

Development of GridBeans UNICORE Tutorial 25.-26.07.2007

Valentina Huber, Bastian Demuth



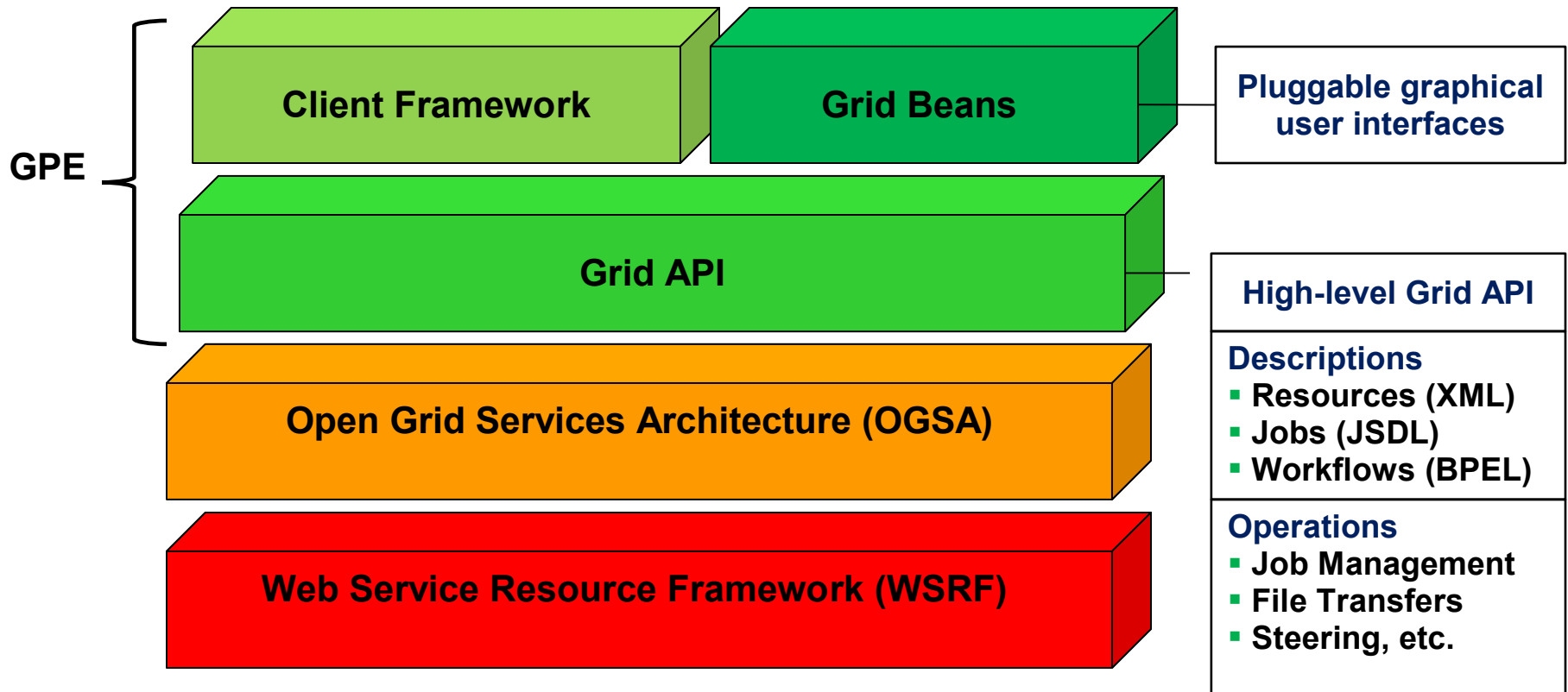
Forschungszentrum Jülich
in der Helmholtz-Gesellschaft

UNICORE Grid Programming Environment

- ▶ Grid Programming Environment (GPE) developed by Intel
 - ▶ Graphical user interfaces and interoperable GridBeans for application development
 - ▶ High-level API for programming Grid Clients
 - ▶ Language independent definition
 - ▶ Java reference implementation
 - ▶ supports Unicore-based and Globus Toolkit infrastructures



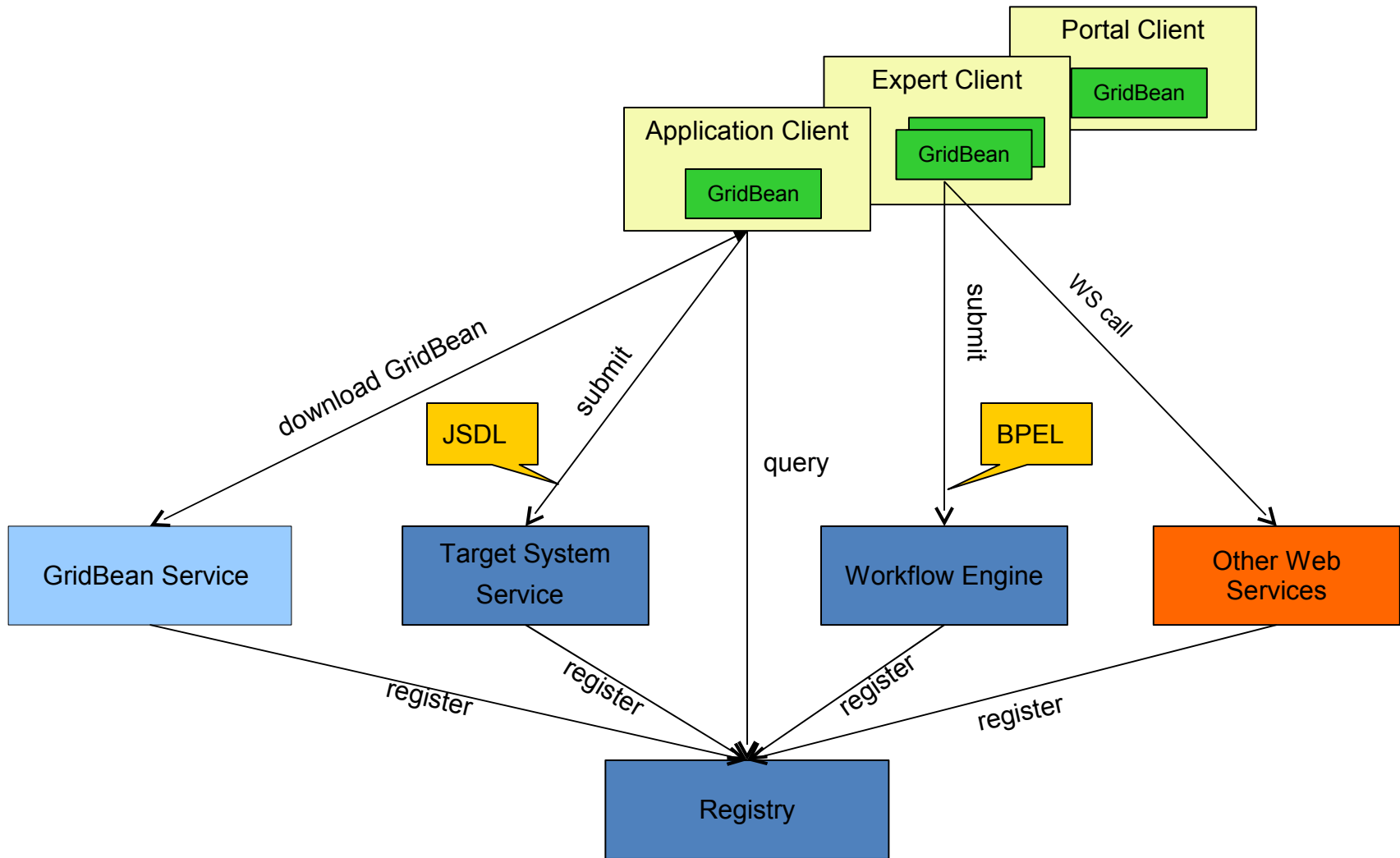
UNICORE GPE Architecture



XML = Extensible Markup Language
JSDL = Job Submission Definition Language
BPEL = Business Process Execution Language



UNICORE Process-GPE



UNICORE GridBeans

- ▶ Extensions for GRID clients
- ▶ Generate task (“job”) descriptions
- ▶ Provide pluggable GUIs
- ▶ Based on high level GRID API

- ▶ GridBean predecessor: UNICORE Plugin
- ▶ GridBeans vs. UNICORE Plugins
 - ▶ support new grid standards (JSDL, BPEL, web services, etc.)
 - ▶ independent from UNICORE clients
 - ▶ The same code could be used in application/expert/portal clients (for portal only GUI part has to be re-implemented)
 - ▶ easily distributed via GridBean Download Service

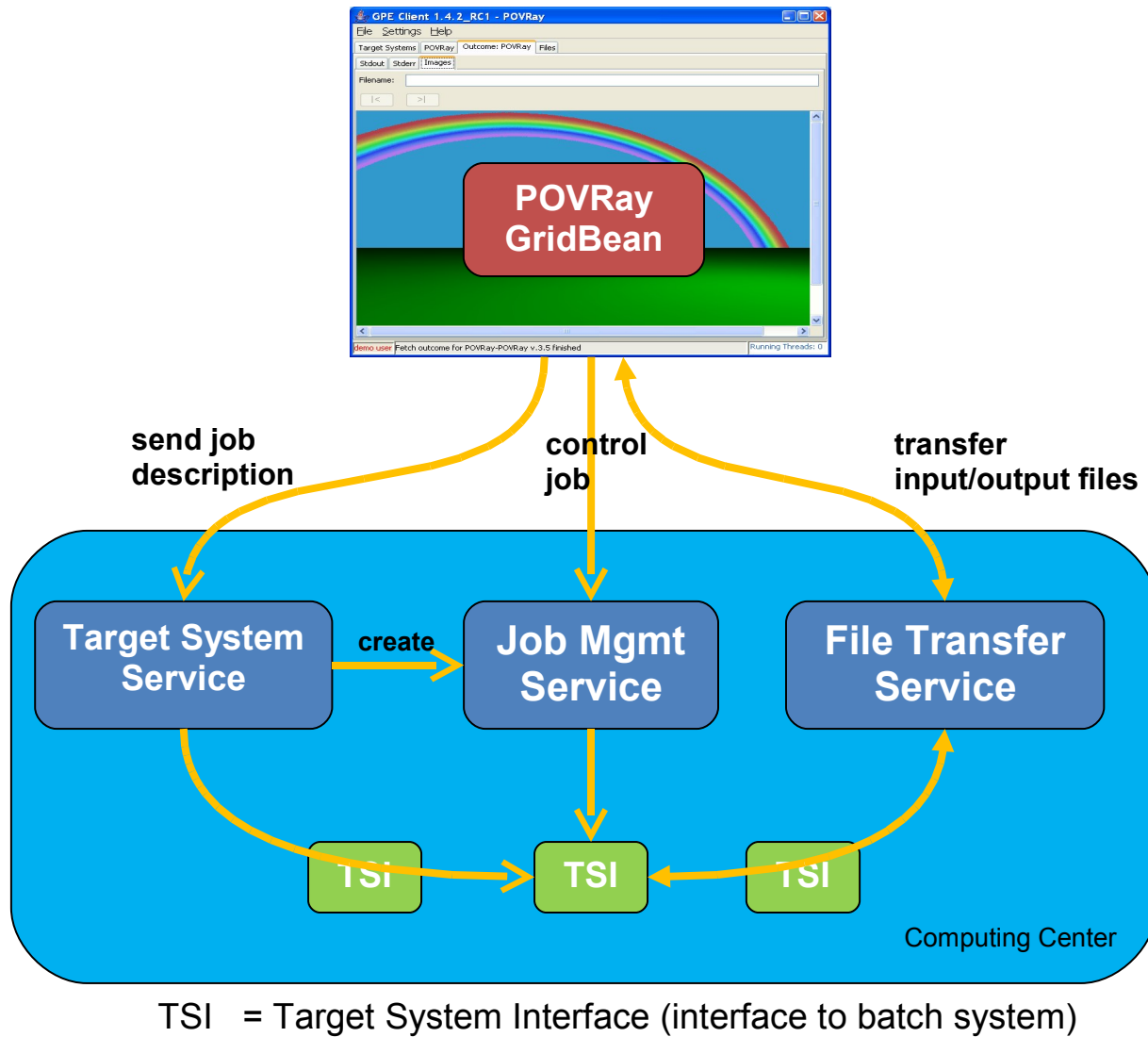


UNICORE GridBean Use Cases

- ▶ Computational GridBean
 - ▶ One-step job generator
 - ▶ GridBean is requested to create and execute JSDL document
 - ▶ Examples: POVRay, Generic, Script
- ▶ Workflow GridBean
 - ▶ Nested atomic jobs and dependencies
 - ▶ BPEL workflow submitted to the workflow engine
 - ▶ Examples: FileCheck
- ▶ Data-Staging GridBean
 - ▶ Transfer of input/output files, streaming data
- ▶ Service GridBean
 - ▶ Monitoring jobs, querying registry, invoking other web-services
 - ▶ Example: Destroy
- ▶ Custom GridBean
 - ▶ Everything else

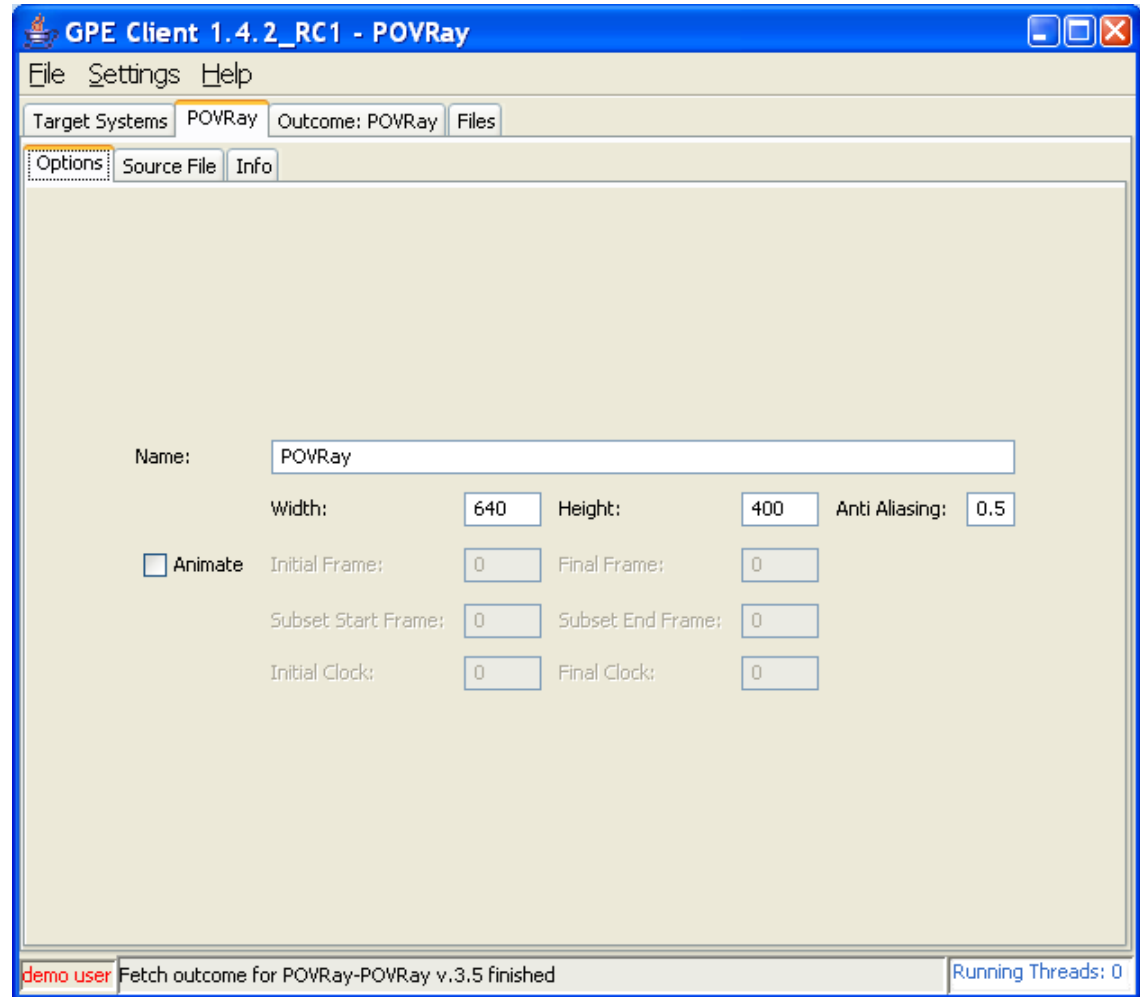


UNICORE Running a Simple Job



UNICORE GridBean Example: POV-Ray

▶ Job Preparation



UNICORE GridBean Example: POVRay

▶ Job Monitoring

The screenshot shows the GPE Client 1.4.2_RC1 - POVRay interface. The main window displays a table of job details:

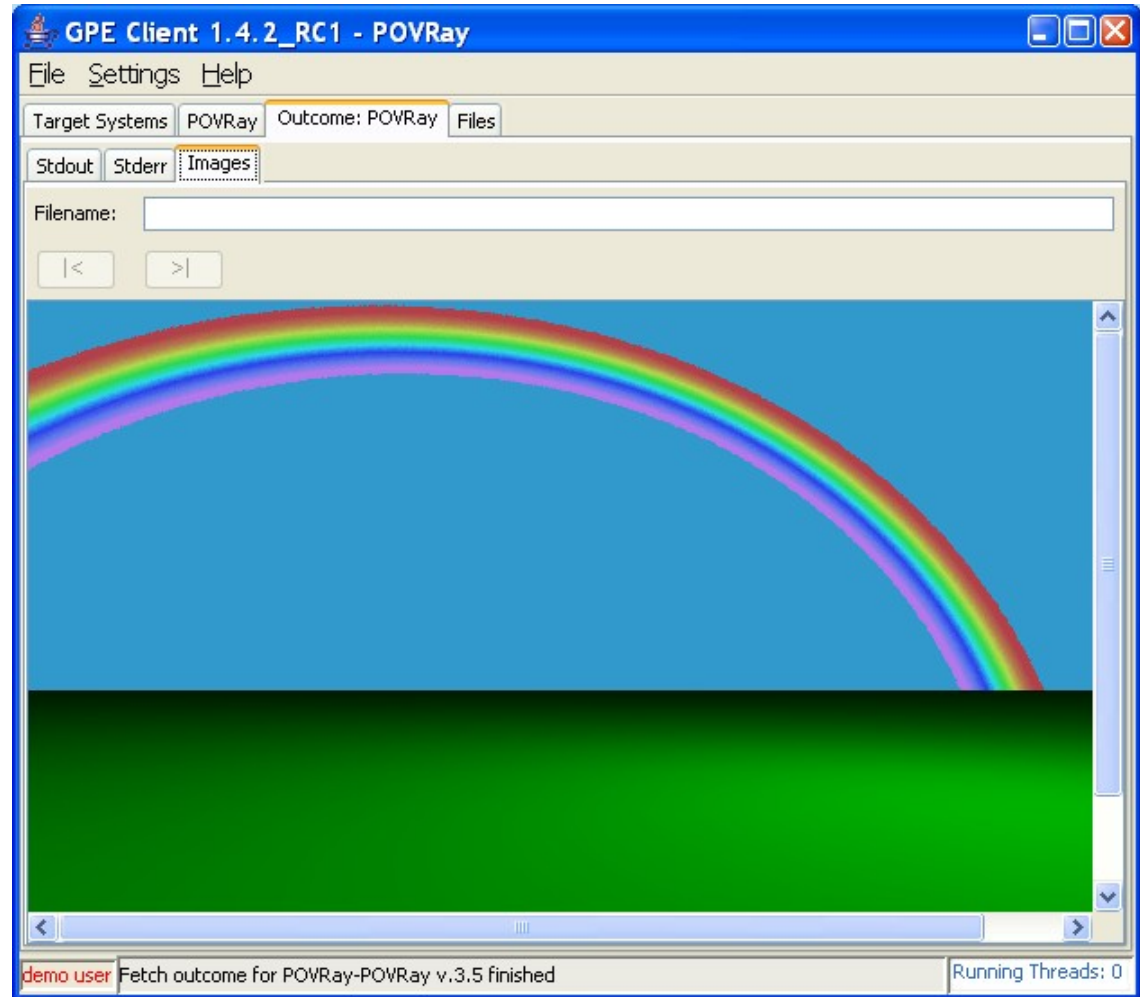
Job Name	Application	State	Submission Time
POVRay	POVRay-3.5	SUCCESSFUL	Jul 13, 2007 2:51...
POVRay	POVRay-3.5	SUCCESSFUL	Jul 13, 2007 3:00...

The interface also shows a 'Registries' section with 'zam025c14.zam.kfa-juelich.de' and a 'Target Systems' section with 'Workflow_zam025c14' and 'zam025c14'. The status bar at the bottom indicates 'demo user Fetch outcome for POVRay-POVRay v.3.5 finished' and 'Running Threads: 0'.



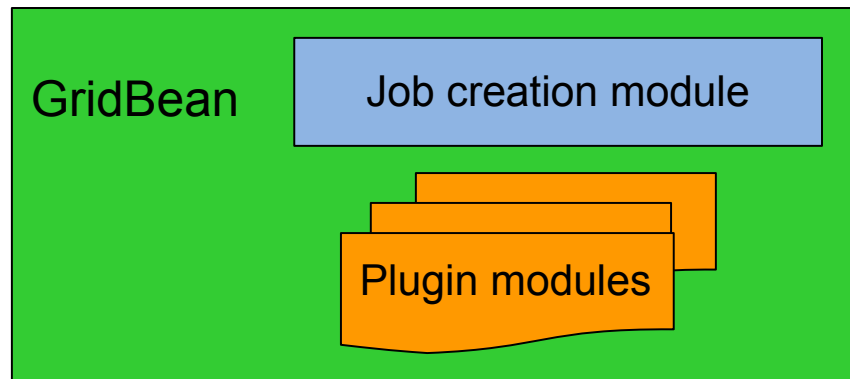
UNICORE GridBean Example: POVRay

- ▶ Fetching output



UNICORE GridBean Structure

- ▶ Nice separation of the data model and its graphical views
 - ▶ Job creation module (GridBean model)
 - ▶ Defines internal data representation
 - ▶ Translated to actual job specification (JSDL or BPEL)
 - ▶ One or more plugin modules
 - ▶ GUIs for getting user input and presenting job output
 - ▶ Differ depending on environment (e.g. application/portal clients).



UNICORE GridBean Configuration File

▶ META-INF/gridbean.xml

```
<gb:GridBeansInfo xmlns:gb="http://gpe.intel.com/gridbeans">
```

```
  <gb:Name>Script</gb:Name>
```

Name of GridBean

```
  <gb:Author>Valentina Huber</gb:Author>
```

```
  <gb:Version>1.3</gb:Version>
```

```
  <gb:Application>Script</gb:Application>
```

```
  <gb:Description>Script GridBean</gb:Description>
```

GridBean Model

```
  <gb:PluginVersion>1</gb:PluginVersion>
```

```
  <gb:GridBean>de.fzj.gpe.gridbeans.script.ScriptGridBean</gb:GridBean>
```

```
  <gb:Plugin type="gb:Swing">de.fzj.gridbeans.script.ScriptPlugin</gb:Plugin>
```

GridBean GUIs

```
  <gb:Plugin type="gb:Portlet">de.fzj.gridbeans.script.ScriptWebPlugin</gb:Plugin>
```

```
  <gb:ApplicationName>ANY_APPLICATION</gb:ApplicationName>
```

Application on target system

```
  <gb:ApplicationVersion>1.0</gb:ApplicationVersion>
```

```
</gb:GridBeansInfo>
```



UNICORE GridBean Model

- ▶ Maintains the actual state of job settings
- ▶ Stores internal data in pairs: *key* + *value*
- ▶ Keys are QNames (e.g. <width xmlns:'http://gpe.intel.com/gridbeans/povray'>)
- ▶ Values may have any type (Java Objects)
- ▶ Model must implement **IGridBeanModel** interface
- ▶ Superclass **AbstractGridBean** implements IGridBeanModel methods for storing named GridBean parameters:
 - ▶ get(QName key)
 - ▶ set(QName key, Object value)



UNICORE Job Type Hierarchy

- ▶ GridBean model creates a job of one of these types:

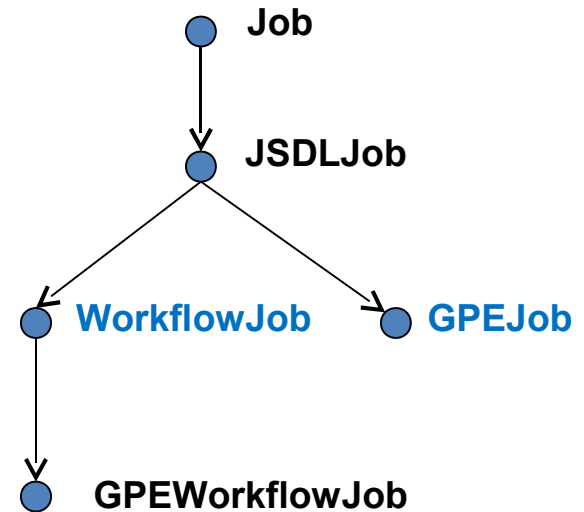
- ▶ **GPEJob** to invoke atomic job
- ▶ **WorkflowJob** to produce workflow

- ▶ GPEJob specifies the job attributes:

- ▶ Resource requirements
- ▶ Files to stage-in/out
- ▶ Application name and version
- ▶ Named application parameters (fields/options/variations)
- ▶ Application working directory

GPE IDB

- ▶ WorkflowJob specifies workflow in a subset of the BPEL language with GPE extensions



UNICORE IGridBean Interface

- ▶ Model must implement **IGridBean** interface methods
 - ▶ **setupJobDefinition(Job job)** generates JSDL document from the data in the model
 - ▶ **parseJobDefinition(Job job)** transfers values from the JSDL back into the model
 - ▶ **getInput/OutputParameters()** used for generating job imports/exports



UNICORE Plugin Interfaces/Superclasses

- ▶ Swing GridBean plugins implement `IGridBeanPlugin` interface or use the superclass `GridBeanPlugin`
- ▶ Portlet plugins are subclasses of `PortletPlugin`



UNICORE Swing Plugin

- ▶ plugins provide
 - ▶ One or more input panel(s)
 - ▶ One or more output panel(s)

```
public class ScriptPlugin extends GridBeanPlugin {
    public void initialize(Client client, INode node) {
        super.initialize(client, node);
        GridBeanInputPanel input = new ScriptInputPanel(client, getParameter());
        addInputPanel( input);
        GridBeanOutputPanel output = new ScriptOutputPanel(client, getParameter());
        addOutputPanel(output);
    }
}
```



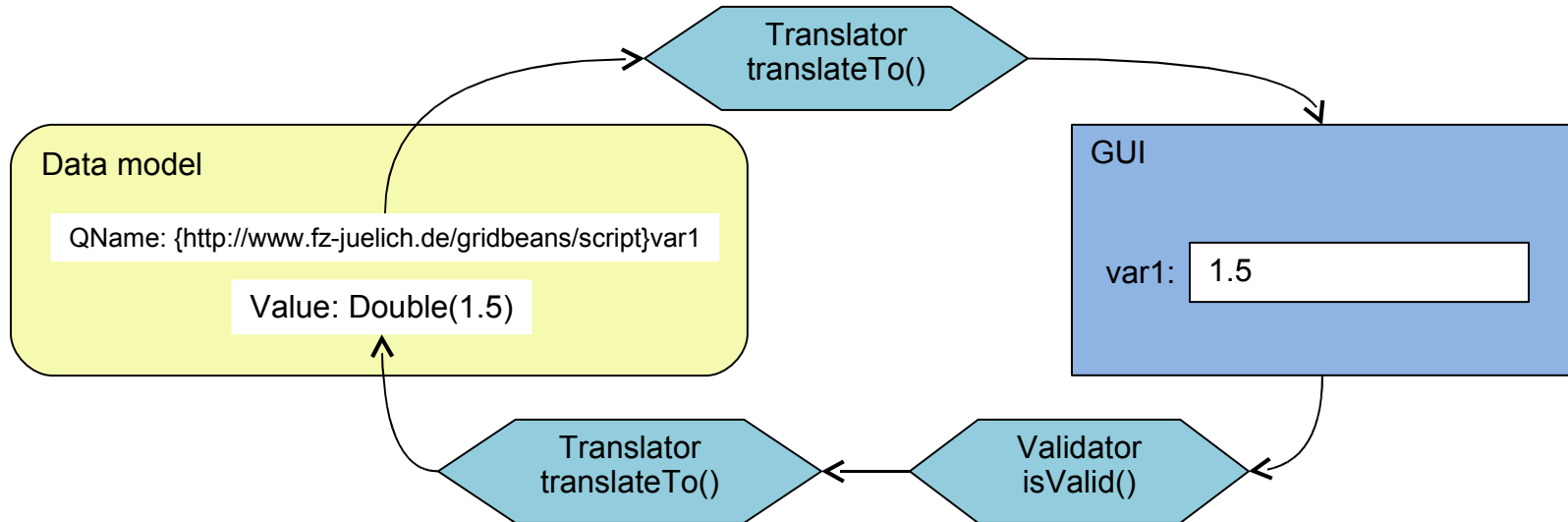
UNICORE Swing Plugin Panel

- ▶ Input panels: preparation and validation of user input
- ▶ Output panels: showing output files (e.g. images)
- ▶ Input and output panels must implement **IGridBeanPanel** interface
- ▶ Superclass **GridBeanPanel** implements IGridBeanPanel methods:
 - ▶ Component `getComponent()`
 - ▶ String `getName()`
 - ▶ void `validate(ErrorSet errors)`



UNICORE Swing Plugin Panel

- ▶ Each graphical component (e.g. JTextField) is bound to a data model element (*key + value*)
- ▶ Validators check input before writing to the model
- ▶ Translators may change data format



UNICORE Swing Plugin Panel

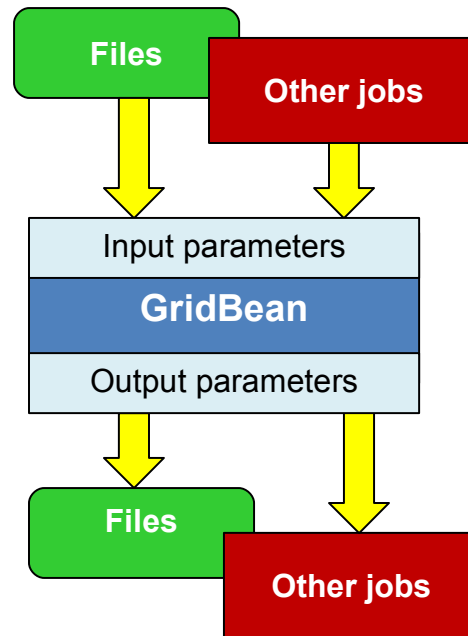
```
public class ScriptInputPanel extends GridBeanPanel {
    public ScriptInputPanel(Client client, INode node) {
        JComboBox shellComboBox = new JComboBox();
        linkComboBox(APPLICATION, shellComboBox);
        setValueTranslator(APPLICATION, ApplicationTranslator.getInstance());
        setValueValidator(APPLICATION, NotNullValidator.getInstance());
    }

    void validate(ErrorSet errors) {
        if (envField.getText().length() == 0)
            errors.addError(new Error("Please specify environment variable."));
    }
}
```



UNICORE Input and Output Parameters

- ▶ Input parameters declare input files, obtained from a file storage or other jobs
- ▶ Output parameters declare output files to be transferred to a file storage or other jobs



UNICORE Input and Output Parameters

- ▶ Four different types of input and output parameters:
 - ▶ **GPE File** (file at a local or remote location)
 - ▶ **URL** (file at a remote location represented by it's URL)
 - ▶ **XML** (arbitrary XML data from a service invocation or a workflow variable)
 - ▶ **File Set** (set of GPE Files)
- ▶ Workflow jobs can use all parameter types
- ▶ Atomic jobs can only use GPE Files and File Sets



UNICORE Further Information

- ▶ GridBean Developer Guide on GPE4GTK Project web page
<http://gpe4gtk.sourceforge.net>
- ▶ gpe4unicore and gridbeans sources
<http://sourceforge.net/projects/unicore>



UNICORE Echo Application

▶ Echo application in simpleidb

```
<idb:IDBApplication>
  <idb:ApplicationName>Echo</idb:ApplicationName>
  <idb:ApplicationVersion>1.0</idb:ApplicationVersion>
  <jSDL:POSIXApplication xmlns:jSDL="http://schemas.ggf.org/jSDL/2005/11/jSDL-posix">
    <jSDL:Executable>/bin/echo</jSDL:Executable>
    <jSDL:Argument>$ARG</jSDL:Argument>
  </jSDL:POSIXApplication>
</idb:IDBApplication>
```



UNICORE EchoGridBean Structure

- ▶ EchoGridBean.java
- ▶ EchoPlugin.java
- ▶ EchoInputPanel.java
- ▶ gridbean.xml



UNICORE EchoGridBean.java (1)

```
public class EchoGridBean extends AbstractGridBean implements IGridBean {

    public static final QName ARG = new QName(
        "http://www.fz-juelich.de/gridbeans/echo", "Arg");

    public EchoGridBean() {
        set(ARG, null);
    }

    public void setupJobDefinition(Job job) throws GridBeanException {
        if (job instanceof GPEJob) {
            GPEJob gpeJob = (GPEJob) job;
            gpeJob.setApplicationName("Echo");
            gpeJob.setApplicationVersion("1.0");
            gpeJob.setWorkingDirectory(GPEConstants.JobManagement.TEMPORARY_DIR_NAME);
            gpeJob.setId("Echo");
            gpeJob.addField("ARG", (String) get(ARG));
        }
        else {
            throw new GridBeanException("Invalid job type");
        }
    }
}
```



UNICORE EchoGridBean.java (2)

```
public void parseJobDefinition(Job job) throws GridBeanException {
    if (job instanceof GPEJob) {
        try {
            GPEJob gpeJob = (GPEJob) job;
            set(ARG, gpeJob.getField("ARG"));
        } catch (Exception e) {
            throw new GridBeanException("Failure parsing job");
        }
    }
}

public String getName() { return "Echo"; }

public List<GridBeanParameter> getInputParameters() {
    return new ArrayList<GridBeanParameter>();
}

public List<GridBeanParameter> getOutputParameters() {
    return new ArrayList<GridBeanParameter>();
}
}
```



UNICORE EchoPlugin.java

```
public class EchoPlugin extends GridBeanPlugin {  
  
    public void initialize(Client client, INode node) {  
        super.initialize(client, node);  
        addInputPanel(new EchoInputPanel(client, getParameter()));  
    }  
  
}
```



UNICORE EchoInputPanel.java

```
public class EchoInputPanel extends GridBeanPanel {

    public EchoInputPanel(Client client, INode node) {
        super(client, "Echo", node);
        try {
            buildComponents();
        }
        catch (DataSetException e) {
            client.getMessageAdapter().showException("Failure creating panel", e);
        }
    }

    private void buildComponents() throws DataSetException {
        add(new JLabel("Echo:"));
        JTextField argTextField = new JTextField(20);
        add(argTextField);
        linkTextField(EchoGridBean.ARG, argTextField);
        setValueValidator(EchoGridBean.ARG, NotNullValidator.getInstance());
        setValueTranslator(EchoGridBean.ARG, StringValueTranslator.getInstance());
        setDescription(EchoGridBean.ARG, "Echo string");
    }
}
```



UNICORE gridbean.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<gb:GridBeansInfo xmlns:gb="http://gpe.intel.com/gridbeans">
  <gb:Name>Echo</gb:Name>
  <gb:Author>Valentina Huber</gb:Author>
  <gb:Version>1.0</gb:Version>
  <gb:Application>Echo</gb:Application>
  <gb:Description>Echo GridBean</gb:Description>
  <gb:PluginVersion>1</gb:PluginVersion>
  <gb:GridBean>gpe.gridbeans.echo.EchoGridBean</gb:GridBean>
  <gb:Plugin type="gb:Swing">gpe.gridbeans.echo.EchoPlugin</gb:Plugin>
  <gb:ApplicationName>Echo</gb:ApplicationName>
  <gb:ApplicationVersion>1.0</gb:ApplicationVersion>
</gb:GridBeansInfo>
```

