Designing Grid Tag Libraries and Grid Beans

Mehmet A. Nacar[1,2], Marlon E. Pierce[1], Gordon Erlebacher[3] and Geoffrey C. Fox[1,2]
[1]Community Grids Lab, Indiana University
[2]School of Informatics, Indiana University
[3]School of Computational Science, Florida State University
{mnacar, marpierc, gcf}@indiana.edu
erlebach@scs.fsu.edu

Abstract: *We present a detailed description of the implementation of a library of Grid tag libraries and Grid beans for Grid Web portal development. Grid tags provide Java Server Faces (JSF) custom components for Grid services. They enable the definition of attributes to the Grid service parameters in a dynamic way embedded into JSF view pages. In addition, Grid beans provide client proxies to the Grid services. Grid tags and beans together provide a platform to develop Grid portlets easily. In addition to standard Grid job submission and remote file operation tags, we also provide management and monitoring capabilities for Grid tasks. This system can persistently store bean features and job parameters, which results in a permanent storage for archiving and reference.*

Keywords: Java Server Faces, Grid portals, portlets, tag libraries

1. Introduction

Grid Web portals are gateways to science applications and data simulations. For instance, TeraGrid [1] computing resources are accessed through gateway portals that provide higher level user interfaces to basic TeraGrid services. The Globus [2, 3] provides implementations for the core services, such as file transfers, remote job submission, and resource information retrieval.

Most of the Grid services are accessible by using command-line tools or Web services clients. GCE Shell [4] supports Grid services by using a command-line tool. Grid portals provide user friendly Web interfaces. Both portals and command line environments are implemented using the Java CoG abstractions [5] that wrap Grid services to support an additional client-programming layer on top of Grid services. Java-based portals may be built out of standard components, called portlets, which are standardized in the JSR 168 specification. Grid portlets are frontend clients of Grid services. Typical capabilities include client tools for interacting with the following services:

- Credential generation and management
- Job Submission
- File transfer and operations
- Monitoring and persistence
- Resource management

It is our observation that portlets work well for exchanging relatively complete applications between projects. There needs to be a way, however, to construct portlets themselves out of reusable components. The capabilities listed above can be implemented as individual portlets, but typically we want to combine these common Grid capabilities into more specific portlets for a particular application. For example, a portlet developed to submit a computational material science code needs to be composed out of job submission, file transfer, and job monitoring capabilities into a single portlet, rather than composed as a combination of existing job submission, file transfer, and job monitoring portlets. Java Server Faces (JSF) [7] is a Web development framework that provides component model to build dynamic web pages, and forms the basis for our approach. JSF applications can be deployed as portlets by using the JSF portlet bridge [8], which provides JSR 168 compatible libraries. The JSF component model can be customized to extend new tag libraries. We have used the JSF tag library framework to design Grid tags and beans to simplify Grid portlet development.

In this paper, we discuss significant revisions and improvements to tag libraries based on the lessons we have learned in the previous work [9]. Our Grid tag libraries enable the design of Grid portlets out of basic Grid tasks. We have changed the design of tag libraries, and we provide additional Grid tags and beans. Instead of using generic task tags, we have used specific tags such as *myproxy*, *jobsubmit*, *fileoperation*, and *filetransfer,* building off the work described in [5]. The new specification brings additional features for application developers. First, the new specification provides more attributes specific to Grid tags that are self-contained and can be customized easily. Second, composite tasks can contain an unlimited number of subtasks (limited to system resources), unlike the previous work, which was restricted to three multi-staged tasks. Both implementations are currently limited to "one deep" nested composite tasks, but the new approach will enable us to build recursively nested subtasks. The third advantage of Grid tags is that it gives liberty to developers to use their own Grid beans library or add more Grid beans to the existing ones.

We summarize related works in the second section. The third section explains JSF Grid beans and tags. We introduce applications and conclude with future works.

2. Related Work

Grid portlets have been developed by a number of groups. GridSphere's Grid portlets [10] provide a set of capabilities that supports Grid services available by the Globus toolkit, including GRAM, Grid FTP, MDS, GRIS, MyProxy, Web Service Resource Framework (WSRF) for GT4 and Open Grid Services Architecture (OGSA). These portlets are built on JSP and use the Grid portlet services of GridSphere. GridSphere Grid portlets are strictly dependent on the GridSphere portal framework; as a result these portlets are not portable among portal containers.

Reasonable Server Faces (RSF) [11] is another Web framework that works to separate the presentation and logic. It enables HTML pages to be totally independent from the backing beans. It also supports simple beans in the request scope. The beans are outside of local JVM and are created in the Spring container [12]. Similar to JSF, RSF supports custom components. In this case, components do not present any view behavior, unlike JSF. In other words, RSF components are non-visual. This is one advantage of RSF in terms of developing Grid tags. RSF tags are described in XML and do not directly tie to technology specifications like JSTL [13]. RSF tags must comply with rendering technologies like IKAT [14].

OGCE portlets [15] are built on Velocity and provide access to common Grid services through the Java CoG abstraction layer [5]. OGCE also provides portlets for Condor and Storage Resource Broker services. These portlets are compliant with JSR 168 and portable among portal frameworks. For example, one can deploy OGCE portlets on either GridSphere or uPortal. Each portlet provides a single Grid capability. JSR 168 does not support inter-portlet communication in its specification; however, OGCE portlets has to share session data to access proxy credential.

3. JSF Grid Tags and Grid Beans

We aim to provide a set of Grid tags in JSF that can be used to build Grid portlets. Our tag libraries provide common Grid capabilities such as proxy credential management, job submission, file operation, and workflow by means of multi-staged tasks. Grid tags are associated with Grid beans to access Grid services. Grid bean methods are bound to tags with attributes. These can then be used to simplify the building of new Grid portlets.

3.1 Grid tags

Grid services are interfaced by Java CoG abstractions. These programming interfaces have capabilities to generate proxy certificates, submit jobs, transfer files and make file operations. They also provide composite task submissions and their handling.

JSF technology helps to build user interfaces based on an object-oriented component approach. JSF tags are built from Java classes that can be extended using JSF component model. New components derive from JSF base component classes. Each component should define its attributes, which can bind values, methods or actions. A full discussion explaining how to extend JSF components is beyond the scope of this paper. We recommend [16] for a tutorial on this subject.

The main goal is to make Grid portlet development easier by encapsulating standard Grid operations with JSF tags. These tags can be assembled to create composite tasks. In traditional Web frameworks such as Velocity and JSP backing bean objects and HTML tags are mixed within the server pages. Instead JSF eliminates this intervention by proposing JSF tags that separate backing bean and server pages.

3.2 Use case example

Typically a Grid portlet must do several related tasks in response to a user-generated event. These may be thought of as simple workflows. These workflows can be considered the nodes of a Directed Acyclic Graph (DAG), which are Grid tags are designed to support. The DAG, or composite task, is called '*multitask'* in our approach. Multitasks only allow dependent task units and prevent parallel tasks. Figure 1 shows a multitask with sub-tasks and their dependencies. In this example, Task A makes a directory. Task B transfers an input file form a remote host to newly created directory, and Task C is responsible for submitting a job on the remote computer. When Task C completes, Task D transfers output file to another location. The following explains the scenario in detail through the use of Grid tags.

This example demonstrates a composite Grid task with Grid tags. The JSF snippet below (Listing 1) shows how a portlet developer would create a custom Grid portlet. First, a myproxy tag generates a proxy credential form `gf1.ucs.indiana.edu` myproxy server. Second, using this credential, it makes a directory on the TeraGrid resource `cobalt.ncsa.teragrid.org`. Third, it transfers an input file called *input_file* from `gf1.ucs.indiana.edu` to `cobalt.ncsa.teragrid.org`. Forth, it then executes a script called *execute*. When the execution is completed outputs are written to the file named *result*. If an error occurs it is also written to the file named *error*. Finally, result file is transferred back to `gf1.ucs.indiana.edu`.

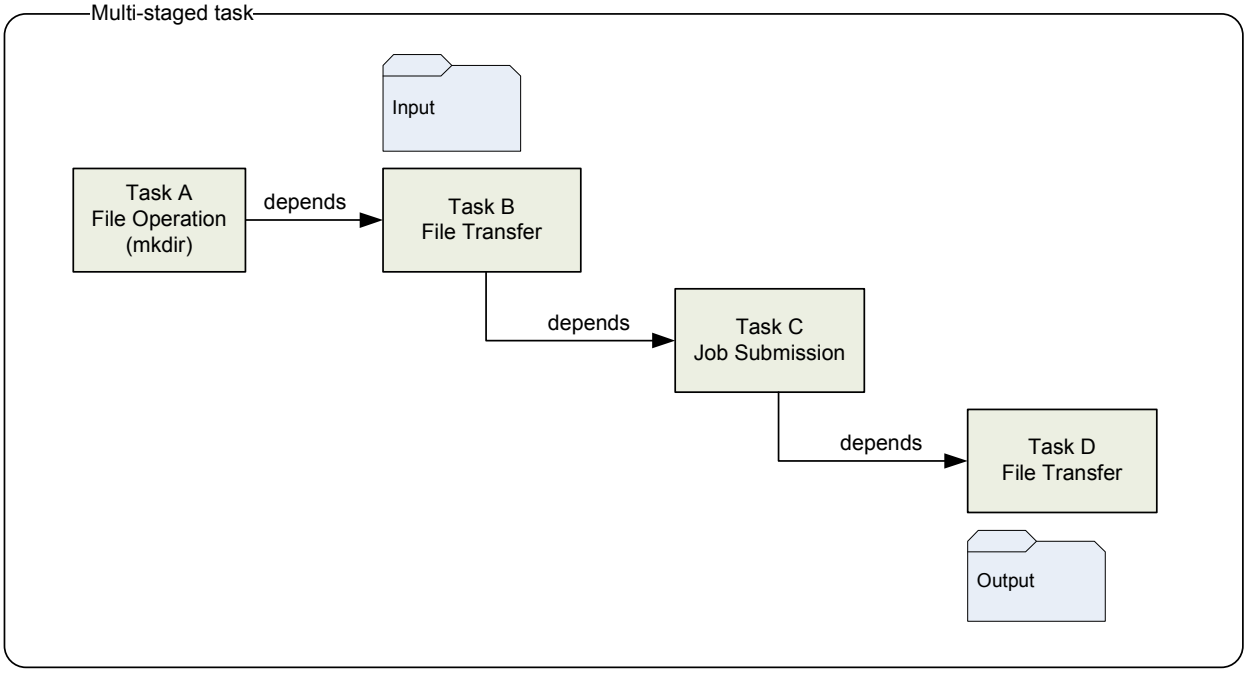

Figure 1. A typical multistage Grid job involves four sub-tasks: moving an input file to a particular execution host, submitting the job, and moving the output to a storage host.

The `<%@taglib uri="http://www.ogce.org/gsf/task" prefix="o"%>` tag is used at the top of the page to define the custom tags called with the *"o"* namespace. Application developers must define Grid operations in a Web form. The <o:submit> tag is a submitting button for the composite task that is bound to a JSF action method [16]. The <o:multitask> defines composite task and <o:dependency> defines their dependencies. The tasks <o:myproxy>, <o:fileoperation>, <o:filetransfer> and <o:jobsubmit> are unit tasks for this composition. The dependency tags indicate that taskA must complete successfully before taskB will run, taskB must complete successfully before taskC can be run and taskC must complete successfully before taskD could run. Complete XML schema specifications of Grid tags can be found at [17]. Each Grid tag is associated with UI component and tag class that is explained in great detail in section 3.4.

```
<%@taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<%@taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<%@taglib uri="http://www.ogce.org/gsf/task" prefix="o"%>
<f:view>
   <h:form id="myform" >
      .......
      .......
      <o:submit id="test" action="next_page" />
         <o:multitask id="mytask" taskname="test" persistent="true" >
            <o:myproxy id="proxy" hostname="gf1.ucs.indiana.edu" port="7512"
                       lifetime="2" username="manacar" password="******" />

            <o:fileoperation id="taskA" command="mkdir"
                             hostname="cobalt.ncsa.teragrid.org"
                             path="/home/manacar/tmp/" />

            <o:filetransfer id="taskB"
from="gridftp://gf1.ucs.indiana.edu:2811/home/manacar/input_file"
to="gridftp://cobalt.ncsa.teragrid.org:2811/home/manacar/tmp/input_file" />

            <o:jobsubmit id="taskC" hostname="cobalt.ncsa.teragrid.org"
                         provider="GT4" executable="/bin/execute"
                         stdin="tmp/input_file" stdout="tmp/result"
                         stderr="tmp/error" />

            <o:filetransfer id="taskD"
from="gridftp://cobalt.ncsa.teragrid.org:2811/home/manacar/tmp/result"
to=" gridftp://gf1.ucs.indiana.edu:2811/home/manacar/result" />

            <o:dependency id="dep1" task="taskB" dependsOn="taskA" />
            <o:dependency id="dep2" task="taskC" dependsOn="taskB" />
            <o:dependency id="dep2" task="taskD" dependsOn="taskC" />

         </o:multitask>
      </o:submit>

   </h:form>
</f:view>
```

Listing 1: Grid tag libraries are used to build a sample Web form.

3.3 Grid Beans

Grid tags and beans work together to perform Grid tasks. Grid tags provide the JSF components for Grid applications, while Grid beans provide the business logic of Grid applications. We have implemented Grid beans in a generic and standard way to support underlying Grid technologies. We have also attempted to design our tag libraries to support other Grid bean implementations. The Grid beans are generic tasks that may be extended using other toolkits besides Globus. For example, the JobSubmitBean

for job submission uses Globus resources in our implementation. Developers can create their own beans with other toolkits. For example, Condor can be used for job submissions rather than Globus. However, this requires that Grid bean method names should be standardized and required bean methods has to be provided. For example, actions methods should be called *submit* in all beans. Parameter names should also be consistent throughout the beans e.g., hostname, provider, username and executable etc.

Our Grid beans are listed below.

- MyproxyBean: This bean generates user proxies and stores the Grid credential in the session.
- JobSubmitBean: Executes GRAM job submissions.
- FileOperationBean: Performs common file and directory operations like *rm, mkdir, put, get*
- FileTransferBean: Transfer files among GridFtp servers
- MultitaskBean: Creates composite tasks and execute them.

Note that these are independent of the JSF framework. The Grid tag libraries shown in Listing 1 are built from these, as we describe in the next section.

3.4 Design and management of Grid tags

Grid tag libraries are built using JSF custom component development techniques. A standard JSF tag requires at least two classes to be implemented: the ComponentTag and IUComponent classes must be extended. Tag names and attributes have to be defined in a *tld* file and this file is added to *web.xml*. Component names and classes are defined in *faces-config.xml*. A full explanation of JSF custom tag development is available from [16].

Custom component classes extend the UIComponentBase class and are normally associated with HTML or other rendered widgets (input fields, buttons, etc.) in the user interface. We have implemented several custom UI components, including UISubmit and UIMultitask, as discussed here. Components can access a map (specifically, a java.util.Hashmap) of attributes and child components. If the component is visual like UISubmit (which we associate with the HTML <submit> button), it also implements encoding and decoding methods to process HTML markup. If the component is non-visual (i.e. does not need to be converted into HTML), it is associated with a null renderer. UIMultitask class is a non-visual component. In addition, the JSF ComponentTag class extension has to implement release(), setProperties(), getComponentType(), and getRendererType() methods. The setProperties() method binds attribute values and methods to the associated UIComponent.

In JSF, the tags and attributes are used to render displays and communicate attribute values (see Listing 1). We encapsulate the actual logic of the page (associated with user button clicks) in several beans that are called by the UISubmit's action method. Besides tag and component classes, there are core beans as following:

- ResourceBean: A general bean to collect property values used in JSF form pages. By default it loads property values from a *resources.properties* file.
- FactoryBean: Manages multiple Grid beans (super class of JobSubmitBean, FileOperationBean and FileTransferBean) and MultitaskBean instances for a single user
- MonitorBean: Monitors and manages bean executions
- TaskListener: Catches Grid bean execution stages and propagates events back to the monitoring bean.
- ComponentBuilderBean (CBB): Retrieves Grid components from JSF pages (see Listing 1) and builds internal hierarchical directed graph of the grid actions to be taken

ResourceBean, MonitorBean and ComponentBuilderBean are managed by JSF's session handling mechanisms and are declared in the faces-config.xml file. CBB is not normally used directly by developers in their JSF pages. They instead interact with this object through Tag libraries. Application developers can directly use ResourceBean and MonitorBean to build up pages.

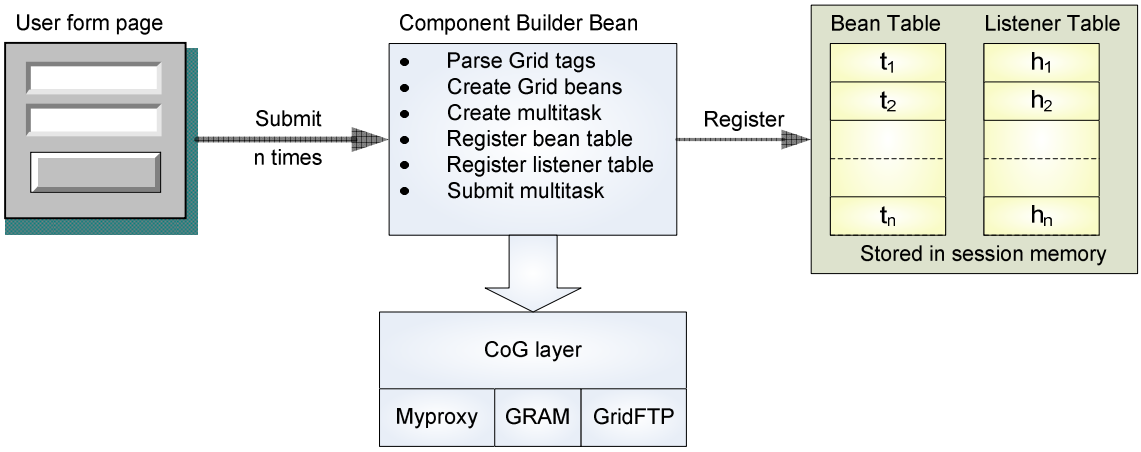

Figure 2. Shows architecture of ComponentBuilderBean and its components

Figure 2 shows the architecture of components. In this diagram, bean and listener tables are in the HttpSession and tables store bean and listener objects in a Hashmap. CBB handles user requests on the server side using Grid bean property values provided by ResourceBean. The actions are fired off by the Grid *submit* tag that is bound to the submit method of CBB. Its action listener catches the event and calls required methods to parse custom components. FactoryBean then constructs corresponding sub-tasks. Next, CBB constructs a taskgraph using MultitaskBean. CBB adds child components which are Grid beans and their dependencies. It then submits the taskgraph and passes the control to the submit button's action attribute. The JSF engine handles the value of the action attribute, while a navigation rule points to the destination page based on the attribute value.

The above classes (particularly the Factory Bean) are designed to accommodate a common use case in Grid portlets that is not handled well by JSF: we need to construct many beans for encapsulating many submissions by a single user in a single session. JSF manages the sessions (lifecycle) of beans but these are statically configured in faces-config.xml, so we need an approach to create and manage lots of Grid beans. We must also address a disparity of time scales: JSF event processing may take milliseconds, while the corresponding backend action may take much longer. Our solutions are described in the following section.

3.5 Design Principles

We have used the strategy of returning immediate results to the user such as passing the control to the next page since Grid operations can take a long time to complete. Thus, a user submits the job in one page and is not required to wait until the job finishes. Instead, users are able to monitor their jobs in another page. To maintain this scenario, either we need to keep callbacks for each job or to store listeners for each job in the servlet HttpSession object. We have therefore used CBB that take care of each request in the session. Then we stored bean instances and their listeners into tables (Hashmap) among the session with *taskname* key. The *taskname* key is created by putting the user-defined taskname (collected from Web form input) and the timestamp together to provide a reasonable ID.

Grid tags launch Grid operations. Keeping track of lifecycles and archiving are also important aspects of Grid portlets. Thus, we define a <o:handler/> tag in Listing 2 that provides capabilities allowing users to manage lifecycles manually such as canceling, suspending, and resuming the jobs. The <o:handler> tag is visual and it is rendered as HTML button. The session tables only persist until the servlet session expires or terminates. So we need to have mechanism to persistently preserve them in a permanent storage. The *persistent* attribute of the multitask tag switches archiving on and off (see Listing 1). A context server [9] provides archival facilities that store bean values and the status in a structured way.

```
<f:view>
  <h:form id="first" >
    <h:dataTable value="#{tasklist.tasks}" var="task">
      <h:column>
        <f:facet name="header">
          <h:outputText value="Handler" />
        </f:facet>
        <o:handler id=”delete” action="#{monitor.cancel}" >
            <f:param id="task" name="taskname" value="#{task}"/>
        </o:handler>
      </h:column>
    </h:dataTable>
  </h:form>
</f:view>
```

Listing 2. The *handler* tag is used with <h:dataTable> to create a table of tasks and enable cancellation actions.

Figure 3 illustrates the user interaction with the Grid beans and tags is illustrated. When user hits a submit button, CBB takes control. CBB first constructs a multitask with the components defined by the Grid tags. CBB also submits the multitask and manages its lifecycle with associated listeners. After the submission is completed, control is passed to MonitorBean shown on the right. MonitorBean interacts with the session to retrieve the information of submitted tasks.

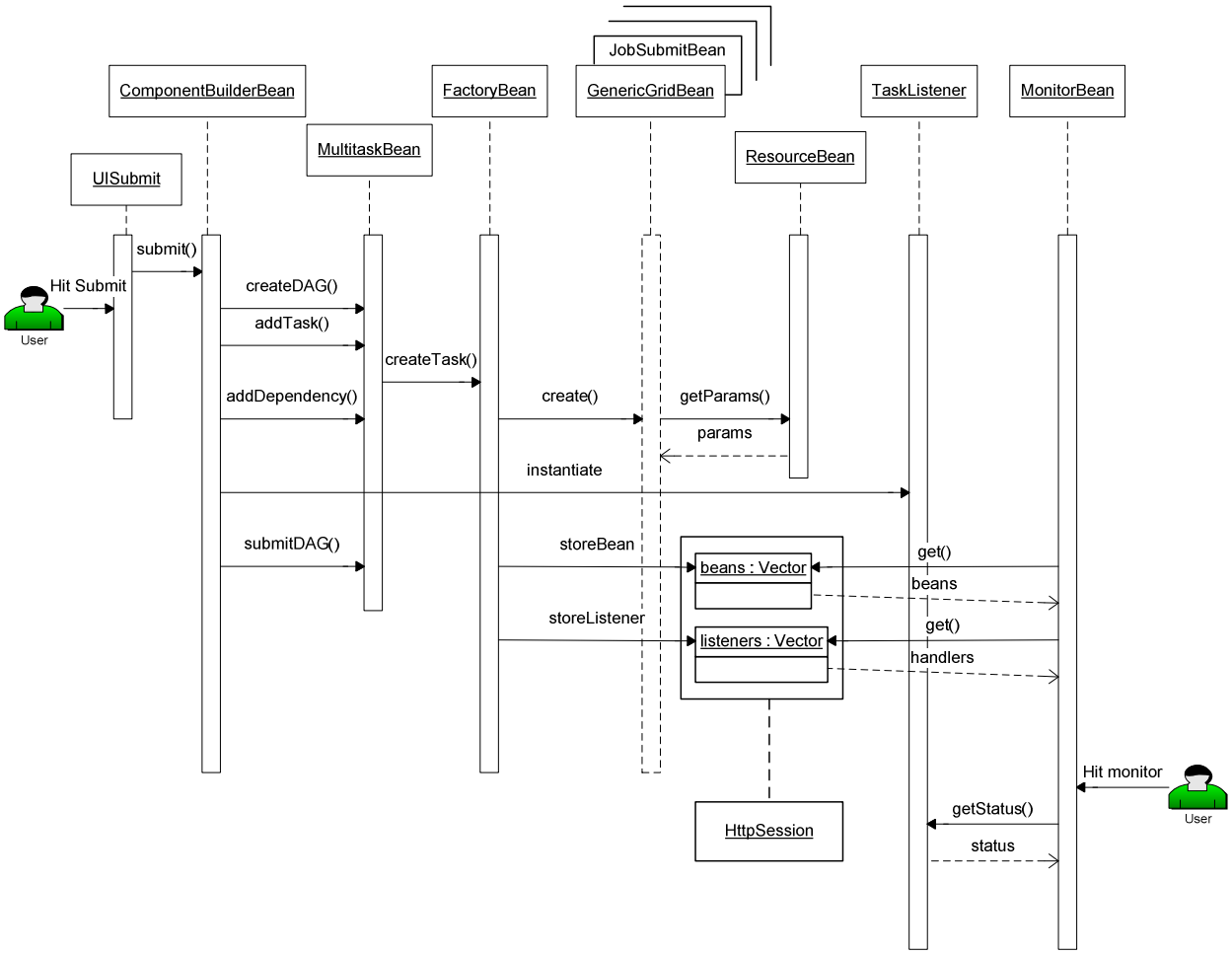

Figure 3: Sequence diagram for Grid tags and beans including user interaction.

3.6 Monitoring and management of jobs

Monitoring pages are responsible for keeping track of submitted tasks. Grid tasks usually take time to process. Consequently, managing the persistence of the tasks and archiving the results and input parameters are important for portal users. CBB provides a mechanism to store task handlers into persistent storage in the user's workspace. Monitoring pages collect status information and task parameters from user's workspace with a key named *taskname*.

In general, CBB provides status information and updates archival storage accordingly. This has an important advantage that caches the monitoring information in the session. On the other hand, CBB stores URL handlers of submitted jobs which are provided by the Globus API. A URL handler is important for persistence. In case the user logs out or a session expires, the handler can always be accessible from archive and the user can retrieve status information with it.

Monitoring pages check the status of submitted tasks. We model task with Java Bean class called JobData. Each submitted task has an associated JobData object. The collection of JobData objects is stored in a java.util.List. Job status information is displayed in HTML using the JSF HtmlDataTable component (which JSF converts to an HTML <table>). Properties stored in the JobData object include taskname, input parameters, output and error file locations, start time, finish time and status.

Portal users can manage the tasks that resume, cancel or resubmit jobs. The MonitorBean supports these capabilities for active (running) tasks. The MonitorBean allows users to manage their job archive: failed tasks may be deleted or renamed for resubmission. Successful task results and output files can be downloaded or transferred to permanent storages.

3.7 Additional Topics: Collecting User Input Values and Handling Navigation

Our Grid tags are primarily non-visual components in a JSF page that are associated with submit button actions. However, many of the tag attributes (e.g., which host to use or input file to copy) must come from user input. This is done using Web forms. Thus, Grid tags are embedded into a complete JSF page that contains a Web form that has visual input and output text elements. There are only two exceptions: the <o:submit> and <o:handler> tags are bound to a button that triggers series of actions behind the scenes. Since Grid tags are unable to get inputs from the page, we need a mediator to communicate these user-provided inputs to our Grid tags.

ResourceBean provides a simple way to represent common property values across the application. We define common property values for Grid beans such as hostname, provider, username etc. Each of these values corresponds to Grid tag attributes. Thus, ResourceBean gets its value from the Web form dynamically and assign it to the Grid tag attribute. ResourceBean enables users to enter dynamic values in the form and submit their tasks with these values.

JSF page navigation is somewhat complicated compared to JSP page navigation, as the JSF pages' links and HTML form actions do not directly point to the next page to load. Instead, JSF navigation rules for a particular web application are configured in the faces-config.xml file. Similar to standard JSF, advanced navigation controls the page with constant values as well. The <o:submit> button provides *action* attribute (see Listing 1) that assign a constant value for the destination page. Action methods and action listener methods of the <o:submit> tag are hidden from the application developers to reduce the complexity. But the navigation is left to application developers. The advantage of this architecture is that users need not wait on the submit page until it is completed. Instead they are directed to the destination page immediately (i.e., asynchronously).

4. Applications and Future Work

The Grid tags and beans described here are used by several science portal applications. The Virtual Laboratory for Earth and Planetary Materials (VLab) portal is mainly focused on computational methods of material science. In this case, scientists launch PWSCF [18] simulations and get visual results. VLab

job submission portlets enable material scientists to launch and monitor simulations through Grid portal. We have developed VLab portlets to use Grid tags and beans to facilitate issuing credentials, file operations, remote job executions and file transfers.

Grid tags are also being developed for use in the Common Instrument Middleware Architecture (CIMA) crystallography portal [19]. CIMA provides access to X-Ray crystallography, instrument and sensor data. Sample data includes CCD images of crystals as well as laboratory conditions such as temperature and humidity. The CCD images may also be post-processed. One of the post-processing applications used is SAINT [20], used to integrate CCD image frames, sort reflection lists, scale, filter, and merge reflections. In this case, crystallographers launch a SAINT application using *multitasks* to initiate an image analysis. This process results in image files that are being downloaded to a portal server and are made available for users.

In this paper, we have described the design, implementation and usage of Grid tags and beans for developing Grid portlets. This extends and improves both the interface and implementation of our previous work. Using a fine-grained component architecture enables the construction of science application specific Grid portlets in terms of reusable Grid tags. Dynamically monitoring and tracking status changes of tasks is another important aspect of Grid portals. We are considering the use of Ajax [21] technology in the next release. In the current architecture, multitasks are currently limited to "one-deep" graphs and are not recursive. We will consider adding this capability. We also need to investigate supporting the second-generation of portlet API, JSR 286 [22].

5. Acknowledgements

This work is supported by the National Science Foundation's Information Technology Research (NSF grant ITR-0428774, 0427264, 0426867 VLab) and Middleware Initiative (NSF Grant SCI 0330613) programs.

6. References

[1] Charlie Catlett, "The Philosophy of TeraGrid: Building an Open, Extensible, Distributed TeraScale Facility," ccgrid, p. 8, 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'02), 2002.

[2] Foster, I. and Kesselman, C. Globus: A Toolkit-Based Grid Architecture. In Foster, I. and Kesselman, C. eds. The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann, 1999, 259-278.

[3] Foster, I., Kesselman, C. and Tuecke, S. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International Journal of High Performance Computing

[4] Mehmet A. Nacar, Marlon Pierce and Geoffrey C. Fox Developing a Secure Grid Computing Environment Shell Engine: Containers and Service Special issue on Grid computing in Journal of Neural Parallel and Scientific Computations (NPSC), Volume 12, pages 379-390, 2004

[5] Kaizar Amin, Gregor von Laszewski, Rashid Al Ali, Omer Rana, and David Walker, An Abstraction Model for a Grid Execution Framework, Journal of Systems Architecture, Volume 52, Issue 2 , February 2006, Pages 73-87, Parallel, Distributed and Network-based Processing.

[6] Abdelnur, A., Chien, E., and Hepper, S., (eds.) (2003), Portlet Specification 1.0. Available from http://www.jcp.org/en/jsr/detail?id=168.

[7] Craig McClanahan, Ed Burns, Roger Kitain. Java Server Faces Specification. Version 1.1.

[8] Apache portal bridges Web site: http://portals.apache.org/bridges/

[9] Mehmet A. Nacar, Mehmet S. Aktas, Marlon Pierce, Zhenyu Lu and Gordon Erlebacher, Dan Kigelman, Evan F. Bollig, Cesar De Silva, Benny Sowell, and David A. Yuen VLab: Collaborative Grid Services and Portals to Support Computational Material Science Dec 30, 2005 Special Issue on Grid Portals based on SC05 GCE'05 Workshop, Concurrency and Computation: Practice and Experience.

[10] Michael Russell, Jason Novotny, Oliver Wehrens: The Grid Portlets Web Application: A Grid Portal Framework. Parallel Processing and Applied Mathematics (PPAM) 2005: 691-698.

[11] Reasonable Server Faces Web site: http://www2.caret.cam.ac.uk/rsfwiki/Wiki.jsp?page=Main

[12] Spring container Web site: http://www.springframework.org/

[13] Bayern S., JSTL in Action. Manning. 2002.

[14] IKAT Web site: http://www2.caret.cam.ac.uk/rsfwiki/Wiki.jsp?page=IKAT

[15] Jay Alameda, Marcus Christie, Geoffrey Fox, Joe Futrelle, Dennis Gannon, Mihael Hategan, Gregor von Laszewski, Mehmet A. Nacar, Marlon Pierce, Eric Roberts, Charles Severance, and Mary Thomas The Open Grid Computing Environments Collaboration: Portlets and Services for Science Gateways March 2006 Concurrency and Computation: Practice and Experience Special Issue for Science Gateways GGF14 workshop

[16] How to write your own JSF components. Web site: www.exadel.com/tutorial/jsf/HowToWriteYourOwnJSFComponents.pdf

[17] http://grids.ucs.indiana.edu/users/manacar/GridTags/GridTagsInterface/GridTagsXMLSchema.xsd

[18] S. Scandolo, P. Giannozzi, C. Cavazzoni, S. de Gironcoli, A. Pasquarello, and S. Baroni, First-principles codes for Computational Crystallography in the Quantum-ESPRESSO package, Z. Kristallogr. 220, 574-579 (2005).

[19] Hao Yin, Donald F. McMullen, Mehmet A. Nacar, Marlon Pierce, Kianosh Huffman, Geoffrey Fox and Yu Ma Providing Portlet-Based Client Access to CIMA-Enabled Crystallographic Instruments, Sensors, and Data Technical Report April 21 2006 with short poster version for 7th IEEE/ACM International Conference on Grid Computing (GRID 2006). Barcelona, Spain.

[20] SAINT Web site: http://xray.utmb.edu/saint.html

[21] Michael Mahemoff. Ajax Design Patterns. O'Reilly. 2006

[22] Stephen Hepper. Java Portlet Specification Version 2.0 Early Draft. July 2006.