A GLUE Meta-Packaging proposal for Grid Middleware and Applications: A joint project between LCG, EDG and iVDGL

Olof Barring[1],Germán Cancio[1], Flavia Donno[1], Saul Youssef[2], Jorge Rodriguez[3], Alain Roy[4]
(1) CERN, (2) Boston University, (3) Florida University, (4) Wisconsin University

v.1.0

3 February, 2003
D R A F T

Table of contents

Document Log

Issue	Date	Author	Comment
0.1	27/11/2002	Flavia Donno	First Version after discussion with German Cancio
0.2	27/11/2002	German Cancio & Flavia Donno	Clarify installation steps, add a possible implementation
0.3	27/11/2002	Flavia Donno	Add tables and open issues
0.4	28/11/2002	German Cancio & Flavia Donno	Clean up, more definitions, clarify configuration steps
0.5	17/12/2002	Alain Roy & Flavia Donno	Add "Audience" paragraph, changes in installation step, references
0.6	19/12/2002	Flavia Donno	Change title accordingly to Ian Bird and David Foster's suggestions.
0.7	16/01/2003	Flavia Donno & German Gancio & Alain Roy	Add Alain's comments on configuration; add Flavia and German attempt for template description
0.8	20/01/2003	Alain Roy	Added Appendix C, Configuration Scripts, and empty Appendix E (examples) and did some reformatting.
0.9	27/01/2003	Flavia Donno & Alain Roy	Some correction + Example of Metapackager Description in Appendix D
1.0	03/02/2003	Flavia Donno	Cleaning up. Add notes from last phoneconf meeting

Audience

This document is addressed to Grid middleware developers, to middleware distributors who certify the software and provide distribution repositories to be used by other sites (such as the NSF Middleware Initiative[NMI], the Virtual Data Toolkit[VDT], European DataGrid[EDG], the LCG certification group), and to organizational entities such as the HEP Intergrid Collaboration Board, the Joint Technical Board and the LCG/Grid Deployment Board as input to take the adequate actions for software distribution, installation and configuration.

Introduction

The process of software creation and deployment goes through various phases. During the development phase, developers setup a directory structure for the code, put it in a CVS repository, use ***software building tools*** to build the software on a specific OS or on multiple platforms with multiple flavors (threaded, debug, 32-bits, etc.).
Once the software is built, ***packaging tools*** are used to create a software *binary* distribution that can be installed at other locations. At each site, different ***software distribution tools*** can be deployed to start from some software repository, which can be local, and use it to install software on the machines managed by a system administrator. The repository can also serve local users to install the software on private desktops. (A simple example of such repository can be a CD and a software distribution tool can be the software used by the specific OS to read the CD and install the software starting from it). Once the software has been installed, ***software configuration tools*** are used to properly configure the software reflecting the local site policies and configurations.
In what follows we only discuss about the latter three software tools: packaging, distribution and configuration tools.

Glossary

Service:	In this context, a service is just an installation of a software on a machine.
Package:	It is a unit of software that physically contains files and provides certain functionalities. It could offer client and/or server components, APIs and CLI interfaces. Also configuration scripts could be part of a Package.
Packaging Format:	It is the format chosen to prepare a given Software Package for distribution. Examples are cpio, tar, rpm, pkg.
Building Packager:	It is the tool used to convert a given Package in a given Packaging Format. Examples are the rpmbuild on Linux, pkgmk on Solaris.
Software Bundle:	It is a collection of Software Packages that offer a given functionality and therefore such Packages are bundled together.
Software Update:	It is the process of updating components of a Software Bundle (Packages).
Software Patching:	It is the process of applying patches to fix bugs reported for components of a Software Bundle (Packages).
Software Upgrading:	It is the process of passing from one release of a Package or Software Bundle to the next.
Software Un-installation:	It is the process of uninstalling a Package or Software Bundle from a system. Once uninstalled the Package or Software Bundle is still available for installation and known to the distribution system.
Software Removal:	It is the process of removing a Package or a Software Bundle from the system. Once removed the Package or Software Bundle is not longer available for installation. It is removed from the local software repository.
Software Configuration:	It is the process of adapting the software to the individual Virtual Organization and site-specific settings. This usually implies changing configuration files.
Software Un-configuration:	It is the process of undoing the steps implied by the configuration step and that affects the installation, in order to later uninstall a package or to upgrade it
Management Packager:	It is used to install, upgrade, uninstall a given Package distributed in a given Packaging Format.
Meta-packager:	It is a packaging tool built on top of one or more Management Packagers. A Meta-packager is used to manage (install/uninstall/upgrade) Software Bundles, by invoking the Management Packager with the right operations. Examples are pacman, updaterpms.
System Packager:	It is the Management Packager together with the supported Packaging Format used natively by the operating system (eg. RPM for most Linux distributions, PKG for Solaris).

Scope of this document

In this document we try to discuss possible solutions for the following problem that LCG has to face: give support in terms of distribution, installation and configuration of GRID middleware and application software for large and middle size farms, manually or automatically managed and for desktop users. In particular we would like to provide solutions for first time installations, updates, patches, upgrades and un-installation of software bundles. The resulting installation must be the same in terms of functionality and setup no matter what is the size of the installation and the procedure followed among the supported ones. We give the freedom of choosing the most appropriate packager and packaging format at a given site.
We assume that experiment specific software does not need special care with respect to middleware. So the rules that apply to the middleware software are also valid for experiment specific software.
An open issue that will not be discussed here is the problem of versioning and ensuring that the right releases of the software are actually deployed and distributed coherently.

State of the Art

At CERN, LCG plans to rely on the new version of the LCFG [R1] package for installation and management of large PCs farms. LCFG was originally developed by Edinburgh University. The development of the LCFG package has continued by WP4 in the European DataGrid (EDG) project to customize and extend it to fit the EDG needs. LCFG offers a complete solution for farm installation and management starting from the installation and configuration of the OS, local accounts and services management, software installation, upgrade and un-installation, etc. The Meta-packager tool coming with LCFG is called updaterpms that relies on the RPM Linux RedHat System Packager and so on software being distributed in RPM Packaging Format.
EDG/WP4 plans to phase out LCFG in spring/summer 2003, replacing it with EDG developments. However, the new solution will rely on system packagers such as rpm for Linux and pkg for Solaris.

Although LCFG is a powerful tool, LCG recognizes that other farm managing tools can be adopted to manage farms at a site and that there is the need of accommodating their requirements. Many sites in Europe use farm management tools that provide installation and configuration tools for software bundles distributed in system native formats, such as rpms for Linux RedHat and PKG for Sun/Solaris.
In the US, the ROCKS system based on RedHat Kickstart is also used for farms management.

Also, a user sitting at his/her desktop should be able to install application software that does not impose requirements on the system in user space, without requiring root privileges. This for instance allows for support of installation on demand in the GRID environment at job execution.
For now in LCG there is not real answer to this problem.

The Virtual Data Toolkit is based on the PACMAN [R2] tool. PACMAN is a user-oriented meta-packager tool that can accommodate distributions in several package formats using the appropriate packagers. PACMAN is mostly interactive and is based on PACMAN description files that describe the process of installing, uninstalling and configuring the software for a specific installation. PACMAN is a good tools for desktop installations.

Recommendations

Here we give some recommendations that will help with the definition of a common solution.

Installation recommendations

For software developers

A. Software developers should provide their software in at least three formats:
   1. Source format that can be compiled by end-users and system administrators.
   2. Binary format that uses the appropriate packaging technology for each supported platform. For example, RedHat Linux uses RPM, and Solaris uses PKG. There will be one binary package per supported platform.
   3. Binary format that doesn't require any extra software (like RPM) to install. Instead, users will use a custom installer provided by the software developers, or they will simply copy the software into its final destination. There will be one binary package per supported platform. This binary package is particularly useful for users that do not have root access and do not have the time or ability to build from source.

B. Complete instructions for installing and uninstalling each package must be provided.

C. Software packages must be relocatable, except in cases when they absolutely cannot be. If you think your software cannot be relocatable, please reconsider it carefully.

For meta-packagers

D. You should describe the contents of the software bundle in a machine-readable format that can be automatically translated into descriptions that will work for different meta-packages. The format is specified by the GLUE meta-packaging group. Translations may be imperfect, but will allow software bundlers to automate most of the work in building a bundle that can be distributed by a meta-packager.

For site administrators

E. Site administrators should install the software however they like.

Configuration Recommendations

For software developers

The recommendations for software developers that follow are not absolute requirements. Note, however, that following them will accelerate the adoption of your software by making it easier for others to use it. Note that some of the recommendations may be hard to follow. We strongly urge you to think about simplifying your installation and configuration steps so that these can be done easily. Your users will thank you!

A. The software should be able to be installed separately from configuring the software. Software and configuration files can be distributed in the same or different packages. Some installation programs combine installation and configuration, but if at all possible, there should be a way to cleanly separate installation and configuration. This will allow software to be easily deployed on large farms of computers. For the same reason, the installation process should be able to run without interaction from a user.

B. For system-wide configuration (as opposed to per-user configuration), there should be a configuration tool that allows the configuration to be edited easily. It is hard to have a general-purpose configuration tool that can do all possible configurations, but a tool that can handle the most common cases is essential. Ideally, this tool should be able to be used without user-interaction (by providing a set of inputs to the configuration tool before it runs) so that it can be used to deploy a farm. This tool should allow reconfiguration of the software after it has been configured for the first time. If the configuration requires changes to shared system files such as xinetd.conf, there should be an option to remove the changes.

   We describe in Appendix A how to make this configuration tool in a way that will enable your software to be as flexible as possible

C. Each software package has a wide variety of configurable options, and little can be said about what should be configurable. However, at a minimum the location of log files and state files (such as process id files), and the port numbers used by programs that access the network should be configuration. When possible, reasonable defaults (such as /var for log files used by privileged processes) should be used when configuration values are not supplied.

D. For software that requires processes to continually run, scripts should be provided to start and stop by the scripts. These should be System-V style scripts. These scripts should check to make sure that the software is properly configured. If it is not and some automatic configuration is possible, it should be done.

E. Upon completion of the configuration step, scripts should exist that allow a user to set up their environment to use the software.

For meta-packagers

F. Meta-packagers that intend to install on farms of computers should preserve the separation between installation and configuration steps. Meta-packagers that are not intended for installation on farms do not need to maintain this separation, if it provides a better user interface.

Appendix A: Configuration Scripts

Software developers should provide a configuration script for their software. As described above, this script should allow users to configure the software for the most common cases with an easy interface. We realize that the idea of having a script that is both easy to use and complete is impossible: these scripts will likely only be able to handle the most common configuration options. Given that incompleteness, software developers should realize that the more complete the scripts are, the more easily software can be configured on large farms.

By using some conventions, we hope we can generate an implementation that simplifies what a person needs to change when creating a metapackager setup: from the metapackager independent description (see Appendix D) generate a partially complete and correct description for any metapackager to use to make an installation with no editing, thus saving us countless hours of headaches. This would allow, for example, the creation of LCFG configuration components or Pacman configuration packages without user intervention.

In this appendix, we suggest using a configuration script that follows several conventions. For software that provides such a script, the conversion of metapackager independent descriptions can be more complete. For software that does not provide such a script, significantly more work needs to be done by the metapackager.

Configuration scripts that meet this recommendation are assumed to work with configuration data that can be expresses as name-value pairs. Other configuration data cannot be expressed with the current metapackager independent descriptions, but perhaps it will be in the future, if it is necessary.

Configuration scripts that conform to this specification are required to take at least four types of parameters, and each one can be specified as many times as desired.

--value <name> <value>
This sets a single named attribute to have the specified value.

--empty <name>
This specifies that the given attribute has the empty value. Not all software will need this.
Ex. GLOBUS_LOCATION = “”

--undefined <name>
This specifies that the given attribute should be undefined. Not all software will need this.
Ex. #GLOBUS_LOCATION=”” The attribute GLOBUS_LOCATION is not defined.

--input <file>
This specifies that there is a file that defines values. Each line of the file can contain a single --value, --empty, or --undefined , argument. These lines look just like they do on the command line, except that they are not subject to the substitutions performed by the shell. Names and values may be quoted (single or double) to allow spaces.

We expect that LCG will provide a sample configuration script that can be used as an example to begin writing your own configuration scripts that conform to this description.

References

[R1] LCFG Documentation,
http://datagrid.in2p3.fr/distribution/datagrid/wp4/edg-lcfg/documentation

[R2] PACMAN Documentation,
http://physics.bu.edu/~youssef/pacman/

[R3] VDT Documentation,
http://www.lsc-group.phys.uwm.edu/vdt/

[R4] NMI NSF Middleware Initiative,
http://www.nsf-middleware.org/