Enabling Grids for E-science in Europe

www.eu-egee.org

JRA1 All Hands Meeting, Padova, 15.11.2004

Installation and configuration of gLite services

Robert Harakaly, CERN
Integration Team

EGEE is a project funded by the European Union under contract IST-2003-508833

Contents

Enabling Grids for E-science in Europe

- Introduction
- Configuration management
- Service installation and set up
  - Demo
- C/C++ services & clients configuration

Problem

Enabling Grids for E-science in Europe

- We need to build an infrastructure supporting installation and configuration under several constraints:
  - Multi-platform environment (Linux, MS Windows)
  - Multiple packaging systems (RPM, tgz, MSI, …)
  - Compatible with standard installation tools (apt, yum, MS Installer, etc.)
  - Compatible with installation/configuration systems (Quattor, SMS, etc.)
  - But we cannot depend on any of them and we need to allow manual installation as well!

Enabling Grids for E-science in Europe

Configuration Management

Service

Enabling Grids for E-science in Europe

- We are using **Service-based installation and configuration** method
- The main building block is the **Service**: it is a set of components providing a given functionality
  - From our point of view the term service has much wider scope than normal, and we consider as a service not only the real services (glite-io-server, …), but also the clients and utilities (CA certificates are regrouped in the security service).
- A Service is always deployed as a simple unit
- Each service is described using an XML file called: ***Service Description File (SDF)***
- The SDF file contains all needed information about the service
  - the service components
  - the service dependencies
  - …
- The SDF is automatically created from templates by the build system

Service Description File (example)

Enabling Grids for E-science in Europe

An example of the service description file:

```
<service name="glite-io-client" version="1.2.3" release="1">
  <description>
    gLite IO-service client.
  </description>
  <components>
    <component name="glite-data-io-base" version="1.1.0" age="1" build="1 "
      arch="i386"/>
    <component name="glite-data-io-quanta" version="1.1.0" age="1" build="1 "
      arch="i386"/>
    <component name="glite-data-io-client" version="1.1.0" age="1" build="1"
      arch="i386"/>
    <component name="glite-data-io-gss-auth" version="1.1.0" age="1" build="1 "
      arch="i386"/>
    <component name="glite-data-config-service" version="1.1.0" age="1" build="1"
      arch="i386"/>
  </components>
  <dependencies>
    <external name="glite-essentials-cpp" version="1.0.1" age="1_EGEE" arch="i386"/>
    <external name="vdt_globus_essentials" version="VDT1.2.0rh9" age="1" arch="i386"/>
    <external name="gpt" version="VDT1.2.0rh9" age="1 " arch="i386"/>
  </dependencies>
</service>
```

Service Description File (example II)

- In addition to what was already mentioned, service description files enable to define **multi-role services**:

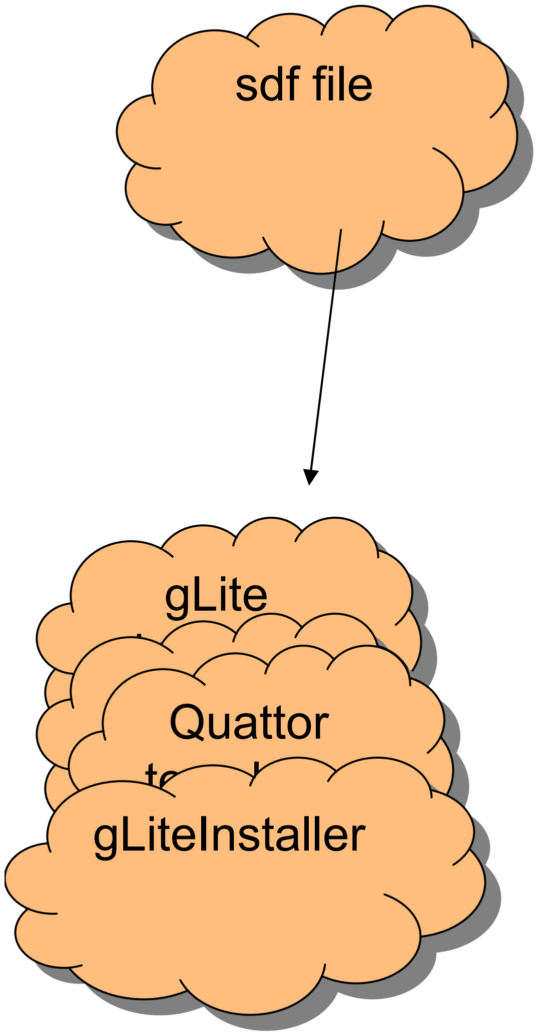
```
<service name="glite-io-server">
  <role name="fireman">
    <components>
      ...
      <component name="io-resolve-fireman"/>
    </components>
  </role>
  <role name="fr">
    <components>
      ...
      <component name="io-resolve-fr"/>
    </components>
  </role>
</service>
```

Derived files

Enabling Grids for E-science in Europe

- To simplify the management service description, all packaging and installation related scripts like .spec, glite installation scripts, quattor templates are derived from the service description file

Service description file



Service description file

```
<service name="glite-io-client">
  <description>
    glite-io-service-client
  </description>
  <components>
    <component name="glite-data-io-base"
      version="1.1.0" age="1" build="1" arch="i386"/>
    <component name="glite-data-io-quanta"
      version="1.1.0" age="1" build="1" arch="i386"/>
    <component name="glite-data-io-client"
      version="1.1.0" age="1" build="1" arch="i386"/>
    <component name="glite-data-io-gss-auth"
      version="1.1.0" age="1" build="1" arch="i386"/>
  </components>
  <dependencies>
    <external name="glite-essentials-cpp"
      version="1.0.1" age="1_EGEE" arch="i386"/>
    <external name="vdt_globus_essentials"
      version="VDT1.2.0rh9" age="1" arch="i386"/>
    <external name="gpt" version="VDT1.2.0rh9" age="1"
      arch="i386"/>
  </dependencies>
</service>
```

gLite installation utility

Multiplatform installation utility for gLite service/node installation. **It is in the design stage.**

- utility providing higher installation and configuration comfort with built-in target management (installation/uninstallation), online help, support of multiple package download protocols, different installation tools, etc.
- Built-in environment checking and configuration
- Graphical user interface in the interactive mode
- uses Quattor/Ant NDF/SDF files as a basic information for installation process
- installs the packages (RPMs) that are missing
- supports installation using different tools like yum, apt, SMS …, or using native built-in methods

Well known installer script:

```
#!/bin/sh
# Copyright (c) Members of the EGEE Collaboration. 2004
# See http://eu-egee.org/partners/ for details on the copyright holders
# For license conditions see the license file or http://eu-egee.org/license.html
# glite-io-client installer v. 0.2.0
# glite-io-client installer installs the gLite I/O Client
# Usage: glite-io-client_installer [-u|-v|--help]
#   -u      uninstall
#   -v      print version
#   --help  print script usage info
# Return codes: 0 - Ok
#   1 - if a file could not be downloaded
##########################################################
function parseRPMList(){
  newRPMLIST=""
  localRPMLIST=`rpm -qa`
  for i in $RPMLIST
  do
    g=`echo $i | sed -e 's/\.i386\.rpm//g'`
    g=`echo $g | sed -e 's/\.noarch\.rpm//g'`
    if [ -z "`echo $localRPMLIST | grep $g`" ]; then
      newRPMLIST="${newRPMLIST} $i"
    else
      echo "$i is already installed. It will be skipped."
    fi
  done
```

Node

- In general, the main deployment units are the nodes.
- Node consist of the set of installed services and (possibly) of some additional packages.
- Node is described by XML file called: Node Description File (NDF)
- NDF file is built from a corresponding set of service definition files.

Node can consist of:

- one service (LB), in that case the NDF can be identical to the corresponding SDF
- more services (LB + WMS), the NDF is a join of SDF files of LB service and WMS service

Use of NDF does not exclude the custom installations

Node Description file (example)

Enabling Grids for E-science in Europe

```
<node name="IO-FPS">
  <description>
    gLite data IO and FPS node
  </description>
  <services>
    <service name="glite-io-server" version="1.2.3-5">
      ...
    </service>
    <service name="glite-fts">
      ...
    </service>
    <service name="glite-fps">
      ...
    </service>
  </services>
  <depends>
    <!-- CA set -->
  </depends>
</node>
```

Service configuration file

- SDF and NDF files describe the services and their dependencies.
- They are mostly used during the installation process
- Service Configuration File contains the service configuration parameters
- Configuration types:
  - **Post-installation configuration:** Environment setup needed after the installation of the service create the working environment for the service (creation of DBs, …) in future should shrink to a minimum
  - **Service configuration:** Configuration of the service at startup and at run-time

Today the separation between these two types of configuration is fuzzy since post-installation configuration creates also the configuration files for the service.

We hope that the proposed configuration schema will clearly separate these two types of configuration.

Service configuration file

```
<service name="glite-io-client">
  <parameters>
    <io-client.Server value="lxb123.cern.ch"/>
    <io-client.ServerPort value="23456"/>
    <io-client.EncryptName value="true"/>
    ...
  </parameters>
</service>
```

Service configuration file II

```
<service name="IO-server">
  <role name="fireman">
    <parameters>
      ...
      <firemanEndPoint value="http://lxb2024.cern.ch:8080/
        org.glite.data.catalog-service-fr/services/
        FiremanCatalog"/>
    </parameters>
  </role>
  <role name="fr">
    <parameters>
      ...
      <replicaEndPoint value="http://lxb2024.cern.ch:8080/
        org.glite.data.catalog-service-fr/services/
        ReplicaCatalog"/>
      <disableDelagation value="false"/>
    </parameters>
  </role>
</service>
```

Service configuration file III

Enabling Grids for E-science in Europe

Configuration parameters are defined for each used role.
Each service/role can contain definitions of multiple instances:

```
<service name="XYZ">
  <instance name="instance1">
    <!--instance1 parameters -->
  </instance>
  <instance name="instance2">
    <!--instance2 parameters -->
  </instance>
</service>
```

Configuration file validation

Enabling Grids for E-science in Europe

- XML Schema file is used for validation of the service configuration file, configuration parameters documentation
- The goal is to minimize the configuration problems due to typos, non-correct values in the production use.
- Three levels of validation to enable flexible development
  - **Strict:** Validation errors will cause that the configuration file will be rejected
  - **Loose validation:** Parameters described in schema will be checked as with strict validation but allows an usage of additional parameters not described in schema
  - **No validation:** No validation of the configuration file

XML Schema

Enabling Grids for E-science in Europe

```
<schema>
  …
    <xs:element name="io-client.ServerPort">
     <xs:annotation>
      <xs:documentation>
       TCP port number of gLite IO server
      </xs:documentation>
     </xs:annotation>
     <attribute name="value">
      <xs:simpleType>
       <xs:restriction base="xs:integer">
        <xs:minInclusive value="1"/>
        <xs:maxInclusive value="65535"/>
       </xs:restriction>
      </xs:simpleType>
     </attribute>
    </xs:element>
  …
</schema>
```

Enabling Grids for E-science in Europe

Service installation and set up

Service installation and configuration

Enabling Grids for E-science in Europe

- For installation and configuration a deployment module for a number of services/nodes was created in the CVS
- A deployment module contains the service description file, service configuration file and configuration scripts.
- The build system derives all necessary files from the SDF, publishes them on the web page and packages the configuration file and scripts into the service deployment package
- We provide two types of distribution packages:
  - RPMs
  - Binary tarballs
- Supported installation:
  - Installation scripts for RPMs and tarballs
  - Quattor based installation

  Installation scripts and quattor templates are derived from the SDF by the build system.

  Downloadable from the gLite web page
  - Manual installation using standard rpm utility
- Post-installation configuration
  - Using configuration scripts
  - Quattor configuration (should arrive soon)

Post Installation Configuration

Enabling Grids for E-science in Europe

- Configuration parameters described in the service configuration file
- Configuration is done using Python modules
  - Each service (external, and internal) is represented by a Python class
  - Enables reusing of service configuration (condor, globus, …)
- Configuration files/scripts location:
  - ${GLITE_LOCATION}/etc/config
    - glite-global.cfg.xml
    - glite-<service>.cfg.xml
  - ${GLITE_LOCATION}/etc/config/scripts
    - Python configuration modules for each service

Configuration customization

Enabling Grids for E-science in Europe

- Possibility for the user to customize the gLite services
- Configuration order:
  - ${GLITE_LOCATION}/etc/config/glite-<service>.cfg.xml
  - /etc/glite.conf
  - ~/.glite/glite.conf
  - ~/.glite/glite-<service>.cfg.xml

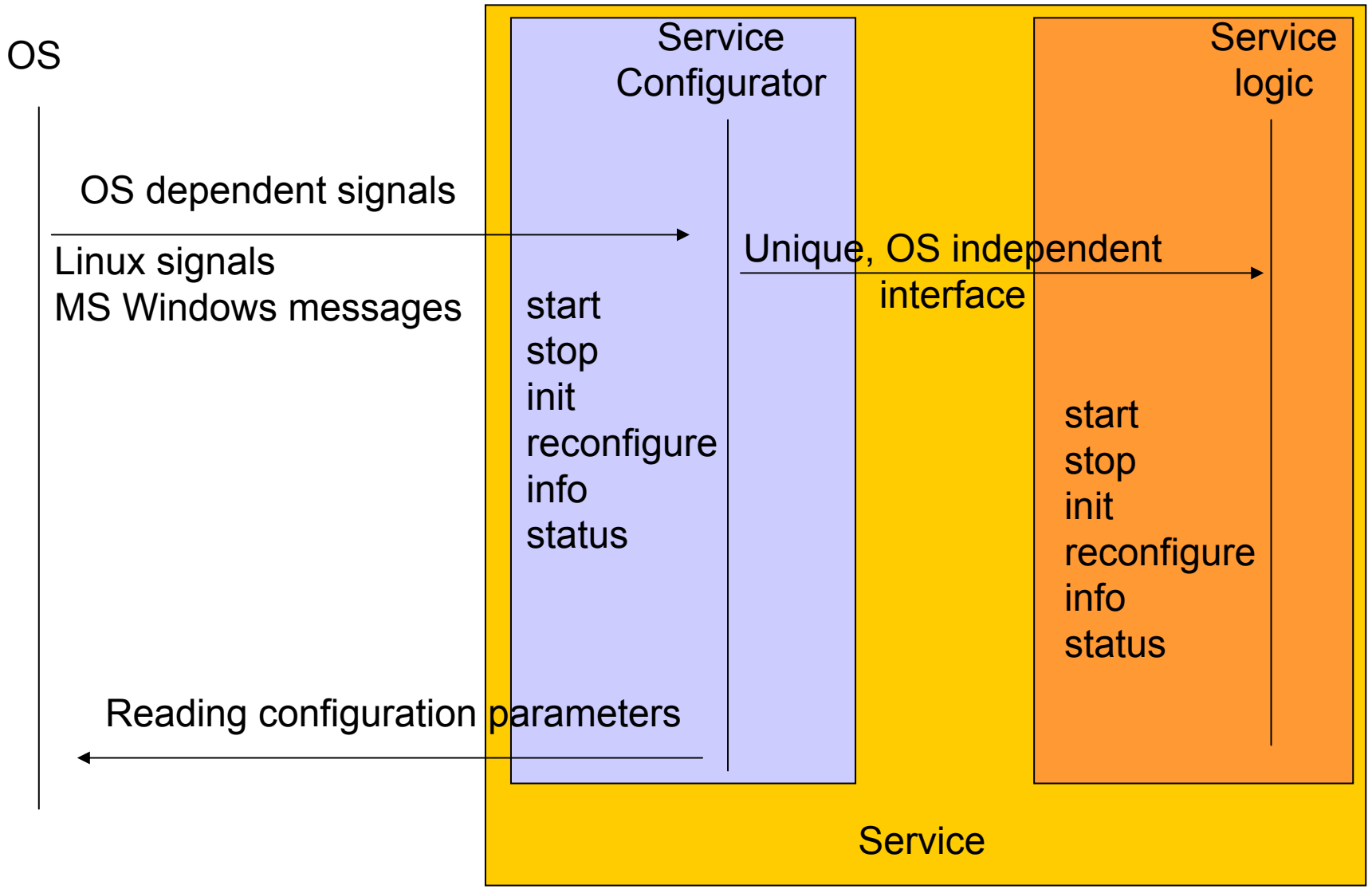
C/C++/Perl services and clients configuration

C/C++/Perl services and clients configuration

Enabling Grids for E-science in Europe

- Configuration based on the Service Configurator approach
- Service Configurator is designed as an internal part of the service
- In some cases this approach is already (partially) implemented (Tomcat, …) and the implementation is using this functionality
- Role of the Service Configurator :
  - Hide OS dependent issues from the service (messaging, …)
  - Create an unique interface for the management of the services
  - Load service configuration and forward it to the service logic
  - Enable addition of the additional common functionality (instrumentation) to the services, by providing an unique interface to it

Interfaces


OS
Service Configurator
Service logic
OS dependent signals
Linux signals
MS Windows messages
Unique, OS independent interface
start
stop
init
reconfigure
info
status
start
stop
init
reconfigure
info
status
Reading configuration parameters
Service

Interfaces II

C++ interface to be implemented in the service

```
/** Initialize the Component. Parameters passed during the
  * Initialization are not supposed to change (static
  * parameters) */
  virtual int init(const Params& params) = 0;
/** Start the Component. This method is called to start the
  * execution or restart it after a reconfiguration */
  virtual int start() = 0;
/** Stop the Component. This method is called to stop the
  * execution either is case of shutdown and in case of
  * reconfiguration */
  virtual int stop() = 0;
/** Reconfigure on the fly (dynamic parameters) the Component.*/
  virtual int reconfigure(const Params& params) = 0;
/** Pause Component. Optional */
  virtual int pause() = 0;
/** Resume Component. Optional */
  virtual int resume() = 0;
/** Get service information */
  virtual char *info() = 0;
/** Get service status */
  virtual char *status() = 0;
```

Client configuration

- C/C++/Perl/Python and JAVA clients can use simplified interface from the previous slide.
- C++ interface to implement in the client applications

```
/** Initialize the Component. Parameters
  * passed during the Initialization */
  virtual int init(const Params& params) = 0;
/** Reconfigure on the fly (dynamic parameters) the
  Component.*/
  virtual int reconfigure(const Params& params) = 0;
/** Get service information */
  virtual char *info() = 0;
/** Get service status */
  virtual char *status() = 0;
```

C++ service configuration demo

Enabling Grids for E-science in Europe

- The Service Configurator for C++ service demo will be presented by Paolo after this presentation
- Since the proposition and the Paolo's implementation were developed in parallel, the general approach is the same, but the implementation doesn't follow exactly the presented proposition
- From other side, the modification of implementation to follow the presented proposition, is simple

Hot issues

Enabling Grids for E-science in Europe

- We need to have more information about how to install/configure services:
- Post installation configuration
  - what actions are necessary?
- Service configuration
  - Which configuration parameters are valid/available
  - Complex parameter types: are there any? how to express them?
- Service description information
  - Service relationships: remote database connections, dependencies between remote services, how to express and validate them?