A basic tutorial for Rapid

Jos Koetsier and Jano van Hemert

for version 2.0—May 19, 2011

http://research.nesc.ac.uk/rapid/

In this tutorial, we will learn how to build portals for computational science applications using Rapid. Rapid is designed to quickly prototype, develop and deploy portlets that become part of a web portal. Its main purpose is to allow rapid development of user interfaces for dedicated tasks and applications that need access to remote compute resources. It has many features to make it easy to create these portlets as it can connect to different compute resources, handle remote file systems and deal with all aspects of submitting compute jobs.

The tutorial expects students to be comfortable with typing commands on a command line (terminal), downloading archives from the web and extracting these, and with basic editing of text files. We begin the tutorial with installation instructions of several software components and then start to develop a simple portlet. We will refine this portlet gradually to incorporate more features provided by Rapid.

Preparation of the development system

The first task is to set up the development environment. Rapid depends on the following software packages.

- Java (version 1.5+)
- Ant (version 1.7.1+)
- A JSR-168-compliant portal container, such as Liferay Portal [1]

[1]

To install Liferay simply go into its directory structure and execute the `startup.sh` script found in the `bin/` directory of the Apache Tomcat server. Wait for a few seconds before testing the installation worked by point any web browser at http://localhost:8080.

Task 1: Install where necessary Java and Ant

Task 2: Download, unpack and install Liferay

Task 3: Download the rapidportlet software from http://research.nesc.ac.uk/rapid/ and unpack it

Configuration files

After unpacking the rapid portlet distribution, a number of directories and files will be created in the `rapidportlet` directory. All configuration information will be put into the `configuration` directory. The two important files are called `rapid.xml` and `rapid.properties`.

We first customise `rapid.properties`. Important properties in this configuration file are `rapid.portletname` and `liferay.category`; these determine the name of the portlet and in which category they will be placed.

Task 4: Edit the `rapid.properties` file in your text editor of choice and change the portlet name and category.

The second file is the main configuration file that we will continue to edit throughout this tutorial. The file `rapid.xml` contains

```
rapid.portletname = name
rapid.title = a title
rapid.title.short = shorttitle
rapid.keywords = rapid portlet keyword
liferay.category = rapid
```

Figure 1: An example of the `rapid.properties` file

a definition of the resources, the logic of the task and XHTML markup that defines the user interface of the portlet. Figure shows an outline of a simple portlet. It contains one page, some XHTML and two buttons, one to refresh the page and one that refers to a non-existing page.

```
<?xml version="1.0" encoding="UTF-8"?>
<rapid xmlns="http://www.nesc.ac.uk/rapid"
xmlns:x="http://www.w3.org/1999/xhtml">
   <page name="pageone">
      <x:h1>First Page</x:h1>
      <button display="refresh">
         <navigate nextpage="pageone"/>
      </button>
      <button display="next page">
         <navigate nextpage="pagetwo"/>
      </button>
    </page>
    ...
    ...
</rapid>
```

Figure 2: A very simple outline of a `rapid.xml` file

Task 5: Add the (now missing) second page with some XHTML and a button that refers back to the first page.

Deploying Rapid portlets

The configuration is now ready to be translated into a portlet. By calling Rapid, we translate the XML into a WAR file that can be deployed directly in Liferay.

Task 6: Call this file `rapid.xml` and copy it into the `configuration` directory of Rapid.

Task 7: Issue the command `ant` from the `rapidportlet` directory.

Task 8: Copy the resulting WAR file from the `portlet` directory into the `deploy` directory of the Liferay Portal.

Task 9: Wait a few seconds, log into the portal and add the portlet (see Figure for instructions).

Task 10: Declare and initialise variables to contain the Username, Password and Filename.

We now add a file browser to the portlet that will enable the user to select a file from a remote SFTP server. The file browser will display the full path of the file a user selects.

Rapid uses variables to maintain state. Before a variable can be used it has to be initialised and given a default value. For this exercise we require three variables: two variables are needed to authenticate to the SFTP server, one for the username and one for the password. A third variable is used to store the full path of the file the user selected.

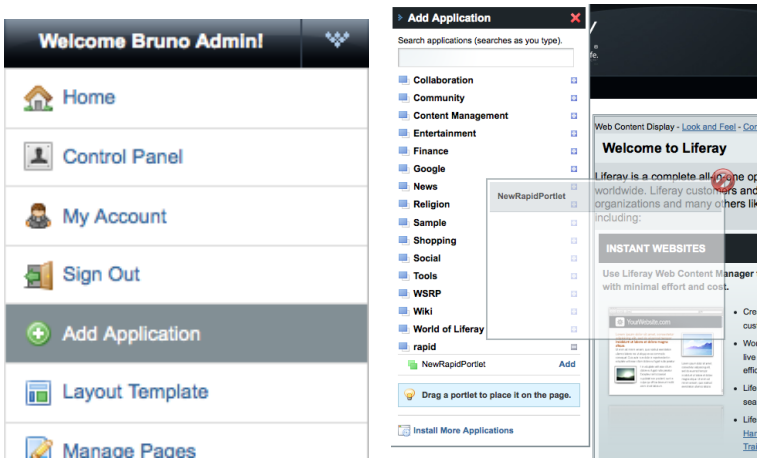

Figure 3: To add a new portlet to Liferay first click on "Add application" then find your named portlet under your chosen category (default: rapid)

```
<initialise>
   <variable name="usernameVar">
      <single>
         <value>username</value>
      </single>
   </variable>
   <variable name="passwordVar">
      <single>
         <value/>
      </single>
   </variable>
   ...
   ...
</initialise>
```

Figure 4: Initialising variables in Rapid's main configuration file

Transferring files

Next we define the SFTP server that the user will be able to browse. The SFTP server requires a unique name that will be used to refer to it throughout the `rapid.xml` file. Furthermore, we have to specify its URL, a username and a password. Here, we use the values of the variables we specified previously for the username and password. The notation Rapid uses for variable substitution is a dollar sign ($) followed by the name of the variable in brackets. To retrieve the value of the username we use `$(usernameVar)` and for the password we use `$(passwordVar)`. This notation can be used throughout all parts of a Rapid XML file.

```
<ssh name="mysftpserver">
   <url>sftp://my.host/path</url>
   <username>$(usernameVar)</username>
   <password>$(passwordVar)</password>
</ssh>
```

Figure 5: Definition of a file system resource reachable via SFTP with a username and password

Of course simply defining and displaying the values of variables is not very interesting. We need to allow the user to manipulate these variables. Rapid can use different methods to enter a value, such as radio buttons, drop-down lists and text boxes. In this exercise we will add two text boxes, one for the username and one for the password.

Task 11: Add a SFTP file server definition to the initialisation section of `rapid.xml`.

```
<page name="...">
   ...
   <variable name="usernameVar">
      <text cols="20"/>
   </variable>
   ...
</page>
```

Figure 6: Interface elements to allow manipulation of variables in a portlet

We add a file browser to allow a user to select a file. We also want to display to a user the path of the file they choose. Adding a file browser is similar to adding a text box as we did in the previous section. The file browser is simply another way for the user to manipulate the value of a variable. The value of the file chosen by the user can displayed simply by using the `$(filenameVar)` notation.

Task 12: Add a textbox to allow a user to enter the username

Task 13: Add a textbox to allow a user to enter the password. Use the `password="true"` attribute to stop characters appearing on the screen.

Task 14: Add a file browser.

Task 15: Display the full path of the file chosen by the user.

Task 16: Deploy the new portlet, login to the sftp server and select a file.

Example portlet: word count

We will create a portlet to submit a very simple computational job: counting the words in a text file. So far, we enabled a user to select a file. Here, we will extend the portlet to use the Unix word count

```
<page name="...">
   ...
   <variable name="filenameVar">
      <browser filesystem="mysftpserver"/>
   </variable>
   ...
</page>
```

Figure 7: Adding a file browser to a Rapid portlet

program (normally installed under `/usr/bin/wc`) to determine how many characters, words and lines are in a file selected by a user.

Figure shows the steps of the whole process we enable via the portlet. The user will be able to browse and select a file from a file server (Step 1). The file is copied to a another computer (Step 2) where the computation takes place (Step 3)—here counting characters, words and lines in a text file. The result will be copied into a directory that is available externally via HTTP; this allows the portlet to access it remotely and display it directly in the portal (Step 4).

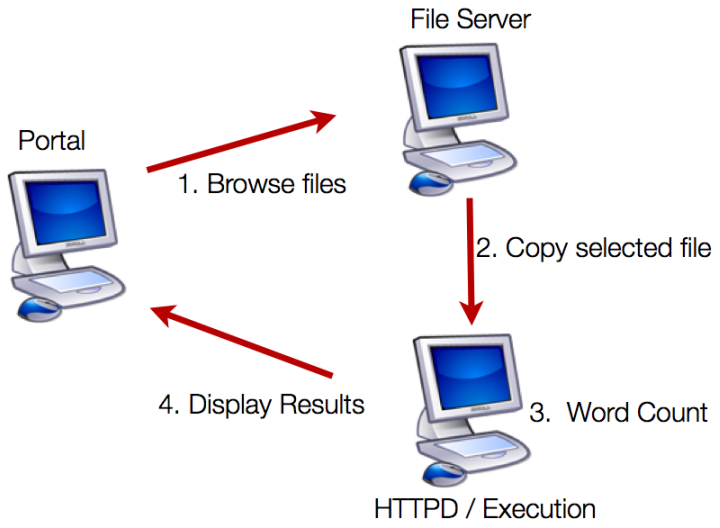

Figure 8: Browsing a remote file system, selecting and copying a file, then executing Unix' word count program and displaying the result directly in the portal

In the `initialise` section of the portlet we have to add a simple outline of the job. We have to specify the location of the executable and the parameters it uses. The word count program (in its simplest form) takes one parameter: the location of the file. The output of the program is written to `stdout` and can be redirected to a file using the `<stdout>` element. The output file will be written to a directory that is available remotely via a web server; this allows the portal to fetch and display the results.

Task 17: Insert the appropriate compute job definition in the initialisation of your `rapid.xml`

The input files to a job often reside on another file system and need to be transferred to the computer that will perform the computing step. Here, the computer responsible for executing the word

```
<initialise>
   <posix>
      <executable>/usr/bin/wc</executable>
      <parameter index="0">/tmp/inputfile</parameter>
      <stdout>/Users/compute/Sites/outputfile</stdout>
   </posix>
</initialise>
```

Figure 9: Definition of a compute job in the initialisation section of Rapid's main configuration file

count program is the same computer running the portal. Rapid can transfer files both before—`source` file transfers—and after— `target` file transfers—the execution of a program. All file transfers have to be initialised in the `initialise` section as shown in the example in Figure . In this example, the portlet copies a file selected by a user, which is represented by the variable `$(filenameVar)`, into the `/tmp/` directory of the execution host, where it is named `inputfile`.

```
<initialise>
   <datastage>
      <source>
         <filesystem>mysftpserver</filesystem>
         <path>$(filenameVar)</path>
      </source>
      <filename>/tmp/inputfile</filename>
   </datastage>
</initialise>
```

Figure 10: Definition of a file transfer where a filename is substituted from another user interface element such as a file browser. Here we show a stage in before the computing step where an input file is transferred from a remote SFTP server

A SYNCHRONISATION ISSUE exists in the example shown in Figure . If two users were to access the portlet at the same time, the second user's input file will overwrite the first. We need to make the input and output filenames unique for every computational job. A `uuid` variable exists to make this possible. This variable will be set to a random new `uuid` every time a job is initialised in a Rapid portlet.

```
<initialise>
  ...
   <variable name="uuidVar">
      <uuid/>
   </variable>
   ...
</initialise>
```

Figure 11: Definition of a uuid variable, which is assigned a random string every time a portlet is initialised

Using a uuid variable, we can change the `<filename>` element of our data transfer to ensure a file with a reasonably unique filename is created. The name of our input file will then resemble

`/tmp/inputfile-fe7f0cb0-f606-4801-85ac-b20867443290`. Similarly, we have to ensure the output file, containing the word count is unique. We need to append also the `uuid` to the filename of the output file.

```
<initialise>
   ...
   <datastage>
      ...
      <filename>/tmp/inputfile-$(uuidVar)</filename>
      ...
   </datastage>
   ...
</initialise>
```

Figure 12: New definition of part of the data stage in that incorporates a random input string into the input file

Task 18: Add a `uuid` variable to the initialisation section of your portlet.

Task 19: Append a `uuid` to the name of the input file.

Task 20: Append a `uuid` variable to the name of the output file.

Computational resources

The computational resource is where the programme executes. Two steps are required to setup a resource. First we have to indicate how to reach the computational resource, which we do by specifying a file system resource. Then we define a computational resource that will be accessed through the file system indicated in the first step. If the computational resource is the same as the portal we access the computational resources through a `local` filesystem. Alternatively, if the computational resource is installed on a remote system we can access it through SSH, using the `sftp` filesystem. Next we have to define a computational resource and connect it to the file system. An example of such a connection is shown in Figure .

```
<fork name="compute">
    <filesystemname>computeFS</filesystemname>
</fork>
```

Figure 13: An example of a simple compute resource that executes via a fork

Task 21: Add a new `sftp` file system that refers to the local portal computer in the `initalise` of your Rapid XML file. Name this file system `computeFS`.

Task 22: Add a `fork` execution engine

When we submit a compute job via a portlet it needs to know which resource to submit to as Rapid allows the definition of multiple execution engines. To specify which execution engine to use, we use a `submitto` element as shown in Figure . In order for a

```
<initialise>
...
   <submitto>computeFS</submitto>
...
</initialise>
```

Figure 14: An example of setting the compute resource

portlet to actually submit a compute job a user has to press a button in their browser. In Rapid, this translates to adding a button

that contains the <submit/> tag as shown in Figure .

```
<button display="submit">
   <navigate nextpage="pageone"/>
   <submit/>
</button>
```

Figure 15: An example of a submit button that also changes the portlet interface the next page

Task 23: Set the fork execution engine to the execution engine to submit jobs to.

Task 24: Add a new button or edit an existing one to allow the user to submit a job.

Job monitoring

When a compute job is submitted we want to be able to monitor it and check its results. However, as soon as the job was submitted all its corresponding variables were reset to the portlet's initial values. Two elements exist that enable us to retrieve the settings of the variables at the time when the job was submitted: <joblist> and <setjob>. The element <joblist> iterates through each job. Within this element the values of previously submitted jobs are retrieved and can be displayed. Additionally, the following useful tags are defined.

- <status/> displays the current status of the job (suspended, running, completed, error)
- <date/> displays the date the job was submitted at
- <jobid/> is an identifier that Rapid generates to uniquely identify each job submitted
- <selection name="selectionname"/> displays a radiobutton that allows the user to select a job. Used in conjunction with <setjob>

```
<page name="...">
   <x:h1>Select a job</x:h1>
   <x:br/>
   <joblist>
      <selection name="myselection"/>
      <jobid/>
      <status/>
     <x:br/>
  </joblist>
</page>
```

Figure 16: Basic definition of a compute job monitoring page with a selection mechanism

The element <selection> is used to allow a user to select a job which he or she wants to have a closer look at. It displays a radiobutton the user can click. The selected job can later be retrieved using the <setjob> element.

In this section we display the contents of the output file we generated using the word count command. The simplest way to accomplish this using XHTML is by using an iframe.

```
 <page name="...">
    <x:h1>Results</x:h1>
    <x:br/>
    <setjob selection="myselection">
       File: $(filenameVar)
       <x:br/>
       <x:iframe src ="http://my.httpd.ac.uk/outputfile-$(uuidVar)" width="40%" height="50"/>
       <x:br/>
    </setjob>
</page>
```

Task 25: Add job monitoring to your portlet

Task 26: Deploy the portlet once more into Liferay

Adding more features

The portlet created so far is very basic. Several features can be added such as the following.

- Submit to a Condor pool, PBS cluster or Sun Grid Engine.
- Let the user choose from multiple computational resources where to submit a compute job.
- Add persistence by saving the state of a portlet to a file or a database.
- Use Jython to dynamically change the user interface

To add these features make use of Rapid's user manual[2]; it explains in detail the tags and constructions required to add these features.

[2]