Grid Software Object Specification

Gregor von Laszewski[*]and Jeremy Millar[†]

January 12, 2000

Contents

1 Introduction

The Grid provides a large collection of heterogenous computing resources to the user, in terms of both hardware and installed software. In order to run an application, the user must locate a machine with the appropriate software.

Currently, this search is carried out by hand, typically by obtaining accounts on various machines and inspecting their installed software. Publication of software deployment data in a metadata registry (similar to the Globus MDS) will allow this process to be automated to some extent. Rather than inspecting each machine, the user will be able to search a central registry for the required combination of machine and software.

Moreover, deployment metadata will allow for automated scheduling of jobs on the Grid. In this case, the user need only specify the computational requirements, e.g. architecture, software, performance metrics. The proper Grid middleware component will search the metadata registry for the required combination of capabilities and schedule and run the job.

[*]Argonne National Laboratory, MCS Division
[†]Univ. of Tennessee, Dept. of Computer Science

This paper proposes Grid object specifications for the various types of software necessary to support ease of search and automated scheduling. Section 2 presents these specifications in detail. Section 3 presents a series of concrete examples.

2 Object Class Definitions

The Grid provides an LDAP-based directory service (e.g. MDS) for registering and storing metadata. This section defines objects classes for the representation of software and software deployments within this service.

The following object classes are defined:

- Grid::Software
- Grid::Daemon
- Auxiliary classes

Previous proposals have included a Grid::SoftwareRepository object class. This class was used to store the information associated with a RIB software repository. In practice however, this class tended to duplicate much of the functionality of the Grid::Software object class. Because of this, the Grid::SoftwareRepository class has been eliminated and its attributes have been assimilated into the Grid::Software class.

2.1 Grid::Software

The Grid::Software object class represents software packages installed on a Grid::ComputeResource[1]. Grid::Software objects include applications (PETSc), libraries (ROMIO), and utilities (xxgdb).

Because instances of Grid::Software are normally accessed through a Grid::ComputeResource, they are placed in the DIT below the ComputeResource. In the case of a shared filespace, the metadata should be replicated beneath each ComputeResource.

```
Grid::Software OBJECT CLASS
  SUBCLASS OF Grid::Top
  RDN = sw (software)
  CHILD OF {
    Grid::organization;
    Grid::organizationalUnit;
    Grid::ComputeResource;
    Grid::ResourceManager;
    Grid::ResourceManagerQueue;
    Grid::Root;
  }
```

[1] Defined elsewhere

```
AGGREGATES {
  Grid::Version;
}

MUST CONTAIN {
  name                       :: cis, singular;
  description                :: cis, singular;
  dateOfInformation          :: cis, singular;
  deploymentStatus           :: cis, singular;
  domainClassification       :: cis;
  keywordClassification      :: cis;
  documentation              :: cis;
}

MAY CONTAIN {
  executableDirectory        :: ces;
  includeDirectory           :: ces;
  libDirectory               :: ces;
  sharedlibraryDirectory     :: ces;
  manpathDirectory           :: ces;
  rootDirectory              :: ces;
  dateOfInstallation         :: date, singular;
  installPrefix              :: ces;
  installedBy                :: dn;
  administeredBy             :: dn;
  configuration              :: ces;
  environment                :: ces;
  state                      :: cis;
  interface                  :: ces;
  quality                    :: cis;
  bugs                       :: cis;
  notes                      :: cis;
}
```

name: The name of the software.

description: A short description of the software.

dateOfInformation: The date the software was installed.

deploymentStatus: Status of the software deployement, e.g. pending.

domainClassification: A classification of the software domain, e.g. Parallel Processing Tools, Communication Libraries.

keywordClassification: A keyword to enable simple classification and searching, e.g. message passing library.

documentation: Pointers to the software documentation, e.g. http://www.mcs.anl.gov/mpi/mpich/.

executableDirectory: The directory containing the software executable, e.g. /usr/local/bin.

includeDirectory: The directory containing include files for the software, e.g. /usr/local/include.

libDirectory: The directory containing the libraries for the software, e.g. /usr/local/lib.

sharedlibraryDirectory: The directory containing the shared libraries for the software.

manpathDirectory: The directory containing the man pages for the software, e.g. /usr/local/man.

rootDirectory: The root directory (if present) of the software.

dateOfInstallation: The date the software was installed.

installPrefix: The starting path of libraries, header files, etc.

installedBy: The party responsible for installing the software.

administeredBy: The party responsible for administering the software.

configuration: The parameters given at configuration time.

environment: Environment variables which must be set for the software.

state: The state of the object.

interface: Input/output parameters of the software (experimental).

quality: The quality of the installation, e.g. production, test.

bugs: Known bugs.

notes: Notes and remarks.

2.2 Grid::Daemon

The Grid::Daemon object class represents services running on a Grid::ComputeResource, e.g. gloperf, sqld. Grid::Daemon objects are stored in the DIT in a manner similar to Grid::Software objects.

```
Grid::Daemon OBJECT CLASS
  SUBCLASS OF Grid::Software
  RDN = da (daemon)
  CHILD OF {
    Grid::ComputeResource;
    Grid::Root;
```

```
  }

  AGGREGATES {
    Grid::DaemonAccess;
  }

  MAY CONTAIN {
    hostname                :: cis, singular;
    hostnameDN              :: dn, singular;
    name                    :: cis, singular;
    parameter               :: cis;
    developerContact        :: dn;
    daemonAdministrator     :: dn;
  }
```

hostname: The hostname of the machine the daemon is running on.

hostnameDN: The distinguished name of the host the daemon is running on.

name: The name of the daemon.

parameter: A tuple of the form (attribute=value) necessary to configure the daemon.

developerContact: Contact information for the daemon developers.

daemonAdministrator: The party responsible for administering the daemon.

2.3 Auxiliary Object Classes

The following sections define auxiliary object classes. These classes are included as aggregates in the classes defined above. They are not meant to serve as standalone objects.

2.3.1 Grid::Version

Most software packages are released with a release tag or version number. Often, this tag can be broken down into convenient attributes (e.g., major and minor version numbers). Breaking the release tags into as many separate attributes as possible will facilitate sophisticated searching.

```
Grid::Version OBJECT CLASS
  MUST CONTAIN {
    releaseTag                  :: cis, singular;
  }

  MAY CONTAIN {
    releaseDate                 :: cis, singular;
```

```
    releaseMajor                :: cis, singular;
    releaseMinor                :: cis, singular;
    releasePatch                :: cis, singular;
    releaseBeta                 :: cis, singular;
```

releaseTag: The tag or version number provided with the software.

releaseDate: The date of the release.

releaseMajor: The major release number.

releaseMinor: The minor release number.

releaseBeta: The beta release number.

releasePatch: The patch release number.

2.3.2 Grid::DaemonAccess

The Grid::DaemonAccess object class describes the access information for a Grid::Daemon object.

```
Grid::DaemonAccess OBJECT CLASS
  MAY CONTAIN {
    accessProtocol             :: cis, singular;
    accessPort                 :: cis, singular;
    controlPort                :: cis, singular;
    controlProtocol            :: cis, singular;
  }
```

accessProtocol: The protocol used to access the daemon, e.g. telnet.

accessPort: The port used to access the daemon, e.g. 80.

controlPort: The port used to control the daemon.

controlProtocol: The protocol used to control the daemon.

3 Examples

This section presents a series of examples illustrating the use of the proposed object classes. To keep information about deployed software separate from other information concerning a machine, it is recommended that Grid::Software objects be placed in the subtree gn=software.

Thus, a typical DIT would look like:

```
o=grid
  ou=Argonne National Laboratory
    ou=Mathematics and Computer Science Division
```

```
hn=dark.mcs.anl.gov
  gn=software
    sw=Globus
        ...
    sw=hbm
        ...
    sw=gcc
        ...
    da=sqld
        ...
    da=MDS
        ...
```

3.1 Grid::Software Example

```
dn: sw=Globus, hn=pitcairn.mcs.anl.gov, ou=Mathematics and Computer Science
Division, o=Argonne National Laboratory, o=Grid
name: Globus
description: The Globus toolkit provides the infrastructure to build
metacomputing applications
dateOfInformation: Wed Jan 12 08:40:13 2000 GMT
deploymentStatus: Installed
domainClassification: Parallel Processing Tools
keywordClassification: Grid middleware
documentation: http://www.globus.org/documentation
executableDirectory: /soft/pub/bin
includeDirectory: /soft/pub/include
libDirectory: /soft/pub/lib
manpathDirectory: /soft/pub/man
dateOfInstallation: Wed Jan 12 08:40:13 2000 GMT
installPrefix: /soft/pub
installedBy: cn=Gregor von Laszewski, ou=Mathematics and Computer Science
Division, o=Argonne National Laboratory, o=Grid
administeredBy: cn=Joe Bester, ou=Mathematics and Computer Science Division,
o=Argonne National Laboratory, o=Grid
configuration: --enable-hbm -enable-gloperf
quality: Production
bugs: bugs@globus.org
releaseTag: 0.9.9.a
releaseDate: Fri Aug 28 14:02:08 1998 GMT
releaseMajor: 0.9
releaseMinor: 9
releasePatch: a
```

3.2 Grid::Daemon Example

```
dn: da=MDS, hn=pitcairn.mcs.anl.gov, ou=Mathematics and Computer Science
Division, o=Argonne National Laboratory, o=Grid
name: Local Argonne MDS
daemonAdministrator: cn=Gregor von Laszewski, ou=Mathematics and Computer
Science Division, o=Argonne National Laboratory, o=Grid
developerContact: g=Globus, g=projects, o=Grid
parameter: backupinterval 3600
accessProtocol: LDAP
accessPort: 389
controlProtocol: HTTP
controlPort: 13555
```