Revision 0.5.4

GRIDportal White Paper

Martin Matusiak

22nd November 2005

GRIDportal source code and full documentation available at the project site.
`http://gridportal.dynalias.org/`

Contents

List of Figures

Preface

This documented aims to describe GRIDportal in some level of detail, focusing not on its internals but rather its positioning in the software stack and how it interacts with other software components.

The way this document is laid out, it lends itself to sequential reading so you will probably find it most beneficial to read the chapters in the order they are presented.

0.1. Intended audience

This documented is intended for system engineers/administrators who are evaluating GRIDportal for their needs or indeed deploying GRIDportal on their grid sites. If you are reading this, it is understood that you are familiar with NorduGrid/ARC middleware and thus I make no effort to explain what it is and what services it provides. The NorduGrid documentation[1] does a much better job of that and I refer you to it.

After reading this white paper you should understand

- what GRIDportal is and how it relates to grid middleware and the http server.
- whether or not GRIDportal is a good choice for your grid site.

0.2. Chapter guide

Chapter 1 - Introduction aims to give you a high level introduction to GRIDportal, what its purpose is and what needs have brought it into existence.

Chapter 2 - Installation aims to explain what is needed to install GRIDportal and walk you through the installation procedure.

Chapter 3 - Architecture aims to give a fairly comprehensive explanation of how GRIDportal interacts with all the software components which surround it.

Chapter 4 - NorduGrid aims to give a brief introduction to NorduGrid as a basis for an explanation of how GRIDportal is affected by NorduGrid, both the NorduGrid network and the middleware. The crucial point here is the authentication mechanism NorduGrid provides.

Chapter 5 - Design aims to give a quick overview of how GRIDportal is designed internally and how it operates.

[1]NorduGrid documentation `http://www.nordugrid.org/papers.html`

1. Introduction - the bird's eye view

1.1. The problem

Grid resources offer plenty of processing power and many common scientific applications have been written to operate on grids. But there exists a certain gap in usability when using these applications on a workstation on the one hand and accessing them on a grid resource on the other. Widely used scientific software, such as Matlab and Abaqus, is commonly targeted at the Windows platform, where the user interacts with the software through a graphical user interface (GUI). Many software packages also offer a backend module, without a GUI, for heavier computations than a workstation can process in a reasonable amount of time. These backend modules can be installed on a grid resource and offered to users, but there is an inherent drawback to this approach.

In the most common scenario, software running on a grid resource does not offer a graphical user interface. Instead, the user must utilize a command shell through a Unix account on the grid resource in order to make use of the software. Since the bulk of the potential user base for grid-enabled applications comes from a Windows background, the transition to the Unix environment is not entirely trivial. In most cases, the facilitator of a grid resource will have to provide an introductory course to Unix, in order to bring the user up to speed on how to make use of the grid.

While a Unix course does solve the problem, it has several drawbacks:

1. The grid administrator would rather focus on managing the grid without being held up in user support.
2. Every new user must take the course to be able to use the grid resource.
3. There is no prospect of resolving the above two issues long term.

Users also do not know anything about grids to begin with, so additional schooling in the operation of a grid would be necessary.

1.2. The solution

A solution to the usability gap problem was devised by Jonas Lindemann from the Division of Structural Mechanics at Lund University, Sweden. He proposed a web based application portal which would serve as an intermediary between the user and the grid. His project, entitled Lunarc Application Portal, was forked into GRIDportal. The objectives of the portal are:

- To provide an easy-to-use interface to the applications on the grid.
- To set focus on the applications and hide the specifics of grid computing from the user to the extent possible.
- Not to compromise existing security policies of grid computing.

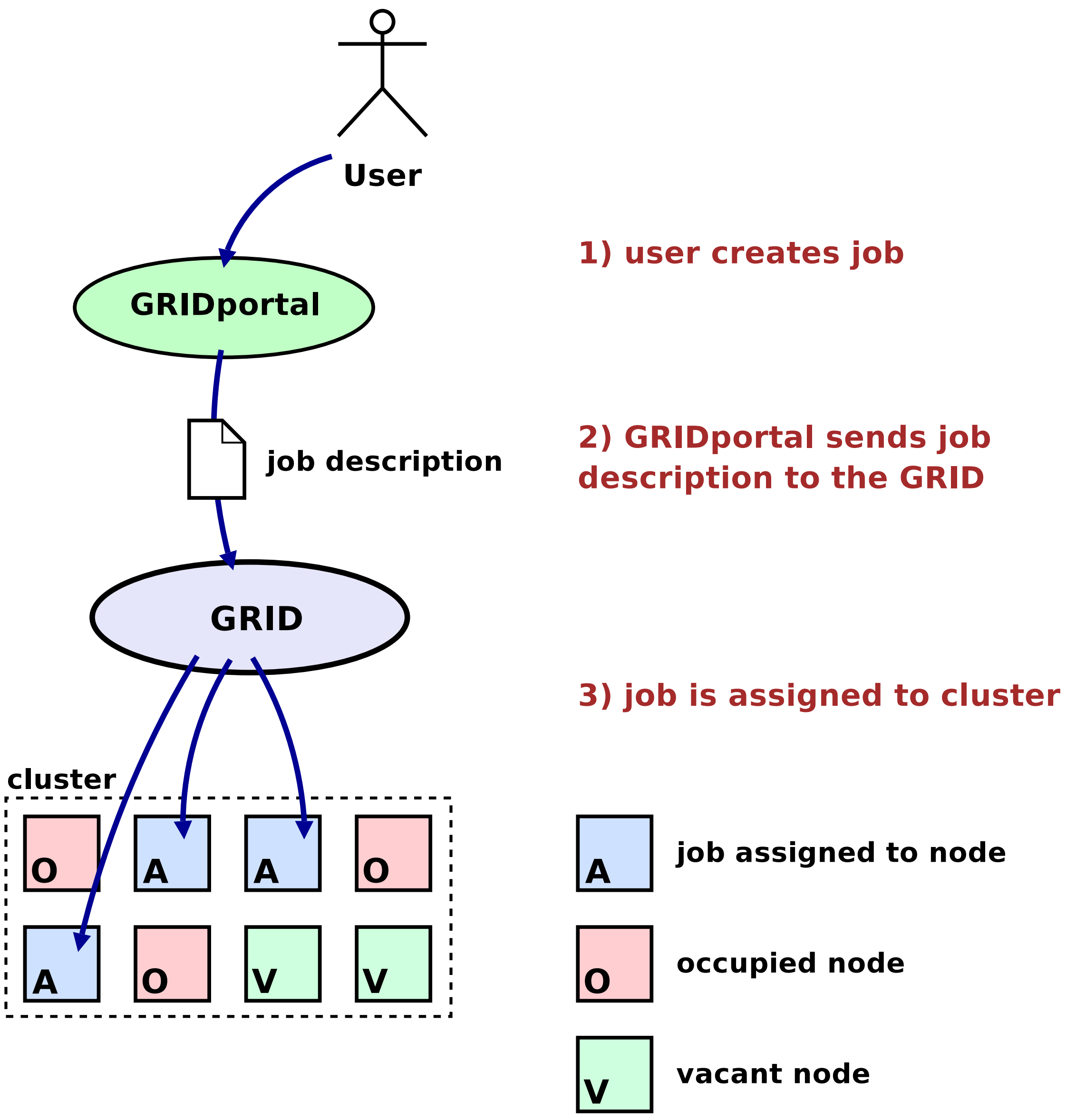

Figure 1.1.: Conceptual model

As shown in the Conceptual model, the portal fills the usability gap between the user and the grid resource. The portal provides a simple interface to the user, where he/she can define the job they wish to execute on the specific application. The job is then written to a job description file and sent to the grid scheduler. The scheduler decides where and when the job is executed, depending on resource availability. While the job is running, the user will receive status updates via email. Once it is finished, the user can retrieve the results through the portal.

2. Installation

2.1. System requirements

GRIDportal depends on the following software.

2.1.1. A *nix operating system

GRIDportal was designed and tested on Linux but should work just as well on other Unices, perhaps with some minor changes.

2.1.2. Apache HTTP server 1.3.x/2.0.x

The most commonly used HTTP server worldwide. Other web servers can probably be used as well, with some effort.

2.1.3. Webware for Python

Webware for Python is a suite of software components for developing object-oriented, web-based applications. The suite uses well known design patterns and includes popular features such as a fast application server (WebKit), Python Server Pages (PSP), and a CGI wrapper.

Note: WebKit must run as a Unix daemon to serve GRIDportal web pages.

Note: Webware requires Python.

GRIDportal is written specifically for Webware.

2.1.4. HyperText HTML code generation library (bundled)

Although Webware is meant to serve as a server side scripting platform, building html pages is more convenient when using John Dustman's HyperText HTML library. The library is contained within the GRIDportal source code.

2.1.5. NorduGrid/ARC middleware

The NorduGrid middleware (or Advanced Resource Connector, ARC) is an open source software solution distributed under the GPL license, enabling production quality computational and data Grids.

GRIDportal functions are mapped to NorduGrid/ARC commands. The portal essentially translates user input through a web page into commands invoked on the command prompt.

2.1.6. Command shell

The command shell is the gateway between the portal and the middleware. In most cases it will be bash, but other shells should work just as well.

2.2. Install procedure

```
# $File: src/gridportal/INSTALL

1. In order to run GRIDportal, you must have Webware installed and WebKit
   running on your system. The installation guide for WebKit[*] is
   located at the Webware website and I will not duplicate it here. You
   should set up WebKit as a system service running under its own user
   account.

         You will also need to install pyGlobus[**], it has to be on the
         PYTHONPATH.

2. Next, create a working directory as per the instructions in the WebKit
   install guide, for instance:

    $ python /path/to/Webware/bin/MakeAppWorkDir -l -cvsignore \
    -c context gridportal

   The directory structure inside gridportal/ should now look like this:

    404Text.txt Cache/ ErrorMsgs/ Logs/ WebKit.cgi lib/ AppServer
    Configs/ Launch.py Sessions/ context/

3. Now unpack the GRIDportal source code and copy the contents of
   src/gridportal/ to /path/to/context/.

4. Next, open /path/to/Webware/WebKit/Configs/Application.config and add
   under ''Contexts'':

    'default': '/opt/gridportal/context',

5. Now restart the WebKit application server and you should be able to
   access gridportal. The exact URL will depend on your webserver/Adapter
   configuration.

6. I'm assuming you created a special user account for webware/webkit.
   You should now create a homedir for this user. The homedir is crucial,
   because this is where GRIDportal will write all of its data. For
   example:

    $ mkdir /var/webware

   Now make sure this user owns the homedir:

    $ chown webware:webware -R /var/webware

7. Finally, set this url in the conf_main.py config file in the root
   gridportal directory:

    env_homepath = "/var/webware"
```

```
* Webkit installation guide
  http://www.webwareforpython.org/Webware/WebKit/Docs/InstallGuide.html

**PyGlobus
        http://dsd.lbl.gov/gtg/projects/pyGlobus/
```

3. Architecture

3.1. What is GRIDportal?

In order to understand how GRIDportal functions (and how it should function), it is important to understand exactly what it is and what it isn't.

3.1.1. What it is

GRIDportal is a collection of server side scripts, written in Python, which run on an HTTP server and make up a web site. The portal fills a gap between the user, expecting a web interface to the grid, and the grid, designed to be accessed through a command line interface. In order to function, the portal depends on a series of other software. Considering that the complete set of interconnected software makes a fairly complex construct, it is helpful to consider the portal seen from different perspectives, discussed in the following sections.

GRIDportal is distributed under the General Public Licence (GPL).

3.1.2. What it isn't

GRIDportal is not a replacement for any of the following.

- An HTTP server.
- A server side scripting framework.
- A grid scheduler and/or management system.

3.2. The HTTP perspective

The HTTP perspective strives to explain how an HTTP request is passed onto the portal and handled by the portal. It could be said to be the top-most perspective of the portal, as it is the part that the user is immediately in contact with. The process can be divided into a series of steps:

1. The user requests a page of the portal through a web browser.
2. The request is sent to the grid's Apache web server, listening on port 80.
3. Apache interprets the request and decides that it is meant for GRIDportal. This translations process is handled by the special mod_webkit plug-in to Apache, which is part of WebKit.
4. The request is forwarded to the WebKit server listening on port 8086 (but only visible from the machine running Apache).
5. WebKit identifies that the page requested is part of the GRIDportal application.

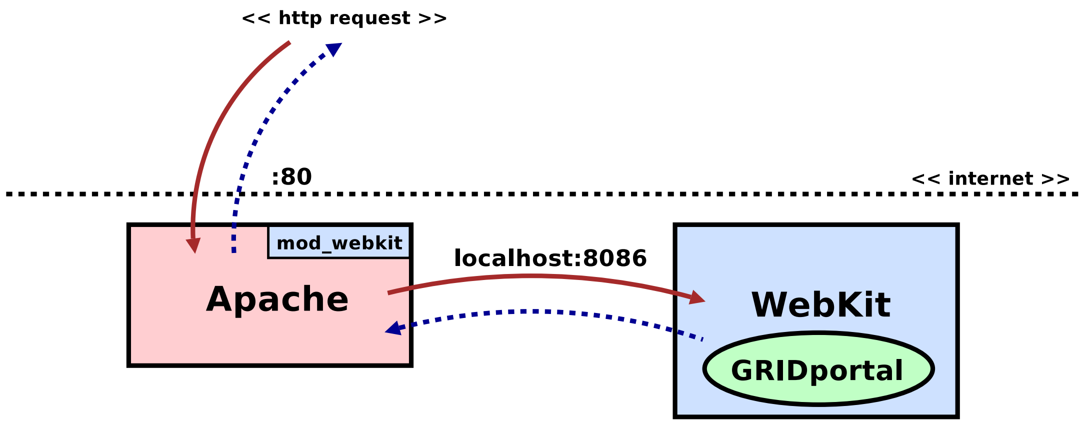

Figure 3.1.: HTTP request model

6. WebKit runs the script file which was requested. The result of the run is a web page produced by GRIDportal.
7. The web page is sent back to Apache.
8. Apache forwards the response to the user's web browser.

3.3. The WebKit perspective

3.3.1. Webware, WebKit and GRIDportal

In order to understand how WebKit operates, it may be helpful to understand how WebKit relates to Webware and GRIDportal.

3.3.1.1. Webware

Webware is a framework which allows building web-based, object-oriented applications in Python. In other words, it's a collection of software components which combined offer this capability.

3.3.1.2. WebKit

One of those components is *WebKit*, an application server, that is the part which runs the applications. WebKit works with *servlets*. Any Python module (a Python module is a Python source code file) invoked through WebKit (that is through a web page) is a servlet. WebKit uses Python's Just-In-Time compiler, which compiles a module into byte code the first time it is run and then caches the byte code file for future runs. (This effectively means that if a module in GRIDportal is changed, WebKit must be restarted to know about this change.)

3.3.1.3. Others

Other components of Webware include CGIWrapper, a CGI interface to Webware, Python Server Pages, for writing pages where Python code is mixed with html (like the php scripting language) and several others.

3.3.1.4. GRIDportal

GRIDportal is an application running on the WebKit application server. Which makes all Python modules within GRIDportal servlets.

Because GRIDportal runs within Webware, it can access all the components which Webware provides. Consider GRIDportal to be the application and Webware to be the library of classes and functions it relies on.

3.3.2. WebKit page serving

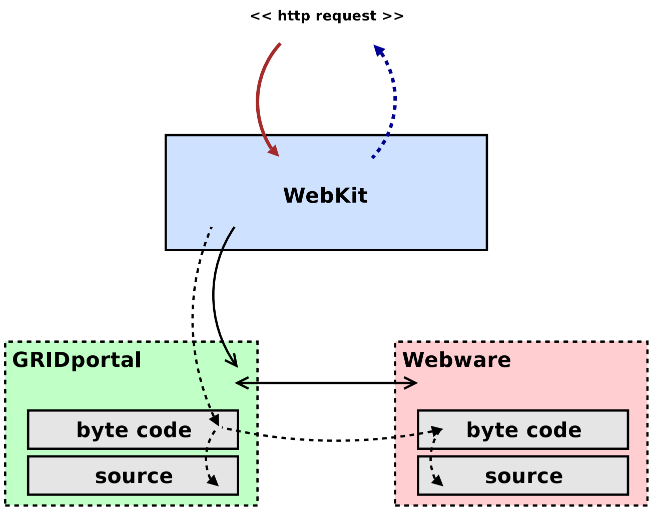

Figure 3.2.: WebKit model

As a request comes in for a web page, WebKit's interaction with GRIDportal is illustrated by the full arrows. A description is given below.

1. An HTTP request comes in for a URL.
2. WebKit finds the Python module (file with .py extension) corresponding to the URL.
3. If the module depends on Webware, required modules from Webware are used when running the module.
4. WebKit runs the module.
5. The result of the run is a web page, which is returned as a response to the HTTP request.

What happens behind the scenes is shown with the dotted arrows. A description is given below.

1. An HTTP request comes in for a URL.
2. WebKit finds the Python module (file with .py extension) corresponding to the URL.
    a) If the module exists in compiled form (file with .pyc extension), this file is used.

   b) Otherwise the module is compiled into byte code.

3. If the module depends on Webware, required modules from Webware are used when running the module. For each of these, the following applies:

   a) If the module exists in compiled form (file with .pyc extension), this file is used.
   b) Otherwise the module is compiled into byte code.

4. WebKit runs the compiled module.

5. The result of the run is a web page, which is returned as a response to the HTTP request.

3.4. The frontend perspective

The frontend perspective strives to explain in simple terms how the portal interacts with the user. I have included three common scenarios to explain what the user does and what the outcome is.

Please note that the model is a simplified one and does not aspire to illustrate the entire communication stream between the user and the portal.

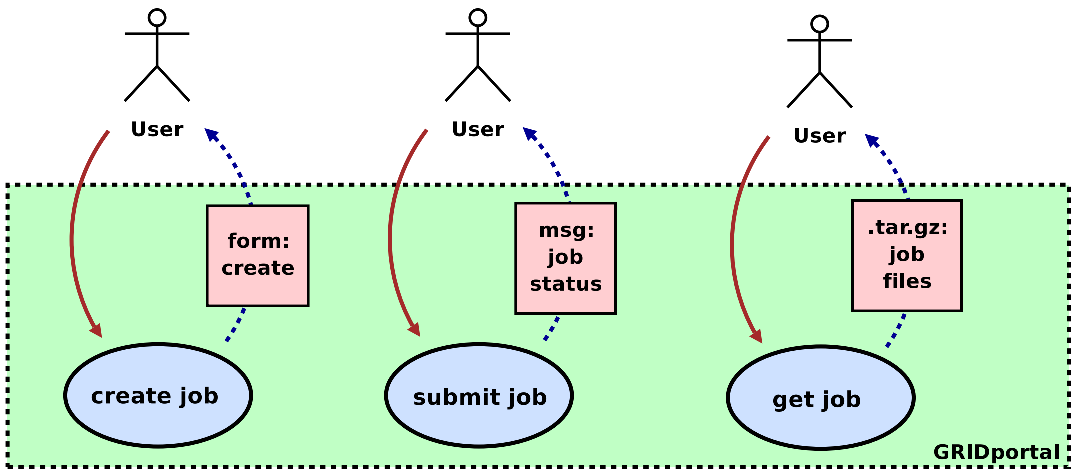

Figure 3.3.: Frontend model

First the user will want to create a job for submission to the grid.

1. The user invokes the create job function.
2. The portal returns an html form to the user which lists all the required job parameters and attributes.
3. The user fills in the form, including any input files necessary to the job. (The input files are transferred through HTTP.)
4. The portal validates the input of the form and returns a message saying the job has been created successfully.

Now the job has been created, validated and is eligible for submission.

1. The user invokes the submit job function and selects the job for submission.
2. The portal submits the job and returns a status report for the job. (Depending on the state of the job queue and the job itself, no job is guaranteed to be accepted by the grid.)

A job which has been submitted and has completed can then be retrieved by the user.

1. The user invokes the get job function and selects the job to get.
2. The job files are presented to the user in an html form. The user can select files individually and view them on-line or download them. The user may also choose to download all the files in a .tar.gz archive for offline use.

3.5. The backend perspective

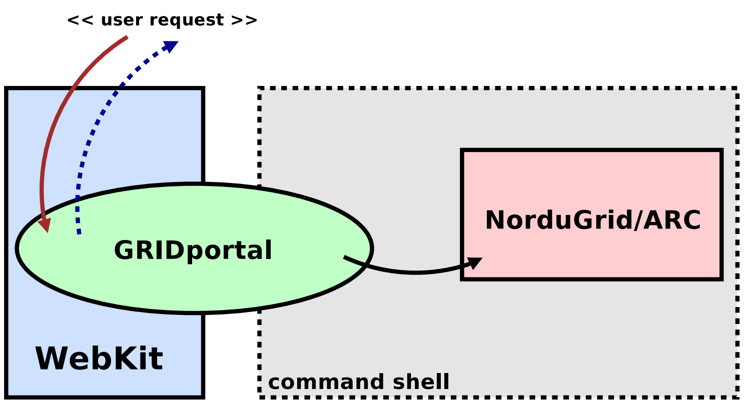

Figure 3.4.: Backend model

The backend perspective strives to show how the portal interacts with the grid middleware. It must be stressed that all interaction with the grid itself, that is job submission, monitoring, queue management and the like, is handled on a level beneath that which the portal operates. The portal interfaces the NorduGrid/ARC middleware, which provides all these services.

A user request to the portal happens in the form of a web page being processed. The resulting information is translated by the portal into middleware language (that is a middleware command with arguments). As the portal runs in a Python environment, it can access the command shell and execute commands directly through it. The requested command is now passed onto the middleware.

For example, let's consider a hypothetical job submission.

1. The user describes the type of job to be submitted through the portal, including all relevant parameters, such as the application it is to use, the job input file(s), the job parameters etc.
2. The portal retrieves all information about the job, captures the input file(s) and generates a job description file in middleware format.

3. All of the files comprising the job are passed onto NorduGrid/ARC via the command shell. The middleware adds the job to the job queue.

The user may now wish to monitor the job.

1. The user issues a job status check request through the portal.
2. The portal passes the job status request to the middleware via the command shell and reads the response.
3. The response is parsed by the portal and reported to the user.

Once a job completes, the user wishes to retrieve the results.

1. The user issues a job retrieval request through the portal.
2. The portal passes the request to the middleware via the command shell. All the files from the job are downloaded to a location on the file system.
3. The portal lists the files to the user for download.

The job may now be changed, submitted again or removed.

3.6. The physical perspective

Given that the portal is a means of bringing the user in touch with the grid, to the reader it may seem that any mention of a grid is conspicuously absent so far in this chapter. That is so for a reason. As mentioned, the portal does not interface the grid directly, it does so only through middleware.

But because the grid presents the ultimate goal in this chain, I have included this model as an attempt to illustrate how the portal is positioned in the larger perspective. As an example of a possible deployment environment, I present Norgrid Cluster at the Norwegian University of Science and Technology, Trondheim, Norway.

3.6.1. Case study: Norgrid Cluster, NTNU

Norgrid is a cluster of 64 nodes, located at the Norwegian University of Science and Technology. A node can only be reached through the cluster frontend machine, which serves as an intermediary between the user and the cluster. All jobs to the cluster are scheduled on the frontend.

The cluster runs on the Rocks Cluster Distribution and deploys the Torque PBS (Portable Batch System), which is a cluster resource management system.

NorduGrid/ARC is positioned above Torque, that is the Torque queue has one queue specifically for NorduGrid/ARC. All jobs submitted through NorduGrid/ARC use this queue.

As shown in the diagram, a user request to the portal in logical terms happens as shown by the full arrows. The user communicates with the portal and can access well known applications through the portal.

What actually happens is shown with the dotted arrows. A request sent from the user's browser reaches Apache and is then passed on down the software chain. Finally, the job is executed by NorduGrid/ARC inside a Torque queue.

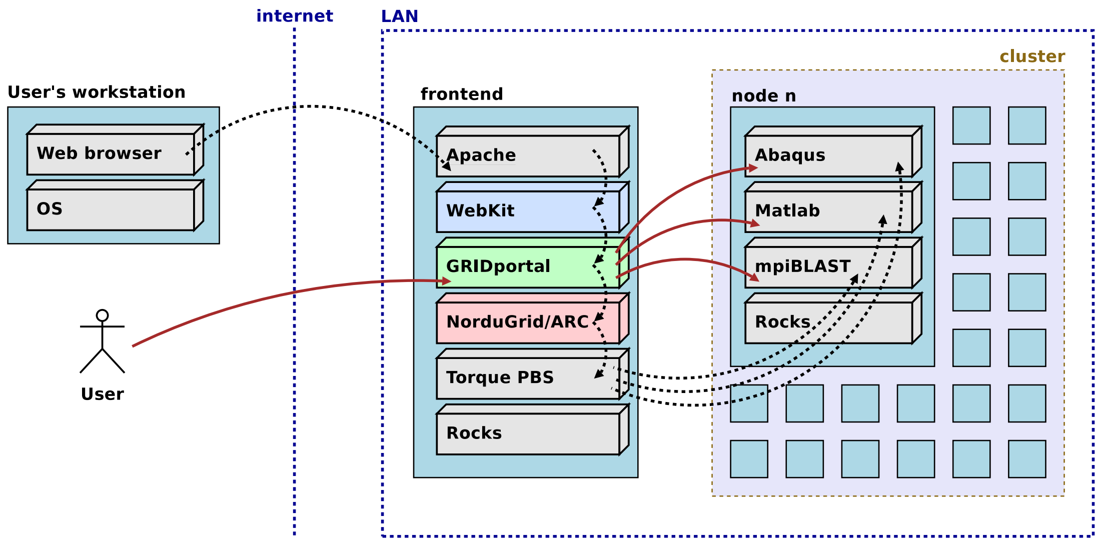

Figure 3.5.: Physical model: Norgrid Cluser

4. NorduGrid

4.1. Network and middleware

Because GRIDportal is closely tied to NorduGrid, it is fairly important to understand the relationship. From NorduGrid's own definition:

> NorduGrid is a Grid Research and Development collaboration aiming at development, maintenance and support of the free Grid middleware, known as the Advance Resource Connector (ARC).

Simply put, NorduGrid produces ARC middleware, which is deployed at a wide range of grid sites, primarily in Scandinavia. All of these sites make up the NorduGrid network of grid resources, which can be accessed from any one of the deployment sites.

The NorduGrid/ARC middleware is the software layer which facilitates access to the NorduGrid network. In practice this means that through the middleware a grid user can access any grid resource in the network, provided he/she has the required credentials, rather than being limited to the local grid.

Because grid usage in this manner no longer is confined to a local grid site, in fact it can span organizational and even national borders, a sound mechanism for authentication is required.

GRIDportal requires NorduGrid/ARC to function, it inherits its features and offers them through a web interface, but it is also bound by its restrictions and limitations. One such restriction is the authentication mechanism.

4.2. Authentication

Access to the NorduGrid network is controlled through a series of steps, described in the sub sections which follow. All of the steps must be completed in order to access a specific grid resource. The absence of one or more steps gives no access whatsoever.

4.2.1. User certificates

Authentication to NorduGrid, that is verification of identity, is handled through user certificates. A user certificate identifies a user uniquely on the NorduGrid network, by name and organization. A sample certificate identifier can look like this:

```
/O=Grid/O=NorduGrid/OU=ntnu.no/CN=Martin Matusiak
```

A user wishing to access a grid resource within the NorduGrid network (or simply a user wishing to use GRIDportal) must apply for a user certificate to a certificate authority. The authority will verify that the user is legitimate and validate the certificate. A certificate is password protected.

In practice this means generating a certificate request by using the middleware and emailing the request to the certificate authority. The certificate identifier (the information about who the user is and which organization the user belongs to) is contained within the certificate itself.

A certificate consists of two files, called a *key pair*. One key is private, the other is public. Upon generating a certificate request, a certificate is created. But it is not *valid*, ie. it will not be accepted in use, until it has been *signed* by a certificate authority, which is why the public key has to be emailed to the certificate authority for signing. Once the authority has processed the request and signed the certificate, it is returned to the user and can now be used for authentication.

Upon obtaining a signed user certificate, the user still has no access to the NorduGrid network.

4.2.2. User proxies

Since NorduGrid is a network designed to comprise thousands of grid resources worldwide, it becomes problematic to keep track of individual users as they access many different grid resources. To the local administrators, this would mean managing user accounts for both local and remote users.

Another concern is uniformity of user accounts, that is ideally a user accessing different grid resources should have the same access priviliges to each grid, otherwise the work flow is interrupted and the question of securing the grid from abuse is raised.

A final consideration is that regardless of how many layers of abstraction the grid middleware may have, the user commands accessing the application ultimately have to be executed by a user on the system, at the operating system level.

For all of these reason, NorduGrid/ARC handles *authorization*, that is access control, to the NorduGrid network through user proxies. A proxy is an intermediary between the user and the NorduGrid network. A valid proxy gives access to the NorduGrid network.

A proxy is a file generated by the user and valid for a limited period of time (typically 24 hours). Then the proxy is submitted to the NorduGrid as authentication key. Once a proxy expires, a new one must be generated to access the NorduGrid network.

Proxy generation is handled by the middleware and requires a signed user certificate. The user is asked for his/her password (the password corresponding to the user certificate) along the way.

With a valid proxy, a user can access the NorduGrid network, which in practice means being able to submit jobs, check status on jobs etc. But without completing the final step (described in 4.2.3), all of these requests will be denied because no grid will accept them.

4.2.3. Virtual organizations

A user in possession of a valid user proxy will be authorized to use the NorduGrid network but without completing this last step, the user will not be authorized locally by any grid within the network, so actual grid resources are still off limits.

As mentioned in the previous step, when using a grid resource, there is a mapping from the grid user to the local user. This mapping is facilitated by the system of Virtual Organizations. A user wishing to access grid resources on the NorduGrid network (or simply a user wishing to use GRIDportal) is required to join a Virtual Organization. This membership then grants priviliges of grid access on the network.

4.3. Implications for GRIDportal

4.3.1. Network and middleware

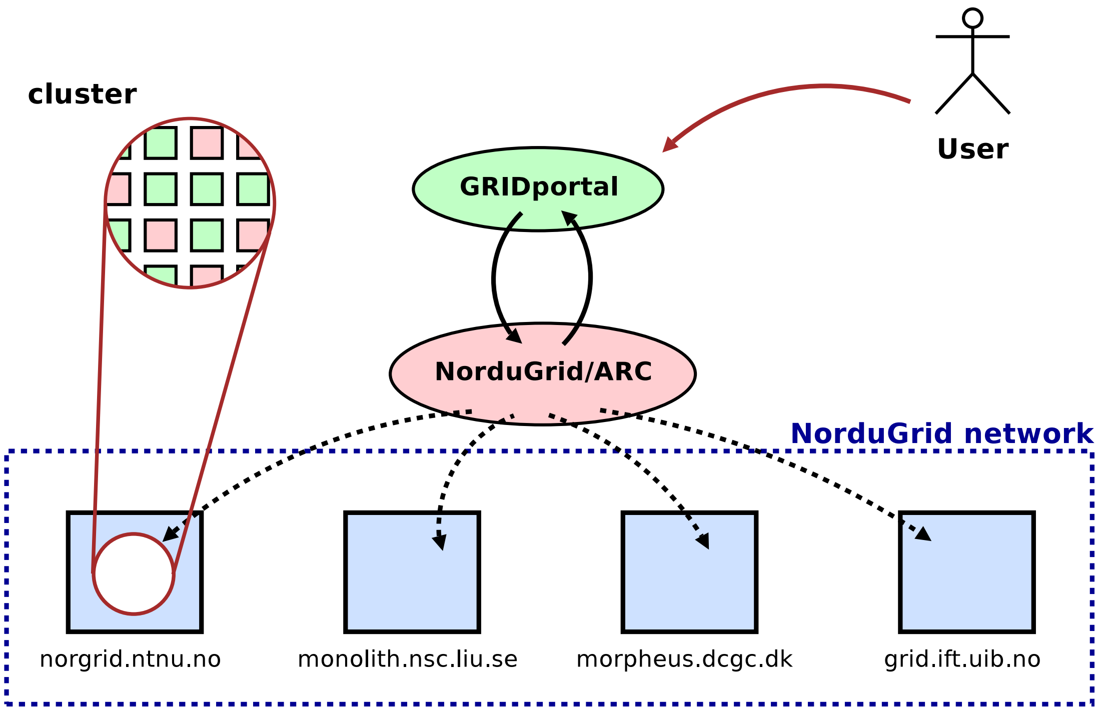

Figure 4.1.: GRIDportal a gateway to the NorduGrid network

GRIDportal is an interface built on top of the NorduGrid/ARC middleware, which allows user friendly access to NorduGrid/ARC services. It does not replace or overlap any part of NorduGrid/ARC, it simply co-exists with it. The advantage of building on top of NorduGrid/ARC (as opposed to building directly on top of a grid resource) is that GRIDportal can offer the full repertoire of NorduGrid/ARC services instead of being confined to a single local grid. Jobs submitted through GRIDportal can be assigned to any grid within NorduGrid, provided the user has been authorized to use that particular grid resource.

4.3.2. Authentication

As mentioned, GRIDportal is tied to NorduGrid/ARC and inherits both its benefits and restrictions. One such restriction is the authentication mechanism, which is fairly complex, mixing user interaction with manual intervention from certificate authorities and grid administrators. Since the primary goal of GRIDportal is to make grid usage simpler to the users, this barrier is very inconvenient.

For the moment, GRIDportal does *not* provide a way around this. It is the philosophy of NorduGrid to generate both the certificate and the proxy on the client side and GRID-portal does not challenge this point of view. But even though there are tools to handle this on the client side, none of them are very convenient for new users.

It should be the aim of GRIDportal in future development to solve this problem and provide a user friendly authentication method, without compromising security.

5. Design

5.1. Project status

As mentioned, GRIDportal is a fork off Jonas Lindemann's "LUNARC Application Portal" (LAP). The bulk of the code base in GRIDportal originates from LAP and is Jonas' work.

5.1.1. GRIDportal changes/additions

5.1.1.1. Source code

This section lists the major changes (source code only) since the project was forked.

conf_main.py New configuration module to keep common config settings out of source code.

conf_blast.py New configuration module specifically for BLAST.

BlastJobPage Class added to process Blast jobs.

BlastTask Class added to process Blast jobs.

DocUserGuide, DocCreateJob, DocEditJob, DocDeleteJob, DocSubmitJob, DocKillJob, DocCleanJob, DocGetJob, DocTar Classes added to provide user documentation.

MatlabJobPage Class added to process Matlab jobs.

MatlabTask Class added to process Matlab jobs.

FormDescPage Class added to display help for html forms.

Form Added function for select tag and input field label descriptions (URLs for labels).

LunarcPage Changes in the menu.

ManageJobPage Function deleteJobYes() implemented.

Ui Hacked the source to include more parameters.

5.1.1.2. Documentation

LUNARC Application Portal was inherited by GRIDportal with no documentation whatsoever. In the course of the GRIDportal project, the following documentation has been produced.

- the User's Guide (integrated into GRIDportal)
- GRIDportal White Paper (this document)
- GRIDportal API reference
- GRIDportal Specification
- GRIDportal Test Document

5.2. Schematics

This section explains the significance of the most important classes which make up GRIDportal.

5.2.1. Page

This diagram illustrates two of the key classes in GRIDportal - *LunarcPage* and *SecurePage*. Both are derived from the base class *Page*, supplied by WebKit. *SecurePage* also inherits from *Configurable* (from Webware/MiscUtils).

LunarcPage draws the main page template common to [almost] all the pages on the portal website, including the menu. It also writes the stylesheet.

SecurePage implements an authentication routine, so all classes derived from *SecurePage* require a valid session.

Most classes in GRIDportal which deal with displaying pages derive from either *LunarcPage* or *SecurePage*.

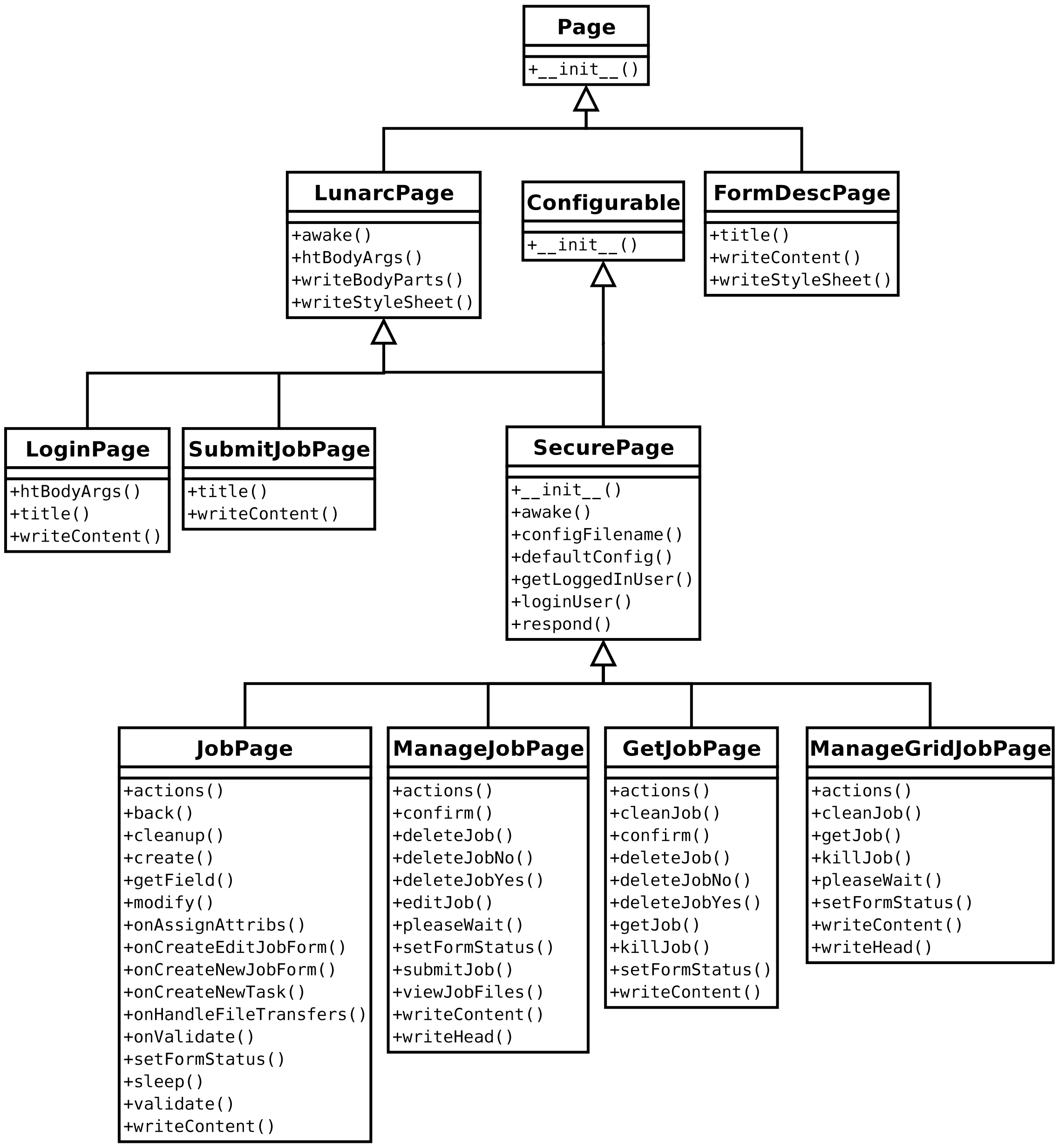

Figure 5.1.: UML diagram: Page

5.2.2. Task

In GRIDportal, a ”job” is implemented by the class Task. Jobs for different applications require different implementations and are sub-classed from Task.

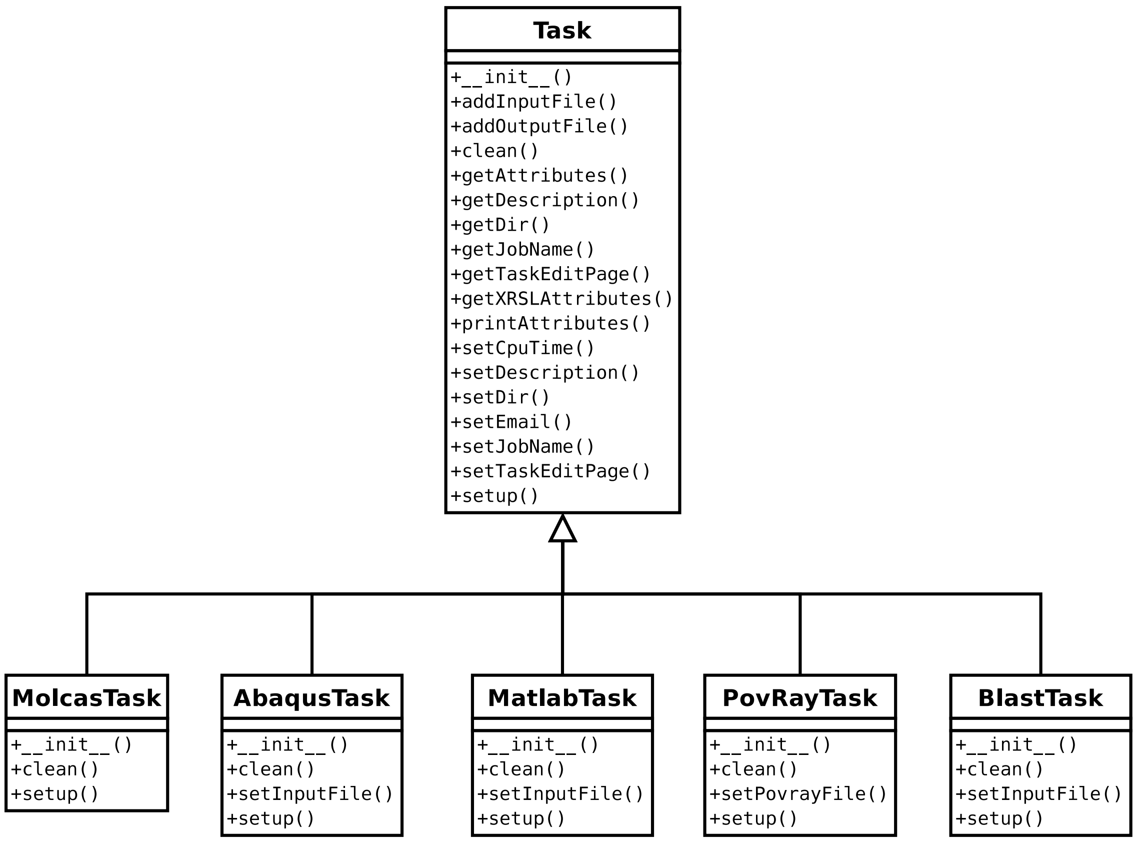

Figure 5.2.: UML diagram: Task

5.2.3. JobPage

Similarly, JobPage is the common denominator for the processing of all pages - create/edit job forms as well as processing of html forms. Then there are derived classes for jobs for specific applications.

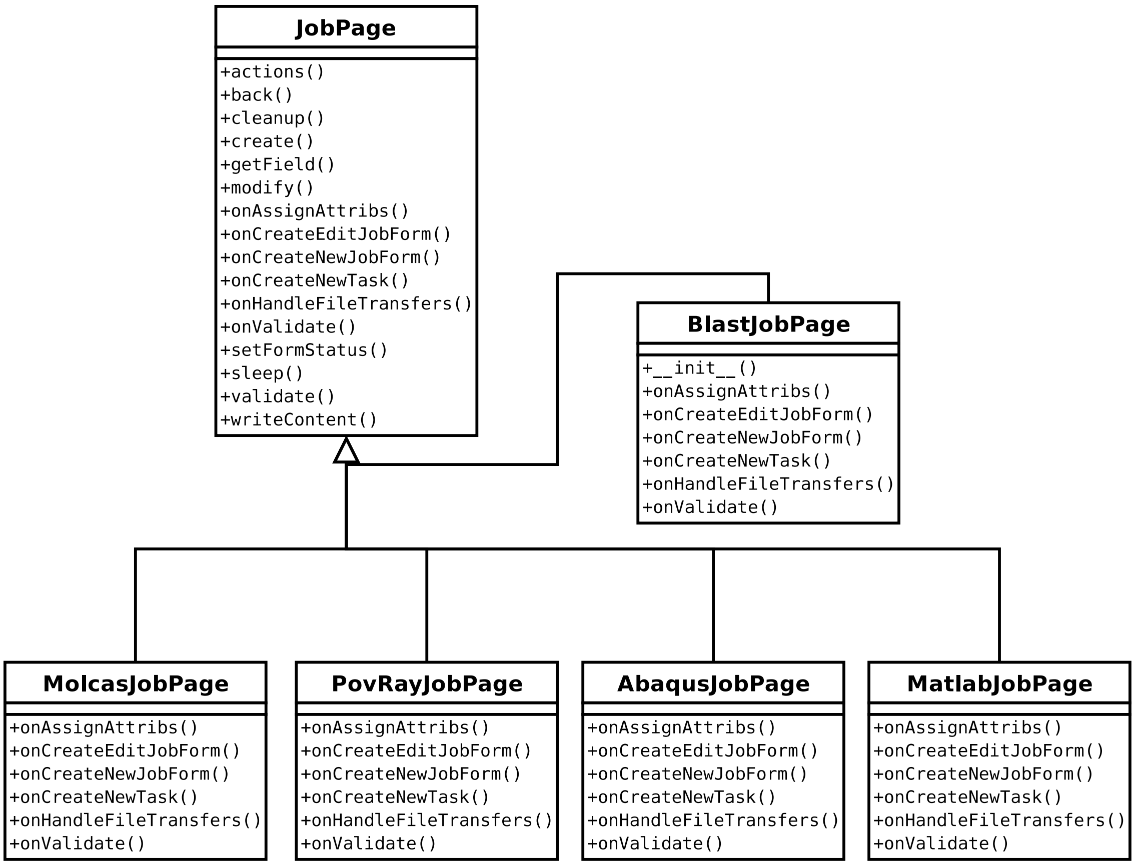

Figure 5.3.: UML diagram: JobPage

5.2.4. BlastJobPage

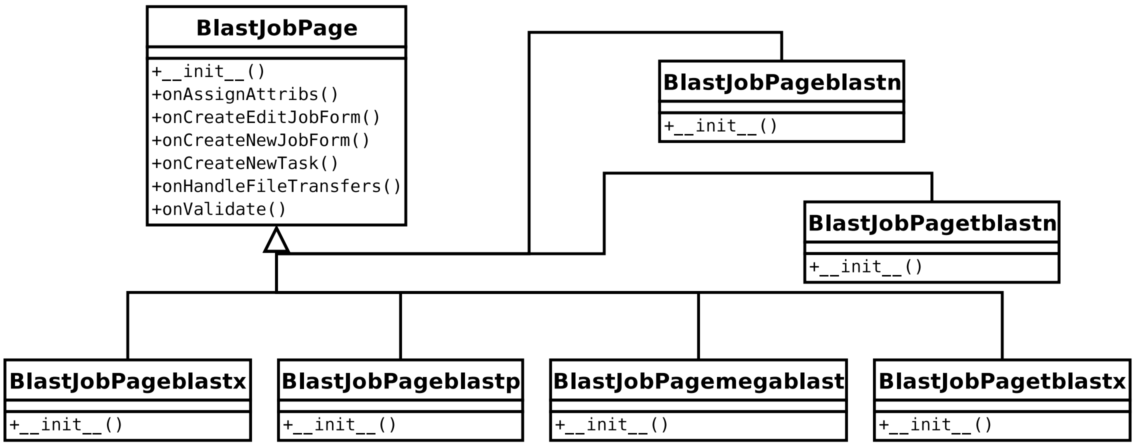

Figure 5.4.: UML diagram: BlastJobPage

The BLAST application has several variants, each of which requires slightly different input parameters. Efforts at packing all the code into one class failed and thus subclasses were written for the separate cases.

5.2.5. Ui

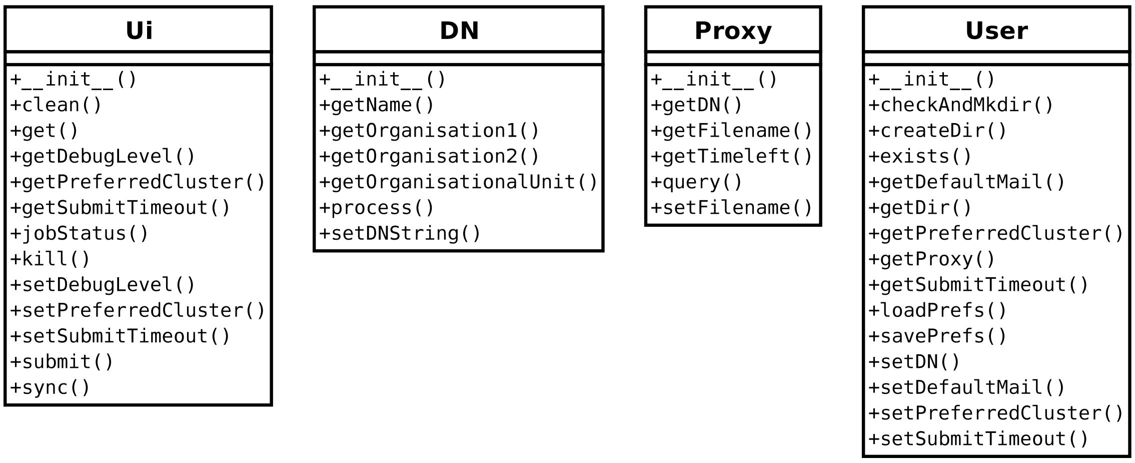

Figure 5.5.: UML diagram: Ui

Here are the classes from pyARC.py, which supply the interface to the NorduGrid/ARC middleware.

The *DN* class parses the distinguished name (ldap terminology).

The *Proxy* class handles proxy validation, it uses commands from the Globus Toolkit package (part of NorduGrid/ARC).

The *Ui* class handles all job related functions - job submission, status check, syncing job list, killing job, getting jobs etc. It uses native NorduGrid/ARC commands.

I have also included the *User* class, which is declared in Lap.py but is not unrelated to *DN*. The *User* class deals with managing data about the user (authenticated through a certificate). It creates the directory structure (using the DN of the user), where user data is stored, and manages user preferences.

5.2.6. PropertyList

PropertyList defines a common class for dealing with property lists, implemented as a dictionary.

XRSLAttributes derives from *PropertyList* and declares attributes common to all NorduGrid/ARC jobs.

TaskDescriptionFile is a general class to describe the file where a task is described in a *PropertyList*.

XRSLFile is derived from *TaskDescriptionFile* and serves to write xrsl job file, which serve as input to NorduGrid/ARC.

Form and *Table* are included here to save space but completely unrelated. Both classes handle html code generation for common tags. In fact, all html code inside forms and tables passes through one of these two classes.

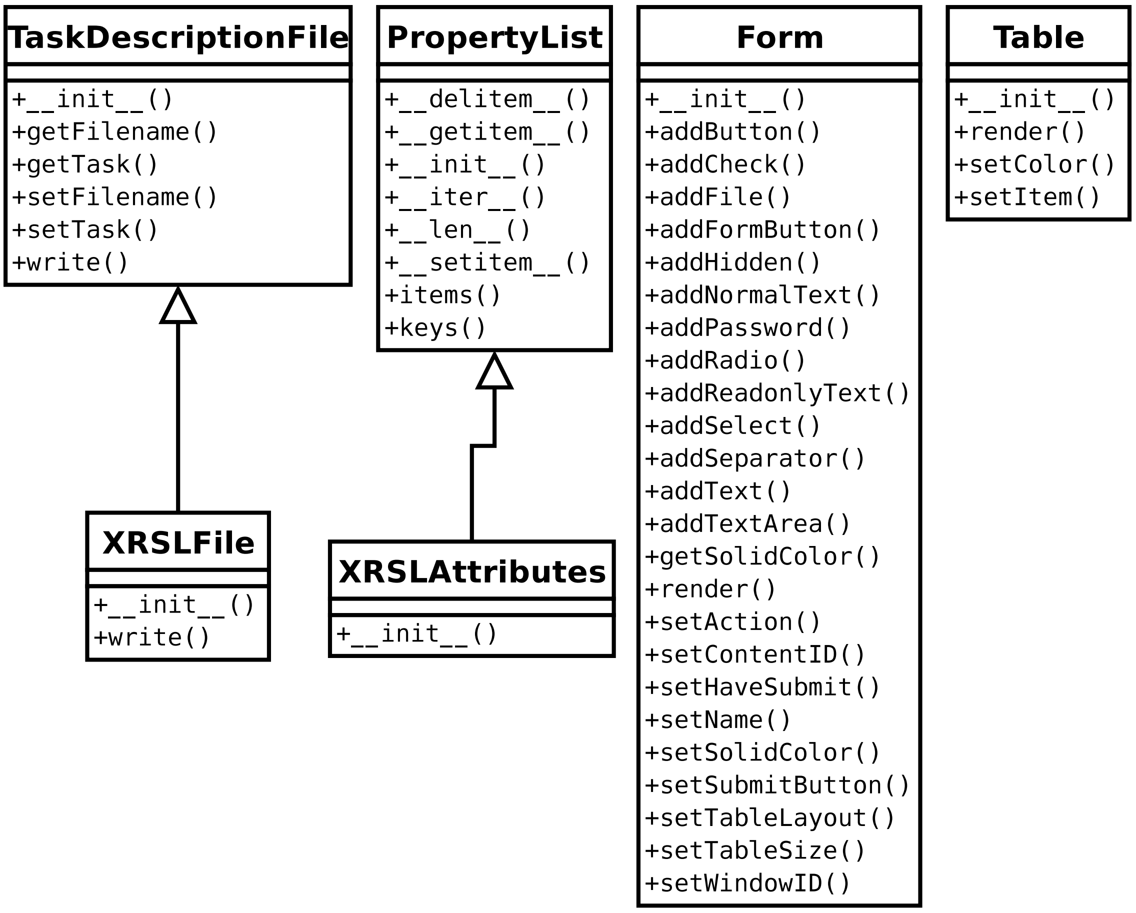

Figure 5.6.: UML diagram: PropertyList

5.2.7. Menu

The Menu* classes all serve to create the dhtml menu on the portal website. Essentially they simply menu creation in Python source instead of writing JavaScript manually.

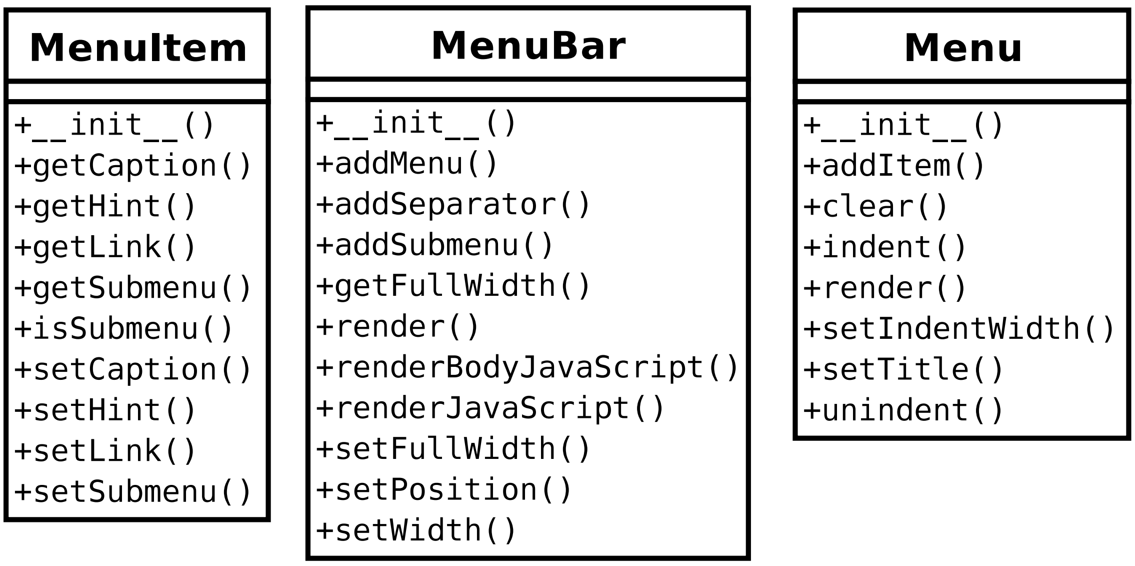

Figure 5.7.: UML diagram: Menu

5.3. Session handling

Session tracking in GRIDportal is handled with the session mechanism provided by WebKit. All pages which require a valid session are sub-classed from *SecurePage*, which implements a session verification procedure.

A session is valid for a set period of time and renewed with every user request to GRIDportal. Inactivity will eventually expire the session. The length of which it is valid is set in the WebKit configuration, by default it is 60 minutes.

5.3.1. File system hierarchy

A user of GRIDportal will normally log into the portal, create a job, submit the job and wait for it to finish. Large jobs can run for several hours so unless the user is loading pages all this time, his/her session will expire. But the user will return after the job has finished to collect results so there must be some mechanism to store data about the user, given that sessions are short lived.

Instead of relying on a database solution, GRIDportal uses a file system hierarchy to store information about the user and any jobs he/she may have created. The root path of this hierarchy depends on the WebKit configuration and is defined in conf_main.py. I refer to it here as *%root_path%*.

User data is stored according to user certificate identifier information. A user whose certificate identifier is **/O=Grid/O=NorduGrid/OU=ntnu.no/CN=Martin Matusiak** will have his/her information stored under *%root_path%/NorduGrid/ntnu.no/Martin Matusiak*. A more general form of this path, which I will use henceforth, is:

%root_path%/NorduGrid/<organization>/<user>.

5.3.2. Proxy storage

A user without a valid session is logged into GRIDportal by way of a valid user proxy file. The proxy file is stored on the server and used by GRIDportal to access NorduGrid/ARC on behalf of the user.

The proxy is stored under *%root_path%/NorduGrid/<organization>/<user>/lap_proxy*.

5.3.3. Job storage

5.3.3.1. Job specification

Jobs are created by the user through GRIDportal and subsequently stored on the server. Jobs are stored under *%root_path%/NorduGrid/<organization>/<user>/job_<jobname>*, where <jobname> is the name given by the user on the job creation form. (In practice a new job created with the same name as an old job will overwrite the old one.)

Job specifics depend on the application but the most common format consists of several special files.

job.task Common for all jobs, this is where GRIDportal stores information about the job in *pickle* (serialized) format.

job.xrsl Common for all jobs, this is the file GRIDportal generates from *job.task* as a job description used for job submission to NorduGrid/ARC.

run.sh Application specific but considered a good way of specifying the execution string to an application (as opposed of writing all of this to *job.xrsl*, which is entirely possible).

input files Depending on the application, there can be one or more other files in the job directory. These are input files from the user, required for the job. In practice they are job descriptions and/or data sets in the format the application requires them to be.

5.3.3.2. Job results

Once a job has been submitted and completed, the user will want to retrieve the results. Every job created can be ran more than once, the results of each run will be stored separately and can be accessed at any time until they are deleted by the user (or the job as a whole is deleted). A job retrieved from the grid is stored under:

%root_path%/NorduGrid/<organization>/<user>/job_<jobname>/<jobid>

<jobid> is assigned by NorduGrid/ARC to distinguish between jobs on the same grid. When the results of a job are retrieved through NorduGrid/ARC, all the files belonging to a job are stored under the above mentioned job path. A special subdirectory *gmlog* contains debugging information about the job run.

A. References

A.1. Software

- Apache HTTP server `http://httpd.apache.org/`
- BLAST `http://www.ncbi.nlm.nih.gov/BLAST/`
- HyperText HTML code generation library `http://dustman.net/andy/python/HyperText`
- Matlab `http://www.mathworks.com/`
- mpiBLAST `http://mpiblast.lanl.gov/`
- NorduGrid middleware `http://www.nordugrid.org/`
- Webware for Python `http://www.webwareforpython.org/`

A.2. Documentation

- WebKit documentation `http://www.webwareforpython.org/Webware/WebKit/Docs/index.html`
- Webware documentation `http://www.webwareforpython.org/Webware/Docs/index.html`

B. Glossary

cluster A cluster is a collection of homogeneous (ie. identical) pc-computers (nodes) connected in a local area network. Each node has its own, distinct memory and hard drive. Clusters are fairly cheap because each node is a regular desktop computer/small server in terms of computing power and is purchased mainstream at a low cost.

Clusters have a dedicated frontend computer which hosts the scheduler for the cluster and manages user accounts, access permissions etc. The cluster frontend is the only gateway to the cluster nodes. Some variant of a shared file system, often nfs, is used to distribute data for jobs across nodes in a cluster

Most importantly, a cluster is a cheap way of obtaining a lot of processing power, as compared to large shared memory machines.

grid First of all, a grid is not a cluster. A grid is a general term used to describe many connected resources which combined offer great processing power. A grid can be a network of clusters, it can also include shared memory machines. NorduGrid specifically consists of clusters.

GRIDportal The term GRIDportal can mean one of two things. It is both the *project* whose goal it is to produce the software product, and the *product* itself. In most cases I refer to GRIDportal as the software product, but on occasion the alternate definition is applied.

LunARC Application Portal (LAP) The original grid application portal software written by Jonas Lindemann. GRIDportal is a fork of the Lunarc Application Portal. Sometimes referred to as LAP.

NorduGrid The international collaboration which produces NorduGrid/ARC. Sometimes used to mean the NorduGrid network.

NorduGrid/ARC The middleware from NorduGrid. Sometimes referred to simply as middleware.

NorduGrid network The collection of grid sites which deploy NorduGrid/ARC and thus offer grid resources to the whole network.