# AN XML-BASED CONFIGURATION AND MANAGEMENT SYSTEM FOR THE GLITE MIIDDLEWARE

Joachim Flammer, Alberto Di Meglio, Guillermo Diez-Andino Sancho,
Robert Harakaly, Lanxin Ma, Marian Zurek
CERN, Geneva, Switzerland

*Abstract*

gLite is the next generation middleware for grid computing, composed out of more than 25 different services, implemented in different languages, using different technologies and coming with individual configuration needs. Past experience has shown that configuration and management of such a distributed system are one of the biggest challenges.

In order to ease the configuration and deployment of the different gLite services, we have developed a configuration system that offers the administrators of gLite an easy to use, extendable and flexible system across all services, yet fitting the needs of the different systems. It is based on an XML-encoded common configuration storage format across all services with pre-configured configuration templates guiding the user in the proper selection of the configuration scenarios.

In a second step, we are working on a prototype to implement a common management layer into the different services in order to allow remote control and monitoring of the services.

## INTRODUCTION

gLite [1] is the next generation middleware for grid computing. Born from the collaborative efforts of more than 80 people in 12 different academic and industrial research centres as part of the Enabling Grids for E-SciencE (EGEE) project [2], gLite provides a leading-edge, best-of-breed framework for building grid applications, tapping into the power of distributed computing and storage resources across the internet.

Currently, the gLite middleware (figure 1) consists of more than 25 different services, implemented in different languages, using different technologies and coming with individual configuration needs.

## CONFIGURATION AND MANAGEMENT

### *Challenges*

The development of a substantial part of the software has been started before gLite in different projects, e.g. within the scope of the European Datagrid Project (EDG) [3] and the LHC Computing Grid project (LCG) [4]. The consequence of this evolutionary development is a lack of a common configuration system – rather each service is equipped with its own specific configuration. In addition, no management functionality is built into the services.

As gLite can be run in multiple operational scenarios, supporting hundreds of configuration options to adapt for the site specific environment, setup and composition of the middleware, configuration can be quite challenging and requires in-depth knowledge of the different services. In addition, the middleware is run in a widely distributed environment, spread over different sites and continents.

Typical operational questions are: *How to configure a set of 1000 distributed machines? How to find out about the configuration of a specific service? How to restart a service without disturbing other services?*

Past experience has shown that configuration and management are one of the biggest challenges of such a distributed system.

### *Design requirements*

When designing a new configuration and management system for a distributed set of middleware components, several aspects have to be taken into account in order to fulfil the needs of the user community as well as the different services. We consider the following seven requirements as the corner stone for a successful implementation of a distribute configuration and management system:

- *Modularity*
  Installation and configuration are two separate steps in the process towards a running service. Also, they require different approaches for updates and changes. One of the fundamentals of the system has to therefore be a clear separation between these two steps.
- *Homogeneity*
  For a system composed out of more than 25 different services, a user clearly does not want to learn 25 different configuration systems. The configuration system therefore has to offer a unique interface to the user both in defining the configuration of the system as well as interacting with the services (e.g. starting them).

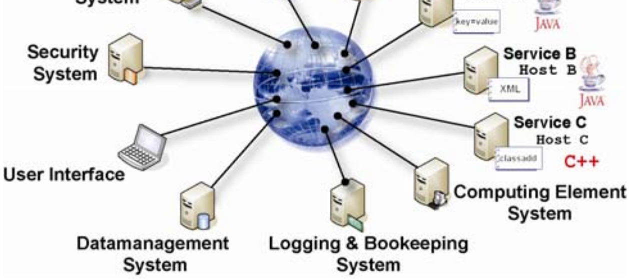


Figure 1: gLite middleware.

- *Flexibility*
  The composition of the gLite services, the machines and the network depends very much on the individual setup. Different users therefore have very dissimilar ways of configuring the system according to their needs. The configuration system has to be flexible enough to cope with all aspects a user might require.
- *Guidance*
  Configuring a distributed system that is composed of different services can already be quite challenging due to the sheer amount of available and necessary configuration parameters. The configuration system has to give the user every possible means of support to understand the different configuration options and to choose the subset of parameters that have to be adapted for the particular setup.
- *Documentation*
  Although everybody agrees to the fact that documentation is essential, most of the documentation is left to the end of the development process as an add-on. A better approach is to couple the documentation directly to the configuration and allow an automatic creation of the different kinds of documentation material.
- *Remote instrumentation*
  Having a distributed system running 24h/24h requires the possibility for remote control in order to learn about the present configuration, understand the problem as well as to act on the services without disturbing other services. The configuration and management system therefore has to implement the necessary instrumentation in each service to allow remote intervention.
- *Security*
  Certainly configuration information can contain a set of vulnerable information like passwords. In addition, the control of the services has to be clearly restricted to a pool of authorised users. The configuration and management system therefore has to have the same level of security as the underlying grid services.

Having studied presently available configuration and management systems like YAIM [5], none of them has been found to fully support all these requirements. We have therefore developed a common configuration management system for the gLite middleware, taking into account all seven requirements.

The implementation of the different requirements via a gLite wide configuration system is done in two steps: The first step implements a common configuration layer, isolating the user from the diversity and complexity of the individual configuration systems and offering a unique interface to configure, start and stop the individual gLite services. The second step foresees the implementation of the necessary configuration interfaces directly into the services together with a distributed infrastructure for the configuration and management of the different services.

While the first part is already implemented in the gLite middleware, the second part currently exists as a prototype. The second part of this paper will describe the two parts in details.

## GLITE CONFIGURATION SYSTEM

### *Separation of installation and configuration*

As described earlier, the gLite middleware is split into several services that can be installed separately. Each service comes with its own deployment module for installation and is presently distributed as a set of RPM's as well as tarballs. Each service deployment module comes with all the necessary configuration elements to configure the service: A set of configuration file templates, containing as much as possible preconfigured values, the configuration script, to configure and control the service, as well as the necessary documentation.

All the gLite services share the same format for the configuration files and the same (command line) interface for the configuration scripts.

### *Configuration scripts*

The configuration scripts are a set of platform independent, both service specific as well as common (e.g. for the MySQL configuration) scripts, implemented in Python. Python has been chosen because it combines the advantages of a scripting language with an object oriented approach. The script offers the possibility to configure, start and stop a service, to query the configuration information, to show the status of a service as well as some service specific options (e.g. removing part of a service).

### *Configuration files*

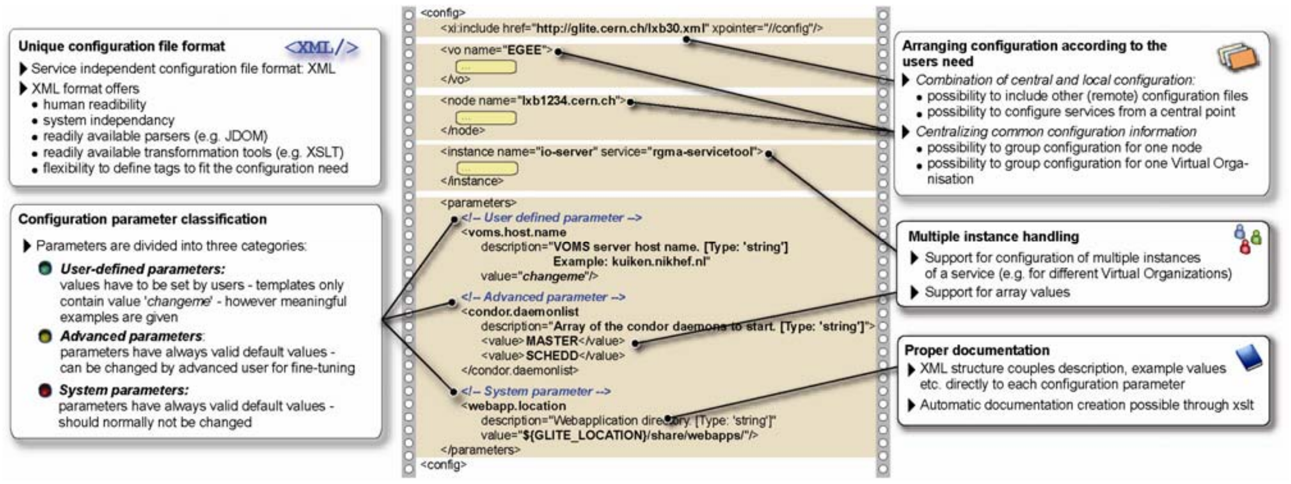
The central part of the configuration system is based on a common set of configuration files, implemented using XML, that contain all of the necessary configuration information for the different services. While still being human readable, XML allows a proper arrangement as well as automated treatment of the configuration information using standard technologies like XSLT transformation. Figure 2 shows an excerpt of a gLite configuration file together with the main characteristics of the syntax of the configuration files.

The common configuration files and scripts protect the user from the underlying complexity and diversity of the individual services. Rather, the configuration system presents the user with a homogeneous set of configuration files and formats, as well as a command line interface with a unique look-and-feel.

### *Configuration parameter classification*

In order to guide the user in choosing the set of configuration parameters needed to set up the configuration, all of the configuration parameters are split up into three categories:

*User-defied parameters* contain values that have to be set by the users. In the configuration file templates, these values are filled with the value of 'changeme' as no default values exist for these values. However meaningful examples are given as part of the parameter description. Passwords and central machine names are examples of such values.



**Unique configuration file format** <XML/>

- Service independent configuration file format: XML
- XML format offers
  - human readibility
  - system independancy
  - readily available parsers (e.g. JDOM)
  - readily available transformation tools (e.g. XSLT)
  - flexibility to define tags to fit the configuration need

**Configuration parameter classification**

- Parameters are divided into three categories:
  - ● ***User-defined parameters:*** values have to be set by users - templates only contain value '*changeme*' - however meaningful examples are given
  - ● ***Advanced parameters***: parameters have always valid default values - can be changed by advanced user for fine-tuning
  - ● ***System parameters:*** parameters have always valid default values - should normally not be changed

```
<config>
    <xi:include href="http://glite.cern.ch/lxb30.xml" xpointer="//config"/>
    <vo name="EGEE">
        ...
    </vo>
    <node name="lxb1234.cern.ch">
        ...
    </node>
    <instance name="io-server" service="rgma-servicetool">
        ...
    </instance>
    <parameters>
      <!-- User defined parameter -->
      <voms.host.name
          description="VOMS server host name. [Type: 'string']
                       Example: kuiken.nikhef.nl"
          value="changeme"/>
      <!-- Advanced parameter -->
      <condor.daemonlist
          description="Array of the condor daemons to start. [Type: 'string']">
          <value>MASTER</value>
          <value>SCHEDD</value>
      </condor.daemonlist>
      <!-- System parameter -->
      <webapp.location
          description="Webapplication directory. [Type: 'string']"
          value="${GLITE_LOCATION}/share/webapps/"/>
    </parameters>
<config>
```

**Arranging configuration according to the users need**

- *Combination of central and local configuration:*
  - possibility to include other (remote) configuration files
  - possibility to configure services from a central point
- *Centralizing common configuration information*
  - possibility to group configuration for one node
  - possibility to group configuration for one Virtual Organisation

**Multiple instance handling**

- Support for configuration of multiple instances of a service (e.g. for different Virtual Organizations)
- Support for array values

**Proper documentation**

- XML structure couples description, example values etc. directly to each configuration parameter
- Automatic documentation creation possible through xslt

Figure 2: The central part of the gLite configuration system: excerpt of an XML configuration file

*Advanced parameters* always come with default values and normally do not need be changed by the user. However, advanced users can change these values in order to fine-tune the behaviour of a service.

*System parameters* also always come with default values. However, in contrast to the advanced parameters, their values should normally not be changed as the service depends intrinsically on the specific value.

### *Multiple instance handling*

The configuration information of gLite cannot be represented using a flat structure. Therefore the configuration system supports both array values for configuration parameters as well as multiple instances for a service, e.g. the configuration of separate instances of the same service running on the same node.

### *Arranging configuration information according to the users need*

Normally, all configuration files are stored in a specific directory on the machine the service is running on. As several services can be run in parallel on a machine, the configuration files are split into one global configuration file and several service specific configuration files.

However, configuring services across multiple nodes requires two additional approaches: Firstly, information, specific to one node can be combined into one node section. Secondly, configuration that is common to the entire site can be stored in a common configuration storage in order not to duplicate configuration information. The host of the central information system is a good example for such information. To allow for this central *site configuration,* the configuration system has the possibility to include other (central or remote) configuration files via the xi:include functionality of XML.

Finally, as the middleware supports multiple Virtual Organizations (VO) that are grouping a set of users and some services having a separate instance for each VO, configuration information can be arranged according to the VO. Service instances can be created automatically iterating over the list of defined VOs.

The grouping of configuration information together with the possibility to overwrite a configuration parameter by a later occurrence and reuse them via their reference in other parameters, allows the user to setup its configuration space in his or her preferred way.

When running the configuration scripts, the configuration system determines the correct value taking into account the hierarchy, grouping and overwriting of the configuration values as well as the iteration over the different VOs.

### *Interlinked documentation*

Proper documentation is one of the essential aspects of any system. In terms of configuration, finding the meaning of a specific configuration value might not be straight-forward. Furthermore, as configuration information is updated, documentation often stays behind.

The gLite configuration system therefore couples the documentation with the configuration value. For each of the values, the documentation contains a description of the parameter together with an example, the type and, where applicable, the unit.

### *Automated processing of the configuration information*

A final advantageous feature of XML is XSLT, a standard, powerful method for performing automatic transformations. Presently, the gLite configuration system makes use of this for automatic generation of the full parameter set (see ‘Arranging configuration information …’ above). Far more applications, however, are possible such as the automated creation of documentation.

Further details about the configuration can be found in the gLite installation guide [5].

## DISTRIBUTED CONFIGURATION AND MANAGEMENT SYSTEM

Clearly, the common configuration system can only be the first step in an implementation of a configuration management system. While implementing a common layer of configuration offers the user a unique interface, it does create an unnecessary extra layer between the service and the user. Also, despite being able to store the configuration information remotely, all interaction with services is only locally possible by connecting to the corresponding machines. In order to solve the second part of the challenge, we are working on a prototype to implement a gLite wide unique service instrumentation interface for direct configuration and management together with a central management service. Figure 3 gives an overview of the prototype implementation.

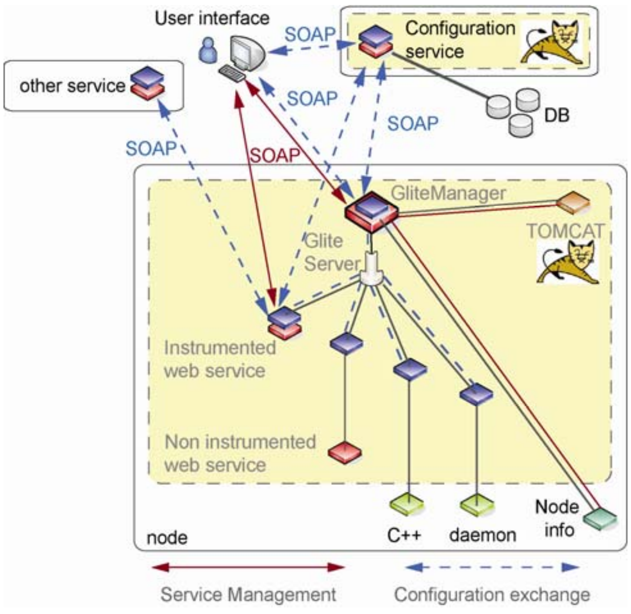


Figure 3: gLite configuration management system prototype

There are two entry points to control a given service from the management point of view:

A common *GliteService* interface will be responsible for the configuration of the individual service as well as providing the management interface for the control of the service. For gLite services that are implemented as Java web services inside Tomcat, this interface is implemented as a common, extendable interface inside the service. For non-instrumented Java web services as well as C++ services and daemons, a web service stub, implementing the corresponding interface, acts as an interface between the service and the outside world. The outside communication (e.g. to a user interface) is based on SOAP.

A central *GliteManager*, on the other hand, acts as a central entry point to a node by collecting the information from different sources (all of the services running on the node, the tomcat container as well as information from the node) and by enabling the central management of the different parts. The GliteManager implements the standard GliteService interface thus presenting itself as a standard gLite service. The GliteManager is implemented as a Java web service inside TOMCAT based on JMX. All inside communication to TOMCAT and to the other web application is done via JMX. All outside communication uses SOAP.

A user is then able, via a user interface, to either contact the central GliteManager on a node to find out about the presence and status of the different services or to directly contact the different gLite services implementing the GliteService interface.

The configuration of the services on the other hand is served via one or more *Configuration Services* that support several back ends for the storage of the configuration information such as databases or files. The user can define /query/update the configuration of a service via a user interface that connects to the configuration service. A gLite service then gets its configuration via the GliteService interface described above from the gLite Configuration Service. The service can equally contact another gLite service via the same interface to learn about the configuration of the other service, adapting its own configuration in this way.

Finally, to assure sufficient security, the system will be based on the normal grid security services using the Virtual Organisation Membership Service (VOMS) that is already used for normal grid operations within gLite.

## CONCLUSIONS

Our experience has shown that a proper configuration system is essential for the success and usability of a service, especially when running in a distributed environment and using a composite system.

The gLite configuration system has been successfully used in production for a year, constantly adapting to new requirements as they emerge. The initial feedback from the users has shown that the implemented configuration system has been well received, fitting their needs and adapting very well to their requirements for configuring the gLite middleware. One common request is, however, to improve the tools to manipulate the configuration files, as XML is sometimes considered not sufficiently human readable.

For the configuration management system, the first studies and prototype implementation look very promising. The management system however requires changes inside the different gLite services and therefore has to be carefully balanced with the different services.

## ACKNOWLEDGEMENTS

This work has been funded by the European Communities under contract IST-2003-508833.

## REFERENCES

[1] http://www.glite.org
[2] http://egee-intranet.web.cern.ch/egee/intranet/gateway.html
[2] http://eu-datagrid.web.cern.ch/eu-datagrid/
[3] http://lcg.web.cern.ch/LCG/
[4] http://grid-deployment.web.cern.ch/grid-deployment/documentation/LCG2-Manual-Install/
[5] gLite Installation Guide, http://glite.web.cern.ch/glite/documentation