Grid Portal Developers Kit

Building a Portal Using GPDK: A Developers Tutorial

Jason Novotny
jdnovotny@lbl.gov

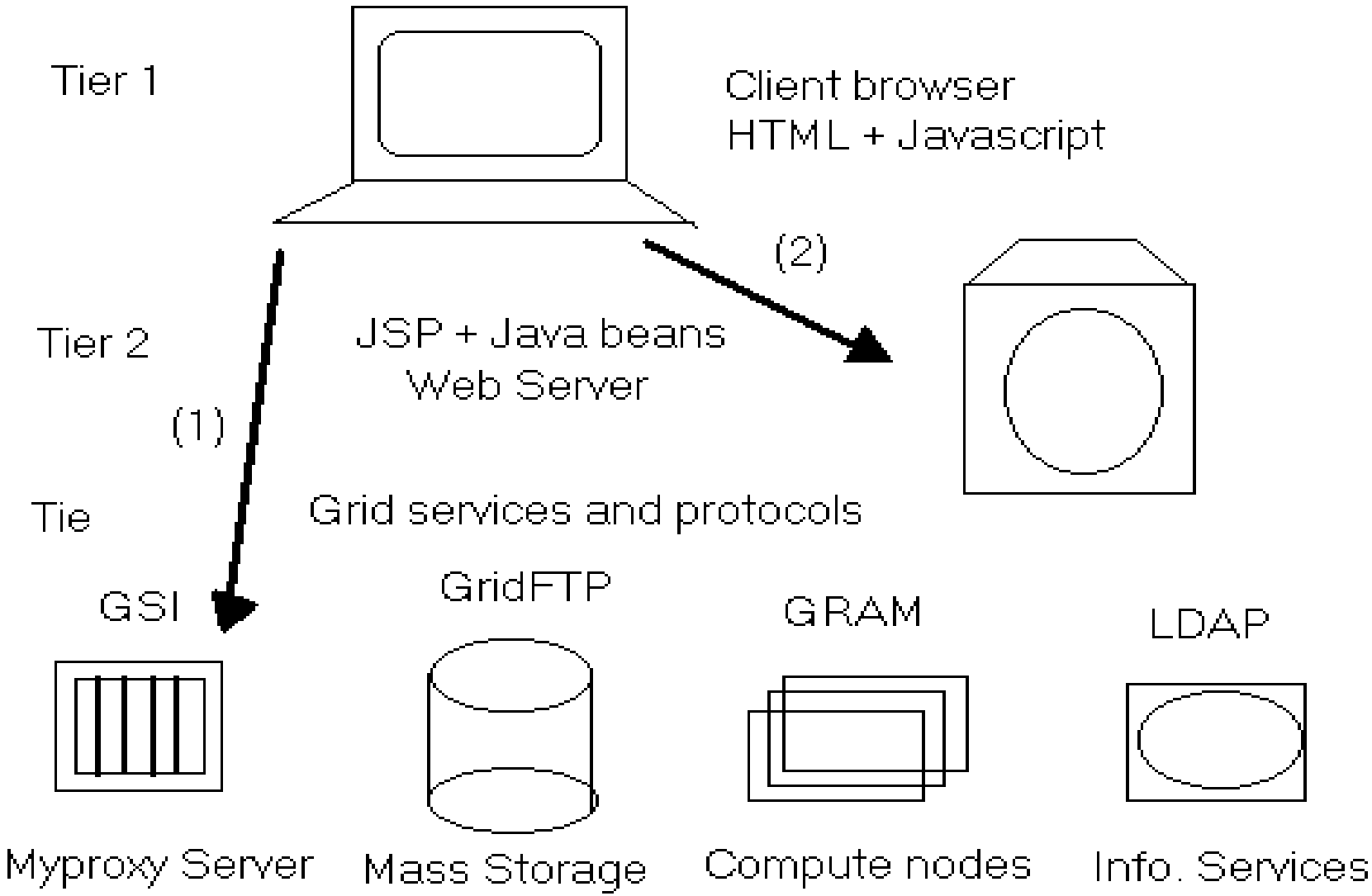
Tutorial Outline

- Brief overview of GPDK
- Installing GPDK
- Core GPDK beans
  - Security, Job Submission, File Transfer, Information Services
  - User Profiles, Logger classes, etc.
- Installing the GPDK demo portal
- GPDK internal architecture
- Advanced topics as time permits

Brief Overview of GPDK

- ***A "Grid portal" is a customizable, personalized web interface for harnessing Grid services and resources.***
- ***The Grid Portal Development Kit (GPDK) is intended to provide a core set of modular, reusable components for accessing Grid services in the form of Java beans.***
- ***GPDK makes use of the Java CoG kit for access to Grid services***
- ***GPDK takes advantage of the Tomcat servlet container, the latest open source reference implementation of the Sun Servlet and Java Server Pages specifications.***
- ***GPDK provides a complete development environment including template projects that can be easily extended to support additional services or customized problem solving environments.***

Portal Architecture



Obtaining GPDK

- GPDK project information available at http://dast.nlanr.net/Projects/GridPortal
- GPDK available off CVS

  cvs -d :pserver:gpdk@palomar.extreme.indiana.edu:/work/cvs login

  (just hit enter)

  cvs -d :pserver:gpdk@palomar.extreme.indiana.edu:/work/cvs co gpdk

GPDK prerequisites

- JDK v1.3 or higher. Tested with Sun JDK v1.3 on Linux and Windows
- Tomcat servlet container v 3.1.1 or 3.2.1. Also tested with v 3.3 milestone releases (requires jar files be placed in lib/common)
- A Portal certificate used for retrieving credentials from a Myproxy server
- (Optional) Secure web server e.g. Apache for portal deployment

Generating a Portal Certificate

- From any Grid node (where Globus has been deployed) issue the following:

  %>grid-cert-request* -dir /tmp -nopw -cn FQDN -host FQDN

  where FQDN is the fully qualified hostname of the portal

- Follow the directions provided by grid-cert-request to obtain a signed certificate.
- Move /tmp/userkey.pem and the newly signed certificate to $TOMCAT_HOME/certs or the Apache certificate/key locations, making sure to edit httpd.conf to reflect the new certificate and key locations.
- Copy the Grid CA certificates found in /etc/grid-security/certificates/* on a Grid node to apache/config/ssl.crt directory if using Apache.
- On the Myproxy server, make sure the myproxy-server.config has the common name (CN) of your certificate in the list of allowed hosts that can retrieve credentials

GPDK Directory Layout

- INSTALL, README -- docs on setting up GPDK
- build.pl, build.xml -- Perl build script with ANT makefile
- etc/ -- contains gpdk.properties
- src/ -- contains core GPDK service beans
- lib/ -- contains required Java classes including CoG, IAIK security libraries, Netscape LDAP classes and ANT
- template/ --contains template Java classes, JSP and web pages used for creating a new project
- docs/ -- Additional GPDK developer documentation and Javadoc generated API documentation
- projects/ -- portal projects including default "demo" project

Development Environment

- GPDK uses ANT, a Java based Makefile tool to compile, deploy and build new portal projects
- Makefile targets are specified in XML e.g.

```
<target name="dist" depends="compile">
    <jar jarfile="lib/${project}.jar"
        basedir="${classes.dir}"/>
    <copy todir="${tomcat.home}/lib">
<fileset dir="lib"/>
    </copy>
    <delete dir="${classes.dir}"/>
  </target>
```

Development Environment

- build.pl is the principal script to compile, deploy, and create new GPDK projects.

[help] # Print this message and exit

[demo] # Build the demo GPDK portal

[update] # Update project from CVS

[new <project name>] # Create a new portal project locatednn projects/<project name>

[all] # Compile GPDK performing the following:

[clean] # Delete existing jar file

[compile] # Compile src java files

[docs] # Make javadoc output of classes

[dist] # Make gpdk.jar library and copy to /usr/local/tomcat-latest/lib

[<target>] # Build target specified in build.xml

Building GPDK

- Need to have JAVA_HOME and TOMCAT_HOME set appropriately
- Edit etc/gpdk.properties
- "build.pl all" compiles core GPDK classes and creates a JAR file, gpdk.jar, which is deployed to $TOMCAT_HOME/lib
- Javadoc API of GPDK source code is generated in docs directory
- Creates $TOMCAT_HOME/gpdk which contains gpdk.log, ContactsList, and creds directory for storing users' delegated credentials

The GPDK properties file

- etc/gpdk.properties contains GPDK specific parameters used by all GPDK generated portals:

```
TOMCAT_HOME=@TOMCAT_HOME@

# Location of the myproxy-server to use

MYPROXY_SERVER=localhost

# Default lifetime of users' proxy certificates in minutes

MYPROXY_DEFAULT_CRED_LIFETIME=60

# Path to portal certificate

PORTAL_CERT=/usr/local/portalcert/usercert.pem

# Path to portal private key

PORTAL_KEY=/usr/local/portalcert/userkey.pem

# Path to CA certificates

PORTAL_CA_CERT_DIR=/etc/grid-security/certificates
```

GPDK properties file cont.

```
# (optional) Path to gridmap file if any

GRIDMAP_FILE=/etc/grid-security/grid-mapfile

# Path to GSI ssh and scp

GSI_SSH=/usr/local/bin/ssh1

GSI_SCP=/usr/local/bin/scp1

# MDS settings

#DEFAULT_MDS_URLS="ldap://mds-
    alliance.ncsa.uiuc.edu:391/o=Globus, c=US"

#DEFAULT_MDS_URLS=ldap://gis.ipg.nasa.gov:4321/o=Grid

DEFAULT_MDS_URLS="ldap://binkley.lbl.gov:2000/dc=lbl, dc=gov,
    o=Grid"

# CoG debug level

COG_DEBUG_LEVEL=1
```

Core GPDK bean capabilities

Security

- Retrieve credentials from Myproxy server and create a proxy from uploaded proxy file

Job Submission

- Submit jobs defined by a JobBean using Globus GRAM API or GSI enhanced SSH.

Information Services

- Provides MDS connection pool to multiple LDAP servers
- Query and Result beans provide an API for querying LDAP servers for information.

File Transfer

- Provides a connection pool to multiple GSI FTP servers as well as beans for copying files using Grid FTP API or GSI enhanced SCP

Myproxy Bean

- Uses Java CoG Myproxy API beneath the covers
  - set/getUsername()
  - setPassword()
  - set/getLifetime()
  - set/getProxyFilename()
  - set/getMyproxyServer()
  - DestroyProxy()
  - LoadProxyFromFile() throws MyproxyException
  - LoadProxyFromStringBuffer() throws MyproxyException
  - RetrieveCredential() throws MyproxyException

MyproxyBean example

```
MyproxyBean myproxy = new MyproxyBean();
myproxy.setUsername("novotny");
myproxy.setPassword("bogus");
try {
    X509Certificate cert = myproxy.retrieveCredential();
} catch (MyproxyException me) {
    System.err.println("Unable to retrieve credential");
}
```

SecureService interface

- Grid service beans extend SecureServiceBean which implements SecureService.
- SecureService has two methods:
  - void setProxyFilename(String proxyFilename)
  - String getProxyFilename()
- SecureServiceBean implements SecureService
  - Instantiates Logger used by other grid service beans
  - Creates a GlobusProxy based on proxyFilename

Job Beans

- JobBean describes a Job

  Boils down to Globus RSL job descriptor

  Setter and getter methods:

  - executable
  - arguments
  - directory
  - queue name
  - max memory
  - num. Processors
  - Etc.

JobSubmission interface

- JobSubmission extends SecureService
  - set/getHostname()
  - void newJob(String jobName, JobBean jobBean)
  - JobInfoBean getJobInfoBean()
  - void runBlockingJob() throws JobSubmissionException
- JobSubmissionBean is an abstract implementation of JobSubmission interface

GRAM and SSH submission

- GramSubmissionBean and GSISSHSubmissionBean extend JobSubmissionBean and uses **Java CoG**
- GramSubmissionBean supports interactive submission (blocking) and batch queue submission (non-blocking)
- GSISSHSubmissionBean uses Java exec to invoke GSI enhanced SSH
- After a Job has been submitted, JobInfoBean contains submission info:

  Timestamp, job name, job ID (callback URL), job RSL, job status

- JobHistoryBean provides a hashtable wrapper for

Simple GRAM Batch Job Submission

```
JobBean jobBean = new JobBean()
jobBean.setExecutable("/bin/ls");
 GramSubmissionBean jobSubmission = new GramSubmissionBean();
jobSubmission.setProxyFilename("/tmp/x509up_u500");
jobSubmission.setHostname("modi4.ncsa.uiuc.edu");
jobSubmission.newJob("simple job", jobBean);
try {
    jobSubmission.runBatchJob();
} catch (JobSubmissionException e) {
    System.err.println("Job submission failed");
}
JobInfoBean jobInfo = jobSubmission.getJobInfoBean();
jobHistory.add("simple job", jobInfo);
```

GassServerBean

- Used primarily for retrieving output from interactive job submissions
- Not recommended as resource costs can be high

e.g. Single GASS server started for each job

```
GassServerBean gassServer = new GassServerBean();
gassServer.setProxyFilename(user.getProxyFilename());
gassServer.start();
jobSubmission.setGassServer(gassServer);
jobBean.setStdout(gassServer.getStdoutURL());
jobBean.setStderr(gassServer.getStderrURL());
try { jobSubmission.runBlockingJob();} catch (JobSubmissionException e)
gassServer.shutdown();
```

Information Services Beans

- MDSPool provides a connection pool to LDAP servers using Netscape Directory SDK

  http://docs.iplanet.com/docs/manuals/dirsdk/jsdk40/contents.htm

- MDSPool is a singleton class

  getInstance() returns a static instance

- MDSPool uses LDAP URLs provided in gpdk.properties

- MDSPool instantiated and destroyed by BasicPortal

MDSQueryBean

set/getMDSUrl()

set/getSearchAttributes()

set/getSearchFilter()

Hashtable queryForContacts() throws MDSException

Hashtable queryForQueueAttrs(String hostname) throws MDSException

Hashtable queryForHostAttrs(String[] searchHostnames) throws MDSException

LDAPSearchResults query() throws MDSException

By default, LDAP referrals are followed

File Transfer beans

- FileTransfer interface extends SecureService interface
  - set/get{Source, Dest}Hostname()
  - set/get{Source, Dest}Directory()
  - set/get{Source, Dest}Filename()
  - Transfer()
- FileTransferBean provides abstract implementation of FileTransfer interface
- GSIFTPTransferBean and GSISCPTransferBean subclass FileTransferBean using Java CoG GridFTP API or exec'ing GSI enhanced SCP

File Transfer beans

- GSIFTPServiceBean provides a session bean for maintaining multiple GSIFTP connections (session connection pool)
- Internal thread checks for server timeouts and disconnects after 15 minutes

GSIFTPClient getConnection(String hostname)

void closeConnection(String hostname)

void destroyConnections()

void start()

void stop()

void run()

User Profiles

- UserProfileBean is a serializable session bean intended to store a user's profile
  - set/getUsername()
  - set/getUserDN()
  - set/getEmailAddress()
  - set/getEmailNotify()
  - set/getX509Certificate()
- UserAdminBean is responsible for saving and loading UserProfileBean's.
- Which users are authorized to access portal?

User Authorization to Portal

- Currently GPDK is designed to create a profile for any user that can retrieve credentials from the Myproxy server specified in gpdk.properties
- UserAdminBean can optionally look in local gridmap file to grant or deny access
- UserLoginBean can be used in conjunction with plaintext password file to grant or deny access.

Additional GPDK classes

- Config class is used to read parameters specified in gpdk.properties
- A Logger is used by all GPDK service beans to log warnings and errors.
- LoggerManager provides a singleton class to manage multiple Loggers

```
LoggerManager logManager = LoggerManager.getInstance()
logManager.addLogger("gpdk", gpdk.log)
Logger logger = logManager.get("gpdk")
```

BasicPortal class

- The BasicPortal class provides static access to portal (GPDK) specific data that is accessible for the lifetime of the server.
- BasicPortal has init() and destroy() methods which are responsible for initializing and shutting down the LoggerManager and the MDSPool.
- Init() method also creates "ContactsList" file containing resource information obtained from querying LDAP server (GIIS)

Building the GPDK demo portal

- GPDK provides template source code, JSPs to demonstrate the GPDK beans and a working portal
- GPDK build script provides target "demo"

  "build.pl demo" creates the GPDK demo portal

  Projects directory in GPDK contains self-sufficient demo portal including README, build scripts, source code and JSP/HTML files
- To build and deploy demo portal:

  "cd projects/demo; build.pl all"

  Creates and deploys demo.jar as well as javadoc API for demo source code files

Template file pre-processing

- Files in template directory are copied and preprocessed to projects directory based on 2 substitutions:

  PROPER_NAME defines the Portal project name e.g. "GPDK" for the demo portal

  PROJECT_NAME defines the Portal package name e.g. "demo" for the demo portal

- ANT accomplishes copying and preprocessing:

  ```
  <copy file="${template.dir}/README.tpl"
  tofile="${project.dir}/README"/>

  <replace dir="${project.dir}" token="@PROJECT_NAME@"
  value="${project}" excludesfile="${project.dir}/build.xml"/>

  <replace dir="${project.dir}" token="@PROPER_NAME@"
  ```

Template source code

- Provides a portal specific @PROPER_NAME@UserProfileBean that subclasses UserProfileBean
- Provides a @PROPER_NAME@Portal class that subclasses BasicPortal
- Provides a @PROPER_NAME@Config class that allows developers to add new initialization parameters to properties file
- Provides a central, controller servlet used to forward control to Page objects
- Provides several Page classes e.g. LoginPage, LogoutPage, UpdateProfilePage to demonstrate the

Template project code

- Provides JSP/HTML to demonstrate GPDK services in web directory
- Provides extensible @PROJECT_NAME@.properties file that can be used to add new portal configuration settings
- Provides README detailing installation instructions and description of template files.
- Provides complete build scripts (build.pl and build.xml) for developing new portal project

Building a New Project in 3 steps

- Step 1: Come up with cool project name e.g. Quake - the Earthquake analysis portal
- Step 2: Use GPDK build script to create new portal:

  build.pl new Quake

  Preprocesses template files according to 2 substitutions:

  - PROJECT_NAME = quake
  - PROPER_NAME = Quake
- Step 3: Compile and deploy Quake project files

  cd projects/quake; build.pl all

  Creates $TOMCAT_HOME/quake directory which contains quake.log, pages.config and users directory to store serialized user profiles

Testing the demo portal

- Startup Tomcat in stand-alone mode

  cd $TOMCAT_HOME/bin; startup.sh

- Go to http://localhost/demo/servlet/demo from your web browser
- Try uploading a proxy or retrieving from the Myproxy server if configured. On success you will be able to create and edit a personal profile
- When portal is first initialized, ContactsList is created containing static MDS info e.g. resource and queue information

GPDK uses Model 2 Architecture

- Model 1 defined as page-centric architecture. Use Java beans directly from JSP pages only
  - Makes unmaintainable JSP code with too much embedded Java
- Model 2 uses MVC design pattern to separate presentation from logic
  - A centralized servlet is used to handle all requests by forwarding to the appropiate JSP "view" page
  - Further broken down by using Page objects to perform the back-end business logic

Controller Servlet

- HttpServlet class has init(), service(), and destroy() methods

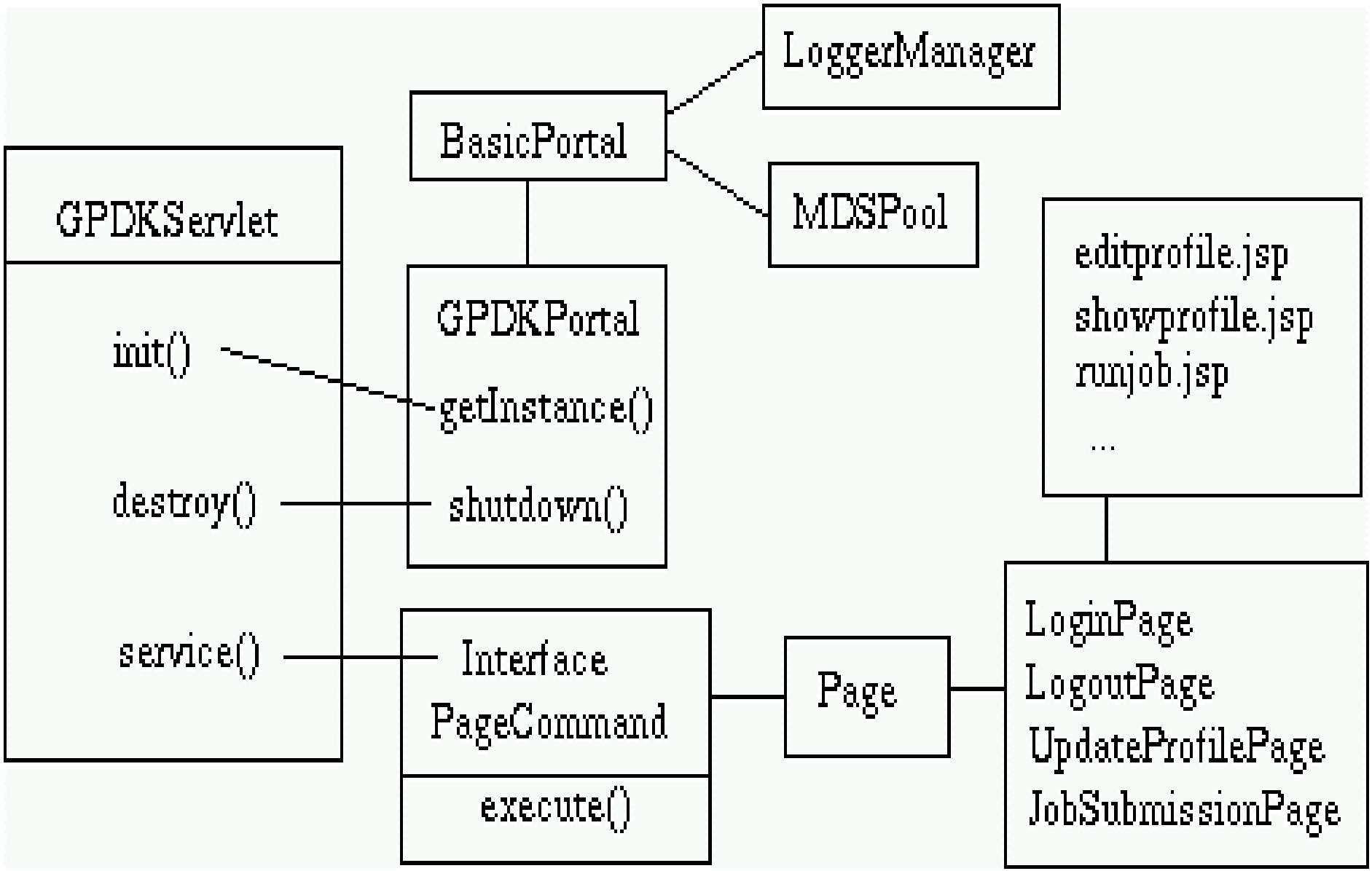
  Init() is responsible for instantiating a @PROPER_NAME@Portal object and loading Page objects

  Destroy() simply shuts down the @PROPER_NAME@Portal object

  Service() handles all requests and forwards to the appropriate Page object

- Additional helper methods allow the logging of HTTP request and session information

GPDK internal architecture



Portal pages

- Controller servlet uses command pattern to forward control to a particular Page object
  - Based on value of "action" parameter in HttpServletRequest object
- PageCommand defines interface all Page objects must implement
  - Execute() is only method defined by interface
- Servlet pre-loads Page objects in init() method based on etc/pages.config file
- Service() method of Servlet responsible for invoking execute() method of Page objects based

pages.config

- Contains 3 columns used to

  Page Action Name

  - Used to identify a particular Page object to a request

  Page class

  - The Page object to invoke by the servlet

  JSP to forward to

  - A JSP page to load after the execute() method completes for the given Page object

pages.config example

- Page action Page object JSP page

  login LoginPage main.jsp

- In login.html, you see:

  <form action="POST" action="/demo/servlet/demo">

  <input type="hidden" name="action" value="login">

  </form>

- When form is submitted, "action" tag will cause the Servlet to invoke the LoginPage.execute() method

- After execute() completes, servlet forwards to main.jsp. (execute() method can also redefine

Creating a New Portal Page

- Edit pages.config e.g.

```
Someaction     SomePage          somejsp.jsp
```

- Create a web page with the action name specified

```
<input type="hidden" name="action" value="someaction">
```

- Create SomePage in src/pages with the skeleton:

```
Public class SomePage extends Page {
    SomePage(String gotoPage) {
        super(gotoPage);
    }
    public String execute(HttpServletRequest req) throws
      CommandException, PageException {  // fill code in here  }
}
```

- OR, use BasicPage to directly forward to a JSP

- Create somejsp.jsp in web/jsp directory

Page Error Handling

- Execute() throws CommandException, PageException
  - If a request parameter doesn't exist or is invalid, throw CommandException
  - If an error occurs in Page, throw PageException(message)
  - Optionally, log error-
    - logger.log("An error occurred")
- When an exception is thrown, Servlet redirects to error.jsp which displays the error message and optionally a stack trace

Portal development (mini-FAQ)

- What if I edit/create a new GPDK service bean/class?

  "build.pl all" in GPDK directory to build new Jar file and deploy to $TOMCAT_HOME/lib

- What if I change gpdk.properties?

  Same as above since the properties file is contained within the JAR file

- What if I edit/create source code in the project directory or the project properties file?

  "build.pl all" in the project directory

- What if I edit/create html/JSP files in the project directory?

  "build.pl prepare" will deploy them to $TOMCAT_HOME and you won't need to restart Tomcat

- What if I edit/create any files in the template directory?

  "build new $project" will update existing $project with

Deploying a Production Portal

- Must set up secure web server e.g. Apache w/ mod_ssl or Stronghold
- WebServer-SG is intended to be a turn-key secure web server with Tomcat

  http://www-itg.lbl.gov/Grid/projects/WebServer-SG.html
- You can use the generated portal certificate with Apache (edit httpd.conf)
- If mod_jserv is used as the connector with Tomcat, add the following to httpd.conf

  Include @TOMCAT_HOME@/conf/tomcat.conf

Production Portal cont.

Add the following to $TOMCAT_HOME/conf/tomcat.conf, where @PROJECT_NAME@ is the project you've created.

```
ApJServMount /@PROJECT_NAME@ /root

Alias /@PROJECT_NAME@
    "@TOMCAT_HOME@/webapps/@PROJECT_NAME@"

<Directory "@TOMCAT_HOME@/webapps/@PROJECT_NAME@">

    Options Indexes FollowSymLinks

</Directory>

ApJservMount /@PROJECT_NAME@/servlet /@PROJECT_NAME@

<Location /@PROJECT_NAME@/WEB-INF/ >

    AllowOverride None

    deny from all

</Location>
```

Production Portal cont.

- If mod_jk (the new and recommended connector is used) add the following to htpd.conf

```
Include @TOMCAT_HOME@/conf/mod_jk.conf

Alias /@PROJECT_NAME@
    "@TOMCAT_HOME@/webapps/@PROJECT_NAME@"

<Directory "@TOMCAT_HOME@/webapps/@PROJECT_NAME@">

    Options Indexes FollowSymLinks

</Directory>

JkMount /@PROJECT_NAME@/servlet/* ajp13

JkMount /@PROJECT_NAME@/*.jsp ajp13

<Location "/examples/WEB-INF/">

    AllowOverride None

    deny from all
```

Securing GPDK

- Make sure login information goes over HTTPS
- Edit Servlet.java and uncomment the following lines in service()

```
if (!req.getScheme().equals("https")) {
    rd = getServletContext().getRequestDispatcher(ROOT_PAGE);
        rd.forward(req, res);
}
```

- ROOT_PAGE is web/index.html and simply redirects to HTTPS:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
    Transitional//EN">
<html><head>
<meta http-equiv="refresh"
    content="1;url=https://@HOSTNAME@/@PROJECT_NAME@/servle
    t/@PROJECT_NAME@">
</head><body></body></html>
```

Additional resources

- Excellent book "Web Development with JavaServer Pages" Kolb & Fields

  http://www.manning.com $10 online!

- Recommend reading the Servlet 2.3 and JSP 1.2 specifications from http://java.sun.com

- Information on the Tomcat servlet engine can be found at:

- http://jakarta.apache.org

- WebServer-SG: A distribution of Apache, Tomcat and mod_ssl

  http://www-itg.lbl.gov/Grid/projects/WebServer-SG.html