDir	<code>org.deisa.jra7.deshl.ui.clt.command.DESHLIisDir</code>
isFile	<code>org.deisa.jra7.deshl.ui.clt.command.DESHLIisFile</code>
list	<code>org.deisa.jra7.deshl.ui.clt.command.DESHLList</code>
makeDir	<code>org.deisa.jra7.deshl.ui.clt.command.DESHLMakeDir</code>
move	<code>org.deisa.jra7.deshl.ui.clt.command.DESHLMove</code>
remove	<code>org.deisa.jra7.deshl.ui.clt.command.DESHLRemove</code>

submit	org.deisa.jra7.deshl.ui.clt.command.DESHLSubmit
status	org.deisa.jra7.deshl.ui.clt.command.DESHLStatus
terminate	org.deisa.jra7.deshl.ui.clt.command.DESHLTerminate
sites	org.deisa.jra7.deshl.ui.clt.command.DESHLSites
fetch	org.deisa.jra7.deshl.ui.clt.command.DESHLFetch
jobs	org.deisa.jra7.deshl.ui.clt.command.DESHLJobs

**Table 1: Supported CLT commands.**

## 5.1 Component Interfaces

The external interface this component presents is through the command line. This is provided by a shell script called `deshl` and command processing classes that implement `DESHLCommand`.

### 5.1.1 `deshl` Shell Script

The `deshl` shell script provides the entry command to the DESHL CLT. The arguments supplied to `deshl` are one of the commands that are configured to map to an implementation of `DESHLCommand`. These commands are discussed in the following sections. An example using the `exists` command would be:

```
$deshl exists ssl://gateway:port/TEST_EPCC_NJS_A/home/File.txt
```

Similarly, help documentation and a description of any optional arguments can be displayed by supplying a `help`, `-h`, option. For example:

```
$deshl exists -h
```

Please note that generally the commands follow the UNIX philosophy of reporting on failure rather than on success, unless such output is specifically required as part of the operation of the function.

### 5.1.2 `File copy`

File or directory copy is provided by the `DESHLCopy` class. This command takes a source file or directory, a target file or directory and optional arguments. This command is also used to import data to or export data from the DESHL file system.

```
deshl copy source target <arguments>
```

Optional arguments:

0: do not overwrite an existing target

1: overwrite existing target

2: non-recursive, do not copy directories

3: recursive, directory copy allowed

If no optional arguments are specified, the default behaviour is non-recursive, no-overwrite

The command gives no output on success, but prints an exception message if any errors are encountered.

This command aims to follow the SAGA specification for data staging [10].

### 5.1.3 *Exists*

Functionality to determine if a remote file or directory exists is provided by the `DESHLExists` class. It takes a single argument, the name of the remote file or directory whose existence is to be verified.

```
deshl exists remote_filename
```

The command prints out an appropriate message on successfully identifying if the path does exist or not, or an exception message if any errors were encountered.

This command aims to follow the SAGA specification for data staging [10].

### 5.1.4 *isDir*

Functionality to determine if a remote path is a directory is provided by the `DESHLIsDir` class. It takes a single argument, the remote path.

```
deshl isDir path
```

The command prints out an appropriate message on successfully identifying if the path is a directory or not, or an exception message if any errors were encountered.

This command aims to follow the SAGA specification for data staging [10].

### 5.1.5 *isFile*

Functionality to determine if a remote path is a file is provided by the `DESHLIsFile` class. It takes a single argument, the remote path.

```
deshl isFile path
```

The command prints out an appropriate message on successfully identifying if the path is a directory or an exception message if any errors were encountered.

This command aims to follow the SAGA specification for data staging [10].

### 5.1.6 *list*

Functionality to list the contents of a remote directory is provided by the `DESHListDir` class. It takes a single argument, the full remote directory name.

```
deshl list directory_name
```

The command prints a directory listing on success, and prints an exception message if any errors were encountered.

This command aims to follow the SAGA specification for data staging [10].

### 5.1.7 *makeDir*

Functionality to create a remote directory is provided by the `DESHLMakeDir` class. It takes the full remote path plus optional arguments.

```
deshl makeDir directory_name <arguments>
```

Optional arguments:

0: fail if parent directory does not exist (default)

1: create parent directories if not existing

The command prints an exception message if any errors are encountered.

This command aims to follow the SAGA specification for data staging [10].

#### 5.1.8 *move*

Functionality to rename a remote file or directory is provided by the `DESHLMove` class. It takes the name of the file or directory to be renamed, the new name plus optional arguments.

```
deshl move source target <arguments>
```

Optional arguments:

0: do not overwrite existing target (default)

1: overwrite allowed

The command prints a directory listing on success, and prints an exception message if any errors are encountered.

This command aims to follow the SAGA specification for data staging [10].

#### 5.1.9 *remove*

Functionality to delete a remote file or directory is provided by the `DESHLRemove` class. It takes the full remote path plus optional arguments.

```
deshl remove path <arguments>
```

Optional arguments:

0: do not delete directories (default)

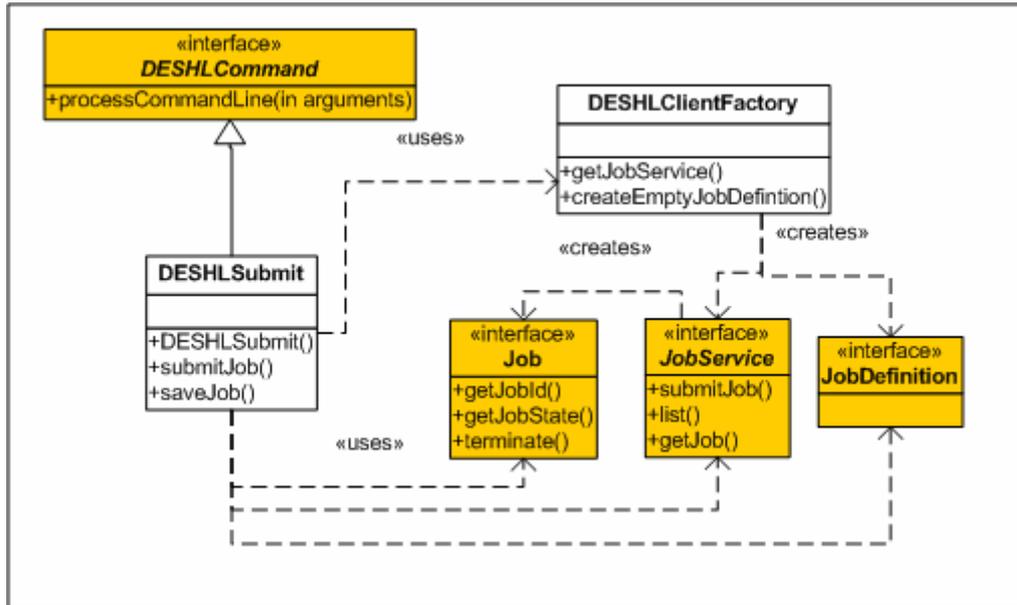
1: allow deletion of directories

The command prints an exception message if any errors are encountered.

This command aims to follow the SAGA specification for data staging [10].

#### 5.1.10 *submit*

Job submission is provided by the `DESHLSubmit` class. This is shown in Figure 4. The Job Management API is discussed in more detail below in Section 6.2.



**Figure 4: UML diagram for submit.**

By default it processes a command called `submit`. This command takes a number of options followed by the batch job script file and any arguments to pass to that file:

```
deshl submit <options> job_script <arguments>
```

The job script can also contain directives. This command aims to follow the `qsub` Open Group specification for job submission [9].

From that specification the following options are supported, as shown in Table 2:

Short Option	Long Option	Name	Description
C	directive	prefix	The prefix used for directives in the job script file
h	help	help	Used to obtain a summary of the available options and command syntax
N	name	job name	The user defined name for the job
q	destination	destination	The site the job should run on
v	variables	environment variables	A single name=value environment variable

**Table 2: Job submission options.**

Job submission also supports directives in the job script file. A directive is a special comment line in that file. It must occur within the heading comment block, where the heading comment block must start from the first line of the file. The only supported comment designator is the hash or pound symbol, '#'. This must be the first character to be a valid comment line. For the comment line to be a directive the comment symbol must be immediately followed by the directive prefix symbol, by default the dollar symbol, '\$', but can be specified on the command line.

Directives are used to allow the user to set values for the SAGA `JobDefinition` [10] attributes in Table 2. The value for a directive is everything following the equals, '=', up to the end of the line, and then trimmed for whitespace. It is up to the user to supply suitable values for the targeted resource manager. The values shown below are examples.

<b>SAGA JobDefinition attribute</b>	<b>Description</b>
SAGA_JobCmd	The job executable path. The only required attribute. By default this is set to the <code>job_script</code> , but the user can override this by setting this directive.
	<code>#\$ SAGA_JobCmd = /my/job.exe</code>
SAGA_JobArgs	An argument to the job. Usually set from arguments on the command line.
	<code>#\$ SAGA_JobArgs = arg</code> <code>#\$ SAGA_JobArgs = another arg</code>
SAGA_JobEnv	An environment variable.
	<code>#\$ SAGA_JobEnv = var1=val1</code> <code>#\$ SAGA_JobEnv = var2=val2</code>
SAGA_JobName	A name for the job.
	<code>#\$ SAGA_JobName = my job</code>
SAGA_FileTransfer	A file transfer local to the destination site, by default all stage ins are to the USPACE, all stage outs HOME. A stage in is designated by a <code>&gt;</code> , a stage out by a <code>&lt;</code> . File values must be a valid URI.
	<code>#\$ SAGA_FileTransfer = file:///dat#HOME&gt;file:///dat#USPACE</code> <code>#\$ SAGA_FileTransfer = file:///out#USPACE&gt;file:///out#HOME</code>
SAGA_HostList	The destination host for the job. This would be the locator for a Unicore NJS the user has access to.
	<code>#\$ SAGA_JobHostList = ssl://gateway.host:4433/SomeNJS</code>

**Table 3: SAGA JobDefinition attribute directives.**

A directive must occupy a single line, with only one directive allowed per line; the directive examples shown in Table 3 are wrapped for space reasons.

The file conventions used are the same as for the data staging commands. Note that only local file transfers to the site are allowed when submitting a job. This is to allow files that must be in the USPACE when the job runs to be staged in and to allow files to be staged out of the USPACE when the job completes. For other file transfers use the appropriate data staging command.

A successful submission results in a message containing the job identifier on standard output and a file saved with the same name as the job identifier. This file stores the necessary information to contact the job when used with DESHL job management commands (such as status).

If the job submission is unsuccessful then an error message will be returned on standard error (no output on standard output). Further information may be available from logs.

### 5.1.11 *status*

Job status is provided by the `DESHLStatus` class. By default it processes a command called `status`. This command takes a number of options followed by the job identifier file created when `submit` was called earlier:

```
deshl status <options> job_identifier_file
```

The output from `status` is a keyword indicating the state of the job at the target site. This command aims to follow the `qstat` Open Group specification for job status [9].

### 5.1.12 *terminate*

Job termination is provided by the `DESHLTerminate` class. By default it processes a command called `terminate`. This command takes a number of options followed by the job identifier file created when `submit` was called:

```
deshl terminate <options> job_identifier_file
```

The result of `terminate` is that the instance that represents the job is destroyed and that the underlying job is removed from the Resource Manager.

This command aims to follow the `qdel` Open Group specification for job termination [9].

### 5.1.13 *sites*

The available sites are provided by the `DESHLAvailableSites` class. By default it processes a command called `sites`. This command only has a help option and no arguments:

```
deshl sites <options>
```

The output from this command is a list of the available sites accessible through the DESHL. The names of these sites can be used in job submission to target the job at that site.

### 5.1.14 *fetch*

Once a job is completed, this can be determined by calling `status`, then the job is finalised and any output retrieved by invoking the `fetch` command. This command only has a help option, the general form is:

```
deshl fetch <options> job_identifier_file <to_dir>
```

The `job_identifier_file` is one previously created by a job submission, whose status shows the job is complete (a job that has already had `fetch` invoked will no longer be known). The optional `to_dir` is the directory to retrieve any output from the job to, if not specified the current directory is used.

On successful completion no output is given but any files the job created will be available in the specified or current directory.

If there was a problem an error message will be displayed on standard error. More information may be available in the logs.

### 5.1.15 *jobs*

All of the user's current job identifiers can be retrieved by invoking the `jobs` command. This command only has a help option, the general form is:

```
deshl jobs
```

This command queries the configured sites for the user's current jobs. A current job is one that has been successfully submitted but not yet fetched. The identifiers for current jobs are collated and presented on standard output with each identifier on a new line. Nothing is output if there are no current jobs.

If there was a problem an error message will be displayed on standard error. More information may be available in the logs.

## 5.2 Component packages and modules

The classes for the CLT exist under the package:

```
org.deisa.jra7.deshl.ui.clt
```

Where `ui` stands for user interface, and `clt` command line tool.

## 5.3 Component Design

The detailed component descriptions can be found in the supporting documentation of the implementation of the classes mentioned above. Please refer to the generated source documentation that will be available with the DESHL release.

# 6. Client Library

The Client Library component exposes two simple programmatic interfaces based on SAGA standards, one for data staging and one for job management. As stated earlier, the Client Library interacts with the Grid Access Library, but hides the complexity of the interaction with the Grid Access Library API. The Client Library is also intended to insulate the application developer from changes in the underlying components.

## 6.1 Data Staging API

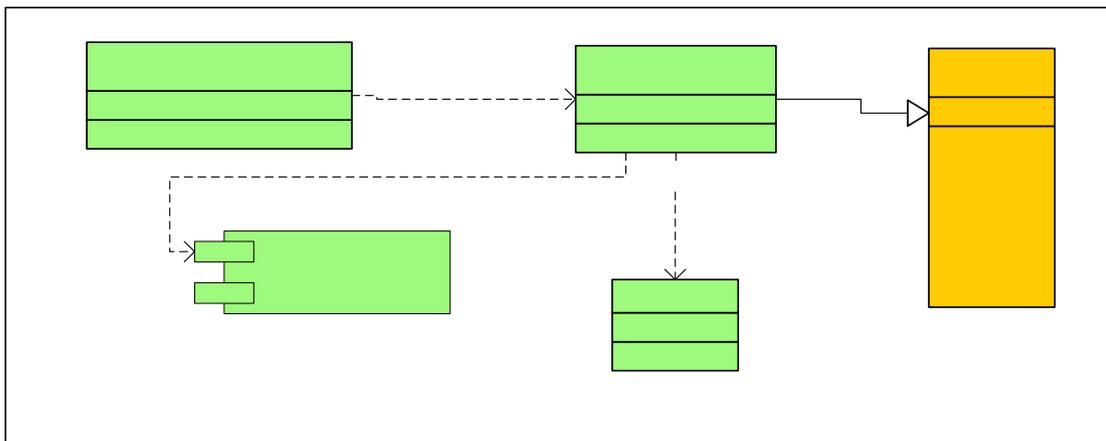


Figure 5: Data staging class diagram

The data staging API exposes one principal interface, *DESHLNSDir*, based on the SAGA standard for data staging, *NSDir* (see Figure 5). An implementation of

*DESHLNSTDir* is provided by the *DESHLNSTDirImpl* class, which is also responsible for interacting with the Grid Access Library classes. An instance of *DESHLNSTDir* is obtained by calling the static factory method *DESHLNSTDirFactory.getInstance()*.

#### 6.1.1 *DESHLNSTDirFactory*

This class acts as a factory class, which defines a single method:

```
public static DESHLNSTDir getInstance();
```

This method creates a static instance of *DESHLNSTDirImpl*, and returns it cast as a *DESHLNSTDir* interface. Only one instance of *DESHLNSTDirImpl* is created by the factory class per java runtime instance. Prior to creating the *DESHLNSTDirImpl* class, the factory class reads a configuration file which must be specified via a runtime system property, "grid.config.file". This file is used to create a configuration object which is then passed to the Grid Access Library to initialise the grid classes. This file must be present, and must be supplied as a comma-separated variable text file with a separate entry on each line. An entry must exist in the configuration file for each site that the client wishes to access. Each configuration entry must consist of the following:

- The full path of the site, in the format `ssl://gateway:portNumber/njsName`
- The full local path of a valid certificate for accessing that site
- The password for that certificate.

#### 6.1.2 *DESHLNSTDirImpl*

This class implements the *DESHLNSTDir* interface, and is responsible for interacting with the Grid Access Library to provide data staging functionality via the Grid Access Library classes. Instances of this class are created and configured from the *DESHLNSTDirFactory* class.

#### 6.1.3 *DESHLNSTDir*

The methods defined by the *DESHLNSTDir* interface are as follows:

- `changeDir` – set the current working directory to another location
- `copy` – copy an existing file or directory within the DEISA file system, or import a file from local storage to the DEISA file system, or export a file from the DEISA file system to local storage
- `exists` – determine if a specified file or directory is an existing location on the DEISA file system
- `getName` – return the full path of the current working directory
- `getNumEntries` – return the number of entries in the current working directory
- `getEntry` – return the name of a specific element in the current working directory
- `isDir` – determine if a specified path is a directory on the DEISA file system
- `isFile` – determine if a specified path is a file on the DEISA file system
- `list` – list the contents of a specified directory
- `makeDir` – create a new directory on the DEISA file system
- `move` – rename an existing file or directory on the DEISA file system
- `remove` – delete an existing file or directory from the DEISA file system

#### 6.1.4 *Component packages and modules*

The Client Library data staging classes exist in the following packages:

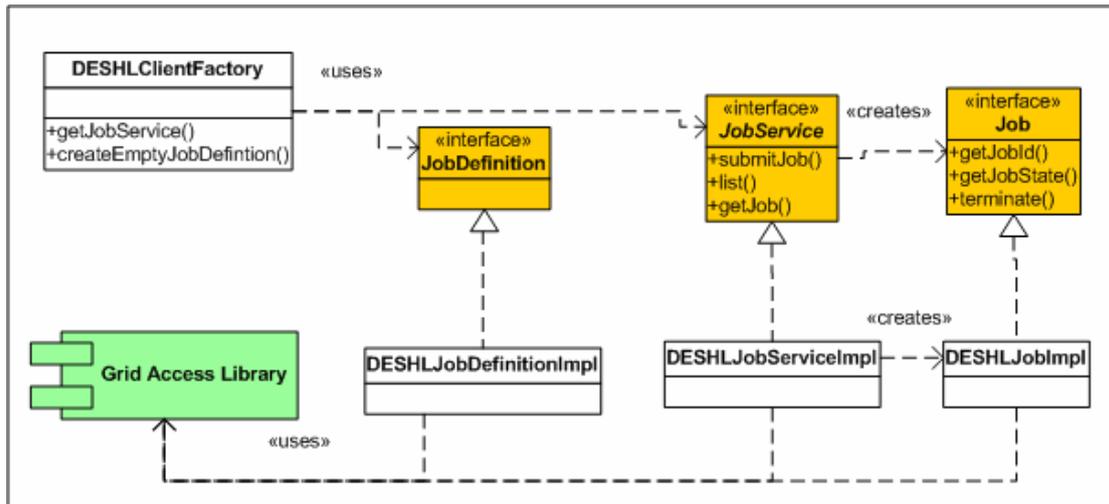
Class	Package
-------	---------

DESHLNSDirFactory	org.deisa.jra7.deshl.client
DESHLNSDir	org.deisa.jra7.deshl.client
DESHLNSDirImpl	org.deisa.jra7.deshl.client.impl

**Table 4: Client Library data staging classes**

## 6.2 Job Management API

The DESHL Job Management API is based on the SAGA Job Management API [10]. The key classes in the Job Management API are shown in the following diagram:



**Figure 6: UML diagram of key classes for Job Management.**

The entry class to the Job Management API is currently the `DESHLClientFactory`. This factory class offers the user access to the SAGA Job Management interfaces – `JobService`, `JobDefinition` and `Job`. Please refer to the SAGA documentation for full details [10]. These interfaces are backed by concrete implementations for DESHL based on the Grid Access Library.

The key classes are explained further below. For the complete and up to date class documentation for all the DESHL implementation classes, refer to the generated source documentation that will be available with the DESHL release.

Unless stated otherwise the methods shown are those implemented from the SAGA API. If a method from the SAGA API is not shown then it has not been implemented.

### 6.2.1 `DESHLClientFactory` class

The `DESHLClientFactory` class is the key entry class to the Job Management API. This class relies on a properties style file for the implementation classes of the interfaces it provides access to. The default properties file is:

```
saga.jobservice.class = org.deisa.jra7.deshl.client.impl.JobServiceImpl
saga.jobdefinition.class = org.deisa.jra7.deshl.client.impl.JobDefinitionImpl
```

**Listing 1: config.properties for DESHLClientFactory.**

This gives the factory the class the fully qualified class names of implementation classes for the SAGA interfaces. The following methods are offered:

```
+ getJobService() : JobService
  { exceptions = DESHLException }
```

Class method that returns the configured implementation of the SAGA `JobService` interface. Any problems with the configuration and loading the implementation class result in a `DESHLException` being thrown.

```
+ createEmptyJobDefinition() : JobDefintion
  { exceptions = DESHLException }
```

Class method that returns the configured implementation of the SAGA `JobDefinition` interface. Any problems with the configuration and loading of the implementation class result in a `DESHLException` being thrown.

### 6.2.2 *DESHLJobDefinitionImpl* class

The `DESHLJobDefintionImpl` implements the SAGA `JobDefinition` interface. Although the SAGA documentation suggests this is a class, it also suggests that the underlying implementation may wish to provide read only attributes and that this factory mechanism provides the best way of offering it. The `JobDefinition` class is purely a holder class for attribute name/values and extends the SAGA `Attribute` interface. A `DESHLAttributeImpl` class provides the underlying implementation of the `Attribute` methods. In general the methods are as specified by SAGA, however some inconsistencies in when exceptions are thrown are rectified. Also all setters return a self reference to allowing chaining.

In a multi-threaded environment access to this class should be synchronised.

The following operations are offered:

```
+ setAttribute( key : String, value : String ) : Attribute
  { exceptions = ReadOnlyAttributeException }

+ getAttribute( key : String ) : String
  { exceptions = NoSuchKeyException }

+ setVectorAttribute( key : String, values : String[] ) :
Attribute
  { exceptions = ReadOnlyAttributeException }

+ getVectorAttribute( key : String ) : String[]
  { exceptions = NoSuchKeyException }

+ listAttributes() : String[]
  { no exceptions }

+ removeAttribute( key : String ) : Attribute
  { exceptions = ReadOnlyAttributeException,
  NoSuchKeyException }

+ doesAttributeExist( key : String ) : boolean
  { no exceptions }
```

This is an additional method to check whether an attribute exists to avoid a `NoSuchKeyException`.

```
+ isAttributeReadOnly( key : String ) : boolean
  { exceptions = NoSuchKeyException }
```

This is an additional method to check whether an attribute is read only to avoid a `ReadOnlyAttributeException`.

```
+ isAttributeSingleValue( key : String ) : boolean
  { exceptions = NoSuchKeyException }
```

This is an additional method to check whether an attribute was set as a single value to determine which getter to use.

```
+ isAttributeVectorValue( key : String ) : boolean
  { exceptions = NoSuchKeyException }
```

This is an additional method to check whether an attribute was set as a vector value to determine which getter to use.

```
+ valueEquals( o : Object ) : boolean
  { no exceptions }
```

This is an additional method to compare two Attribute instances for equal values – same attributes with the same values.

### 6.2.3 *DESHLJobServiceImpl* class

The *DESHLJobServiceImpl* class implements the SAGA *JobService* interface using the Grid Access Library. This shares common configuration details with the Data Staging API implementation that can be contained within a static *DESHLGridConfig* class.

The following methods are offered:

```
+ submitJob( job: JobDefinition ) : Job
  { exceptions = SAGAException }

+ list() : String[]
  { exceptions = SAGAException }

+ getJob( jobId : String ) : Job
  { exceptions = SAGAException }
```

### 6.2.4 *DESHLJobImpl* class

The *DESHLJobImpl* class implements the SAGA *Job* interface using the Grid Access Library. This shares common configuration details with the Data Staging API implementation that can be contained within a static *DESHLGridConfig* class.

This particular implementation requires offering the user a means to ‘fetch’ the job once it is complete. A “clean up” method has been added to the SAGA interface for this purpose.

The following methods are offered:

```
+ getJobId() : String
  { no exceptions }

+ getJobState() : JobState
  { no exceptions }
```

JobState is the SAGA enumeration of possible job states.

```
+ terminate() : void
  { exceptions = SAGAException }

+ cleanUp( toDir : File ) : void
  { no exceptions }
```

This method has been added to allow any additional files generated by the Resource Manager to be returned to the specified directory local to the user. Such files might include standard output and error from the job. It also indicates to the Resource Manager that the user has finished with the job and that it can be cleaned up.

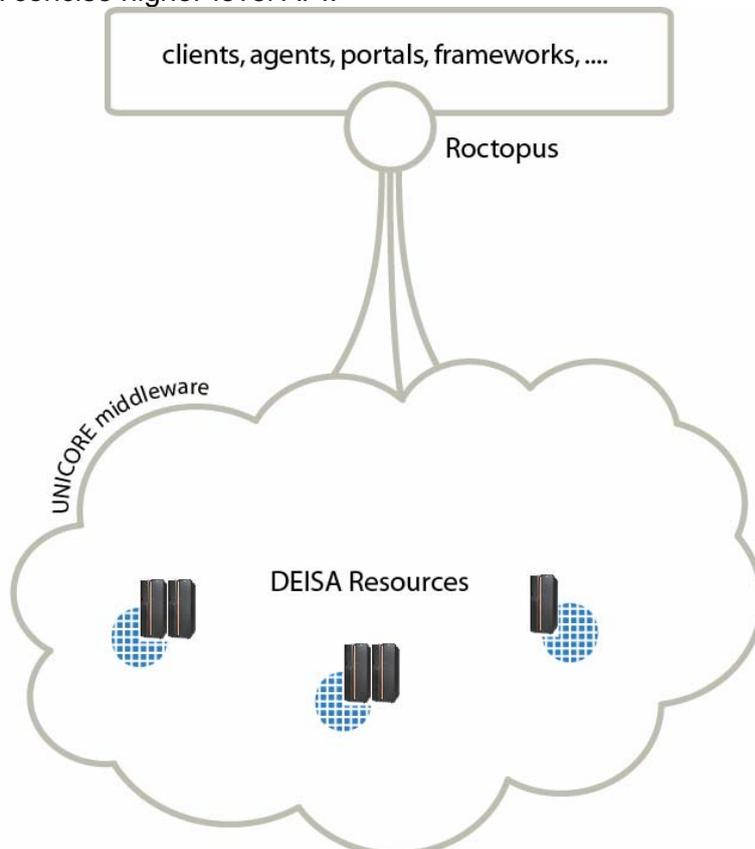
### 6.2.5 Component packages and modules

The SAGA Job Management interfaces can be found under the `org.saga.jobmanagement` package.

The DESHL Job Management can be found under `org.deisa.jra7.deshl.client`.

## 7. Grid Access Library

The Grid Access Library, known as Roctopus, is a high-level API for building Grid agents, portals, frameworks, clients (command line, rich internet applications, 'thick' installed applications, etc). The primary target of the Roctopus API is the UNICORE Grid middleware. Wherever client code for UNICORE might be used, Roctopus can be used as a concise higher-level API.



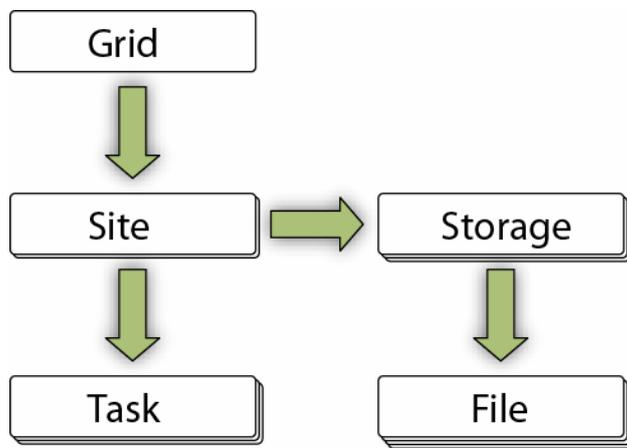
**Figure 7: High level overview**

In JRA7, this Roctopus API provides the foundation for UNICORE access which can be used to present a SAGA-like API for programming Grid access, and consequent command line tools for this (see Figure 7).

The ARCON library for UNICORE provides some assistance for building clients but it is quite low level, and thus does not make client development easy. Moreover the existing UNICORE client is coded in such a way that the Grid access functionality is not easily separated from operation of the GUI, and thus this avenue for re-use is only useful as a reference.

Rather than attempting a programming-language neutral design, this work focuses squarely on an API for Java.

The model is as follows: a Grid consists of a set of links to the Sites for that particular Grid configuration. One can submit a Job to be executed, retrieve a list of executing jobs and query the Attributes (description of the resources) at each Site. A Site contains links to a number of Storages, which consequently contain Files, which a programmer can manage and import/export files with. In addition Roctopus manages a list of sites and the security credentials which are used to access them. This is shown in the diagram in Figure 8.

**Figure 8: The Roctopus Object Model**

The Roctopus API also includes flexible support for job submission and monitoring. Currently UNICORE uses a polling model for checking the status of jobs, transfers, etc. Roctopus presents this as a clean asynchronous interface (currently implemented internally by repeated polling), where interested parties are able to register for notifications. Operations for cleanup and staging are non-blocking - concurrency issues are taken care of by the API.

In addition, Roctopus provides tidy, uncluttered access to the raw information from a UNICORE Grid - information that can be combined, re-processed and presented in interesting new ways.

The public Roctopus API consists almost entirely of interface classes. In the style of many other APIs in java, multiple implementations are to be expected. So, for example, the next generation of the UNICORE software will be called UNICORE/GS

and is being developed in the UniGrids project. It is foreseeable, and quite likely, that in the course of the DEISA project, and in the eDEISA project (currently a proposal), the UNICORE infrastructure will be upgraded to UNICORE/GS. The tools which will be developed on top of the Roctopus API can be ensured of a smooth upgrade path when the infrastructure switches to UNICORE/GS, as essentially the API will remain the same.

## 8. Requirements Satisfaction

### 8.1 Data Staging

Table 5 shows the major features for data staging specified in the 'Functional Scope for DESHL v2.0' [3] that are satisfied by this design. A full description of each requirement can be found in the referenced document. A full qualification of the satisfaction is given later.

Rqmt #	Requirement description	Satisfied?
FR 1 - 4	Upload a remote file to the DEISA file system from local storage	Yes
FR 5 - 8	Upload and append an existing file	No
FR 9 - 12	Download a remote from the DEISA file system to local storage	Yes
FR 13 - 16	Download and append a file	No
FR 17 - 20	Delete a file	Yes
FR 21 - 24	List the contents of a directory on the DEISA file system	Yes
FR 25 - 28	Determine a specified file exists on the DEISA file system	Yes
NR 1	Should adhere to SAGA standards	Partial
NR 2	UI must install on Linux and Windows	Yes
NR 3	UI may install on DEISA HPC system	Yes

**Table 5: Requirements satisfaction for data staging**

#### FR 1 - 4

Implemented as specified

#### FR 5 – 8

This feature has been dropped from the requirements and therefore has not been implemented.

#### FR 9 – 12

Implemented as specified

#### FR 13 – 16

This feature has been dropped from the requirements and therefore has not been implemented.

#### FR 17 – 20

Implemented as specified

#### FR 21 – 24

Implemented as specified

#### FR 25 - 28

Implemented as specified

#### NR 1

SAGA is an evolving set of standards, and a pragmatic approach to design has been taken with regards to complying with SAGA and delivering a functional system within the project timescales.

#### NR 2 – 3

The client is composed of a set of Java classes and libraries requiring Java v1.5 or later. Therefore, any platform capable of running an appropriate version of the Java runtime java should be viable as a platform for installation.

## 8.2 Job Management

Table 6 shows the major features specified in the 'Functional Scope for HSM v1.0' [12] that are satisfied by this design. A full description of each requirement can be found in the referenced document. A full qualification of the satisfaction is given later.

Rqmt #	Requirement description	Satisfied?
FE-1	OGSA-based services	Partial
FE-2	Command Line Tool	Yes
FE-3	Secure and reliable	Yes
FR-1	Submit a batch job to a site through UI	Yes
FR-2	Submit through an OGSA-based service	Partial
FR-3	OGSA-based job submission service	Partial
FR-4	Secured OGSA-based job submission service	Partial
FR-5 – 8	Terminate a batch job	Yes
FR-9	Monitor job status through UI	Yes
FR-10	Monitor job status through OGSA-based service	Partial
FR-11	OGSA-based service for monitoring a job	Partial
FR-12	Secured OGSA-based service for monitoring a job	Partial
FR-13 – 16	Suspend a job	No
FR-17 – 20	Resume a job	No
NR-1	OGSA-based services adhere to standards	Partial
NR-2	UI must install on Linux and Windows	Yes
NR-3	UI may install on DEISA HPC system	Yes

**Table 6: Requirements satisfied by the design**

From the previous release with the change of focus the OGSA based requirements are now marked as partial. This is because this work is being undertaken by UniGrids and it was decided that this work should not be replicated in DESHL but that DESHL should be designed such that when available, DESHL would use the UniGrids software. This was achieved by adhering to the SAGA API implemented using the Grid Access Library. The Grid Access Library is currently based on UNICORE but will be updated to use the UniGrids software. DESHL clients are insulated from this by the SAGA API.

### FE-1 OGSA-based services

The Job Management is based on the SAGA API. The OGSA based services will be provided by UniGrids, rather than replicate this in DESHL implementing the SAGA API provides a standards based API that allows the UniGrids OGSA based services to be plugged in.

### FE-2 Command Line Tool

A command line tool is provided.

### FE-3 Secure and reliable

Based on Unicore.

### FR-1 Submit a batch job to a site through UI

Provided by the CLT.

**FR-2 Submit through an OGSA-based service**

Provided by the Job Management API based on the SAGA API.

**FR-3 OGSA-based job submission service**

This will be provided when available from UniGrids through the Grid Access Library. Users of the CLT and Job Management interface will not see any changes.

**FR-4 Secured OGSA-based job submission service**

Provided by the Grid Access Library, currently based on Unicore, will be provided by UniGrids when available.

**FR-5 – 8 Terminate a batch job**

Provided in the Job Management API.

**FR-9 Monitor job status through UI**

Provided in the CLT.

**FR-10 Monitor job status through OGSA-based service**

The Job Management API uses the Grid Access Library (see FR-11).

**FR-11 OGSA-based service for monitoring a job**

Provided by the Grid Access Library, currently based on Unicore, will be provided by UniGrids when available.

**FR-12 Secured OGSA-based service for monitoring a job**

Provided by the Grid Access Library, currently based on Unicore, will be provided by UniGrids when available.

**FR-13 – 16 Suspend a job**

This has not been provided in this design. This will be provided in a subsequent release.

**FR-17 – 20 Resume a job**

This has not been provided in this design. This will be provided in a subsequent release

**NR-1 OGSA-based services adhere to standards**

The following standards will be used directly:

- SAGA
- JSDL

Indirectly the Grid Access Library will provide full OGSA-based access to Unicore when it becomes available. This also allows for the possibility of supporting OGSA based access to any other DRM.

**NR-2 UI must install on Linux and Windows**

Based on the assumption that the client is Java based (see AS-1 in [4]) all the client code will install on Linux and Windows provided they meet that assumption and that a suitable JVM is present.

**NR-3 UI may install on DEISA HPC system**

If a DEISA HPC system meets AS-1 in [4] then the client will install.