FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

Bachelorarbeit: Design and Development of a Replacement for GPE in the UNICORE Rich Client

Christian Böttcher
Matr. Nr.: 4017423
Fachhochschule Aachen, Campus Jülich
Fachbereich 9: Medizintechnik und Technomathematik
Scientific Programming

Jülich, August 2017

Diese Arbeit ist von mir selbstständig angefertigt und verfasst. Es sind keine anderen als die angegebenen Quellen und Hilfsmittel benutzt worden.

Unterschrift

Diese Arbeit wurde betreut von:

1. Prüfer: Prof. Dr. - Ing. Andreas Terstegge
2. Prüfer: Dipl.-Inform. Björn Hagemeier

Abstract

UNICORE is a collection of server and client software that offers easy and uniform access to various computing resources like HPC systems. Since there are different job scheduling systems, with different job properties, UNICORE needs an abstract job model.

Part of the UNICORE Software is the UNICORE Rich Client. This is an eclipse-based graphical client, that offers views for the grid of known UNICORE servers, as well as editors for jobs and workflows. At the moment, the Rich client uses the Grid Programming Environment (GPE) for multiple client tasks. GPE as a whole is meant to help with the development of *Grid-Applications*, both on the server and client side. UNICORE uses parts of the client-side features of GPE to provide a model for job submission and editors for jobs, as well as client representations of the various UNICORE servers.

The problem is, that GPE is out-dated, over-engineered and its documentation is not very comprehensive. Because of this, code maintenance is more difficult than it should be.

This thesis aims to offer an alternative to GPE for UNICORE. This means a suitable job model, an implementation of the required operations, such as job submission, as well as the relevant GUI elements.

To make the job model as extensible as possible and to allow easy maintenance in the future, the Eclipse Modelling Framework (EMF) is used.

The EMF model can then be used with the tool EMF Forms to automatically generate GUI elements for editing a job.

In this thesis the existing code of the Rich Client will be analysed in respect to it's dependency on GPE, the relevant parts of GPE will be analysed for their function and demands for a GPE replacement will be formulated. After that, the first steps towards an implementation will be described.

Contents

List of Figures

1. Introduction

1.1. Motivation

Job scheduling systems have been around since the early days of supercomputing. As the High Performance Computing (HPC) systems and their use cases became more complex, an efficient management of jobs became more difficult. Nowadays there is a large variety job scheduling software that also runs on different hardware.

UNICORE (UNiform Interface to COmputing REssources) is, as the full name states, a uniform interface to different computing resources, that supports different job schedulers and operating systems. A more complete introduction to UNICORE is given in chapter 2.

A part of UNICORE is a GUI Client, the UNICORE Rich Client (URC). One of it's tasks is the modelling of jobs. Since different job schedulers have slightly different properties for jobs, an abstract (job-) model is required. An existing abstract model for job submission is JSDL (see section 2.2), which is already used in UNICORE. In the URC JSDL is currently implemented with GPE (Grid Programming Environment; see section 2.3), which came with additional properties. Other GPE-features that are used by UNICORE are class-representations of different grid-resources (e.g. storage or job submission systems), workflow abstraction and GridBeans (see 2.1.3).

The problem is, that GPE is over-engineered and outdated. This makes maintenance and development more difficult than necessary. To avoid these problems in the future, a replacement for GPE, that supports all relevant features of the UNICORE Rich Client, is needed.

1.2. Structure

In the next chapter important background information is given. This includes information about UNICORE and its components, the technologies used or mentioned in this thesis (such as GPE or EMF) and the exact problem definition for this thesis.

In chapter 3 the relevant UNICORE component is analysed regarding its capabilities and its usage of GPE.

Chapter 4 describes the design and implementation of the solution. In chapter 5 the process of generating the GUI with EMF Forms is described. Last, in chapter 6, a conclusion is drawn and an outlook on the future of the project is given.

2. Background

This chapter explains some background information about UNICORE and other technologies used in this paper.

2.1. UNICORE

UNICORE [**UNICOREws**] is an open-source system of server and client software. It allows easy and secure access to distributed data and computing resources.

2.1.1. Grid Resources

The UNICORE Servers provide various resources. These resources are job execution systems (called target systems), storage services, factories for target systems and storages, workflow management services and registries. All services are part of a *Grid*, which is structured as a tree (see figure 2.1).

Figure 2.1.: Example of the Resources on the Public TestGrid

2.1.2. URC

A part of UNICORE is the *UNICROE Rich Client* (URC) [**UNICOREURC**]. The URC is a graphical client that offers views for browsing the grid of known UNICORE Servers, editing and managing jobs and workflows and related functions.

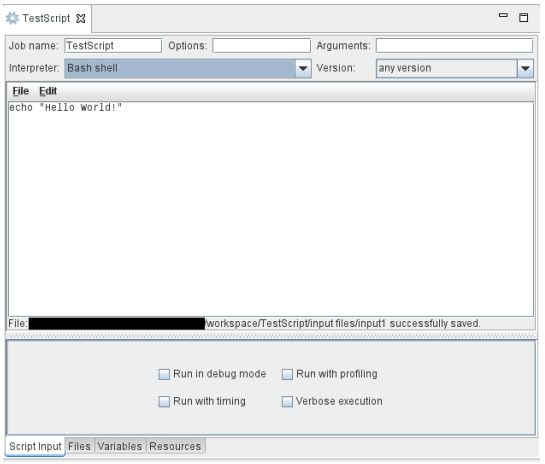

Figure 2.2.: Editor for a Job with the Script GridBean

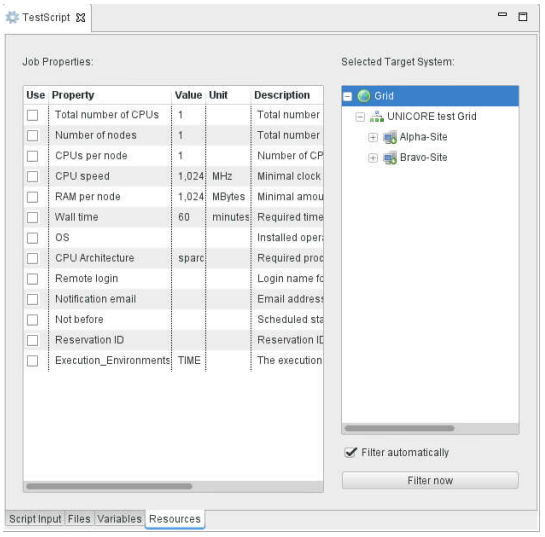

Figure 2.3.: Editor for Job Resources

Since the URC is based on the eclipse-platform, it is made up of multiple plug-ins, which define their interfaces as so called extension points. These extension points can be used by other plug-ins to extend their functionality. The plug-ins are developed independently and can be replaced easily, if the replacement extends to the same extension points.

2.1.3. GridBeans

Gridbeans are templates for UNICORE Jobs. These templates can support multiple different applications and application versions. A Gridbean may also add additional GUI elements to make the job-editing process for specific applications easier. The Gridbean implementation and concept are taken from GPE.

2.2. Job Submission Description Language (JSDL)

JSDL [**JSDLSpecification**] is an XML specification that is used to describe resource requirements and execution information for job submissions. It does not describe the state of a running job. Instead, JSDL describes resource requirements, job name and description, required file transfers before and after job execution and the command to execute.

JSDL is extensible, making is possible to add custom job requirements that are individual to the executing system.

```
<?xml version="1.0" encoding="UTF-8"?>
<jsdl:JobDefinition
 xmlns:jsdl="http://schemas.ggf.org/jsdl/2005/11/jsdl" xmlns:jsdl-posix="http://schemas
    .ggf.org/jsdl/2005/11/jsdl-posix"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <jsdl:JobDescription>
    <jsdl:JobIdentification>
      <jsdl:JobName>An Example JSDL Job</jsdl:JobName>
      <jsdl:Description>
        A JSDL file that contains several different JSDL elements for an example job.
      </jsdl:Description>
    </jsdl:JobIdentification>
    <jsdl:Application>
      <jsdl:ApplicationName>example-application</jsdl:ApplicationName>
      <jsdl-posix:POSIXApplication>
        <jsdl-posix:Executable>
          /bin/example
        </jsdl-posix:Executable>
        <jsdl-posix:Argument>-a argument</jsdl-posix:Argument>
        <jsdl-posix:Input>stdin</jsdl-posix:Input>
        <jsdl-posix:Output>stderr</jsdl-posix:Output>
      </jsdl-posix:POSIXApplication>
    </jsdl:Application>
    <jsdl:Resources>
      <jsdl:IndividualPhysicalMemory>
        <jsdl:LowerBoundedRange>1073741824.0</jsdl:LowerBoundedRange>
      </jsdl:IndividualPhysicalMemory>
      <jsdl:TotalCPUCount>
        <jsdl:Exact>2.0</jsdl:Exact>
      </jsdl:TotalCPUCount>
    </jsdl:Resources>
    <jsdl:DataStaging>
      <jsdl:FileName>stdin</jsdl:FileName>
      <jsdl:CreationFlag>overwrite</jsdl:CreationFlag>
      <jsdl:DeleteOnTermination>true</jsdl:DeleteOnTermination>
      <jsdl:Source>
        <jsdl:URI>http://example-server.com/files/stdin</jsdl:URI>
      </jsdl:Source>
    </jsdl:DataStaging>
  </jsdl:JobDescription>
</jsdl:JobDefinition>
```

Listing 2.1: Example JSDL File

2.3. Grid Programming Environment (GPE)

GPE [**GPEWebsite**] [**GPESF**] is an open-source software system that offers support for developing grid-applications independent of the underlying grid-middleware. It includes components for building application-specific user interfaces, service registries and basic grid services that may interface with other grid technologies.

The relevant parts of GPE for this thesis are several client-packages and interfaces. This includes the GridBean concept, job and JSDL abstraction and client-implementations for various UNICORE grid resources.

A more detailed analysis of the capabilities of GPE will be given in 3.3.

2.3.1. Reasons for Replacing GPE

There are several reasons for GPE not being desired anymore:

First of all, GPE is over-engineered. In some places are several unnecessary layers of abstraction that are never used anywhere within GPE. This includes multiple instances of *top-level-interfaces* that don't contain any functions. This leads to many **instanceof** conditions, which defeats the purpose of Object-Oriented-Programming.

An example for this is shown in figure 2.4. GPE differentiates multiple kinds of Jobs, but two classes contain all implementation. In practice, all jobs are passed to functions as a Job object. This requires conditions like

```
if (job instanceof JSDLJob) {
        ((JSDLJob) job).executeJSDLMethod();
}
```

when different job features are to be used.

Since UNICORE only has one kind of job, the Interface-hierarchy in GPE makes maintenance more difficult than necessary and the code more confusing.

An other reason for replacing GPE is the sparse documentation, which has not been updated since 2008. Since only the basic functions of the client and GridBean-API are described and the quality and amount of in-line documentation can range from *none at all* to *well-documented file*, working with the GPE API can be difficult.

Since the URC uses GPE in several places, development in the URC is is more difficult than it should be. This would also facilitate future extensions of the URC that surpass the GPE functionality.

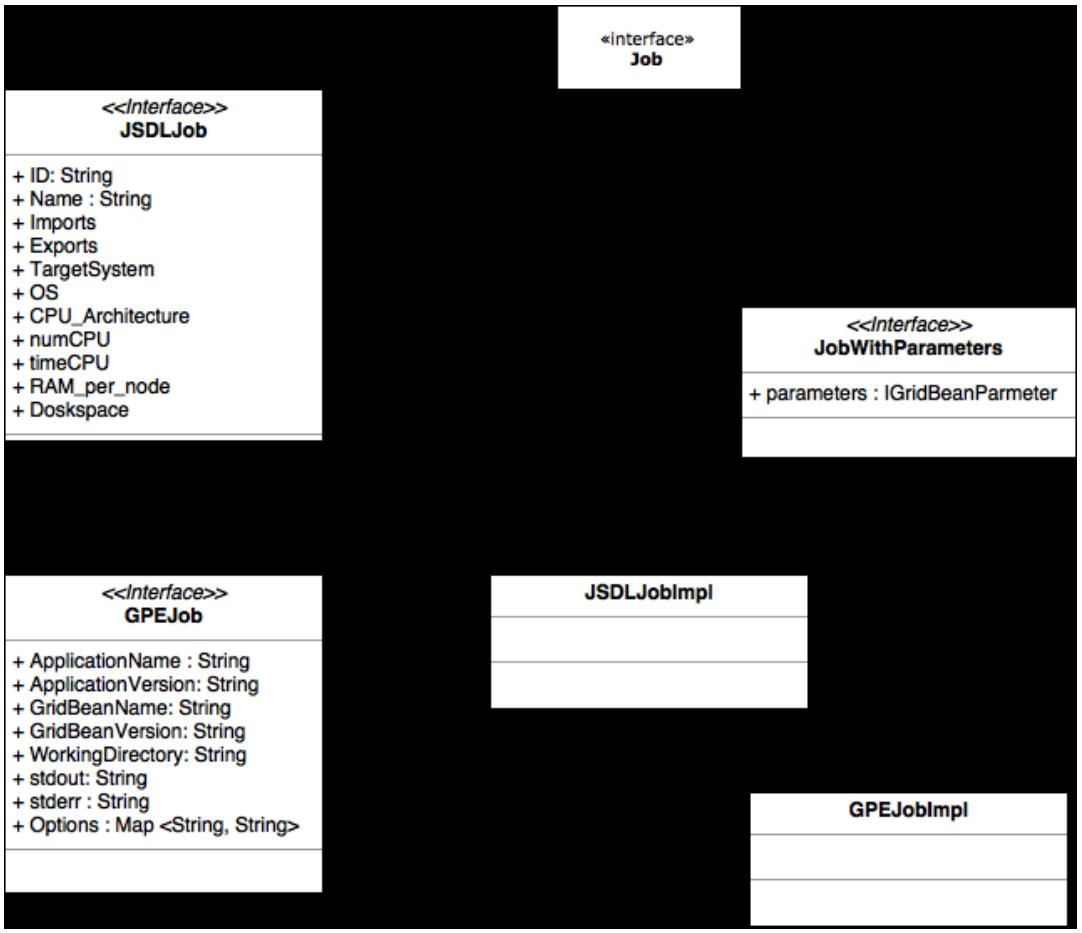

Figure 2.4.: Job Interface Hierarchy

2.4. Eclipse Modelling Framework (EMF)

EMF [**EMFBook**] is a framework that models Java-APIs. EMF allows to generate XML schemas, UML diagrams and Java Interfaces and implementations from the modeled API. The API may be modeled in the EMF ecore format, in XML or UML or in annotated Java Interfaces, and EMF is capable of generating the other formats.

This allows the developer to focus on the design of the API, as EMF is capable of re-generating the Java code if the model is changed. The re-generating of the code preserves manual changes, such as implementations of functions.

The generated code does not only include Interfaces, but also a basic implementation with getters and setters, as well as factories and code for instance (singleton) management.

EMF-generated code also includes notification of *Observers/ Listeners* when attributes are changed. This allows the same object to be edited by multiple means (e.g. GUI views, background thread) at the same time, with all updates propagating to the other editors.

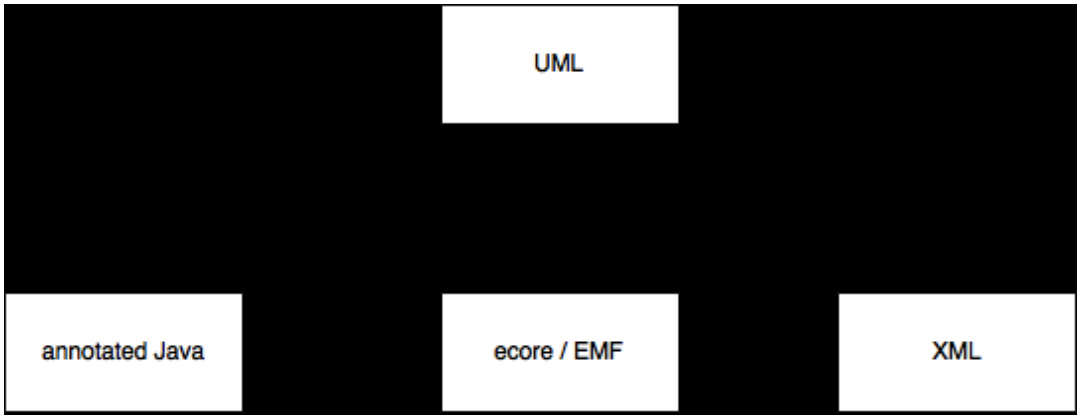

Figure 2.5.: Ecore Relations

2.5. EMF Forms

EMF Forms [**EMFFormsWebsite**] is a part of the EMF Client Platform (ECP), which is a framework for building EMF based client applications.

EMF Forms provide a way for generating form-based GUIs based on an EMF model.

It can generate GUI-elements for multiple platforms, including Java Swing and SWT. This allows an easy inclusion of the generated elements in a graphical Java-client.

As the GUI is generated automatically, changes to the underlying EMF model only require a single re-generation to rebuild the GUI-elements.

To allow for more complex attributes, or attributes wich contain non-EMF classes, to be edited, custom controls can be created and added to an EMF Forms model.

In the EMF Forms context, controls are GUI elements that edit a specific attribute of the corresponding EMF model. There are default controls available for Javas primitive data types, as well as for the default EMF types, which usually consist of a label and a text field for editing the property. EMF Forms also supports various controls for Lists.

EMF Forms also supports references among objects, so that additional views can be opened to edit complex attributes, with the main object still being updated.

2.6. Problem definition

GPE is a dependency for multiple URC plug-ins. Because of this, changes to these plug-ins require usage of GPE. Since GPE is outdated and over-engineered, this is more complicated than it should be.

To solve this, a replacement for GPE for usage in the URC is desired. This replacement could then directly use the UNICORE Java-API to make URC development easier and remove several layers of unnecessary abstraction that are inherent with GPE.

Since GPE is widely used in the URC, a step-by-step approach to designing the replacement is necessary. Currently GPE handles Job and workflow abstraction, as well as several client operations and (remote) file handling. The first step will be the design of a new jobmodel, as well as a library for handling client requests. These features are currently included in the URC plug-in *gpe4eclipse*.

The first step can be divided into several parts: First an analysis of the required capabilities of the replacement, since not all GPE capabilities are utilized by the plug-in. Then, the design of the jobmodel, with respect to the JSDL specification and additional properties defined by UNICORE. Last, the design and implementation for the required

client-classes, as well as the integration with the jobmodel.

Ideally, the replacement would be compatible with the XML-format that GPE uses to describe Jobs and GridBeans, so that transition is as easy as possible, but this is not the highest priority.

The new library will be designed in a way that allows for relatively easy extension of its features, as well as easy maintenance. It should also include the required GUI elements for editing and viewing the modelled jobs.

3. Analysis of the Desired Capabilities

GPE is used by three URC plug-ins: org.chemomentum.rcp.wfeditor, org.chemomentum.rcp.common and de.fzj.unicore.rcp.gpe4eclipse. A short analysis of the interaction between GPE and the respective plug-ins follows.

3.1. Relation between GPE and the URC

GPE is made up of several projects that depend on each other. One of these projects is called *gpe4unicore*. This project contains implementations of the GPE API that use the existing UNICRE Java API. There are also many similar class names in GPE, such as TargetSystemClient (GPE) and TSSClient (UNICORE API) or the JobClient (both). This can lead to confusion when analysing the sourcecode.

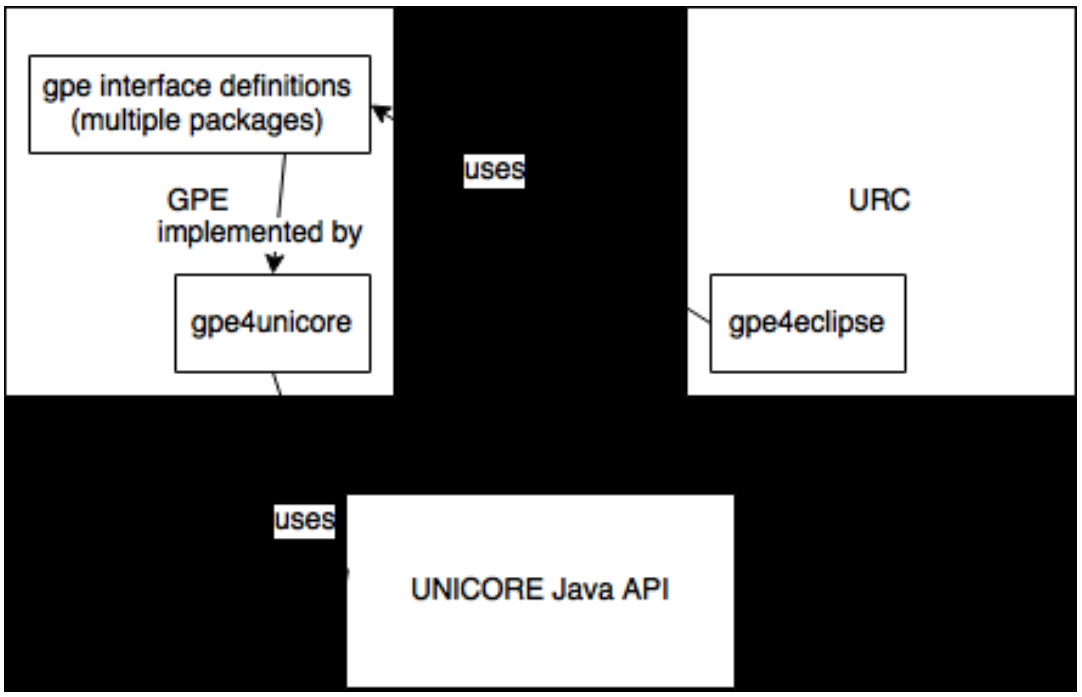

Figure 3.1.: Relation between GPE and URC

3.2. Analysis of the gpe4eclipse Plug-in

The gpe4eclipse plug-in contains 30 packages and sub-packages. The common prefix is de.fzj.unicore.rcp.gpe4eclipse.

These packages contain classes that take care of several tasks. Among this are classes to synchronize the supported GridBeans and Applications with the respective Registry, UNICORE client operations like communicating with the Registry or TargetSystemService and submitting jobs. It also contains classes for the various job properties that are supported by UNICORE and not necessarily included in GPE:

- Site-Specific Requirement
- Reservation ID
- Schedule
- User Email
- XLogin
- Execution Environment

Another big part are the GUI-elements for the various types of Requirement-Values:

- Boolean
- DateTime
- Enum
- RangeValueType (complex type defined by JSDL)
- String

The plug-in also contains multiple classes that are responsible for *File-Stageing*, meaning the copying of files to and from the job-directory before and after execution. These files can be copied from and to anywhere in the UNICORE-Grid or the clients system.

The rest uses the aforementioned parts to deal with job creation, editing, submission and the management of submitted jobs.

In addition to this, the plug-in also contains multiple classes marked as obsolete, and other classes, that are never utilized anywhere.

3.3. Analysis of the used GPE Elements

GPE offers many features, but not all of them are used by the *gpe4eclipse* plug-in. In this section all GPE features that are used by it will be analysed.

As a first step, all GPE-classes that are imported by the plug-in are listed (see A.3).

The GPE classes can be sorted by package and by function, although several classes fit into multiple categories:

- GUI classes
  These classes deal with GUI elements that are provided by GPE. This does not need any analysis beyond their existence, since the GUI will be done with EMF Forms in the future.
- GridBean classes
  These classes provide functionality surrounding GridBeans. This includes The GridBean-Interface and the required Clients for downloading GridBean information.

- Application classes
  These classes mainly consist of the Application Interface and its implementations. Application mainly holds information on a single application, as well as its version, description and information about required input/output files.

- Job and JSDL classes
  These classes provide job-representation, jsdl abstraction and related functions. They contain the hierarchy shown in figure 2.4, though not all implementations are directly used in the *gpe4eclipse* plug-in. They also contain the complex RangeValueType from JSDL as well as abstraction for all JSDL-job-properties.

- Client operations
  These classes contain all client-functionality that is used from GPE. This includes all UNICORE-grid related operations, as well as GridBean-downloads and several remote file operations.

- (Remote) File classes
  These classes are also client-classes, but only deal with file management. The managed files can be in various locations, from the local system to remote systems as well as dynamic locations, such as the working directory of jobs, which isn't defined until job submission.

- Utility classes and Exceptions These classes contain Exceptions or constants and utility functions, that are used by other classes and don't fit into other categories. They offer no additional functionality.

This means that the GPE-replacement has to contain a working jobmodel that contains all JSDL and GPE Job-elements, a GridBean and Application-abstraction that is ideally compatible with the GPE version, a working library of client-classes for the client-operations that were formally done with GPE and file abstraction for local and remote files. The GUI and utility classes, as well as the Exceptions, can be discarded, as they are either unnecessary or have to be rewritten for the new models anyway.

3.4. rcp.common and rcp.wfeditor

The package org.chemomentum.rcp.common only uses a few classes from GPE and all of them are utility classes or contain constants. These can be rewritten or copied when reworking this plug-in to work without GPE.

When rewriting the workfloweditor, it will have to use the new jobmodel. Because of that, a detailed analysis and a design for necessary features of the replacement will have to wait until the jobmodel is finished.

4. Design

4.1. Jobmodel

The jobmodel consists of an abstraction of a *Job* and contains all elements necessary to describe a job, including *GridBean* and *Application*.

4.1.1. Usage of EMF

To make the design process and future maintenance as easy as possible, EMF was used to model jobs. As jobs are to be edited in the URC, using EMF allows the generation of the GUI elements with EMF Forms. This also removes the need to manually write the Interface and implementation code, especially in the future, when the model may need to be altered.

4.1.2. Design

A job can be modelled by a set of attributes. This set of attributes is taken from multiple sources, the GPE Job model, JSDL and the attributes visible in the URC, which are then merged.

The JSDL specification describes the following attributes:

- job name
- job description
- application name
- stdin
- stderr
- executable (path)

- arguments
- import files
- export files
- desired execution system
- OS
- CPU Architecture
- CPUs per node
- CPU time per node
- total CPU count
- RAM per node
- CPU speed
- disk space per node

The GPE job is described by an empty interface with some inheriting interfaces. All Job-classes in GPE combined define the following attributes (attributes already mentioned in JSDL removed):

- job Id
- application version
- GridBean name
- GridBean version
- working directory
- environment variables

In the URC the following job attributes are described (attributes already mentioned removed):

- start time

- reservation id
- notification email
- username

These lists are then combined with job id being merged into job name (combined list is in section A.3).

Some of these attributes can be grouped together:

Desired execution system, OS, CPU Architecture, CPUs per node, CPU time per node, total CPU count, RAM per node, CPU speed and disk space per node can be grouped as JobRequirements, that set requirements of the execution system.

GridBean name and version can be replaced by a single reference to a GridBean object. The same can be done for the application. This results in the following Class-structure:

Figure 4.1.: UML class diagram of Job, JobRequirements, GridBean and Application

4.1.3. Additional Types

As several attributes of Job, JobRequirements, GridBean or Application are more complex than the primitive data types in Java, additional Types have to be created:

- GridFile
  GridFile is necessary to make references to files that are not already in the job-directory. GridFile is an alias for the existing org.unigrids.services.atomic.types.GridFileType, and can refer to any file on a storage in the UNICORE grid.

- DataStage
  DataStage is an abstraction for file imports or exports. It has a name and a *delete on termination* flag. Inheriting from DataStage are the classes StageOut and StageIn . These contain a GridFile attribute for the remote file and a String attribute for the name of the corresponding file in the job-directory. Exports (StageOut) also have an *overwrite* flag for the destination.

- CPUArchitecture
  CPUArchitecture is an enumeration that contains various CPU architectures. It includes all JSDL-mandated entries, as well as several additions that were included in GPE (see listing A.2).

- OSType
  OSType is an other enumeration. This contains various operating systems. As in CPUArchitectures, all JSDL mandated types and GPE additions are included(see listing A.1).

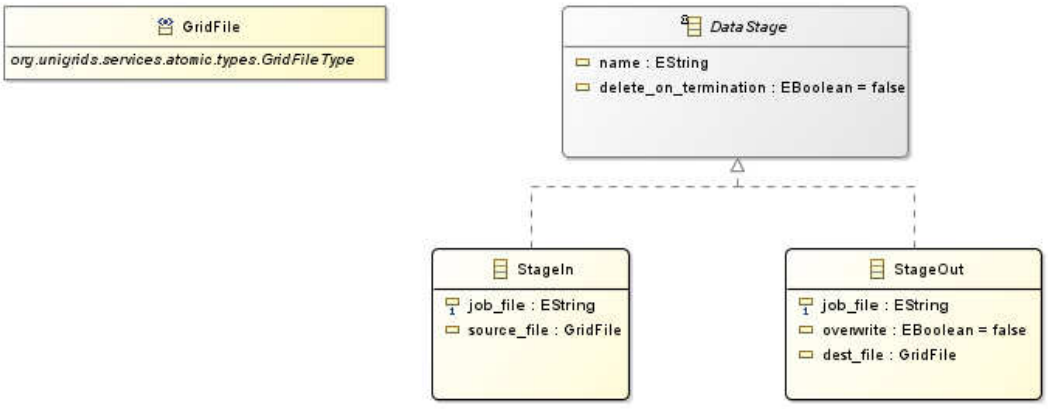

Figure 4.2.: UML class diagram of the stage sub package

- JSDLType
  JSDLType is an abstract class representing any custom JSDL property. As JSDL is extensible, the jobmodel has to support custom properties. The only attribute of JSDLType is a function that returns it as a JSDL-compatible String.

- StringValueType
  StringValueType is a simple wrapper for a Java String. It is an extension of JSDLType that allows for additional String properties.

- RangeValueType
  RangeValueType is another extension of JSDLType. It contains one or more elements of the following types:

  - UpperBounded Consists of a float value as the upper bound u and a boolean if the bound is exclusive; The resulting interval is $I = [-\infty; u]$ or $I = [-\infty; u)$
  - LowerBounded Consists of a float value as the lower bound l and a boolean if the bound is exclusive; The resulting interval is $I = [l; \infty]$ or $I = (l; \infty]$
  - Exact
    Consists of an exact float value v and a float delta d; The resulting interval is $I = [v - d; v + d]$
  - Range Consists of a single LowerBound and a single UpperBound; The resulting interval is $I = [l; u]$;

  A given (float) value matches a RangeValueType, if it matches any part of the RangeValueType.

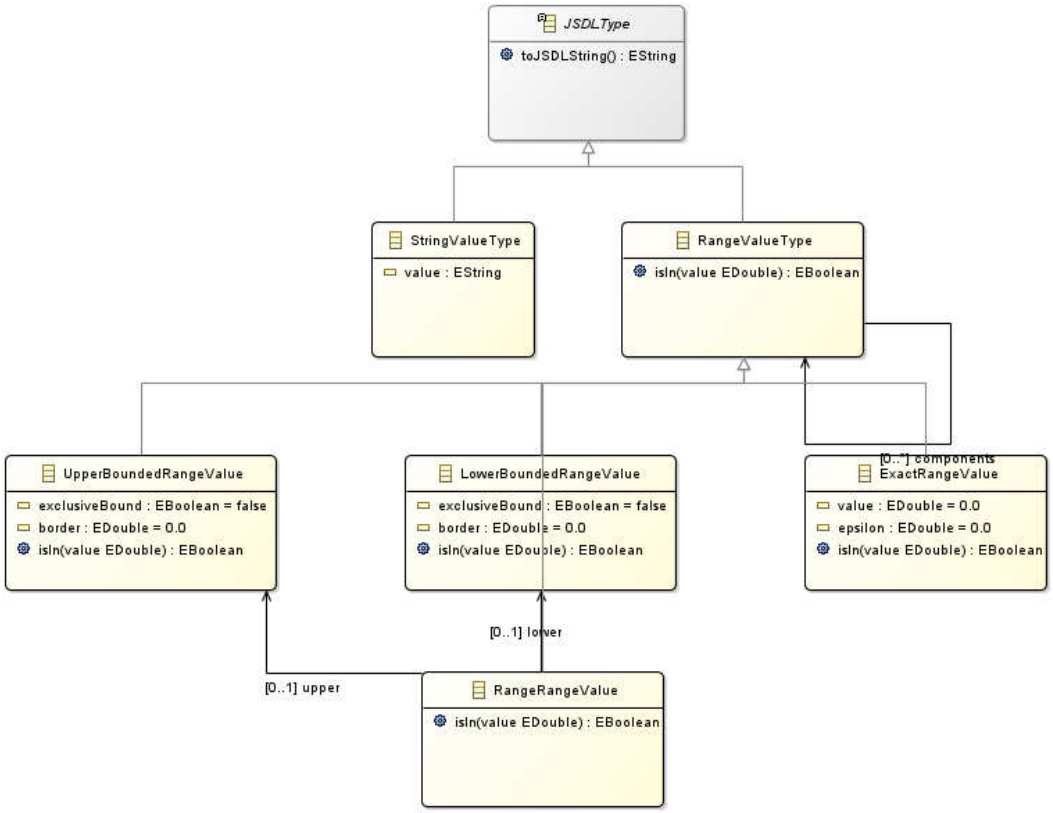

Figure 4.3.: UML class diagram of the jsdl sub package

4.1.4. Additional Properties

As JSDL explicitly allows custom properties to be passed as requirements, the jobmodel has to support user defined attributes as well. This is done by adding a Map<String, JSDLType> as a property of JobRequirements. Additional properties can then be edited and stored in this map.

4.2. Design of the Gridmodel

All Grid-elements have common attributes:

- name
- URI
- termination time

This leads to a base class GridResource that describes a general UNICORE grid element. As these attributes are requested the same way for all types of grid resources, this class can be fully implemented.

4.2.1. Types of Grid Resources

The different types of grid resources are implemented in respective classes that inherit from GridResource. The functionality of the different classes is a merge from GPE functionality and the UNICORE Java API.

- Storage
  Storages are remote storages that contain files and directories. The class representation must have functions for common file operations (list, create, delete, move, copy, chmod), as well as up- and download of files.
- TargetSystem
  The target system is the system that executes jobs. It must have a submit feature, as well as functions that return supported applications, attached storages or submitted jobs.
- StorageFactory
  The storage factory is responsible for creating new storages and listing all storages it created.

- TargetSystemFactory
  The target system factory is responsible for creating new target systems and listing all target systems it created.

- SubmittedJob
  Submitted jobs are shown as grid-elements under target systems. The class SubmittedJob inherits from GridResource and Job. It must have functions for requesting the status of the job (e.g. running or completed), as well as controlling the job (pause, resume, abort, restart) and getting additional job information (get submit time, get estimated end time, get job storage).

- Registry
  The registry returns all services that are registered with it. That means that the class representation must have functions to list all known target system factories and all known storage factories. No function for storages or target systems is needed, as they are not directly contained in the registry, but accessible by their factories. The registry class should also have functions for filtering the factories by various attributes such as supported applications or name.

4.2.2. Implementation

Like GPE, these classes make use of the existing UNICORE Java API, and mostly only have to match the jobmodel data to the API data format.

SubmittedJob

As Java does not allow classes to inherit from multiple other classes, SubmittedJob could not inherit the implementation from both GridResource and Job. To solve this, the interface IGridResource was created. It contains all functions of GridResource.SubmittedJob then implements the Job and IGridResource interfaces, while inheriting the implementation from JobImpl. The IGridResource functions are then delegated to a GridResource member that is created in the constructor.

```
package de.fzj.unicore.rcp.gridmodel;

public class SubmittedJob extends JobImpl implements Job, IGridResource {

        private IGridResource delegateGridResource;
        ...
        public SubmittedJob(EndpointReferenceType epr, IClientConfiguration icconf, Job
             j) {
```

```
                super();
                this.delegateGridResource = new GridResource(epr, icconf);
                ...
        }
        ...
        @Override
        public String getURI() {
                return delegateGridResource.getURI();
        }
}
```

Listing 4.1: Example for Delegation in SubmittedJob

5. GUI generation with EMF Forms

To make use of the jobmodel in the URC, GUI-elements are required, so that the user may edit and manage jobs. As the jobmodel was created with the EMF, tools such as EMF Forms can be used to generate most necessary elements.

5.1. Alternatives to EMF Forms

While it does ultimately not matter how the URC GUI is implemented, there are some arguments that support the usage of EMF Forms. First, EMF Forms works well with EMF/ ecore, which means that the new jobmodel can be directly used. While this is supported by other tools (e.g. Extended Editing Framework (EEF)), EMF Forms has the ability to directly integrate it's views into the Eclipse Client Platform, on which the URC is based, whereas other tools generate GUI code, which will then have to be manually handled. While EMF Forms too requires some manual work, this will be done with so-called renderers, which render the GUI from the view-file at runtime. This means that small changes to the view will be directly integrated and require no re-generation of the GUI code. Another feature of EMF Forms is the automatic content validation for generated GUIs.

5.2. Required GUI Elements

The only element of the jobmodel that should be directly edited by the user is the Job. But not all job properties need to be editable by the user. The following job properties should not be writeable to the user:

- GridBean
  The GridBean is defined at job creation. Since it defines compatible applications and may define additional job properties and requirements, changing it in an existing job is impractical, since this would require a re-rendering of the view.

- working directory
  The working directory is set server-side after job submission. This is only relevant in SubmittedJobs, to access the job files.

For all other job properties a suitable editing element should be created.

The GUI elements can be separated into 3 groups. General (GridBean, job name, application), Files (imports and exports) and Requirements (view for editing the JobRequirements property). For most of these, EMF Forms is capable of generating a suitable default control.

5.3. Required Custom Controls

Some job properties have type that is not supported by EMF Forms directly. For these, EMF Forms offers *custom controls*, which allow the definition of a custom GUI element which is capable of editing a single job attribute. These properties are the following:

- Application
  Since the list of available applications is defined by the GridBean and by supported Applications on the reachable target systems, this list must be regularly updated while editing a job. Since EMF Forms offers no automatic way to do this, this must be done manually.

- GridFileType
  GridFileType is not supported by EMF Forms. To be able to edit the imports and exports, it is necessary to be able to select a remote GridFile. This must also be done manually.

- slected target system
  While the selected target system could be edited as a string, this is inconvenient for the user. A more practical approach would present all target systems to the user as before (see figure 2.3).

- JSDLType
  To be able to edit additional job properties or requirements, it must be possible to edit a general JSDLType. As JSDLType is an abstract class without any attributes, EMF Forms can not generate a default control for it.

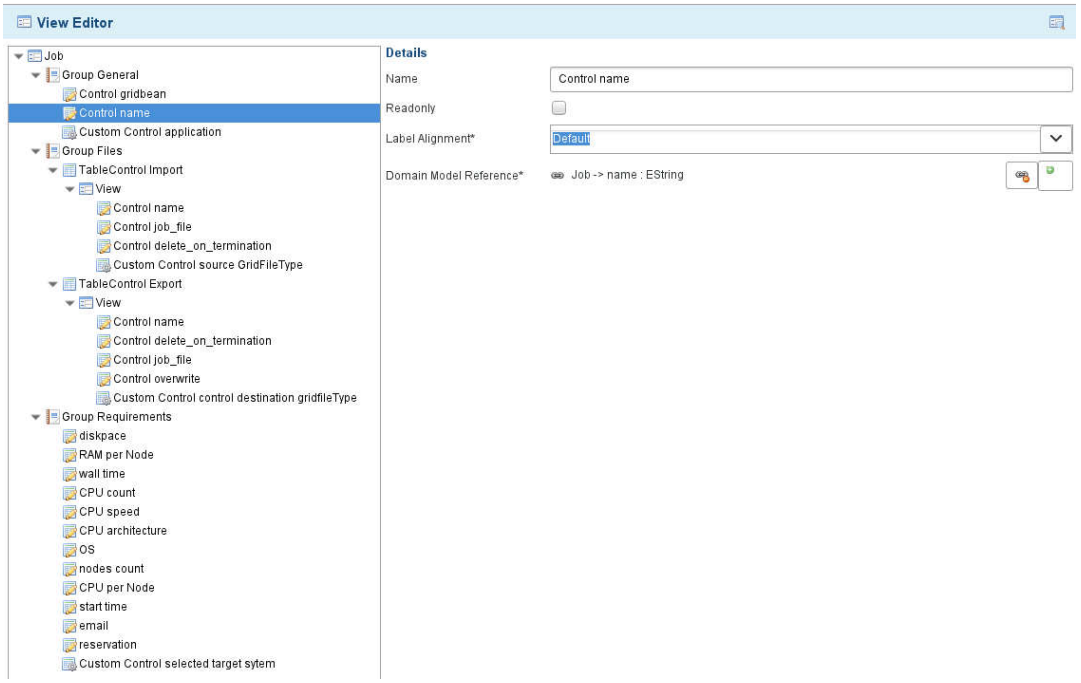

Figure 5.1.: The EMF Forms View Editor

5.4. Preview of generated GUI

The previews in figures 5.2, 5.3 and 5.4 are not yet complete and do not contain any custom controls. This means that the automated check for completion and errors does not work properly and colors some fields red (e.g. GridBean).

Figure 5.2.: Preview: Generated GUI for General Job Settings

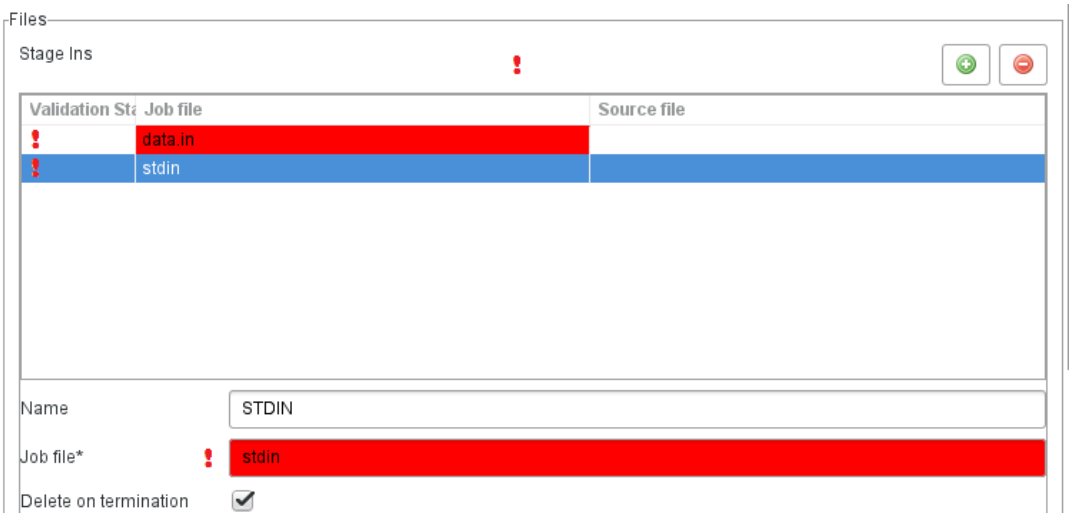

Figure 5.3.: Preview: Generated GUI for Data Imports

Figure 5.4.: Preview: Generated GUI for Job Resources

6. Conclusion and Further Development

6.1. Conclusion

The first steps toward a replacement for GPE have been done: The UNICORE Rich Client has been analysed in respect to the capabilities that are dependant on GPE, a suitable jobmodel has been designed as a replacement for the GPE jobmodel, a replacement for the GPE client-classes has been designed and the replacement for the GPE GUI elements has been planned. In the implementation process the Eclipse Modelling Framework has been used for the jobmodel, to make the initial process and future changes easier and EMF Forms has been used to require less manual work on the GUI elements.

6.2. Further Development

There are several steps to be taken until a full replacement for GPE in the URC is ready.

6.2.1. Finish GUI Implementation

The first step would be to complete the GUI implementation as described in chapter 5. This requires implementation of the custom controls (section 5.3). After that, the view created with EMF Forms is ready to be used.

6.2.2. Integrate Jobmodel, Gridmodel and GUI into URC

After completing the GUI, the elements described in this thesis have to be integrated in the URC. This does not only require changes in the classes that use GPE, but also changing the plug-in layout, since several extension points use GPE classes. This work

will be more focussed on the eclipse client platform and require a better understanding of its features.

6.2.3. Work on a Replacement in the workflow plugin

With a working jobmodel and the respective URC plug-in (*gpe4eclispe*), it's now possible to start replacing the workflow functionalities in GPE. Similiar as with the *gpe4eclispe* plugin, this would require a more detailed analysis of the URC workflow plug-in, as well as GPE in respect to workflows. The replacement can then be developed building on the new jobmodel.

6.2.4. Replace GPE functions in the rcp.common plugin

While the org.chemomentum.rcp.common plug-in does not depend heavily on GPE, it still requires some reworking to work without GPE.

A. Listings, Lists and Figures

This chapter contains all listings, lists and figures that offer more detail on some parts of this thesis, but either are too detailed or too long for the main part.

A.1. Enumerations

```
unknown("Unknown"),
linux("LINUX"),
mac_os("MACOS"),
win95("WIN95"),
win98("WIN98"),
windows_R_Me("Windows_R_Me"),
winNT("WINNT"),
windows_2000("Windows_2000"),
windows_XP("Windows_XP"),
msdos("MSDOS"),
solaris("Solaris"),
sunOS("SunOS"),
freeBSD("FreeBSD"),
netBSD("NetBSD"),
openBSD("OpenBSD"),
bsdunix("BSDUNIX"),
aix("AIX"),
z_OS("z_OS"),
os_2("OS_2"),
os9("OS9"),
netWare("NetWare"),
tru64_unix("Tru64_UNIX"),
irix("IRIX"),
osf("OSF"),
mvs("MVS"),
os400("OS400"),
javaVM("JavaVM"),
win3x("WIN3x"),
```

```
winCE("WINCE"),
NCR3000("NCR3000"),
dc_os("DC_OS"),
reliant_unix("Reliant_UNIX"),
sco_unixWare("SCO_UnixWare"),
sco_openServer("SCO_OpenServer"),
sequent("Sequent"),
u6000("U6000"),
aseries("ASERIES"),
tandemNSK("TandemNSK"),
tandemNT("TandemNT"),
bs2000("BS2000"),
lynx("Lynx"),
xenix("XENIX"),
vm("VM"),
interactive_unix("Interactive_UNIX"),
gnu_hurd("GNU_Hurd"),
mach_kernel("MACH_Kernel"),
inferno("Inferno"),
qnx("QNX"),
epoc("EPOC"),
ixWorks("IxWorks"),
vxWorks("VxWorks"),
mint("MiNT"),
beOS("BeOS"),
hp_mpe("HP_MPE"),
nextStep("NextStep"),
palmPilot("PalmPilot"),
rhapsody("Rhapsody"),
dedicated("Dedicated"),
os_390("OS_390"),
vse("VSE"),
tpf("TPF"),
caldera_open_unix("Caldera_Open_UNIX"),
attunix("ATTUNIX"),
dgux("DGUX"),
decnt("DECNT"),
openVMS("OpenVMS"),
hpux("HPUX"),
other("other");
```

Listing A.1: OSType enum

```
sparc("sparc"),
powerpc("powerpc"),
x86("x86"),
x86_32("x86_32"),
x86_64("x86_64"),
parisc("parisc"),
mips("mips"),
ia64("ia64"),
arm("arm"),
other("other");
```

Listing A.2: ProcessorType enum

A.2. Used GPE Classes

```
com.intel.gpe.client.impl.configurators.UnicoreCommonKeys;
com.intel.gpe.client.impl.security.SecurityProviderImpl;
com.intel.gpe.clients.all.ClientAdapter;
com.intel.gpe.clients.all.exceptions.GPEGridBeanPluginUnavailableException
    ;
com.intel.gpe.clients.all.exceptions.GPEJobFailedException;
com.intel.gpe.clients.all.gridbeans.BaseGridBeanClient;
com.intel.gpe.clients.all.gridbeans.GridBeanClient;
com.intel.gpe.clients.all.gridbeans.GridBeanInfo;
com.intel.gpe.clients.all.gridbeans.GridBeanInfo;
com.intel.gpe.clients.all.gridbeans.GridBeanJob;
com.intel.gpe.clients.all.gridbeans.GridBeanJobWrapper;
com.intel.gpe.clients.all.gridbeans.InternalGridBean;
com.intel.gpe.clients.all.gridbeans.InternalGridBean;
com.intel.gpe.clients.all.gridbeans.InternalGridBeanImpl;
com.intel.gpe.clients.all.gridbeans.panels.BaseGridBeanInputPanel;
com.intel.gpe.clients.all.gridbeans.panels.BaseGridBeanOutputPanel;
com.intel.gpe.clients.all.gridbeans.panels.IGridBeanInputPanel;
com.intel.gpe.clients.all.i18n.Messages;
com.intel.gpe.clients.all.i18n.MessagesKeys;
com.intel.gpe.clients.all.providers.GridBeanProvider;
com.intel.gpe.clients.all.providers.OutcomeProvider;
com.intel.gpe.clients.all.requests.SubmitRequest;
com.intel.gpe.clients.all.utils.FileTools;
com.intel.gpe.clients.all.utils.GPEFileFactoryImpl;
```

```
com.intel.gpe.clients.api.Application;
com.intel.gpe.clients.api.Application;
com.intel.gpe.clients.api.Client;
com.intel.gpe.clients.api.ClientFactory;
com.intel.gpe.clients.api.DownloadGridBeanClient;
com.intel.gpe.clients.api.DownloadGridBeanClient;
com.intel.gpe.clients.api.FileSystem;
com.intel.gpe.clients.api.GridBean;
com.intel.gpe.clients.api.GridFile;
com.intel.gpe.clients.api.Job;
com.intel.gpe.clients.api.JobClient;
com.intel.gpe.clients.api.JobType;
com.intel.gpe.clients.api.RegistryClient;
com.intel.gpe.clients.api.SelectionClient;
com.intel.gpe.clients.api.StandaloneClient;
com.intel.gpe.clients.api.Status;
com.intel.gpe.clients.api.StorageClient;
com.intel.gpe.clients.api.TargetSystemClient;
com.intel.gpe.clients.api.TargetSystemFactoryClient;
com.intel.gpe.clients.api.WSLTClient;
com.intel.gpe.clients.api.apps.ArgumentMetaData;
com.intel.gpe.clients.api.apps.ParameterMetaData;
com.intel.gpe.clients.api.async.AsyncClient;
com.intel.gpe.clients.api.async.IProgressListener;
com.intel.gpe.clients.api.async.Request;
com.intel.gpe.clients.api.async.RequestExecutionService;
com.intel.gpe.clients.api.async.SyncClient;
com.intel.gpe.clients.api.cache.DefaultProperties;
com.intel.gpe.clients.api.cache.GPEClientProperties;
com.intel.gpe.clients.api.configurators.FileProviderConfigurator;
com.intel.gpe.clients.api.configurators.NetworkConfigurator;
com.intel.gpe.clients.api.configurators.UserDefaultsConfigurator;
com.intel.gpe.clients.api.exceptions.FaultWrapper;
com.intel.gpe.clients.api.exceptions.GPEException;
com.intel.gpe.clients.api.exceptions.GPEFileAddressUnresolvableException;
com.intel.gpe.clients.api.exceptions.GPEInvalidQueryExpressionException;
com.intel.gpe.clients.api.exceptions.
    GPEInvalidResourcePropertyQNameException;
com.intel.gpe.clients.api.exceptions.GPEJobNotSubmittedException;
com.intel.gpe.clients.api.exceptions.GPEMiddlewareRemoteException;
com.intel.gpe.clients.api.exceptions.GPEMiddlewareServiceException;
com.intel.gpe.clients.api.exceptions.GPEQueryEvaluationErrorException;
```

```
com.intel.gpe.clients.api.exceptions.GPEResourceNotDestroyedException;
com.intel.gpe.clients.api.exceptions.GPEResourceUnknownException;
com.intel.gpe.clients.api.exceptions.GPESecurityException;
com.intel.gpe.clients.api.exceptions.
    GPETerminationTimeChangeRejectedException;
com.intel.gpe.clients.api.exceptions.
    GPEUnableToSetTerminationTimeException;
com.intel.gpe.clients.api.exceptions.
    GPEUnknownQueryExpressionDialectException;
com.intel.gpe.clients.api.exceptions.GPEUnmarshallingException;
com.intel.gpe.clients.api.exceptions.GPEWrongJobTypeException;
com.intel.gpe.clients.api.gpe.GPETargetSystemFactoryClient;
com.intel.gpe.clients.api.jsdl.JSDLElement;
com.intel.gpe.clients.api.jsdl.JSDLJob;
com.intel.gpe.clients.api.jsdl.OSType;
com.intel.gpe.clients.api.jsdl.OSType;
com.intel.gpe.clients.api.jsdl.OperatingSystemRequirementsType;
com.intel.gpe.clients.api.jsdl.OperatingSystemRequirementsType;
com.intel.gpe.clients.api.jsdl.ProcessorType;
com.intel.gpe.clients.api.jsdl.ProcessorType;
com.intel.gpe.clients.api.jsdl.RangeValueType;
com.intel.gpe.clients.api.jsdl.RangeValueType;
com.intel.gpe.clients.api.jsdl.ResourceRequirementType;
com.intel.gpe.clients.api.jsdl.gpe.GPEJob;
com.intel.gpe.clients.api.security.GPESecurityManager;
com.intel.gpe.clients.api.security.GPESecurityManager;
com.intel.gpe.clients.api.transfers.FileProvider;
com.intel.gpe.clients.api.transfers.GPEFileFactory;
com.intel.gpe.clients.api.transfers.IAddressOrValue;
com.intel.gpe.clients.api.transfers.IGridFileAddress;
com.intel.gpe.clients.api.transfers.IGridFileSetTransfer;
com.intel.gpe.clients.api.transfers.IGridFileTransfer;
com.intel.gpe.clients.api.transfers.IProtocol;
com.intel.gpe.clients.api.transfers.Protocol;
com.intel.gpe.clients.api.transfers.ProtocolConstants;
com.intel.gpe.clients.api.virtualworkspace.
    GPEVirtualTargetSystemFactoryClient;
com.intel.gpe.clients.common.ApplicationImpl;
com.intel.gpe.clients.common.JAXBContextProvider;
com.intel.gpe.clients.common.clientwrapper.ClientWrapper;
com.intel.gpe.clients.common.requests.TransferGridFilesRequest;
com.intel.gpe.clients.common.transfers.GridFileAddress;
```

```
com.intel.gpe.clients.common.transfers.GridFileSetTransfer;
com.intel.gpe.clients.common.transfers.GridFileTransfer;
com.intel.gpe.clients.common.transfers.Value;
com.intel.gpe.clients.gpe4gtk.rms.ResourceManagementServiceClient;
com.intel.gpe.clients.impl.SecurityProvider;
com.intel.gpe.clients.impl.WSLTClientImpl;
com.intel.gpe.clients.impl.WSRFClientImpl;
com.intel.gpe.clients.impl.exception.GPEFaultWrapper;
com.intel.gpe.clients.impl.gbs.DownloadGridBeanClientImpl;
com.intel.gpe.clients.impl.jms.JobClientImpl;
com.intel.gpe.clients.impl.jms.StatusImpl;
com.intel.gpe.clients.impl.registry.RegistryClientImpl;
com.intel.gpe.clients.impl.sms.StorageClientImpl;
com.intel.gpe.clients.impl.transfers.U6ProtocolConstants;
com.intel.gpe.clients.impl.tss.ApplicationUtils;
com.intel.gpe.clients.impl.tss.TargetSystemClientImpl;
com.intel.gpe.gridbeans.ApplicationMatcher;
com.intel.gpe.gridbeans.ErrorSet;
com.intel.gpe.gridbeans.GridBeanConstants;
com.intel.gpe.gridbeans.IGridBean;
com.intel.gpe.gridbeans.IGridBean;
com.intel.gpe.gridbeans.IGridBeanModel;
com.intel.gpe.gridbeans.IGridBeanModel;
com.intel.gpe.gridbeans.JobError;
com.intel.gpe.gridbeans.apps.ArgumentMetaDataImpl;
com.intel.gpe.gridbeans.apps.BooleanValidArgumentRange;
com.intel.gpe.gridbeans.apps.StringValidArgumentRange;
com.intel.gpe.gridbeans.jsdl.GPEJSDLUtil;
com.intel.gpe.gridbeans.jsdl.GPEJobImpl;
com.intel.gpe.gridbeans.jsdl.JSDLJobImpl;
com.intel.gpe.gridbeans.jsdl.JavaToXBeanConverterRegistry;
com.intel.gpe.gridbeans.jsdl.JavaToXBeanElementConverter;
com.intel.gpe.gridbeans.jsdl.JobWithParameters;
com.intel.gpe.gridbeans.parameters.CPUArchitecture;
com.intel.gpe.gridbeans.parameters.EnvironmentVariableParameterValue;
com.intel.gpe.gridbeans.parameters.FileParameterValue;
com.intel.gpe.gridbeans.parameters.GridBeanParameter;
com.intel.gpe.gridbeans.parameters.GridBeanParameterType;
com.intel.gpe.gridbeans.parameters.IEnvironmentVariableParameterValue;
com.intel.gpe.gridbeans.parameters.IFileParameterValue;
com.intel.gpe.gridbeans.parameters.IFileSetParameterValue;
com.intel.gpe.gridbeans.parameters.IGridBeanParameter;
```

```
com.intel.gpe.gridbeans.parameters.IGridBeanParameterValue;
com.intel.gpe.gridbeans.parameters.ResourcesParameterValue;
com.intel.gpe.gridbeans.parameters.ResourcesParameterValue;
com.intel.gpe.gridbeans.parameters.SiteSpecificResourceRequirement;
com.intel.gpe.gridbeans.parameters.SiteSpecificResourceRequirement;
com.intel.gpe.gridbeans.parameters.processing.AbstractProtocolProcessor;
com.intel.gpe.gridbeans.parameters.processing.IParameterTypeProcessor;
com.intel.gpe.gridbeans.parameters.processing.IProtocolProcessor;
com.intel.gpe.gridbeans.parameters.processing.IStepProcessor;
com.intel.gpe.gridbeans.parameters.processing.
    ParameterTypeProcessorRegistry;
com.intel.gpe.gridbeans.parameters.processing.ProcessingConstants;
com.intel.gpe.gridbeans.parameters.processing.ProtocolProcessorRegistry;
com.intel.gpe.gridbeans.parameters.processing.StepProcessorRegistry;
com.intel.gpe.gridbeans.plugins.DataSetException;
com.intel.gpe.gridbeans.plugins.IDataControl;
com.intel.gpe.gridbeans.plugins.IGenericGridBeanPanel;
com.intel.gpe.gridbeans.plugins.IGridBeanPlugin;
com.intel.gpe.gridbeans.plugins.IPanel;
com.intel.gpe.gridbeans.plugins.TranslationException;
com.intel.gpe.gridbeans.plugins.ValueChangeEvent;
com.intel.gpe.gridbeans.plugins.swt.controls.SWTDataControler;
com.intel.gpe.gridbeans.plugins.swt.panels.ISWTGridBeanPanel;
com.intel.gpe.gridbeans.plugins.swt.panels.SWTGridBeanPanel;
com.intel.gpe.gridbeans.plugins.swt.ui.arguments.ApplicationParameterPanel
    ;
com.intel.gpe.gridbeans.plugins.validators.
    FileIsReadableAndNotDirectoryValidator;
com.intel.gpe.util.collections.CollectionUtil;
com.intel.gpe.util.collections.CollectionUtil;
com.intel.gpe.util.defaults.preferences.CommonKeys;
com.intel.gpe.util.defaults.preferences.CommonKeys;
com.intel.gpe.util.defaults.preferences.INode;
com.intel.gpe.util.defaults.preferences.IPreferences;
com.intel.gpe.util.defaults.preferences.IPreferences;
com.intel.gpe.util.observer.IObserver;
com.intel.gpe.util.preferences.PropertiesPreferencesFactory;
com.intel.gpe.util.sets.Pair;
com.intel.gpe.util.swing.controls.CloseListener;
com.intel.gpe.util.swing.controls.configurable.ConfigurableDialog;
com.intel.gpe.util.swing.controls.configurable.GPEPanel;
com.intel.gpe.util.swing.controls.configurable.GPEPanelResult;
```

```
com.intel.gpe.util.swing.controls.configurable.GUIKeys;
com.intel.gpe.util.swing.controls.configurable.IConfigurable;
com.intel.gpe.util.swing.controls.configurable.MessageProvider;
```

Listing A.3: GPE classes used by gpe4eclipse

A.3. Job Attributes

- job name
- GridBean name
- GridBean version
- job description
- application name
- application version
- stdin
- stderr
- executable (path)
- arguments
- import files
- export files
- desired execution system
- OS
- CPU Architecture
- CPUs per node
- CPU time per node
- total CPU count

- RAM per node
- CPU speed
- disk space per node
- working directory
- environment variables
- start time
- reservation id
- notification email
- username