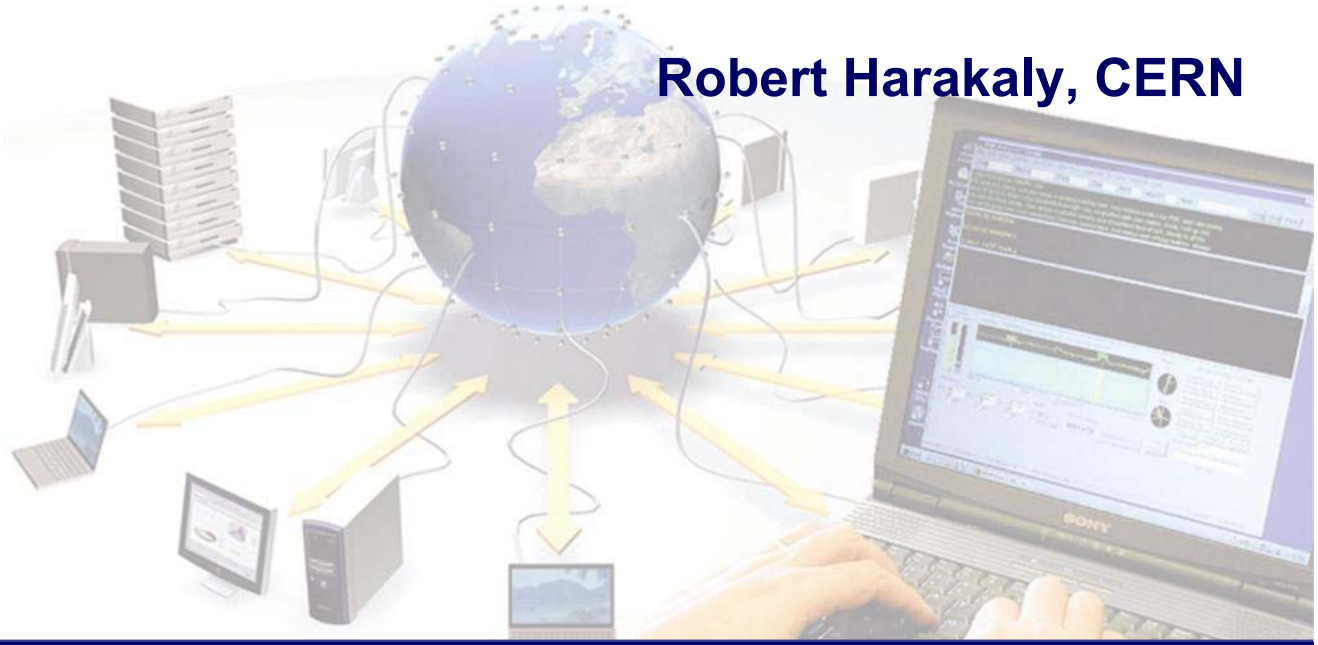
Enabling Grids for E-science in Europe

www.eu-egee.org

JRA1 All Hands Meeting, Padova, 15.11.2004

Installation and configuration of gLite services

Robert Harakaly, CERN



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

Contents

Enabling Grids for E-science in Europe

- Configuration management
- Installation
- C/C++ services & clients

Problem

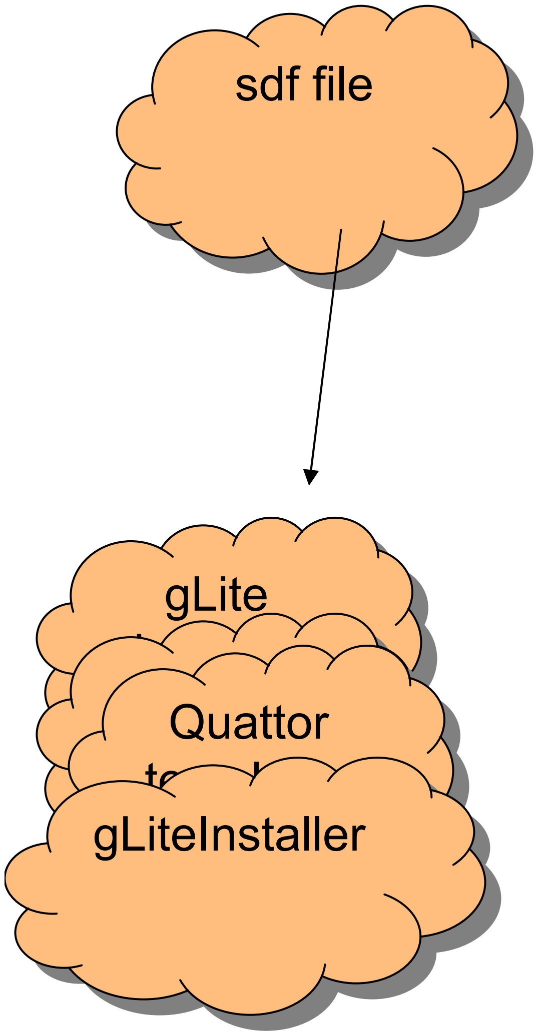
Enabling Grids for E-science in Europe

- We need to build structure supporting installation and configuration in:
  - Multi-platform environment (Linux, MS Windows)
  - Multi-package (RPM, tgz, MSI, …)
  - Supporting standard installation schemas (apt, yum, MSInstaller)
    - but not depending on any of them
    - Supporting additional installation as Quattor, etc.

Proposed solution

- Service based installation and configuration
  - Main building block is the service
  - Detailed description of the service and strict parameter control
    - Each service has its own XML description file (.sdf.xml) defining all needed information
    - Service description contains information (XML Schema) for configuration parameter checking
  - Any other description file (.spec, quattor templates, installation scripts, etc.) is derived from the service description file
  - Post-installation configuration done by using of the Python scripts
- It enables controlled installation and post-install configuration using different methods as:
  - gLiteInstaller
  - installation scripts
  - quattor installation
  - manual installation
  - …

Service description file



Service description file

```
<service name="glite-io-client-service">
  <description>
    glite-io-service client.
  </description>
  <components>
    <component name="glite-data-io-base"
      version="1.1.1" age="1" build="1" arch="i386"/>
    <component name="glite-data-io-quanta"
      version="1.0.0" 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>
```

```
template pro_software_glite_io_client;
# Generated automatically by the build system.
# 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 installation template v. 0.2.0
#
# Global dependencies
# glite-io-client-service dependencies
"/software/packages"=pkg_repl("glite-essentials-cpp","1.0.1-1_EGEE","i386");
"/software/packages"=pkg_repl("vdt_globus_essentials","VDT1.2.0rh9-1","i386");
"/software/packages"=pkg_repl("gpt","VDT1.2.0rh9-1","i386");
# glite-io-client-service RPMS
"/software/packages"=pkg_repl("glite-data-io-base","1.1.1-1","i386");
"/software/packages"=pkg_repl("glite-data-io-quanta","1.0.0-1","i386");
"/software/packages"=pkg_repl("glite-data-io-client","1.1.0-1","i386");
"/software/packages"=pkg_repl("glite-data-io-gss-auth","1.0.0-1","i386");
"/software/packages"=pkg_repl([illegible]);
"/software/packages"=pkg_repl("glite-config","0.2.0-2","noarch");
```

```
Summary: glite-io-service client
Name: glite-io-client-service
Version: [illegible]
Release: 1
Copyright: [illegible]
Vendor: EGEE
BuildArch: i386
Requires: glite-data-io-base, glite-data-io-quanta,
  glite-data-io-gss-auth, [illegible]
BuildRoot: [illegible]
Source: [illegible]
```

gLite installation utility

- Multiplatform deployment utility for the installation
- Uses directly the sdf file as a basic information for the installation process
- Installs individual packages (RPMS, MSI, …) or binary tarballs,
- using pluggable installation tools like yum, apt, SMS …, or using own 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
#
# The glite-io-client installer installs the gLite I/O Client
#
# Usage: [illegible]
#   -v print version
#   -h print script usage info
# Return codes: 0 - Ok
#   1 - if a file could not be downloaded
###############################################################
# Parse the RPMLIST to strip out the RPMS that are already installed
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
```

Service Description File

- In addition to what was already mentioned, service description file enables to define:
  - Muti-role services

```
<service name="glite-io-server">
   <role name="LSF">
     ...
   </role>
   <role name="PBS">
    ...
   </role>
   ...
</service>
```

Node Description file

```
<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

```
<service name="glite-io-client-service"
  version="1.1.1">
   <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="PBS">
    <parameters>
      ...
    </parameters>
  </role>
  <role name="LBS">
    <parameters>
      ...
    </parameters>
  </role>
</service>
```

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>
```

XML Schema

```
<schema>
  <xs:annotation>
    <xs:documentation>
      gLite IO-client configuration parameters schema
    </xs:documentation>
  </xs:annotation>
  <xs:element name="configuration">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="io-client.Server">
          <xs:annotation>
            <xs:documentation>
              Address of gLite IO server serving the client
            </xs:documentation>
          </xs:annotation>
          <attribute name="value" type="xs:string"/>
        <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>
```

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
  - -Loose
  - -No validation

Post Installation Configuration

Enabling Grids for E-science in Europe

- Configuration parameters described in the service configuration file
- Configuration itself is done by Python modules
  - Each service (external, and internal) is represented by Python class
  - Enables reusing of service configuration (condor, globus, …)
- Distributed in the service deployment package (RPM)
- Configuration files location:
  - ${GLITE_LOCATION}/etc/config
    - glite-global.cfg.xml
    - glite-<service>.cfg.xml
  - ${GLITE_LOCATION}/etc/config/scripts
    - Python configuration modules for each service
- See demo at the end of this presentation

Configuration customization

Enabling Grids for E-science in Europe

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

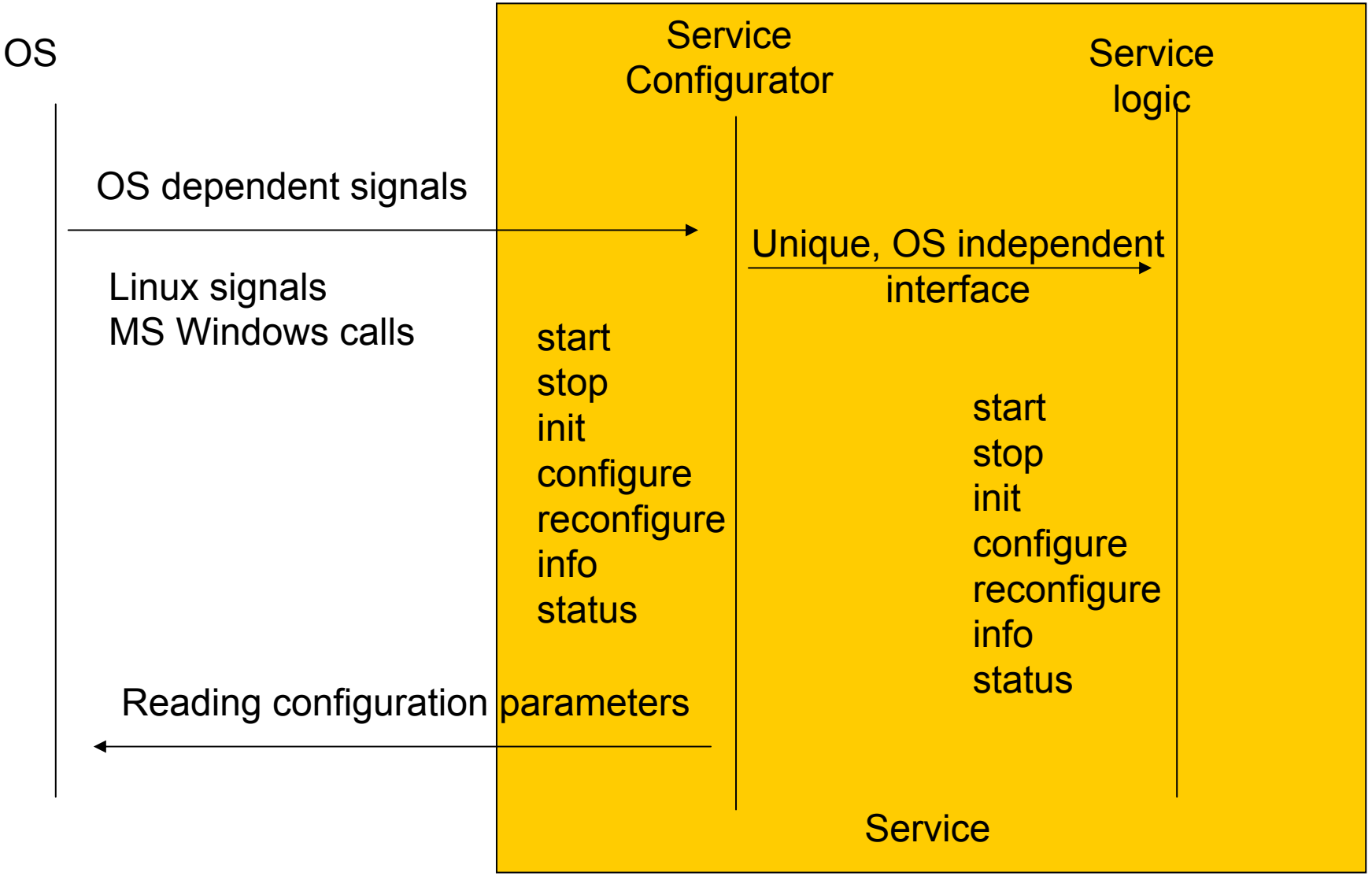
C/C++ services and clients configuration

C/C++ 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
  - Enable addition of the additional common functionality (monitoring, …) to the services, by providing an unique interface to it

Interfaces


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

Interfaces II

Enabling Grids for E-science in Europe

```
/** 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 onfly (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;
```

Client configuration

Enabling Grids for E-science in Europe

- C/C++ and JAVA clients can use simplified interface from the one on the slide 13.

C++ service configuration demo

Enabling Grids for E-science in Europe

- The Service Configurator for C++ service demo will be presented by Paolo at the end of this session.
- 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

- Post installation configuration
  - Actions
- Service configuration
  - Configuration parameters
  - Complex parameter types
- Service description
  - Service relations (mostly databases)