Grid (Portal) Application Development (Approaches, Best Practices, API choices, IOC/DI)

David Meredith
NGS + Grid Technology Group, e-Science Centre,
Daresbury Laboratory, UK

d.j.meredith@dl.ac.uk

Overview

- Portal / Web app architecture
  - JSR 168 portal / portlets – why JSR 168 ?
  - 3 tiered portal architecture (MVC)
    - MVC options (JSF, Portlet API, Webflow, Struts)
  - Web app / portlet development with JSF
- Managing business objects
  - Static singletons / Spring

- Building grid clients with the Java COG.
- Combining the Java COG with Java Server Faces.

Typical Grid Application Development Scenario for 'Science Gateways'

- You need to develop a Grid application to act as a 'science gateway' for your particular requirements.
- Require interfaces / forms for your applications.
- May need to mix the functionality of different services into your application (data Grid services, HPC services).
- Persistence ? Does your application need to save user data – usually e.g. preferences / settings / job descriptions etc…

NGS National Grid Service

Some Key Choices

- Application Style
  - **Online accessible Web application or Portlet**
    - Many users can access application through a browser
    - Provides roaming accessibility (when used with MyProxy which acts a secure certificate store)
    - Traditionally not as good when considering complex rendering requirements - but getting much better with new ‘Web 2’ technologies, e.g. AJAX, powerful Web frameworks).
    - Generally more complex to develop (any non-trivial Web app usually requires backend DB, DAO / transactional service layer, MVC, Security, multi-threading)
  - **Desktop application**
    - User downloads and runs application on own desktop
    - More powerful rendering capability.
    - Not browser accessible (alternative - Web start apps)
  - **Which application style is most suitable for you** ?
    - Really depends on requirements of your application / user requirements. Advantages and disadvantages of each style.
  - **How should i develop the application (which API, DI / IOC** ?)
    - Many choices possible (some best practices will be outlined in talk). Assuming an online accessible portal is suitable for your requirements:
    - Standalone Web Application ?
    - JSR 168 Portlet ?

CCLRC

Some Key Choices

- How do I interact with the Grid middleware (this depends on the Grid middleware of your service Grid). Possibilities:
  - –'**Shell out' to command line scripts / tools** (e.g. 'globusrun'). Requires separate installation of middleware on client machine (nothing wrong with this, just more to install, not as self contained).
  - – **Use Grid client libraries**, (e.g. Java Cog, Python Cog). Such libraries mean you can build your application with its own Grid client libs (this is exactly what we did in practical). No need to install separate Grid middleware – more self contained.
  - – **Use a service** designed to ease access to the Grid e.g. GridSAM (WS for job submission, monitoring).

- Data Requirements
  - – Does your application require access to Data Grid services ? E.g. SRB, OGSA-DAI
  - – Do you require data staging ?
  - – Do you need to stage data in your application (e.g. using Jargon API for SRB) or can you use a service to stage the data (e.g. GridSAM).

Application styles for online accessible applications – Web apps or Portlets ?

- Web application – developed to be ran “standalone” in a servlet container or J2EE application server.
  - All dependencies have to be contained into a single application.
  - ‘Standalone’ development often cleaner.

- Portal / portlets - allow reuse of existing portlet applications - bundled and shared in a single portal.
  - – For example, NGS portal contains separate portlets: Job submit portlet, GridFTP portlet, SRB portlet, LDAP query portlet.
  - – Portlets combine the user interface view into a single portal, usually through a number of ‘tabs.’

Some Open Source JSR 168 Containers

- GridSphere
  - http://www.gridsphere.org
- uPortal
  - http://www.uportal.org
- LifeRay
  - http://sourceforge.net/projects/lportal
- eXo platform
  - http://www.exoplatform.com
- StringBeans
  - http://www.nabh.com/projects/sbportal
- Jetspeed2
  - http://portals.apache.org/jetspeed-2

Limitations of Portlets

- However, (often) in science gateway applications, we need finer grained components that 'mix' the functionality of the separate portlets into a single action or sequence of actions, e.g.
  - "When user clicks button: upload a file, move data to the consuming system (Stage-in), launch application, and move data someplace when application finishes (Stage-out)."
  - This requires the combined functionality of separate "GridFTP" and "Job Submit" portlets.
  - Or maybe OGSA-DAI or SRB portlets or….etc

- This requires both the view and action code of separate portlets to be combined into a single application (back at standalone Web app development – not always true).

- Therefore, endeavor to build applications from reusable components – (this really comes down to some 'best practices' for application development that will be outlined later).
  - But, some good frameworks certainly help by enabling applications to be built using POJO's wired together using 'DI' / 'IOC' (examples shown later).
  - Code can be easily written to be 'reusable' and tested outside of a servlet / portal container.

JSR 168 – Some More Comments

- Assumes developers will be happy to ignore other (well known) development frameworks like Velocity, Struts, and Java Server Faces and learn new Portlet API.
  - 168 involves developing a **GenericPortlet** extension for every single portlet they develop.
  - Can result in complicated **processAction()** and **doView()** methods.
  - Will have to replicate functionality that is provided with web app framework.
  - Assumes developers will be happy with the JSR 168 portlet-style Model-View-Controller.
- Fortunately, it is possible to map these other web application frameworks to portlet actions (portal bridges – e.g. Struts portal bridge).
- I will show examples of the JSF framework for application development.

Why Develop Applications with Java Server Faces ?

- **Web app / Portlet compatible.** Vanilla JSF designed from the outset to be compatible with portals/ portlets and standalone Web apps. This can offer advantages to your development – e.g. the 'NGS openPortal' used earlier.

- **Existing JSF applications** can be ported to run as a Portlet (with some modifications and an awareness of the 'gotachs'). This usually requires (some) modification to your application.

- **No need for portal-bridges.** Some other frameworks involve quite complex portal-bridges (Apache portals bridges proj) – have heard using a portal bridge is comparable to 'trying to fit a square peg in a round hole.' (Stan Silvert, Java One Online 2005)

- **Build application from POJO's** that are free from any servlet or portlet container dependency.
  - POJO's can be written to be reusable and can be easily tested outside of a servlet / portal container.
  - Uses "Inversion of Control' or 'Dependency Injection' to wire pojos together via a configuration file.

- **Results in Reusable UI components or 'widgets'**, e.g. tabbed panes, scrolling tables, calendars, hierarchical trees, table models ..…etc. Apache MyFaces has over 100 reusable components with addition of Oracle ADF (Trinidad). Future – AJAX enabled UI components (hope it will save me from having to learn AJAX and JS)

- **Good tool support**, e.g. drag 'n' drop design of JSF components in JSP using Sun Studio Creator.

Build Applications from reusable POJO / classes by JSF Bean Wiring and Dependency Injection

```
<!-- One shared Grid Util POJO stored in application scope (singleton with no dependency on container) -->
<managed-bean>
   <managed-bean-name>GramJob</managed-bean-name>
   <managed-bean-class>uk.ac.clrc.escience.grid.util.GRAMJob</managed-bean-class>
   <managed-bean-scope>application</managed-bean-scope>
</managed-bean>

<!-- One shared Grid FTP Util POJO stored in application scope (singleton with no dependency on container)  -->
<managed-bean>
   <managed-bean-name>GridFtpUtil</managed-bean-name>
   <managed-bean-class>uk.ac.clrc.escience.grid.util.GridFTPUtil</managed-bean-class>
   <managed-bean-scope>application</managed-bean-scope>
</managed-bean>

<!-- Each user has a UserSession POJO object bound to their HTTPSession -->
 <managed-bean>
    <managed-bean-name>UserSession</managed-bean-name>
    <managed-bean-class>uk.ac.escience.ngs.portal.UserSession</managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
 </managed-bean>

< !– An Interface ‘Backing bean’ with injected references to the beans above -->
 <managed-bean>
    <managed-bean-name> JobSubitBackingBean </managed-bean-name>
    <managed-bean-class>uk.ac.escience.ngs.portal.jobSubmit.JobSubitBackingBean </managed-bean-class>
    <managed-bean-scope>request</managed-bean-scope>
<managed-property><property-name>GramJob</property-name><value>#{GramJob}</value></managed-property>
<managed-property><property-name>GridFtpUtil</property-name><value>#{GridFtpUtil}</value></managed-property>
<managed-property><property-name>UserSession</property-name><value>#{UserSession}</value></managed-property>
  </managed-bean>
```

(In faces-config.xml)

Lets look at the classes

GramJob and **GridFTPUtil**

Already seen *GramJob* and *GridFTPUtil* classes in the practical when they were used in a program that was executed on the command line using “Ant”. There are no JSF or container dependencies in either of these classes.

```
import org.ietf.jgss.GSSCredential;

public class UserSession {
    private GSSCredential userProxy = null;
    private String activeJobName;
    /** No Args Constructor Creates a new instance of UserSession */
    public UserSession() {
    }
    public GSSCredential getUserProxy() {
        return userProxy;
    }
    public void setUserProxy(GSSCredential userProxy) {
        this.userProxy = userProxy;
    }
    public String getActiveJobName() {
        return activeJobName;
    }
    public void setActiveJobName(String activeJobName) {
        this.activeJobName = activeJobName;
    }
}
```

UserSession is a reusable POJO with no dependencies on container

Lets look at the classes

```
import uk.ac.clrc.escience.grid.util.GramJob;
import uk.ac.clrc.escience.grid.util.GridFTPUtil;

public class JobSubitBackingBean {
    private GRAMJob GramJob = null;            // injected via IOC of JSF
    private GridFTPUtil GridFtpUtil = null;    // injected via IOC of JSF
    private UserSession session = null;        // injected via IOC of JSF

    /** No Args Constructor Creates a new instance of UserSession */
    public JobSubitBackingBean () {
    }

    public void executeRun(ActionEvent ae){
      // do job submission logic here using injected GramJob, GridFtpUtil, UserSession
      if(this.getSession().getUserProxy() == null){
        this.showError("Error: Please authenticate yourself – load a proxy certificate");
      } else {
           // create a GRAMJobParameters object called params….
           this.getGramJob(). executeGRAMJob(params, this.getSession().getUserProxy()); // etc
      }
    }

    public GRAMJob getGramJob () {
         return GramJob;
    }
    public void setGramJob(GRAMJob GramJob) {
         this. GramJob = GramJob ;
    }
    // more getters and setters not shown
}
```

JobSubmitBackingBean is a reusable POJO that has the 3 dependencies injected by JSF.

Lets look at the Interface

- The member variables of the POJO’s can be displayed (and modified) in the .jsp view.
- Methods of the POJO’s can be invoked directly from interface to perform logic.
- Functions can be called to process user ‘actions,’ to monitor value change events, to validate user input and much more.

JSP View only displays data and calls methods of backing beans:

```
<h:form id=“executeRunForm">
    <h:outputText value="Job Profile Name: #{UserSession.activeJobName}"/>

    <h:commandButton value=“submit" id=“callexecuteRun”
            actionListener="#{JobSubitBackingBean.executeRun}" />

    <h:message for=“callexecuteRun" styleClass="errorMessage"/>
….
</h:form>
```

Summary – Application Development using IOC / DI

- IOC / DI is a great way to build applications from plain objects which are not tied to any servlet or portlet container (or any other unreachable dependency).

- IOC / DI does encourage the development of reusable classes and UI tag libraries that can be included and reused in any portlet or web app.

- This is at a lower level when compared to combining separate portlets together into a single portal, but helps (significantly) to provide the 'finer' grained control we need when 'mixing' the functionality offered by separate portlets.
  - "e.g. when user clicks button: upload a file, move data to the consuming system (Stage-in), launch application, and move data someplace when application finishes (Stage-out)."

- What we really need to see in the future is the development, and sharing of reusable UI components or 'widgets' for the Grid ! (developed using simple POJO approach so they can be easily 'plugged' into our own applications - much like the widgets provided by the Apache MyFaces project), e.g.

  - GridFTP File browser UI widget + accompanying Backing bean / POJO.

  - SRB File browser UI widget + accompanying Backing bean / POJO….etc.

Tips / Best Practices for Application Development

• Remember to put signed .jar files (e.g. the CogKit libs) into a shared location if your server has more than one Grid app (e.g. <tomcatHome>/shared/lib.

• Monitor your application with jconsole or similar tool (YourKit) – v.cool and useful for exposing mem leaks, thread deadlocks, can monitor the heap, the PermGen space and the frequency of GC allowing you to tune your servers memory allocation requirements.

()

• Compile using lower version JDK, run using latest version JDK (may seem obvious but has caught me out in the past – produces versioning exceptions).

• If you are lucky enough to do some vanilla development, consider using JPA (JSR ???) for your DB access code. Standard. POJO domain model. Simple DAO's. Can choose your ORM tool (e.g. Hibernate, Toplink, Kodo). Supports most relational DB.

Application Development using Spring

- JSF is not the only IOC/DI container available today – Spring is also excellent ! Spring uses a similar concept of wiring POJOs together using an xml file.

- In fact, I combine the use of Spring and JSF in my application development – JSF to do interface, Spring to manage business objects and the DAO's/transactional service layer (offers a v.powerful approach for declarative transaction demarcation – no need for full J2EE container).

- Is a move toward combining JSF and Spring in the latest Web Application development frameworks – consider the Apache Shale project which does just this !

Required Steps to Deploy a JSF Web Application as a Portlet

1) Make necessary modifications to your code:
   a) Use JSF navigation rules only
   b) Use abstracted ExternalContext in web application logic

2) Create portlet.xml deployment descriptor for your JSF application

3) Add jsf-portlet.jar to WEB-INF/lib of your application (JSF RI only, not necessary when using Apache MyFaces)

4) Package JSF portlet into a deployable .war file

4) Deploy JSF portlet application to your portal container and register it with the portal so it can be viewed within the portal interface (specifics vary according to the portal container you are using)

1) Make Some Necessary Modifications to Your Code

a) Ensure page navigation in web application is implemented by JSF navigation rules only (remove any re-direct links). JSF components are rendered so that URL processing is delegated to a ViewHandler which can either create portlet URLs if hosted in a portal, or a FacesServlet URLs if a regular JSF app.

Remove links/re-directs:

```
<a href="./faces/ActiveJobProfile.jsp">Back</a>
```

With JSF navigation rules:

(In faces-config.xml)

```
<!-- below rule applies to all pages because no from-view-id is specified -->
<navigation-rule>
    <navigation-case>
        <from-outcome>goActiveJobProfile</from-outcome>
        <to-view-id>/ActiveJobProfile.jsp</to-view-id>
    </navigation-case>
</navigation-rule>
```

(In Jsp)

```
<h:commandLink action="goActiveJobProfile" value="Back"/>
```

1) Make Some Necessary Modifications to Your Code

b) A JSF portlet application must not rely on any servlet-specific APIs
Use abstracted 'ExternalContext' class to get access to methods and objects that you would normally get from HttpServletRequest, HttpServletResponse, ServletContext.

```
/** to get the ExternalContext */
FacesContext fc = FacesContext.getCurrentInstance();
ExternalContext ec = fc.getExternalContext();

/** wrong ! */
HttpServletRequest req = (HttpServletRequest) ec.getRequest();
Map paramMap = req.getParameterMap();
String cp = req.getContextPath();
Cookies[] cookies  =  req.getCookies();

/** right ! */
Map paramMap = ec.getRequestParameterMap();
String cp = ec.getRequestContextPath();
Map cookies = ec.getRequestCookieMap();

/** useful functions for portlet deployment */
String remoteUserUUID = ec.getRemoteUser();
boolean inRole = ec.isUserInRole("userA");
```

c) Ensure 3rd party libs / packages / dependencies can be deployed to target container (some issues with shared libs and web-app/WEB-INF/libs and security)

2) Create a portlet.xml Deployment Descriptor for JSF web application (using JSF Reference Implementation)

```
<portlet>
   <portlet-name>NgsJobSubmit_Portlet</portlet-name>
   <portlet-class>com.sun.faces.portlet.FacesPortlet</portlet-class>

   <!-- jsf RI required parameter. The "home page" of your JSF application -->
   <init-param>
      <name>com.sun.faces.portlet.INIT_VIEW</name>
      <value>/index.jsp</value>
   </init-param>
   <expiration-cache>0</expiration-cache>
   <supports>
      <mime-type>text/html</mime-type>
      <portlet-mode>VIEW</portlet-mode>
   </supports>
   <supported-locale>en</supported-locale>
   <portlet-info>
      <title>NgsJobSubmit_Portlet</title>
      <keywords>NGS, HPC, Job Submission</keywords>
   </portlet-info>
</portlet>
```

JSF RI does not have a concept of modes, so you (usually) need to disable EDIT and HELP modes in portlet.xml.

2) Create a portlet.xml Deployment Descriptor for JSF web application (using Apache MyFaces)

```
<portlet>
   <portlet-name>NgsJobSubmit_Portlet</portlet-name>
   <portlet-class>org.apache.myfaces.portlet.MyFacesGenericPortlet</portlet-class>

   <!-- MyFaces required parameter. The "home page" of your JSF application -->
   <init-param>
      <name>default-view</name>
      <value>/ActiveJobProfile.jsp</value>
   </init-param>
   <expiration-cache>0</expiration-cache>
   <supports>
      <mime-type>text/html</mime-type>
      <portlet-mode>VIEW</portlet-mode>
   </supports>
   <supported-locale>en</supported-locale>
   <portlet-info>
      <title>NgsJobSubmit_Portlet</title>
      <keywords>NGS, HPC, Job Submission</keywords>
   </portlet-info>
</portlet>
```

MyFaces can support EDIT and HELP modes by subclassing MyFacesGenericPortlet. Only supported with MyFaces 1.0.9+

NGS National Grid Service

3) Add jsf-portlet.jar to WEB-INF/lib of your application (JSF Reference Implementation)

- jsf-portlet.jar is the portlet integration library that must reside with your application in the WEB-INF/lib subdirectory. That way, your application can run as a portlet. javaserverfaces_portlet.class

- Go to http://javaserverfaces.dev.java.net and download javaserverfaces_portlet.class (look under Documents & Files menu item)

- Use"java -cp . javaserverfaces_portlet.class" to install. Based on JSF 1.x release

- This generates jsf-portlet.jar

- or, use jsf-portlet.jar that comes with Sun Studio Creator (free download). Also has a project template for creating a JSF portlet using the JSF RI.

4) Deploy JSF portlet application to your portal container and register it with the portal

This is different according to the portlet container you are using.
For Pluto (Reference Implementation of JSR 168 portlet container), modify 3 Pluto files and add some Pluto boilerplate code to web.xml:

Pluto Files to Modify:

<PLUTO_HOME>/webapps/pluto/WEB-INF/data/
- portletcontext.txt
- portletentityregistry.xml
- pageregistry.xml

a) **portletcontext.txt** (lists the context path of each portlet application)

```
/testsuite
/pluto
/SimpleMyFacesPortletsMixed
/SimpleRIPortlet
/chapter04
/chapter05
/NgsJobSubmissionPortlet1
```

b) **portletentityregistry.xml** (registers the portlet with the portal container)

```
<portlet-entity-registry>

    <application ...>...</application>
    ....

    <!-- Specify portlet application and give it an id value -->
    <application id="12">
        <!-- Specify name of the JSF portlet app as listed in portletcontexts.txt -->
        <definition-id>NgsJobSubmissionPortlet1</definition-id>
        <!-- Give portlet an id value -->
        <portlet id="0">
            <!-- “<portlet app> . <portlet class as specified in portlet.xml>” -->
            <definition-id>NgsJobSubmissionPortlet1.NgsJobSubmit_Portlet</definition-id>
        </portlet>
     </application>

</portlet-entity-registry>
```

c) **pageregistry.xml** (defines the layout of the portlets in the portal)

```
<portal>

    <fragment name="NGS" type="page" >
        <navigation>
            <title>NgsJobSubmit</title>
            <description>NGS Job Submission / JSDL Job Profiles Portlet</description>
        </navigation>
        <fragment name="row1" type="row">
            <fragment name="col1" type="column">
                <fragment name="p1" type="portlet">
                    <!-- <portlet application id> . <portlet id> -->
                    <property name="portlet" value="12.0"/>
                </fragment>
            </fragment>
        </fragment>
    </fragment>

</portal>
```

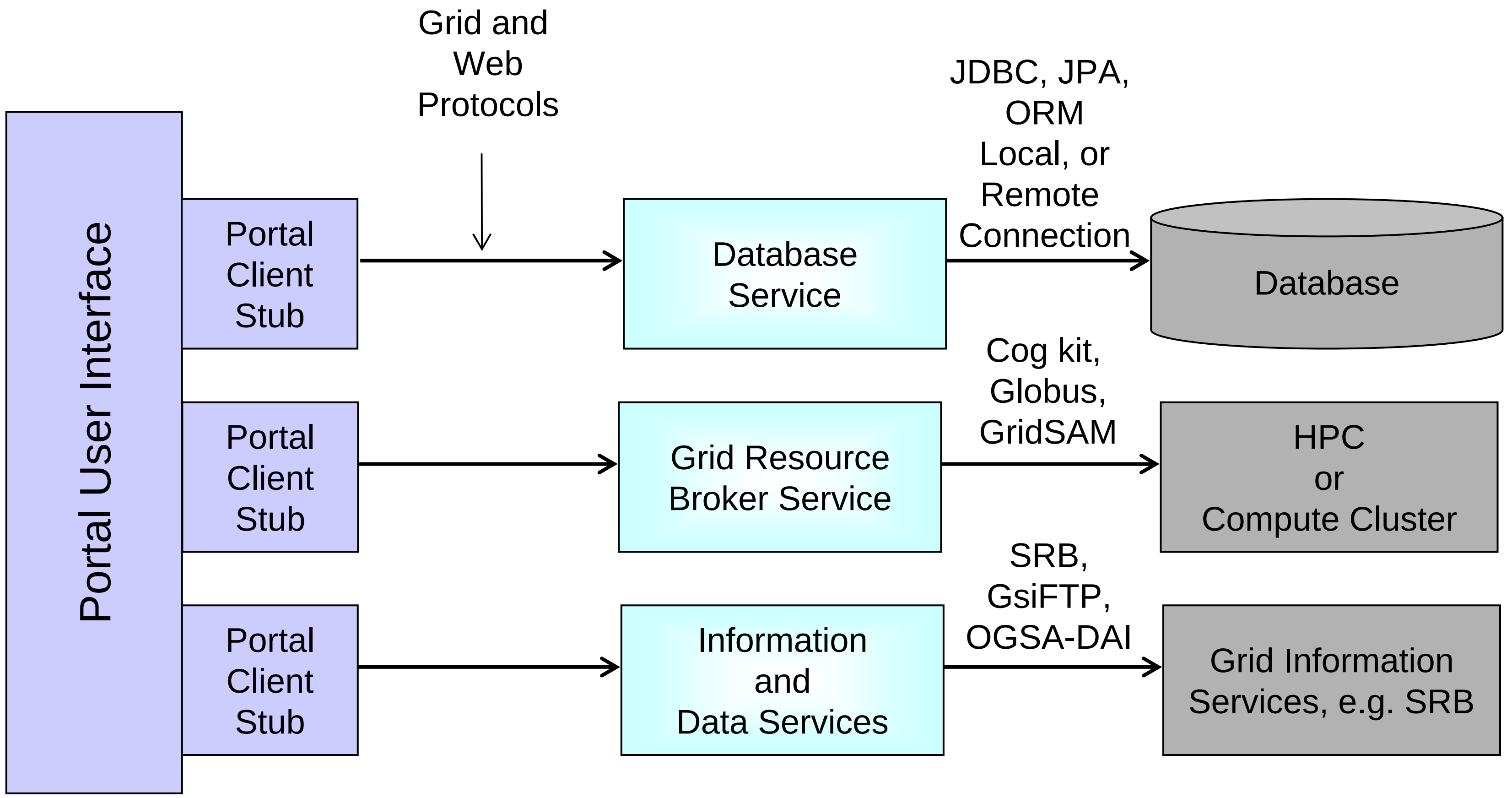
Add Pluto boilerplate code to web.xml

Define a new servlet mapping that wraps the JSF portlet.

Note, this code is added when you use Pluto hot-deploy page

```
<servlet>
   <servlet-name>NgsJobSubmit_PortletServlet</servlet-name>
   <display-name>NgsJobSubmit_Portlet Wrapper</display-name>
   <description>Automated generated Portlet Wrapper</description>
   <servlet-class>org.apache.pluto.core.PortletServlet</servlet-class>
   <init-param>
      <param-name>portlet-guid</param-name>
       <!-- Enter the path to the portlet: <context path>.<portlet name in portlet.xml> -->
      <param-value>NgsJobSubmissionPortlet1.NgsJobSubmit_Portlet</param-value>
   </init-param>
   <init-param>
      <param-name>portlet-class</param-name>
      <param-value>org.apache.myfaces.portlet.MyFacesGenericPortlet</param-value>
   </init-param>
</servlet>
<servlet-mapping>
   <servlet-name>NgsJobSubmit_PortletServlet</servlet-name>
   <url-pattern>/NgsJobSubmit_PortletServlet/*</url-pattern>
</servlet-mapping>
```

Three-Tiered Architecture of Grid



Three-tiered architecture is accepted standard for accessing Grid and other services