Manual: Writing a portlet using the Rapid tool

Jos Koetsier, Srihathai Prammanee

July 16, 2008

Abstract

1 Introduction

In many scientific disciplines, the demand for computational power has increased dramatically and is likely to grow even further in the future. New techniques, such as micro array analysis in biology, generate ever larger data sets that need to be analysed. Sciences such as astronomy and physics are developing more accurate instruments, that, coupled with new and more complicated algorithms require ever larger computing resources. These new innovations and techniques mean that for many applications simple computation on a desktop computer can no longer be considered adequate.

Thankfully, the demand for more computing power is mirrored by the availability of newer and faster High Performance Computing resources. For example, the list of top 500 supercomputers in the world shows that available computing power has increased exponentially for the last fifteen years. New developments in Grid Computing enable users to access even more computing power by allowing users to submit computational tasks transparently to not one, but several, often heterogeneous, resources.

Unfortunately, high performance computing comes at the price of increased complexity. Submitting a computational task to a cluster or grid often requires issuing a set of complex commands through a command line interface. This is further complicated by the necessity of some form of security infrastructure to regulate access to often expensive resources. Users need to request certificates, download proxy certificates and understand the security implications of their actions. Accessing computing power is no longer a matter of double-clicking an application on a desktop, but more and more requires an in-depth knowledge of computer science, which not all scientists possess.

In order to alleviate this burden on the researcher, scientific applications are often wrapped in a user interface. In the grid community, the use of portals is often considered the preferred solution mainly due to its distributed nature. No complicated installation of software is necessary; users can log on to the portal from any computer, simply by using the browser supplied by their operating system. An added advantage is that security is dealt with by the portal. Most portals offer some sort of login mechanism and issue privileges using a role based access mechanism, allowing for fine-grained security to be applied.

When considering the state of the art of current portlets that allow users to submit tasks to clusters or grids, roughly two approaches can be identified. First there is the *generic* portlet. This type of portlet, in effect, wraps a command line job submission. For every parameter a text box or drop-down list is supplied which the user needs to fill out. These portlets allow a wide variety of computational tasks to be submitted, but at the cost of increased complexity. This type of portlet is far from ideal, especially because we set out to shield the user from all this complexity in the first place.

The other approach is to write a *custom* portal application for each computational task. We consider this type of portlet preferable to the generic portlet of the first approach, because of its simplicity. Any user can access the power of large compute clusters without the indepth knowledge required to perform a command-line job submission. However, this approach does come at the cost of increased development time spent on programming the portlets themselves. It is often the case that there is simply not enough time and money available to write a portlet for each new project. There is also the additional problem of maintainability. When the portlet is finished, it requires an expert to maintain it, fix bugs and perform upgrades.

The Rapid project is an attempt to address these issues. We propose an *automatic job submission portlet generation system*. This system allows an expert to specify information about a job submission portlet in an XML file. This can be information about file transfers, options to display to the user and the available compute clusters as well as the security information required to access them. The Rapid system takes this XML file and translates it into a fully working JSR 168 compliant job submission portlet.

This approach gives us the benefits of both portlet types we discussed earlier. On the one hand, users get a job submission portlet that shields them from any complexity and on the other hand the time and effort to write and maintain a generic portlet is greatly reduced. Generating a portlet in this way does reduce flexibility somewhat however; not each portlet that can be written by hand can be generated using Rapid. However, we think that using Rapid will be able to meet most users' needs and will allow scientists to make better use of the available computing power.

2 The Rapid XML Document

The Rapid XML Document is central to generating a new job submission portlet. It contains information about the markup of the portlet, the available grid and HPC middleware, file servers and security information. Using this one document, the RAPID engine is able to generate a new job submission portlet without the need for Java programming.

The `<rapid>` element denotes the root of the Rapid XML document and contains two namespaces: the *rapid* namespace: *http://www.nesc.ac.uk/Rapid* and the *XHTML* namespace: *http://www.w3.org/1999/xhtml*.

Element <rapid>

Root element of the Rapid XML document.

Child Elements

- `<condor>` defines a new Condor job submission instance.
- `<sge>` defines a new Sub Grid Engine job submission instance.
- `<gridsam>` defines a new GridSAM job submission instance.
- `<fork>` defines a new Fork submission instance.
- `<local>` defines a local file system.
- `<ftp>` defines an (s)ftp file system.
- `<http>` defines an http file system.
- `<gsiftp>` defines the gsiftp file system.
- `<initialise>` initialises a new job.
- `<page>` defines the user interface.

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<rapid xmlns="http://www.nesc.ac.uk/Rapid"
       xmlns:x="http://www.w3.org/1999/xhtml">
   <!-- Contents go here -->
</rapid>
```

The child elements of the `<rapid>` element are shown graphically in Figure 1.

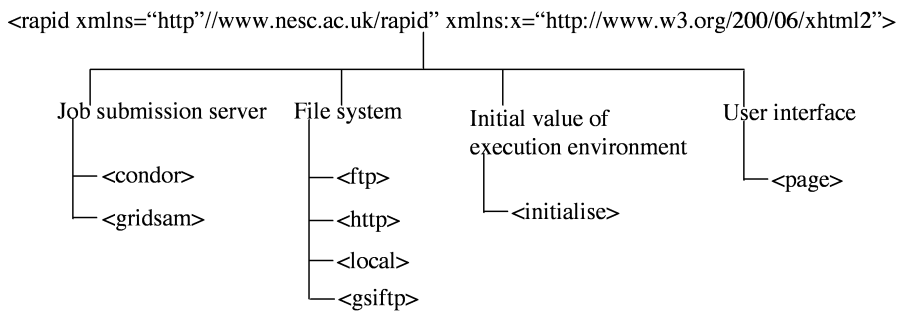

Figure 1: The structure of Rapid XML document

2.1 Job submission servers

A job submission server is the compute resource to which a compute job is submitted. In the *rapid* namespace, the `<gridsam>`, `<condor>`, `<sge>` and `<fork>` elements support GridSAM, Condor, Sun Grid Engine and simple forking a new process, respectively. They are the immediate child elements of the `<rapid>` root element. The elements used for job submission server are illustrated graphically in Figure 2

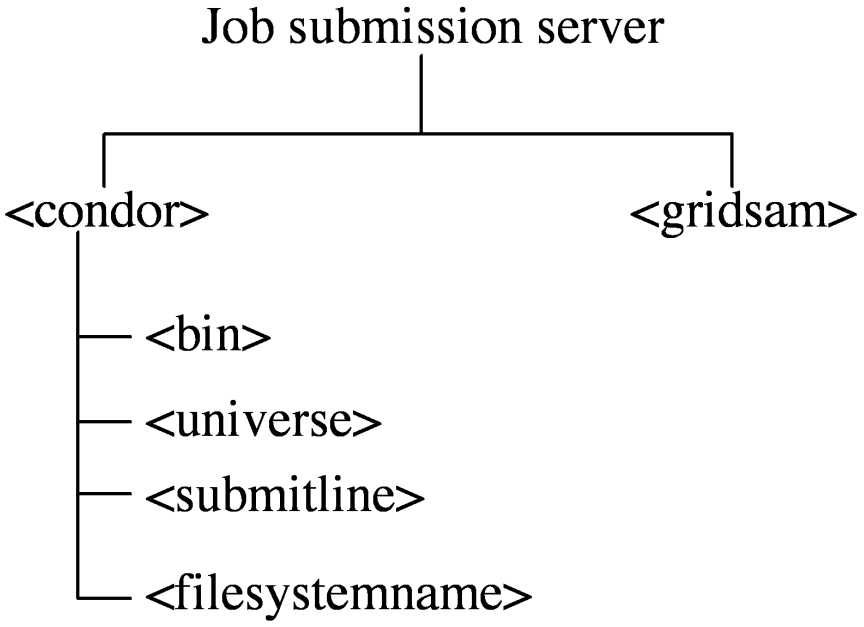

Figure 2: The overall elements defining a job submission server

2.1.1 Gridsam engine

The `<gridsam>` element has one attribute 'name', which uniquely identifies the name of Gridsam instance. The full URL indicating the location of GridSAM is specified in the body of this element.

Element <gridsam>

Attributes

- name : a unique identifier

Example

```
<gridsam name="Manchester">
     https://omii-server:18443/manchester/services/gridsam?wsdl
</gridsam>
```

2.1.2 Condor cluster

The `<condor>` element allows Rapid to submit to a Condor cluster.

Element <condor>

Attributes

- `name`: a unique identifier

Child Elements

- `<bin>` element specifies the location of Condor execution files 'condor_submit' and 'condor_q'. (optional)
- `<condorconfig>` path to the condor_config file. (optional)
- `<universe>` defines the runtime environment under which Condor cluster should execute a job. (optional)
- `<filesystemname>` indicates through which file system (specified in 2.2) the condor submit host is accessed. Supports SSH and Local.
- `<submitline>` defines an additional lines in a Condor submit file. (zero or more)

Example

```
<condor name="condor">
   <bin>/home/condor/condor/bin</bin>
   <universe>vanilla</universe>
   <submitline>priority = 10</submitline>
   <submitline>image_size = 20</submitline>
   <filesystemname>CondorFS</filesystemname>
   <condorconfig>/etc/condor/condor_config</condorconfig>
</condor>
```

2.1.3 Sun Grid Engine

The `<sge>` element allows Rapid to submit to a Sun Grid Engine cluster.

Element <sge>

Attributes

- `name`: a unique identifier

Child Elements

- `<root>`: path to SGE_ROOT. (optional).
- `<filesystemname>`: indicates through which file system (specified in 2.2) the Sun Grid Engine submit host is accessed. Supports SSH and Local.
- `<option>`: lines to add to a SGE submit script. (zero or more)

Example

```
<sge name="sge">
   <root>/home/sge/sge</bin>
   <filesystemname>LocalFS</filesystemname>
   <option>#$ -M admin@nesc.ac.uk</option>
<sge>
```

2.1.4 Fork

The `<fork>` element configures a simple 'fork' submission resource where the job is simply run as a new process.

The <fork> Element

Attributes

- `name`: a unique identifier

Child Elements

- The `<filesystemname>` indicates through which file system (specified in 2.2) the process is to be run. Supports SSH and Local.

Example

```
<fork name="fork">
    <filesystemname>LocalFS</filesystemname>
<\fork>
```

2.2 File systems

File systems are defined using four elements:. `<http>`, `<ftp>`, `<local>` and `<gsiftp>`. These elements are the direct child nodes of `<rapid>` and require a unique 'name' attribute used as a reference throughout the document. All filesystem tags contain a `<url>` child element of which the body specifies the complete URL of the file system. Some elements may encompass additional child nodes, depending on authorisation and authentication. Those file system elements are detailed in the following sections. Figure 3 shows the elements defining file systems.

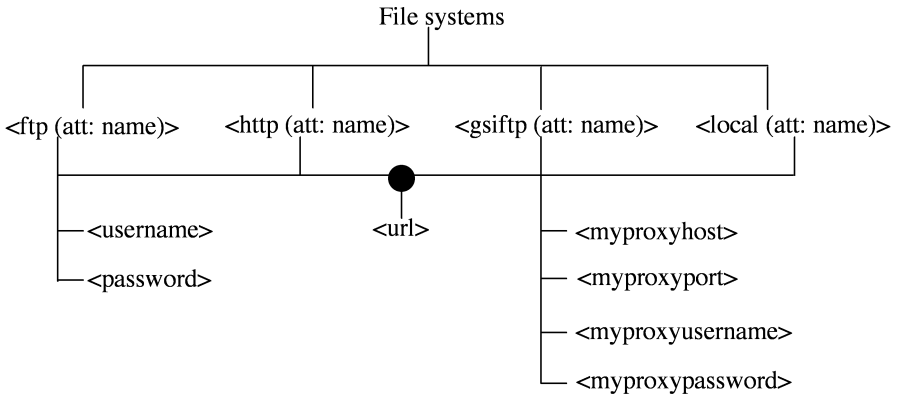

Figure 3: The overall elements defining a file system

2.2.1 HTTP

The `<http>` element is used for both *http* and *https* servers. Neither of them requires authentication.

Element <http>

Attributes

- `name`: a unique identifier

Child Elements

- `<url>` URL of the file system

Example

```
<http name="http">
   <url>http://www.nesc.ac.uk/index.html</url>
</http>
<http name="http2">
   <url>https://www.omii.ac.uk/index.html</url>
</http>
```

2.2.2 Local file system

The `<local>` element allows access to a local file system. Similar to `<http>`, there is no authentication. This file system refers to the Portal file system and not the users' file system. When using GridSAM, extra care has to be taken as during the job submission phase a 'local' filesystem refers to the filesystem GridSAM resides on.

Element <local>

Attributes

- `name`: a unique identifier

Child Elements

- `<url>` URL of the file system

Example

```
<local name="Local File System">
     <url>file:///home/portaluser</url>
</local>
```

2.2.3 FTP

The `<ftp>` element is used for both *ftp* and *sftp* servers and requires authentication using a username and password.

Element <ftp>

Attributes

- `name`: a unique identifier

Child Elements

- `<url>` URL of the file system
- `<username>` username
- `<password>` password

Example

```
<ftp name="myftpserver">
   <url>sftp://host.university.ac.uk/full/path/</url>
   <username>myname</username>
   <password>mypassword</password>
</ftp>
```

2.2.4 GSIFTP

The `<gsiftp>` element describes a Grid FTP server. For authentication, a valid X509 proxy certificate must be obtained through a MyProxy server.

Element <gsiftp>

Attributes

- `name`: a unique identifier

Child Elements

- `<url>` URL of the file system
- `<myproxyhost>` hostname of the myproxy server
- `<myproxyport>` port through which to access the myproxy server
- `<myproxyusername>` username
- `<myproxypassword>` password

Example

```
<gsiftp name="mygisftpserver">
   <url>gsifpt://host.university.ac.uk/full/path/</url>
   <myproxyhost>myproxy.university.ac.uk</myproxhost>
   <myproxyport>7512</myproxyport>
   <myproxyusername>myusername</myproxyusername>
   <myproxypassword>secret</myproxypassword>
</gsiftp>
```

2.3 Initialising a job

The `<initialise>` element is used to give each new job its initial values. It can contain four child elements. The `<datastage>` element involves the file staging details and the `<posix>` element is used to set parameters of a job according to the POSIX model. The compute resource to use is set in the `<submitto>` element and, finally, the `<variable>` element can be used to set variables.

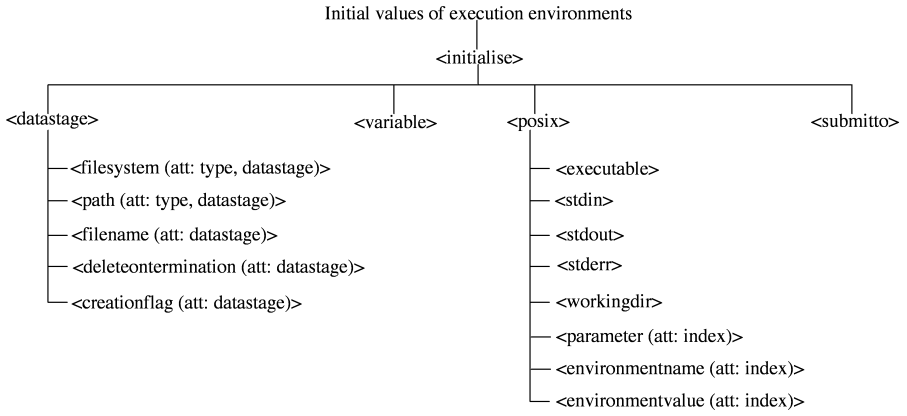

Figure 4: The elements defining the initial values of execution environments

2.3.1 Data types

For each value in the `<initialise>` section, a type has to be declared. There are currently three types, specified by the elements `<single>, <array>` and `<uuid>`. The `<single>` element is used to indicate simple, single values and the `<array>` element is used to specified multiple values. The `<uuid>` element generates a new Universally Unique IDentifyfier (UUID) for each new job that is generated. The values themselves are specified in the body of `<value>` elements which can appear as direct children of `<single>` or `<array>`.

If multiple values in an array are used, multiple jobs will be created at the job submission stage. If multiple array types are used, the maximum number of values is used to determine the number of job to submit, where the first job will use the first value from each array, the second job will use the second value and so on.

Elements <single>, <array>

Child Elements

- `<value>` specifies a value (one or more if parent is `<array>`, one for `<single>`).
- `<min>` sets the maximum possible value. Implies `<value>` is a number.
- `<max>` sets the minimum possible value. Implies `<value>` is a number.
- `<regexp>` sets a regular expression the `<value>` elements must conform to.

- `<errormessage>` error message to display if `<value>` elements contain invalid values.

Example

```
<array>
   <value>parameter1</value>
   <value>parameter2</value>
   <value>parameter3</value>
   <regexp>[0-9]+</regexp>
   <errormessage>Please enter a whole, positive number</errormessage
</array>
```

Elements <uuid>

Automatically creates a new Universally Unique IDentifyer (UUID) when a new job is created and is stored internally as a 'single' datatype.

Example

```
<uuid/>
```

2.3.2 Data staging

Data staging in the Rapid system follows the staging model as defined in JSDL. One *data stage instance* can contain a *source* and *target* transfer. The *source* file transfer copies a file or directory into the job execution engine and is performed before running a job. Once the job execution has finished, the *target* file transfer moves a file or directory out of the job execution engine.

Rapid defines three additional parameters for a data stage instance. First, a filename uniquely identifies the file called in the job execution engine. Second, a creation flag determines how the file will be created in the job execution engine and finally, a deleteontermination flag indicates whether the file will be removed from the execution host once the job is completed.

Element <datastage>

The `<datastage>` element simply groups any datastage elements and has no attributes.

Child Elements

- `<filesystem>` determines the file system used in either the source or target file staging.
- `<path>` determines the full path of either 'source' or 'target' file stage.
- `<filename>` refers to which file or directory will be staged into the job execution file system.
- `<deleteontermination>` determines whether the file will be deleted from the job execution engine once the job has finished. (optional)

- `<creationflag>` refers to the creationflag of the file. Its value can be either APPEND, OVERWRITE, or DONTOVERWRITE. (optional)

Example

```
<datastage>
   <filesystem type="source" datastage="executable_in">
      <single><value> myftpserver </value></single>
    </filesystem>
    <path type="source" datastage="executable_in">
       <single><value>xxx/yyy</value></single>
    </path>
    <filename datastage="executable_in">
       <single><value>.</value></single>
    </filename>
    <deleteontermination datastage="executable_in">
       <single><value>true</value></single>
     </deleteontermination>
    <creationflag datastage="executable_in">
       <single><value>OVERWRITE</value></single>
     </creationflag>
</datastage>
```

Element <filesystem>

Attributes

- `datastage:` unique identifier, groups datastage elements together
- `type:` either 'source' for a source transfer or 'target' for a target transfer

Element <path>

Attributes

- `datastage:` unique identifier, groups datastage elements together
- `type:` either 'source' for a source transfer or 'target' for a target transfer

Element <filename>

Attributes

- `datastage:` unique identifier, groups datastage elements together

Element <deleteontermination>

Attributes

- `datastage:` unique identifier, groups datastage elements together

Child Elements

- `<single>` containing either 'true' or 'false'
- `<array>` containing either 'true' or 'false'

2.3.3 Posix

Job execution in Rapid follows the Unix POSIX model and all its parameters are grouped under the `<posix>` element.

Element <posix>

Child Elements

- `<executable>` denotes the name of the executable to run.
- `<stdin>,<stdout>,` and `<stderr>` elements refer to the standard input, output and error, respectively.
- `<workingdir>` element specifies the directory running the executable.
- `<parameter>`
- `<environmentname>`
- `<environmentvalue>`

Example

```
<initialise>       :
   <posix>
      <executable>
         <single><value>./java</value></single>
      </executable>
      <environmentname index="0">
          <single><value>CLASSPATH</value></single>
      </environmentname>
      <environmentvalue index="0">
          <single><value>.:aaa:bbb/ccc/dddd.jar</value></single>
      </environmentvalue>
      <workingdir>
          <single><value>.</value></single>
      </workingdir>
      <parameter index="0">
          <single><value>para0</value></single>
      </parameter>
      <parameter index="1">
          <array>
              <value>para11</value>
              <value>para12</value>
          </array>
      </parameter>
  </posix>
       :
 </initialise>
```

Element <parameter>

Parameter of the executable.

Attributes

- `index:` index of the parameter, starting at 0.

Example

```
<initialise>
   <parameter index="0">
      <single><value>-al</value></single>
   </parameter>
</initialise>
```

Element <environmentname>

Shell environment value NAME=VALUE. This element refers to the NAME.

Attributes

- `index:` index of the environment value, starting at 0.

Example

```
<initialise>
   <environmentname index="1">
      <single><value>CLASSPATH</value></single>
   </environmentname>
</initialise>
```

Element <environmentvalue>

Shell environment value NAME=VALUE. This element refers to the VALUE

Attributes

- `index:` index of the environment value, starting at 0.

Example

```
<initialise>
   <environmentvalue index="1">
      <single><value>/opt/libs/lib.jar</value></single>
   </environmentname>
</initialise>
```

2.3.4 Variables

Rapid allows variables to be declared under the `<variable>` element, where the 'name' attribute uniquely identifies the variable name. The value of a variable can referred to by using the format: `$(VARIABLE)` .

Element <variable>

Variable to set.

Attributes

- `name`: Name of the variable.

Example

```
<initialise>
   <variable name="myVar">
      <uuid/>
   </variable>
</initialise>

<page>
   The value of myVar is $(myVar)
</page>
```

2.3.5 Execution node

By default, a job is submitted to the execution node specified under the `<submitto>` element. The value of this element refers to a submission engine, previously defined in section 2.1.

Element <submitto>

Example

```
<initialise>
   <submitto>
      <single><value>Manchester</value></single>
   </submitto>          :
</initialise>
```

2.4 Pages

The user interface (UI) is defined in a set of `<page>` elements, that are immediate children of the `<rapid>` root element. When a Rapid portlet starts up, the view associated with the first `<page>` element that is defined within the document is loaded up and displayed. The `<navigate>` and `<submit>` provide the means of navigating between the views.

Element <page>

A page with markup and rapid user interface elements.

Attributes

- `name`: unique name of the page.

Child Elements

- `<datastage>`
- `<posix>`

- `<fileupload>`
- `<submitto>`
- `<navigate>`
- `<variable>`
- `<joblist>`
- `<setjob>`
- elements from the XHTML namespace

Example

```
<page name="page1">
  <!-- page layout is defined here -->
</page>
```

Child elements of the `<page>` can be taken from both the *xhtml* and the *rapid* namespace. The markup of the user interface is written using standard XHTML. The logic of the portlet as well as user input is handled by elements from the *rapid* namespace.

As shown, the `<page>` element shares a number of child elements with the `<initialise>` element. However, the usage of these elements is different. Whereas the `<initialise>` element was used to give parameters in a job a fixed value, the children of the `<page>` element are used to either output a value in a job or allow the user to alter a value by specifying one of the user interface elements described in section 2.4.1.

Finally, the `<joblist>` and `<setjob>` elements can be used to display and monitor jobs.

All children of the page element are shown in a tree diagram Figure 5

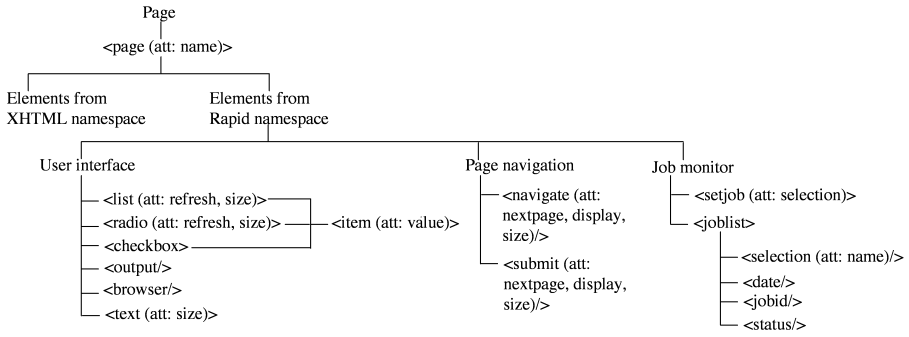

Figure 5: The elements describing a portlet page

2.4.1 User interface elements

Rapid handles input and output of a value in a job by specifying a user interface element. These are currently `<list>`, `<checkbox>`, `<radio>`, `<browser>`, `<text>` and `<output>`. Depending on the type selected in the `<initialise>` section, certain user interface elements cannot be used. For instance, a checkbox cannot be used for a `<single>` type as it would allow a user to input more than one value.

Element <item>

Adds a choice to a `<list>`, `<radio>`, or `<checkbox>`

Attributes

- `value:` The value to set into the job.

Child Elements

- `<radio>` and `<checkbox>` can contain elements from the XHTML namespace.

Example

```
<checkbox refresh="true" size="1">
   <item value="itemone">select item <x:h1>one</x:h1></item>
   <item value="itemtwo">select item <x:h1>two</x:h1></item>
   <item value="itemthree">select item <x:h1>three</x:h1></item>
<\checkbox>
```

Element <list> and <radio>

Shows a list of items, a dropdown list or a set of radio buttons.

Attributes

- `refresh:` if 'true', the page will be reloaded if a selection is made.
- `size:<list>` only. Length of the list. If the length is 1, this user interface element will be a dropdown list.

Child Elements

- `<item>` (one or more)

Example

```
<list refresh="true" size="1">
   <item value="itemone">select item one</item>
   <item value="itemtwo">select item two</item>
   <item value="itemthree">select item three</item>
<\list>
```

Element <checkbox>

Adds one or more checkboxes. Can only be used if the value set in the `<initialise>` section is of type `<array>`

Attributes

- `refresh:` if 'true', the page will be reloaded if a selection is made.

Child Elements

- `<item>` (one or more)

Example

```
<checkbox refresh="true" size="1">
   <item value="itemone">select item <x:h1>one</x:h1></item>
   <item value="itemtwo">select item <x:h1>two</x:h1></item>
   <item value="itemthree">select item <x:h1>three</x:h1></item>
<\checkbox>
```

Element <text>

Adds a text box.

Attributes

- `size:` Size of the text box.

Example

```
<text size="10"/>
```

Element <browser>

Displays a file browser, allowing the user to navigate and select a file or directory. Can only be used as a child of the `<path>` element.

Attributes

- `size:` size of the file browser.

Example

```
<path name="mypath">
   <browser/>
</path>
```

Element <fileupload>

Allows a user to upload a file to the portal's file system. The file upload takes place immediately when updating the page. This file can be staged futher to an execution host by referring to it in a `<datastage>` section. Contains the destination filename in the filesystem of the portal as the body.

Attributes

- `size`: size of the file upload input element (optional)
- `name`: unique name identifying this file upload
- `action`: either 'input' or 'output'. Set to 'output' to display the filename that was uploaded. (optional - default 'input')

Example

```
<fileupload name="myupload" size="10">
   /tmp/my-file-$(uuid).jpg
</fileupload>

<fileupload name="myupload" action="output"/>
```

Element <output>

Displays a value. Can replace values by strings, which can be useful if the values are stored in the job as codes.

Attributes

- `default`: String to display if the job value has not been set

Child Elements

- `<replace>` Replace the value by another string. Specify the value in a 'search' attribute and the replacement in the body.
- `<pre>` Used for `<array>` type. Displays a string before each value in the `<array>`. Can contain XHTML elements.
- `<post>` Used for `<array>` type. Displays a string after each value in the `<array>`. Can contain XHTML elements.

Example

```
<output>
   <replace search="item1">this is item one</replace>
   <replace search="item2">this is item two</replace>
   <replace search="item3">this is item three</replace>
   <pre>before</pre>
   <post>after</post>
<\output>
```

2.4.2 Page navigation

Different pages are navigated through an event button created by either `<navigate>` or `<submit>`. The first purely deals with page navigation whereas the second integrates the capability of submitting a compute job.

Element <navigate> and <submit>

Navigates between pages. `<submit>` also submits a job

Attributes

- `size:` size of the button.
- `nextpage:` next page to load.
- `display:` string to display on the button.

Example

```
<navigate size="12" nextpage="monitor" display="Monitor Job"/>
```

2.4.3 Job monitor

Job monitoring can be implemented using the `<joblist>` and the `<setjob>` elements. The `<joblist>` element iterates through all submitted jobs and allows the user to select a job by clicking on a radiobutton. The `<setjob>` refers to the selection made in the `<joblist>` element and allows a more detailed view of a particular job to be presented.

Element <joblist> and <setjob>

`<joblist>`iterates through all previously submitted jobs. `<setjob>` sets the job selected a `<selection>` element, which is a child element of `<joblist>`.

The elements `<datastage>, <posix>, <submitto>` and `<variable>` elements have all been defined previously. However, if they appear as child elements of `<joblist>` and `<setjob>,` they can only be used to display values of jobs that have been submitted and therefore do not contain input or output child elements.

Attributes

- `selection:<setjob>` only. Sets selected job..

Child Elements

- `<datastage>`
- `<posix>`
- `<submitto>`
- `<variable>`
- `<jobid>`
- `<selection>` Used only in conjunction with `<joblist>`.
- `<date>`
- elements from the XHTML namespace

Element <jobid>

When a job has been submitted it is given a Job ID in the form of a UUID. This Job ID can be displayed using this tag.

Element <status>

The status of a submitted job.

Element <selection>

This tag displays a radiobutton that can be used to select a job.

.

Attributes

- `selection:` unique name identifying the selection

Element <date>

The date the job has been submitted.

Example

```
<joblist>
   <x:table border="1" width="800"> <x:tr> <x:td>
      <selection name="mySelection"/>
   </x:td> <x:td>
       <date/>
   </x:td> <x:td>
       <jobid/>
   </x:td> <x:td>
      <status/>
   </x:td> </x:tr>
   </x:table>
</joblist>

<setjob selection="mySelection">
   <x:h1>executable:
      <posix>
         <executable/>
      </posix>
   </x:h1>
</setjob>
```

3 Generating the portlet

In this section we describe the steps necessary to generate and install the portlet.

3.1 Infrastructure

This section reviews the necessary infrastructure for the Rapid system.

3.1.1 GridSAM

If job submissions through GridSAM are required, we need to set the relevant security options to enable authentication between the portal and the GridSAM instance. This is simply done by copying the file 'crypto.properties' as defined within the OMII GridSAM container into the 'configuration' directory of GridSAM.

3.1.2 Portal

Because RAPID generates portlets, we require a JSR 168 compliant portlet container to deploy our portlet into. Our build scripts are currently written specifically for the GridSphere portal (version 3.0 and higher), but as we use the WAR format for the portlets it is relatively straightforward to deploy them in other portlet containers.

3.1.3 Apache Maven and Ant

Before installing a Rapid portlet, Apache Maven (mvn) and Apache Ant are required. Both packages can be downloaded from the Apache website[1].. Rapid assumes that the 'ant' and 'mvn' commands can be accessed through the current search path.

3.2 Rapid portlet generation and installation

In order to install a new rapid portlet, download and unpack the RapidPortlet tarball. Within the RapidPortlet directory, all the configuration options are set in the 'configuration' directory. The file called 'crypto.properties' is the same as used by the OMII client and is only necessary if job submissions to GridSAM are used.

Another file in the configuration directory, called rapid.properties, sets properties such as the name and title of the portlet that is to be generated. Finally, the configuration directory should contain a file called 'rapid.xml', which specifies the content of the new portlet,

The portlet is generated by issuing the `mvn package` command.from the RapidPortlet directory. The first time this command is executed, a number of jars will be downloaded into the `~/.m2` directory, which can take a long time. If the command is issued again, the downloaded jars will be used and the process will be much quicker. When this process is completed, a 'portlet' directory containing a small ant install script and a 'war' file is generated.

The final portlet will be packaged as a 'war' file into the 'portlet' directory together with a small ant install script. The install script works for the GridSphere portal (version 3.0 and higher) only and assumes the `CATALINA_HOME` environment variable has been set. The portlet is installed by entering the `portlet` directory and issuing the `ant deploy` command.

An in-depth example of Rapid portlet installation can be found in Tutorial: Rapid-based Portlet Installation.

[1] http://www.apache.org/