Architecture Version 1.0

Introduction

This document describes the architecture of the KISTI-NCSA Science Gateway (KNSG) Framework. To fully describe this architecture, this document also provides information about external technologies that were used to build the framework (e.g. Eclipse).

KNSG provides an extensible open source software platform for building HPC software applications that are managed by PTPFlow. The KNSG frameworks provides an easy to use graphical user interface for setting up and launching HPC workflows and views to manage the execution and retrieve results. The initial framework builds upon the Eclipse Rich Client Platform (Eclipse RCP), PTPFlow and Bard.

Purpose

The purpose of this document is to outline the major components of the KNSG Framework architecture for developers who are interested in using and extending the framework for their software applications. All software extension points are documented with the eAIRS CFD workflow use case used as an example concrete implementation.

Scope

The result of version 1.0 is an initial framework for building new HPC applications with some semantic capabilities and an easy to use graphical user interface for setting up, launching and monitoring HPC workflows.

Definitions, Acronyms and Abbreviations

See the Glossary for definitions, acronyms and abbreviations used in this document.

Overview

This document should contain all of the required information for developing new KNSG applications. Where external technologies are used, there will be links to documentation for them. The Use Case section indentifies the initial goal of the project, which is to provide users with an application for setting up, launching and monitoring the eAIRS CFD workflow.

Architectural Goals and Constraints

Development Tools and Resources

Eclipse RCP/Subversion

The Eclipse RCP platform provides an plugin-based architecture for building new applications. Eclipse provides core application functionality (such as drag and drop, window management, extension points and extension capabilities, etc) that allows users to build professional-looking application, with native look-and-feel, on multiple platforms and allowing them to focus on their value-add instead of building the entire application framework from scratch.

Graphical Editing Framework

The graphical editing framework is a toolkit for graphing and editing models. You can learn more about GEF here and here.

C/C++ Development Plugin

The PTPFlow library used included in KNSG uses a small amount of functionality from the C++ Development Plugin provided by Eclipse since PTPFlow was initially intended to be just views and perspectives withing the Eclipse IDE for C+. The C+ IDE was chosen primarily for it's size (it is the smallest Ecli with an easier to use and pse download). This also illustrates the power of KNSG's plugin-based architecture. KNSG version 1.0 puts an easier to use GUI on top of PTPFlow making it easier for novice users to run HPC Workflows while still providing the full capabilities of PTPFlow for advanced users. For more information about PTPFlow, go here.

Java

JavaSE 1.6

The KNSG framework is built against JavaSE 1.6; however, it should also build with JavaSE 1.5 since no JavaSE 1.6 language specific features are being used.

Initial Use Case: eAIRS CFD

The initial use case is the eAIRS RCP application which will be built using the KNSG Application Framework. This version of eAIRS and KNSG will focus on providing an easy to use graphical user interface for setting up the eAIRS CFD Workflow and launching it with PTPFlow. After launching the workflow, PTPFlow's monitoring facilities will be used to monitor the execution of the workflow and then the framework will provide the ability to retrieve workflow results.

Data Catalog

The data catalog consists of repositories that store data used by the KNSG Framework. Below is a description of what a repository is, the API for the repository and a concrete implementation used by the framework.

Repositories

Interfaces and Base Class

IRepository

```
public IDataset retrieveDataset(DatasetId id);
public List<DatasetProperties> listAllDatasetProperties();
public boolean hasDataset(DatasetId id);
public boolean deleteDataset(DatasetProperties properties);
public DatasetProperties getDatasetProperties(DatasetId id);
public boolean isDisabled();
public void setDisabled(boolean disabled);
public boolean isWritable();
```

IDatasetImport

```
public DatasetId importDataset(DatasetProperties properties, List<URI> files);
public void saveDatasetProperties(DatasetProperties properties);
```

BaseRepository

```
protected boolean disabled = false;
public boolean isWritable() {
    return true;
}
```

Local Repository

This represents a concrete implementation of a repository on the user's local machine. It provides methods for managing their imported data on the local machine through the RepositoryView.

LocalRepository extends BaseRepository

```
// Base directory where the local repository is located
protected File baseDirectory;
// Directory containing the datasets
protected File datasetDirectory;
// Directory containing the dataset properties
protected File propertiesDirectory;
```

The repository is responsible for all data used by the KNSG Framework. Users can input data into a repository, add metadata to imported datasets, and drag and drop data onto their Scenarios. The contents of a repository are visualized by the RepositoryView. The next section will discuss in more detail about the KNSG Framework dataset objects.

Dataset

In the KNSG Framework, a dataset consists of the data and the properties (metadata) associated with the data. In the next three sections, the Dataset (containing the data), the DatasetProperties (metadata), and the DatasetBean (the concrete implementation of IDataset), will be discussed in detail.

Dataset Interface

Each dataset object should implement this interface so the system can interact with the dataset object. The KNSG Framework contains one concrete object, a DatasetBean, that generically can handle any kind of data. More specific implementations can be created if needed to handle special kinds of data.

IDataset

```
public String getFriendlyName();
public void setFriendlyName(String friendlyName);
public DatasetId getDatasetId();
public void setDatasetId(DatasetId datasetId);
```

DatasetProperties

This object stores all properties associated with a dataset, including any metadata that is used to describe the type of data.

DatasetProperties implements IUserFacing

```
private String name;
private String description;
private DatasetId datasetId;
private List<IMetadata> properties = new LinkedList<IMetadata>();
```

DatasetBean

This class represents a generic file dataset in KNSG. Datasets can be tagged as inputs for different workflows so that the analysis pages can use this information for adding the correct input to workflows.

DatasetBean implements IDataset

```
private DatasetId datasetId;
private String friendlyName;
private File data;
```

Metadata

Metadata is loosely defined as data that describes data. In the KNSG Framework, any new metadata types should implement the IMetadata interface so the system is aware of it and it is stored properly. The first section below defines the basics of the IMetadata interface and the section that follows the interface definition contains two concrete examples, TagBean and AnnotationBean for tagging and commenting datasets.

IMetadata Interface

IMetadata extends IUserFacing

```
public static final String EXT_PT = "edu.illinois.ncsa.knsg.metadata";
public String getLabel();
public String getValue();
```

Metadata Implementations

Add a tag for a dataset using a TagBean. There is no limit to the number of tags a dataset could have (e.g. eAIRS-Input, result, etc).

TagBean implements IMetadata

```
private String tag;

// metadata type
public String getLabel() {
    return "tag";
}

public String getValue() {
    return tag;
}
```

The AnnotationBean allows users to add comments to a dataset. The annotation consists of a title, comment and date the comment was made.

AnnotationBean implements IMetadata

```
private String title;
private String annotation;
private Date date = new Date();
```

Scenario

A Scenario is the container object for all things that are in a users scenario including including a list of datasets, the workflows that have been executed, and the RMI Service that the scenario uses. The scenario tracks all parts that make up each scenario and the framework is designed to support multiple scenarios. The scenario inherits from the IUserFacing interface, as do all objects in the KNSG Framework that are saved and restored as beans. Here we will only show the Scenario class, see the repository for the other classes.

Scenario extends BaseMember

```
private List<WorkflowBuilderModel> workflows;
private List<IDataset> datasets;
private ServiceInfo rmiService;
```

Managers

Managers provide access to objects in the KNSG framework and manage them. For example, the ScenarioManager allows users to build multiple scenarios simultaneously so that many what-if type analyses can be executed. Managers in the KNSG Framework should extend the BaseManager class, which provides basic services such as saving the state of the managers members and providing change listeners so objects are aware of changes to members of the Manager.

BaseManager

```
protected File stateDirectory;
// The type of members this manager manages
protected String type;
protected List<BaseMember> members;
protected IMemberChangeListenerSupport listeners = new IMemberChangeListenerSupport();
protected IMemberChangedListener myPassThroughListener = new MyIMemberChangeListener();
protected Object lock = new Object();
protected long lastSave = System.currentTimeMillis();

// BaseMember contains the change listener so adding and removing members is slightly different
protected void addMember(BaseMember member);
protected void removeMember(IMember member);
public BaseMember[] getAllMembers();
```

Scenario Manager

The ScenarioManager manages each scenario that is created and allows users to have multiple scenarios open simultaneously. It provides access to each scenario so objects can be added and removed from the scenarios. It also listens for changes so it knows when a scenario has been updated.

ScenarioManager extends BaseManager

```
private static ScenarioManager instance;
private ScenarioManager() {
   type = "scenario";
}

public Scenario getSelectedScenario();
public void addScenario(Scenario scenario);
public void removeScenario(Scenario scenario);
protected BaseMember createMember(File file);
public void removeDatasetFromAllScenarios(DatasetProperties properties);
public void removeDatasetFromAllScenarios(IDataset dataset);
```

The *type* tells KNSG where to store the state information for this manager and the objects that it manages. For example, this manager's state will be stored in user.home/NCSA/KNSG/scenario. So, if user.home was /home/foo, this would result in the state directory being */home/foo/NCSA/KNSG/scenario*

Analysis Framework

The analysis framework allows developers to define new workflows that can be executed by the KNSG framework. By specifying a workflow and a wizard and wizard pages to setup the workflow, users can easily add new analyses to the system. Each part of the framework is described in detail below.

Workflow

The workflow is an XML file containing the Ogrescript to launch a job on an HPC machine. You will need to determine which parts of the workflow you will make configurable and then use the KNSG framework to add the UI parts that will let the user specify values for the workflow. For information on Ogrescript, see the documentation here. Also, for a tutorial that takes you through the creation of a new application and sample workflow, go here. In the next section we will briefly discuss the important UI parts.

Analysis Wizard

The *AnalysisWizard* is the base class for all analysis wizards defined in the KNSG framework. New analyses should use the extension point specified to tell the framework which wizard will setup the parts of the workflow and launch the analysis.

AnalyisWizard extends Wizard

```
public static final String EXT_PT = "edu.illinois.ncsa.knsg.ui.analysisWizards";
protected Scenario scenario;
protected String jobId;
public void setScenario(Scenario scenario);
public abstract String[] getOutputs();
public String getJobId();
public String getResultPath(WorkflowInfo workflowInfo, ParsedLocalEventKey eventKey);
public abstract String[] internalGetResultPath(WorkflowDetails workflowDetails, ParsedLocalEventKey eventKey);

// Writes local file to remote machine and returns absolute path to insert into workflow script
protected String writeRemoteFile(File localFile, String remotePath, String hostURI) {
    return JobUtils.writeFile(localFile, remotePath, hostURI);
}
```

In the Eclipse extension point, you must specify the following items:

- **id** The ID of the new analysis wizard, this needs to be unique since it will be used to retrieve the wizard.
- **name** The name of the new analysis, this will be displayed for users to select which analysis to run.
- **tag** Tag for the workflow, this should match the "experimentId" of the workflow xml file because this is used to retrieve results, otherwise we don't know which results came from which analysis wizards.
- **workflow** Specify the workflow xml file that will be setup and ran by this analysis.
- **class** The wizard class with all of the logic for setting up and running the analysis.

Once you have your analysis wizard, you will need to add wizard pages that provide UI fields that will provide input to your workflow.

Views

All views must use the *org.eclipse.ui.views* extension point. By registering your view with the extension point, Eclipse will automatically make it available to users who include your plug-in in their application. Below you will find some common views provided by the KNSG Framework that can be used directly or extended to provide specific functionality, layout, icons, etc.

Scenarios View

The framework provides a *BaseScenariosView* and its concrete implementation, *KNSGScenariosView*. The *BaseScenariosView* provides basic services such as view selections, a context menu and refresh method for updating the view. The key methods are below:

BaseScenariosView extends ViewPart implements Refreshable

```
public abstract class BaseScenariosView extends ViewPart implements Refreshable {
protected Viewer viewer;

// Abstract Methods subclasses should implement
// Define what the view should look like
protected abstract void internalCreatePartControl(Composite parent);

// Handle selections within the view
protected abstract void handleSelectionChanged(ISelection selection);

// Something has changed, refresh what needs refreshing
protected abstract void handleRefresh();
```

The *KNSGScenariosView* provides a basic tree view of each scenarios objects. Default icons are provided for folders and objects contained in the scenario. By extending this class or the base class, users can use new content providers to change how objects are displayed, any additional data that gets displayed, and icons for those parts. By right clicking on a Scenario, the user can launch jobs on HPC machines using inputs from the Scenario and the Repository and when workflows complete, users can retrieve the results back to the Scenario through menus associated with the HPC job. As previously mentioned in the section regarding Scenarios, the Scenarios object is responsible for tracking which workflows were executed by it and the associated input data and result data and the ScenariosView provides the visualization of the content of each Scenario object..

Repository View

The *RepositoryView* provides a Tree view of all datasets in the *LocalRepository* and content providers that control how the datasets are displayed in the view. It also provides simple drag and drop capabilities that allow users to drag and drop datasets among views that support receiving datasets.

Users can subclass the RepositoryView or implement their own repository view if they want to display the data differently, use different icons, provide new /different functionality, etc. It is recommended that users subclass *RepositoryView* if they need a different implementation since the drag and drop capabilities would be automatically provided to the subclass. One example of a possible subclass would use more metadata to offer more advanced views and organization of the data with a simple example shown in the eAIRS *ScenariosView*. The eAIRS *ScenariosView* categorizes its data based on TagBeans. If a dataset contains a *result* tag, it will be listed in the Scenario under a sub-folder called **Results**. Many customization scenarios are possible using Metadata.

Metadata View

The KNSG *MetadataView* displays all Metadata associated with a dataset. In the KNSG Framework, both the *ScenariosView* and the *RepositoryView* are registered with the Eclipse selection provider so clicking on a dataset in either view will bring up the metadata for the dataset in the *MetadataView*. Registering new views that contain datasets with the Eclipse selection service can automatically take advantage of the *MetadataView* capabilities since it is listening for dataset selection.