



g-Eclipse Middleware Integration

How to integrate a new middleware into the g-Eclipse framework

WP6.4

Document Filename:	MiddlewareIntegration.pdf
Workpackage:	WP6.4
Partner(s):	FZK, PSNC, JKU, UCY, INO, RUR
Lead Partner:	FZK
Config ID:	g-Eclipse MI
Document classification:	PUBLIC

Abstract: This document addresses middleware-developers that would like to see their middleware supported by g-Eclipse. It describes the general procedure of integrating a new middleware into the g-Eclipse framework.



Contents

1	General Techniques	3
1.1	Repositories	3
1.1.1	The Subversion-Repository	3
1.1.2	The CVS-Repository	3
1.2	Tooling	3
1.2.1	Development Environment	4
1.2.2	Bugzilla	4
1.2.3	Build System	4
1.3	UI-Separation	4
1.4	Coding Conventions	5
2	Preparatory Work	6
2.1	Basic Grid Model Types	6
2.2	Authentication and Authorisation	7
2.3	Exemplary Mappings	7
3	Basic Integration	8
3.1	Virtual Organization	8
3.2	Information System	8
3.3	Authentication and Authorisation	8
3.4	Data Management	8
3.5	Job Management	8
4	Advanced Integration	9

1 General Techniques

In this chapter we give a general overview about the basic techniques as used by the g-Eclipse team itself. These may be seen as recommendations for external developers rather than as strong requirements. Nonetheless at least if a contribution to the g-Eclipse sources itself is foreseen or if the resulting plug-ins should be hosted on the g-Eclipse infrastructure, following these techniques is highly desirable.

1.1 Repositories

Currently the g-Eclipse source codes are distributed on two repositories. The first one is a Subversion repository hosted by the Eclipse foundation, the second one is a CVS repository hosted by the partner FZK.

1.1.1 The Subversion-Repository

Since g-Eclipse is an official Eclipse.org project, the Eclipse Foundation provides some infrastructure for g-Eclipse. One piece of this infrastructure is the Subversion repository that is mainly used for the middleware-independent codes of g-Eclipse. To be more precise all source code that is licence-compatible with the Eclipse Public Licence (EPL) can be found in this repository.

The repository location for Committers is:

```
svn+ssh://userid@dev.eclipse.org/svnroot/technology/eu.geclipse
```

where "userid" is your Eclipse Foundation user ID.

The repository location for Contributors is:

```
svn://dev.eclipse.org/svnroot/technology/eu.geclipse
```

For more information about the Subversion-repository and how to access it we refer to:

<http://www.eclipse.org/geclipse/contributing.php>

1.1.2 The CVS-Repository

It is a requirement by the Eclipse Foundation itself that all source code checked in to the Eclipse.org repository has to be licence-compatible with the EPL. Since most of the middleware-specific functionality of g-Eclipse makes use of external libraries that are not compatible to EPL (for instance GPL or LGPL are not compatible to EPL), the g-Eclipse project hosts these parts of the code in an own repository.

To repository location for the CVS repository is:

```
:ext:anoncvs@cvs.fzk.de:/cvs/fzk/geclipse
```

where the g-Eclipse sources can be found in the subdirectory */geclipse/development*.

For more information about the CVS-repository and how to access it we refer to:

<http://wiki.eclipse.org/G-Eclipse-Middleware-Extensions>

1.2 Tooling

The g-Eclipse team itself makes use of a well designed set of tools. These tools are shortly introduced in this section and may be seen as recommendation for external developers.

1.2.1 Development Environment

In order to develop plug-ins for g-Eclipse in principle every development environment for Java can be used. Nevertheless we strongly recommend to use the Eclipse JDT (Java Development Toolkit) and PDT (Plug-in Development Toolkit) for contributing to g-Eclipse. Since extending our framework is mainly about developing Eclipse plug-ins these toolkits provide highly specialised functionality resulting in higher quality and less error prone implementations.

1.2.2 Bugzilla

The Eclipse Foundation hosts a Bugzilla system in order to report and track the status of bugs, feature requests etc. Therefore g-Eclipse itself makes extensive use of this Bugzilla installation.

The Eclipse Bugzilla system can be found at:

<https://bugs.eclipse.org/bugs/>

There one can report new or investigate existing bugs for the various Eclipse projects. When reporting bugs about g-Eclipse you should be aware of the fact, that g-Eclipse is a sub-project of the Eclipse Technology project. So watch out for the Technology project first and then locate the g-Eclipse project within the Technology branch.

1.2.3 Build System

The g-Eclipse project provides not only monthly milestone builds and regular mature releases, but also nightly builds for all the supported architectures. All these builds are managed by a dedicated build system using Luntbuild.

The build-site for g-Eclipse can be found at:

<http://iwr-geclipse.fzk.de:8443/luntbuild/app.do>

There one can access all current builds with all accompanied information such as metrics, JUnit tests, quality reports, etc. Each build consists of a predefined set of plug-ins both loaded from the SVN and the CVS repositories (see Section 1.1). If you would like to have your plug-ins included in these builds you should place them in one of those repositories and contact the g-Eclipse team in order to add your plug-ins to the list of predefined plug-ins for the builds.

1.3 UI-Separation

A basic design principle in the Eclipse world is the separation of core and UI codes. As an Eclipse project g-Eclipse also follows this principle and future middleware implementations should do so as well (though the amount of UI code that has to be written for a new middleware is rather minimal). Separation of UI code means to have two plug-ins for a set of functionality, one containing the core functionality like models and model implementations and the other one containing only UI classes for accessing this core functionality. A good hint to not violate this rule is that your core-plug-ins should not depend on any UI-plug-in (e.g. `org.eclipse.ui.*`, `eu.geclipse.*.ui`, etc.).

The g-Eclipse build system (see Section 1.2.3) also discovers the dependencies among the various plug-ins. When you are in the build system's main page click on the link in the "Latest build" column of the build you would like to investigate and choose "Dependencies.html" from the next page. This will direct you to the dependency table of all plug-ins contained in this specific build.

1.4 Coding Conventions

In general g-Eclipse follows Sun's Java coding conventions and additionally makes use of the Checkstyle plug-in for some more specific conventions.

2 Preparatory Work

Maybe the most challenging part when implementing a new middleware for g-Eclipse is to find a suitable mapping between the g-Eclipse Grid Model and the concepts of the specific middleware. Depending on the complexity of the middleware this may be more or less easy. In any case at least a basic knowledge of the Grid Model on the one hand and the middleware concepts on the other is needed to find the best mapping of the both. Therefore we strongly recommend to get in touch with the Grid Model as well as the Authentication and Authorisation Infrastructure (AAI) as defined by g-Eclipse. A good introduction of both can be found in our Deliverable D1.8 at:

<http://www.geclipse.eu/fileadmin/Documents/Deliverables/D1.8.pdf>

All referred interfaces can be found in `eu.geclipse.core.model`, all referred classes are located in `eu.geclipse.core.model.impl`.

2.1 Basic Grid Model Types

This section gives a brief overview of the most important Grid Model elements.

IVirtualOrganization: A virtual organization (VO) in the sense of Grid computing is a group of people – potentially spread all over the world (therefore virtual) – sharing a common interest. In the sense of g-Eclipse a VO is the central point for accessing the user’s personalized Grid. It provides methods for querying the underlying infrastructure for available resources such as services and computing and storage resources. Which resource types are supported by a specific VO is up to the VO itself. If you are for instance only interested in managing data but not in submitting jobs this will be reflected by your VO in the sense that your VO does then only provide storage resources.

If your middleware does natively provide support for virtual organizations your corresponding implementation of this interface should also provide access information regarding the VO such as server endpoints or user credentials.

Instead of directly implementing the `IGridVirtualOrganization`-interface we strongly recommend to rather extend the `AbstractVirtualOrganization` class that makes it much easier to provide your own model-compliant implementation.

IGridResource: Grid resources are obtained from a VO or from an underlying info service (see next item). They serve as contact and working points within the Grid Model. They represent remote resources that are accessible to a member of a VO in order to manage data, submit jobs etc. The Grid Model itself defines already sub-interfaces of the `IGridResource`-interface for dedicated resources. The following items are such sub-interfaces.

IGridInfoService: An info service is a Grid resource that acts as kind of resource information database within the model. Each VO has to have exactly one dedicated information service. Within your VO the information service is responsible for querying the underlying Grid infrastructure for resources that are available to the VO.

In the ideal case your middleware provides already something like a resource database on the server side. In that case the info service is just the client side interface to that database. There are some middlewares out there that do not have something like a resource database. In that case it may be necessary to specify the available (static) resources as part of your VO definition itself. Nevertheless you then have to provide a shallow info service implementation that does nothing than just returning the resources that are defined for your VO.

Model Element	gLite Concept	GRIA Concept
IVirtualOrganization	VOMS virtual organization.	Locally managed list of GRIA services.
IGridInfoService	gLite BDII service.	Combination of local registry and remote registry service.
IGridStorage	gLite storage elements (SE).	GRIA data services.
IGridComputing	gLite computing elements (CE).	GRIA job services.
IGridJobService	WMS and CREAM submission services and L&B services for job status retrieval.	GRIA job services.
IAuthenticationToken	Globus X.509 proxy certificates and VOMS X.509 proxy certificates.	Java key stores.

Table 2.1: Mappings used for the gLite and GRIA middlewares.

IGridStorage: This interface is the abstraction for any type of storage on the middleware side. This can represent a rather abstract storage device such as some kind of storage resource manager but could also represent a physical storage device such as a hard disk. In any case this interface is the connection of your storage resource and the file system used to access it from the client side. Since each storage element is also an `IMountable` it can be mounted as a folder in the g-Eclipse model view's and can be accessed as it would be a local folder. The only prerequisite here is that there is an EFS¹-implementation registered for the scheme returned by the `IMountable`'s mount points.

IGridComputing: This interface represents endpoints in your infrastructure where one can submit jobs to. The job submission itself is done with a job service.

IGridJobService: A job service is used to submit and manage jobs for your infrastructure. All the client-side job logic should go into your implementation of this interface.

Rather than directly implementing these interfaces for your middleware you should rather watch out for already prepared abstract implementations of these interfaces or their super-interfaces. The general rule is to rather extend the abstract implementation next to your interface as to directly implement the interface itself. This ensures that your implementation will be properly integrated into our Grid Model.

The above list is far from being complete. Nevertheless it covers the most basic use cases of a middleware such as data management and job submission. If you are interested in more advanced features just have a look at the mass of further interfaces that can be found in the `eu.geclipse.core.model` package.

2.2 Authentication and Authorisation

2.3 Exemplary Mappings

The g-Eclipse framework already comes with two Grid middleware implementations (i. e. gLite and GRIA). In table 2.1 we summarize the mappings we used for these two middlewares.

¹Eclipse File System

3 Basic Integration

3.1 Virtual Organization

3.2 Information System

3.3 Authentication and Authorisation

3.4 Data Management

3.5 Job Management

4 Advanced Integration